Fault Detection Methods for Vapor-Compression Air Conditioners Using Electrical Measurements

by

Christopher Reed Laughman

S.B., Massachusetts Institute of Technology (1999)
M.Eng., Massachusetts Institute of Technology (2001)

Submitted to the Department of Architecture
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Architecture: Building Technology

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
Department of Architecture
12 August 2008

Certified by .....
Steven B. Leeb
Professor of EECS and Mechanical Engineering
Thesis Supervisor

Certified by .....
Leslie K. Norford
Professor of Building Technology
Thesis Supervisor

Certified by .....
Peter R. Armstrong
Associate Professor of Mechanical Engineering
Masdar Institute of Science and Technology
Thesis Supervisor

Accepted by .....
Julian Beinart
Professor of Architecture
Chair of the Department Committee on Graduate Students

Thesis Committee

Thesis Supervisor: Steven B. Leeb
Title: Professor of EECS and Mechanical Engineering
University: Massachusetts Institute of Technology

Thesis Supervisor: Leslie K. Norford
Title: Professor of Building Technology
University: Massachusetts Institute of Technology

Thesis Supervisor: Peter R. Armstrong
Title: Associate Professor of Mechanical Engineering
University: Masdar Institute of Science and Technology

Thesis Reader: Leon R. Glicksman
Title: Professor of Building Technology
University: Massachusetts Institute of Technology

Thesis Reader: James L. Kirtley
Title: Professor of Electrical Engineering
University: Massachusetts Institute of Technology

Thesis Reader: Steven R. Shaw
Title: Associate Professor of Electrical Engineering
University: Montana State University

Fault Detection Methods for Vapor-Compression Air Conditioners Using Electrical Measurements

by

Christopher Reed Laughman

Submitted to the Department of Architecture
on 12 August 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Architecture: Building Technology

Abstract

This thesis proposes novel methods that use measurements of electrical terminal variables to identify common mechanical faults in vapor-compression air-conditioners. The importance of air-conditioning in many applications and the current cost of energy both provide powerful incentives for developing fault detection methods, as faults can have a significant impact on the system's functionality and efficiency. While many extant fault detection and diagnostic (FDD) methods depend upon arrays of mechanical sensors, concerns about sensor reliability and the overall complexity of these methods motivated this research into electrically-based FDD methods, which typically incorporate smaller numbers of more reliable sensors. These electrically-based methods use models of the electromechanical energy conversion process to correlate observed changes in the electrical variables to changes caused by faults in the mechanical load. Such an approach allows both electrical and mechanical faults to be identified via the same sensor apparatus, and makes it possible to identify faults that manifest themselves on a wide range of timescales.

FDD methods for three different classes of common faults are studied in this research. The first diagnostic method identifies blockage or leakage in a duct via electrical measurements made at the fan motor terminals. The estimates of the motor's speed and torque developed at the operating point are used in tandem with a fan curve to directly estimate the airflow through a duct system without any additional mechanical measurements. This method was experimentally tested and validated on a commercially available air handler and duct system. In the second class of faults studied, liquid refrigerant, rather than vapor, enters the cylinder of a reciprocating compressor during operation. Since the higher cylinder pressures that result can cause substantial damage and are difficult to measure directly, a method for detecting this fault is proposed that only uses observations of the compressor voltage and current. The performance of this fault detection method was also experimentally validated with electrical and mechanical measurements on a semi-hermetic compressor. The final diagnostic method detects refrigerant leakage in a residential air-conditioning system by identifying changes in the system's cycling behavior. This method also uses measurements of the compressor's electrical power, as well as a small set of temperature measurements, to determine the presence of the fault. This fault detection method was developed and tested on an occupied residence.

Thesis Supervisor: Steven B. Leeb
Title: Professor of EECS and Mechanical Engineering

Thesis Supervisor: Leslie K. Norford
Title: Professor of Building Technology

Thesis Supervisor: Peter R. Armstrong
Title: Associate Professor of Mechanical Engineering
Masdar Institute of Science and Technology

for Katie

Acknowledgments

"You need the willingness to fail all the time.

You have to generate many ideas and then you have to work very hard only to discover that they don't work. And you keep doing that over and over until you find one that does work."

- John W. Backus

Just as it is often said that it takes a village to raise a child, a village-like number of people have helped me as I have attempted to rear and nurture this thesis from its gestational phases to the point in its development represented by the document that you hold in your hands. I owe a tremendous debt to my advisors, Prof. Steve Leeb, Prof. Les Norford, and Prof. Peter Armstrong. I first began working for Steve Leeb as a UROP in January of 1997, and his enthusiasm for engineering and patience in teaching me how to design, build, and analyze complex electromechanical systems is largely responsible for making me the engineer that I am today. My association with Les Norford began in 2001 during my time as a staff research member in LEES, and he has taught me a great deal not only about buildings and building systems, but also about understanding the scope and context of the research I have been involved in, rather than just the research itself. This research grew out of a 3-day trip in 2003 that I took to Purdue University with Professor Peter Armstrong, and he has both helped me to better understand the range of complex interactions which happen in these common air-conditioning systems and showed me how to better

integrate the theoretical material learned in classes with a practical, hands-on approach to building air-conditioning systems.

I would also like to thank the other members of my thesis committee. Professors Jim Kirtley and Leon Glicksman provided valuable insights and helped to clarify this research as it progressed. Prof. Steve Shaw has subtly encouraged and guided me since he helped to supervise my M.Eng. thesis, and has taught me a tremendous amount about solving practical system identification and inverse problems.

Many thanks are also extended to the cadre of undergraduate and graduate students whom I have known during the epoch of my time in LEES. Rob Cox, now of the University of North Carolina, Charlotte, provided me with valuable second, third, and fourth opinions about much of this research as it developed, as well as the experimental platform discussed in Chapter 4, and his sage advice and good humor have been invaluable at all points. Jim Paris helped immeasurably with a wide range of microcontroller and data acquisition problems, and also taught me how to really use Linux and fixed my computer on an all-too-regular basis. Roderick LaFoy and Ryan Bavetta assisted in mechanical design projects with their extensive Solidworks skills and built mechanical systems that are both functional and beautiful. Warit Wichakool also helped solve some challenging circuit design problems in this research and has been a stalwart support throughout. Many other individuals who have come and gone from LEES during my sojurn here have contributed to the feeling of creativity and camaraderie which suffuses this laboratory and have made my experience here all the richer.

This research benefited from the wisdom and experience of a number of individuals in industry. In particular, the results presented in Chapter 2 were made possible in part by data provided by Tim May at Lau Fan Corp.; his patience and willingness to answer my questions improved my understanding of fans immensely. Similarly, the staff of Abco and United Refrigeration, Inc. of Somerville were always willing to answer questions about air-conditioning components and practice. Jim Braun and Haorong Li of Purdue University provided the initial support to investigate electrically-based fault detection methods experimentally, and provided an excellent example of what could be done with fault detection systems for air-conditioners.

My many friends and associates outside MIT have also played an important role in

this thesis by encouraging me and giving me opportunities to explore life outside of 10-050. Glassblowing partners and friends have helped me to explore intuitive and expressive ways of thinking, and have helped to revive my creativity at crucial junctures. My parents and brother have provided unflagging support and have sustained me throughout, lending an understanding ear and words of encouragement when it was needed. Finally, I must thank my lovely wife Katie, whom I got to know as I was beginning this doctoral program, and who has been by my side through all of the failures and successes experienced in the course of this work. Her ready smile and caring ways have carried me through the length of this endeavor; this thesis is dedicated to her.

Essential funding and support for this research was provided by a wide variety of sources, including the Grainger Foundation, the National Science Foundation, NASA Ames Research Center, NEMOmetrics, Inc., and the Office of Naval Research under the ESRDC program. Support and much edification was also provided by the staff of 6.002, with whom I had the privilege of being a teaching assistant for three semesters.

Contents

1 Introduction **31**

1.1 Fault Detection and Diagnostic Systems 34

1.1.1 Fault Detection 35

1.1.2 Fault Diagnosis 41

1.1.3 Fault Evaluation and Decision 43

1.2 Previous Work 45

1.2.1 Air Conditioning System Overview 45

1.2.2 Common Faults in Air-Conditioners 47

1.2.3 FDD Methods for Air-Conditioners 52

1.3 Electrically-based FDD techniques 56

1.4 Research Overview 60

2 Airflow Diagnostics **63**

2.1 Motivation 63

2.1.1 Background 64

2.1.2 Previous Work 69

2.2 Structure of the Diagnostic Method 73

2.3 Rotor Speed Estimation 78

2.4 Torque Estimation 83

2.4.1 Induction Machine Model 84

2.4.2 Motor Simulation 88

2.4.3 Parameter Identification 93

2.5 Fan Dynamics 103

2.6 Experimental Apparatus 111
2.7 Speed Estimation Results 118
2.8 Torque Estimation Results 127
2.8.1 Parameter Estimation 127
2.8.2 Torque-Speed Characteristics 133
2.8.2.1 Hot motor: current-based estimation 135
2.8.2.2 Cold motor: current-based estimation 137
2.8.2.3 Hot motor: torque-speed and current-based estimation . . 139
2.8.2.4 Cold motor: torque-speed and current-based estimation . . 143
2.8.2.5 Summary of estimation for the four different conditions . . 145
2.8.3 Effects of different operating conditions on parameter estimation . . 147
2.9 Airflow estimation 149
2.10 Summary 161

3 Liquid Slugging Diagnostics **163**
3.1 Motivation 163
3.1.1 Background 164
3.1.2 Previous Work 168
3.2 Structure of the Liquid Slugging Diagnostic Method 179
3.3 Experimental Platform 191
3.3.1 Mechanical Systems and Sensors 192
3.3.2 Electrical Systems and Sensors 209
3.4 Experimental Results 213
3.4.1 Preprocessor Characterization 214
3.4.2 Nonslugging Compressor Characterization 217
3.4.3 Steady-State Liquid Slugging 226
3.4.4 Transient Liquid Slugging 236
3.5 Summary 248

4 Refrigerant Charge Diagnostics **249**
4.1 Motivation 249
4.1.1 Background 250

4.1.2 Previous Work 253
4.2 Cycling Detection Methodology 256
4.3 Cycling Diagnostics Using Temperature Measurements 264
4.4 Method Validation 277
4.5 Summary 282

5 Discussion & Future Work **285**
5.1 Airflow Diagnostics 287
5.2 Liquid Slugging Diagnostics 290
5.3 Refrigerant Charge Diagnostics 293

A Fan Estimation Code **297**
A.1 Tachometer Data Processing 297
A.1.1 tachproc2.c 297
A.2 Speed Harmonic Processing 300
A.2.1 speed_est.m 300
A.2.2 spect_gen.m 300
A.3 Estimation Preprocessing 301
A.3.1 volt_est.m 301
A.3.2 GNL.m 303
A.3.3 fdjac2.m 304
A.3.4 findk.m 305
A.3.5 sinfunc.m 305
A.4 Motor Parameter Estimation 305
A.4.1 motorest1.c 305
A.4.2 motorest2.c 321
A.4.3 filter_gen.m 340
A.5 Airflow Prediction Generation 341
A.5.1 speed_flow_proc.m 341
A.5.2 flow_est.m 342

B Liquid Slugging Information **345**

B.1 Solenoid Control Program 345

B.1.1 slug_delay.c 345

B.2 Matlab DQ0 Preprocessor 350

B.2.1 dq0preproc.m 350

B.2.2 GNL3.m 353

B.2.3 findk3.m 354

B.2.4 fdjac23.m 355

B.2.5 cosfunc3a.m 355

B.2.6 cosfunc3b.m 356

B.3 C DQ0 Preprocessor 356

B.3.1 dq0prep.c 356

B.4 Perl Helper Code 376

B.4.1 process.pl 376

B.4.2 indexsift.pl 380

B.5 Compressor Head Drawings 382

B.6 Electrical Schematics 387

C Refrigerant Charge Information **391**

C.1 Thermal Instrumentation 391

C.2 Data Synchronization Scripts 393

C.2.1 eventid1.m 393

C.2.2 syncdata.m 398

C.2.3 postprocess.m 401

C.3 Method Testing 401

C.4 Building Simulation Code 404

C.4.1 house_real.m 404

C.4.2 rfeuler.m 406

C.4.3 rhouse.m 406

List of Figures

1-1 The basic structure of an FDD system (adapted from [73]). 34

1-2 Fault detection using a signal model-based approach (adapted from [73]). . 37

1-3 Fault detection using a system model-based approach (adapted from [73]). . 37

1-4 Multiple faults can exhibit the same fault symptom, and a single fault can exhibit many different fault symptoms. 41

1-5 Schematic illustration of the multiple-simultaneous fault problem. 41

1-6 Schematic diagram of the air-conditioner. 46

2-1 Schematic diagram and picture of air handler. 66

(a) Picture of air handling unit. 66

(b) Schematic diagram of air handling unit. 66

2-2 Picture of broken mount for fan motor in a field-installed AHU, from [31]. . 67

2-3 Structure of the airflow estimation method. 76

2-4 Picture of effect of rotor slot harmonic, from [74]. 79

2-5 MMF pattern of a representative four-pole, 44-slot rotor, from [105]. 81

2-6 Vector representation of the input and output of the Park transformation for an arbitrary variable f, adapted from [79, p. 136]. 85

2-7 Equivalent circuit model for one phase of an induction machine. 88

2-8 Plot of the observed and fitted voltage waveform. The observed waveform is in blue, and the fitted waveform is in red. 91

2-9 Plot of the observed voltage waveform superimposed on a plot of the residual $y_{obs} - y_{fit}$ between the observed voltage and the correponding fitted sinusoidal voltage. The observed voltage is in blue, while the residual is plotted in red. 91

2-10 Plot of the observed and fitted voltage waveforms running the simulation for 35 seconds. The observed waveform is in blue, and the fitted waveform is in red. 92

2-11 Plot of the three-phase set of voltages transformed into the rotating reference frame, referred to the first two cycles of the observed phase A voltage. The d-axis voltage is plotted in blue, and the q-axis voltage is plotted in red. . . 93

2-12 Block diagram representations of simulation and estimation processes. . . . 94

(a) The basic structure of the simulation process. 94

(b) The basic structure of the estimation process. 94

2-13 Structure of an estimation method that incorporates a forward model. . . . 95

2-14 Loss function $V(\mu)$ for the example problem, where μ is varied between 10 and 40. 97

2-15 Block diagram illustrating the structure of the pre-estimation method. . . . 98

2-16 Plot of the observed phase A motor current along with the preprocessed current, which extracts the envelope of the current. The current is illustrated in blue, while the envelope is shown in red. 99

2-17 Plot of the response of the sigmoid function. 101

2-18 Diagram of mass flow through fan. 104

2-19 Vector diagram for forward-curved fans. 107

2-20 Fan curves for three different speeds. 111

2-21 Speed-Flow-Power surface for fan in region of interest. 112

2-22 Picture of the overall air handler apparatus and attached duct system. . . . 112

2-23 Blower wheel. 113

2-24 Mounted motor. 113

2-25 Schematic diagram of the control and data acquisition system. 114

2-26 View of ducting apparatus. 116

2-27 Additional view of ducting apparatus. 116

2-28 Turning vanes. 117
2-29 Spectrum of the motor current for the unblocked fan illustrating the locations of the theoretical and experimentally observed slot harmonics. 119
2-30 Spectrum of the motor current for the 30% blocked fan illustrating the locations of the experimentally observed slot harmonics. 120
2-31 Spectrum of the motor current for the 50% blocked fan illustrating the locations of the experimentally observed slot harmonics. 121
2-32 Spectrum of the motor current for the leaky fan illustrating the locations of the experimentally observed slot harmonics. 122
2-33 Plot of unblocked fan speed as fan operated continuously over approximately 5 days. 123
2-34 Speed over the course of the airflow measurement period. 124
2-35 Plot of leaky fan speed as fan operated continuously over approximately 5 days. 125
2-36 Plot of mean of the fan speed for each of the conditions given in the legend over windows of increasing size. 126
2-37 Plot of the preprocessed observed phase A motor current envelope along with the estimated current envelope. The observed current envelope is illustrated in blue, while the estimated envelope is shown in red. 128
2-38 Observed and fitted motor currents for the hot motor; observed data are in blue, and fitted data are in red. This waveform is zoomed in so the agreement between the observations and the predictions can be seen. 129
2-39 Residual from $i_{a,fit} - i_{a,obs}$. 129
2-40 Observed and fitted motor currents for the hot motor; observed data are in blue, and fitted data are in red. This waveform is zoomed out so that the quality of fit for a larger segment of time can be seen. 130
2-41 Residual from $y_{fit} - y_{obs}$. 130
2-42 Plot of the estimated torque-speed curves for the motor when the parameters of the motor are estimated with different initial guesses for the stator resistance R_s. The value of the initial guess for R_s correponding to each curve is given in the legend. 132

2-43 Observed and fitted motor currents for the hot motor; the observed data are shown in blue, and fitted data are in red. The minimization was performed with only observations of the motor current. 135
2-44 The residual obtained via $i_{a,fit} - i_{a,obs}$. 135
2-45 Plot of the estimated and measured torque-speed curves for the hot motor; the measured curve is in blue, and the estimated curve is in red. 136
2-46 Observed and fitted hot motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data are in red, and observed torque-speed data from the motor is in blue. 137
2-47 Zoomed torque-speed curves. 137
2-48 Observed and fitted motor currents for the cold motor; observed data are in blue, and fitted data are in red. The minimization was performed with only observations of the motor current. 138
2-49 Residual from $y_{fit} - y_{obs}$. 138
2-50 Plot of the estimated and measured torque-speed curves for motor; the measured curve is in blue, and the estimated curve is in red. 139
2-51 Observed and fitted cold motor torque-speed curves for a number of motor starts on different days; fitted data are in red, and the torque-speed data from the motor is in blue. 140
2-52 Zoomed torque-speed curves. 140
2-53 Observed and fitted currents for the hot motor; observed data are in blue, and fitted data are in red. The minimization was performed with observations of both the motor current and torque-speed characteristic. 140
2-54 Residual from $y_{fit} - y_{obs}$. 140
2-55 Plot of the estimated and measured torque-speed curves for hot motor; the measured curve is in blue, and the estimated curve is in red. 141
2-56 Observed and fitted hot motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data is in red, and the torque-speed data from the motor is in blue. These motor parameters were obtained by fitting to the motor current and the torque-speed curve. 142
2-57 Zoomed torque-speed curves. 142

2-58 Observed and fitted currents for the cold motor; observed data is in blue, and fitted data is in red. The minimization was performed with observations of both the motor current and torque-speed characteristic. 143
2-59 Residual from $y_{fit} - y_{obs}$. 143
2-60 Plot of the estimated and measured torque-speed curves for cold motor; the measured curve is in blue, and the estimated curve is in red. 144
2-61 Observed and fitted cold motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data is in red, and the torque-speed data from the motor is in blue. These motor parameters were obtained by fitting to the motor current and the torque-speed curve. 145
2-62 Zoomed torque-speed curves. 145
2-63 Plot of the estimated torque-speed curves for the motor under the set of different conditions. The particular conditions correponding to each curve are given in the legend. 146
2-64 Plot of the unblocked motor speed as a function of time as the motor warms up, starting from room temperature. 147
2-65 Fitted hot motor torque-speed curves for a number of motor starts in different conditions; these conditions are listed in the legend. These motor parameters were obtained by fitting to the motor current only. 148
2-66 Zoomed torque-speed curves. 148
2-67 Cross-section of the duct illustrating the points where the measurements were made. 150
2-68 Fan curves illustrating the power-flow relation at the four speeds of interest. 151
2-69 Illustration of airflow detectability using torque-speed curves that are generated from minimization against only the motor current, as collected for each blockage condition. 154
2-70 Illustration of airflow detectability using a torque-speed curve that was generated from minimization solely against the unblocked motor current. . . . 154
2-71 Illustration of airflow detectability using torque-speed curves that are generated from minimization against the motor current and the torque-speed curve, as collected for each blockage condition. 155

2-72 Empirically measured fan curve for the blower wheel used in experimental air handler. 158

3-1 End cross-section of compression chamber. 169
3-2 Side cross-section of reciprocating compressor, from [14, p. 145]. 170
3-3 Drawing of external compressor appearance. 171
3-4 View of compressor internals, with the inside of the head and the top of the valve plate illustrating the discharge valves and the suction ports. 172
3-5 View of compressor internals, with the bottom of the valve plate and the suction valves. 172
3-6 Picture of valve plate with damaged discharge valves, from [29, p. 26-25]. A healthy discharge valve is shown at the front of the valve plate. 173
3-7 Picture of valve plate with damaged discharge valves, from [29, p. 26-25]. A healthy discharge valve is shown at the front of the valve plate. 173
3-8 Picture of compressor pistons which have been damaged as a result of liquid slugging, from [29, p. 26-29]. The connecting rods should be straight, not bent. 174
3-9 Representative torque-speed curve for a compressor motor. 180
3-10 Lab-frame currents. 184
3-11 Stator currents in stator reference frame. 184
3-12 Loss function showing the effect on $r(\hat{\mu})$ of varying both the initial guess $\hat{\mu}$ and the number of points K used in forming the residual. 189
3-13 Picture of the complete experimental air-conditioning apparatus. 192
3-14 Schematic diagram of the air-conditioner. 193
3-15 First liquid slugging injection apparatus. 194
3-16 Illustration of the injection tube for the first apparatus. 195
3-17 Diagram of the experimental RTU platform with the modifications to the refrigerant piping. The red line indicates the added plumbing and solenoid. 195
3-18 Illustration of mounted suction pressure transducer. 197
3-19 Compressor head for the second iteration of the liquid slugging apparatus. 199
3-20 Second iteration of the liquid slugging apparatus. 200
3-21 Illustration of cylinder pressure sensors (from datasheet). 200

3-22 View of the bottom of the new compressor head. 202
3-23 View of compressor valve plate. 203
3-24 View of compressor with new head and additional sensors installed. 203
3-25 Closeup picture of position sensors. 205
3-26 Picture of installation of position sensors. 205
3-27 Set screw and connecting rod viewed from outside the crankcase through the position sensor mounting hole with position sensor absent. 206
3-28 Slugging apparatus with vapor and liquid reservoirs. 208
3-29 Vapor portion of slugging apparatus. 208
3-30 D-axis stator current input and output of preprocessor. 215
3-31 Q-axis stator current input and output of preprocessor. 215
3-32 D- and Q-axis voltages transformed with phase correction. 215
3-33 D- and Q-axis voltages transformed without phase correction. 215
3-34 Time-varying component of utility voltage angle. 216
3-35 Residual between nonlinear phase estimate and linear phase estimate. . . . 216
3-36 i_{ds} for a representative compressor start transient. 218
3-37 i_{qs} for a representative compressor start transient. 218
3-38 i_{ds} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up. 220
3-39 Zoomed version of adjacent plot. 220
3-40 i_{qs} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up. 220
3-41 Zoomed version of adjacent plot. 220
3-42 P_{rc} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up. 222
3-43 P_d for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up. 222
3-44 Tachometer output during the startup period for the consecutive transient starts with the same starting cylinder position and electrical angle as the stator windings warm up. 223

3-45 i_{ds} for consecutive transient starts with a range of different starting positions when the compressor is warm and the electrical angle is the same. 224
3-46 P_{rc} for consecutive transient starts with a range of different starting positions when the compressor is warm and the electrical angle is the same. . . 224
3-47 i_{ds} when the electrical angle is varied by steps of $\pi/4$ for the compressor when the windings are hot and the initial position of the pistons varies. . . 225
3-48 i_{ds} for electrical angles of $\theta_e = 0$ and $\theta_e = \pi/2$ when the initial piston position is the same and when the stator windings are warm. 226
3-49 i_{ds} for electrical angles of $\theta_e = 0$ and $\theta_e = 3\pi/2$ when the initial piston position is the same and when the stator windings are warm. 226
3-50 Plot illustrating symmetric characteristics of transformation for i_{ds} when the windings are hot and the initial position of the pistons varies. 227
3-51 Transformed currents with pulsatile torques applied to the motor. 229
3-52 Motor speed during torque pulses. 229
3-53 Processed electrical measurements for 67 ms steady-state slug. 229
3-54 Mechanical measurements for 67 ms steady-state slug. 229
3-55 Processed electrical measurements for 133 ms steady-state slug. 231
3-56 Mechanical measurements for 133 ms steady-state slug. 231
3-57 Temperature of the discharge line during liquid injection events. 231
3-58 Pressure of both cylinders during 133 ms steady-state slug. 232
3-59 Processed electrical measurements for 270 ms steady-state slug. 233
3-60 Mechanical measurements for 270 ms steady-state slug. 233
3-61 Processed electrical measurements for 533 ms steady-state slug. 233
3-62 Mechanical measurements for 533 ms steady-state slug. 233
3-63 Filtered squared current for 67 ms steady-state slug. 235
3-64 Filtered squared current for 133 ms steady-state slug. 235
3-65 Filtered squared current for 270 ms steady-state slug. 235
3-66 Filtered squared current for 533 ms steady-state slug. 235
3-67 Simulated i_{ds} for higher load torque in comparison to lower load torque. . . 237
3-68 Simulated i_{qs} for higher load torque in comparison to lower load torque. . . 237

3-69 i_{ds} during transient slugging for 2 slugging tests in comparison to 2 cold nonslugging tests. 238
3-70 P_{rc} during transient slugging for 2 slugging tests in comparison to 2 cold nonslugging tests. 238
3-71 D-axis current for both cold and warm nonslugging tests. 240
3-72 Cylinder pressures P_{fc} and P_{rc} for both cold and warm nonslugging tests. . 240
3-73 D-axis current during transient slugging with the compressor starting in the same position, compared to slightly warm nonslugging starts. 241
3-74 Zoomed picture of the same data. 241
3-75 Q-axis current during transient slugging with the compressor starting in the same position, compared to slightly warm nonslugging starts. 241
3-76 Zoomed picture of the same data. 241
3-77 Cylinder pressures during transient liquid slugging events. 242
3-78 Measured variables during transient slugging start with initial electrical angle $\theta = \pi/2$. 244
3-79 Measured variables during transient slugging start with initial electrical angle $\theta = \pi$. 244
3-80 Measured variables during transient slugging start with initial electrical angle $\theta = 3\pi/2$. 245
3-81 D-axis current during transient slugging for all slugging tests in comparison to the cold nonslugging tests. 246
3-82 Zoomed picture of the same data. 246
3-83 D-axis current during transient slugging for all slugging tests in comparison to the warm nonslugging tests. 246
3-84 Zoomed picture of the same data. 246

4-1 Results of incorrect refrigerant charge survey, from [42]. 252
4-2 Schematic diagram of a residence, adapted from [103]. 256
4-3 Illustration of heat flow in the residence, adapted from [103]. 257
4-4 Test facility for undercharge fault detection experiments. 260
4-5 Full charge experiment with temperatureless charge diagnostic method. . . 262

4-6 Undercharge experiment with temperatureless charge diagnostic method. . 262
4-7 Least squares fits for both regular charge and undercharge data for temperatureless charge diagnostic method. 263
4-8 Measured variables on August 13, 2007, when the system was fully charged. 265
4-9 Measured variables on September 24, 2007, when the system was undercharged. 265
4-10 Normal charge experiment cycling histograms. 267
4-11 Undercharged experiment cycling histograms. 267
4-12 Filtered output of both charge experiments for the histogram cycling detection method. 268
4-13 Validation of technique; validation data was drawn from full charge experiment. 270
4-14 Validation of technique; validation data was drawn from undercharged experiment. 270
4-15 Compressor startup transients during normal cycling behavior for the air-conditioning system in both states of charge. 271
4-16 Compressor startup transients after the unit has been off for a number of hours, in both states of charge. 272
4-17 Scatter plot showing initial compressor power versus the ΔT across the house.273
4-18 Plot of runtime against ΔT over the course of Aug. 13. 274
4-19 Plot of runtime against ΔT over the course of Sep. 24. 274
4-20 Plot of polygon areas for the two different states of refrigerant charge as a function of time. 276
4-21 Circuit description of the thermal behavior for the thermal model. 278
4-22 Simulated fully charged temperature and cycling response over two days. . 279
4-23 Simulated undercharged temperatures and cycling response over two days. 279
4-24 Characteristic curves for both undercharged and fully charged sets of simulated data. 280
4-25 Simulated regularly charged day. 281
4-26 Simulated undercharged day. 281
4-27 Simulated fully charged day with both diagnostics. 281

4-28 Simulated undercharged day with both diagnostics. 281

B-1 Bottom view of compressor head. 383
B-2 Left side view of compressor head. 384
B-3 Cross-section of compressor head. 385
B-4 3-D drawing of compressor head. 386
B-5 Schematic of the control board for the solid-state relays. 388
B-6 Schematic of instrumentation board for cylinder pressure sensors and suction and discharge pressure sensors. 389

List of Tables

2.1 Pre-estimated motor parameters. 128

2.2 Final hot motor parameters minimizing solely against the current. 129

2.3 Measured motor parameters, as measured on 11/30/2007. 130

2.4 Various parameters resulting from starting the simulation with different initial guesses of R_s. 131

2.5 Hot motor parameters obtained by minimizing only against the observations of the motor current. 136

2.6 Cold motor parameters obtained by minimizing solely against the current. . 138

2.7 Hot motor parameters obtained by minimizing against both the current and the torque-speed curve. 141

2.8 Final cold motor parameters minimizing against the current and the torque-speed curve. 144

2.9 Aggregate table of all estimated motor parameters. 146

2.10 Table of all estimated motor parameters for different fault conditions. 149

2.11 Predicted and measured airflow values. 153

(a) Predicted airflow values. 153

(b) Measured airflow values. 153

4.1 Average ΔT and peak transient power during the startup transient for both states of refrigerant charge. 273

4.2 Areas of the trajectories for undercharged and fully charged datasets. 277

4.3 Parameters for the building simulation. 278

Chapter 1

Introduction

Whether the motivation is the environment, resources, or thermodynamics, the topic of energy conservation is presently receiving a great deal of attention. Such considerations can arise in a wide variety of different contexts: the amount of energy needed to manufacture a product, the amount of energy needed to operate a piece of equipment, and the amount of energy needed for transportation are just a few of the the different concerns which can arise. Moreover, a common concern at a time when the price of oil exceeds $140 a barrel is the minimization of the amount of energy consumed by a given task, allowing energy and resources to be better allocated or conserved.

Buildings represent one particularly important and common consumer of energy. Even neglecting the myriad uses of energy which take place inside buildings, the energy required to heat, cool and provide light in buildings amounts to billions of kilowatt-hours every year. Especially apropos to energy conservation considerations is the fact that the installation of air-conditioning in buildings is becoming increasingly commonplace. Over half of the industrial and commercial buildings currently in existence in the U.S. are air conditioned, while approximately 90% of newly constructed homes have central air-conditioning installed [1]. Moreover, 14% of the U.S. primary energy consumption is dedicated to HVAC systems for commercial and industrial buildings, and 32% of the electricity generated is consumed to heat, cool, ventilate, and light commercial buildings [158]. This profusion of air-conditioning systems uses a great deal of energy; by one estimate, 76 billion kWh per year are consumed by packaged air-conditioning equipment [132]. Of the types of air conditioners sold, packaged and unitary air-conditioning equipment forms

the largest segment of the market, cooling 45% of the 58.7 billion square feet of commercial space in the U.S.

While air-conditioning has long been popularly viewed as a luxury [28, 117], it serves many essential functions in buildings today. Perhaps the application which most readily comes to mind is that of increasing the level of thermal comfort in buildings by reducing air temperature and/or humidity, resulting in benefits both to the health and the productivity of the building occupants. Commercial and industrial applications also abound, as air-conditioning is often essential to process control in chemical or manufacturing plants, and it is also needed for many businesses, such as those which deal with the preparation, storage, and sale of food products. Medical applications of air-conditioning are also very common, as hospitals require it for the comfort and well-being of their patients.

This dependence of so many applications on air-conditioning brings a corresponding need for air-conditioners to operate within specified bounds on their performance. The variety of potential consequences of faulty operation depend on the application and the severity of the fault; for faults which result in a loss of air-conditioning, the effects can range from discomfort for home or business occupants, to ruining foodstuffs or temperature-dependent products, to directly affecting the health of hospital patients [21]. Less dramatic faults might not have such extreme short-term consequences, but they can still cause sizeable reductions in equipment efficiency and corresponding increases in energy consumption, or the shortening of equipment life, resulting in service cost increases. Factors causing or contributing to poor behavior are legion, including neglect and improper installation and maintenance, as well as unforeseen electrical and mechanical failures. Field studies suggest that such faults are widespread, as a study of 4,168 air-conditioners found that 72 percent of the air-conditioners had the incorrect refrigerant charge, and 44 percent had improper airflow [100].

Methods for monitoring and evaluating the operation of air-conditioning systems are gaining in importance as air-conditioning becomes more common. Systems designed to implement these methods, often referred to as condition monitoring or fault detection and diagnostic (FDD) systems, have a variety of potential uses, including monitoring the performance of the equipment over time, alerting users to the presence of a fault, improving the quality of service by identifying the cause of faulty operation, or even changing the

operation of the equipment to reduce a fault's effects. These FDD systems can greatly improve the performance of the underlying air-conditioning system and assist the service technician in expeditiously fixing and maintaining equipment.

While the users of air-conditioning systems which incorporate FDD methods will see a direct benefit due to the reduced downtime and improved performance of the air-conditioner, these FDD systems also benefit a number of other parties. For example, an effective FDD method can help a service technician to quickly identify malfunctioning components and reduce the time required to troubleshoot ambiguous problems. This will assist both users and service technicians in ensuring that the system was installed correctly, and providing prognostic capabilities in identifying and monitoring faults which can be addressed during scheduled service calls, rather than in emergency situations. Equipment manufacturers also benefit from the installation of such an apparatus, in that it enables them to better understand how and when faults occur.

Interest in the theoretical and experimental development of FDD methods in general is also growing due to the dependence of society on systems of increasing complexity. These systems can be found nearly everywhere in the early 21st century, from buildings to cars to airplanes to consumer electronics to traffic control systems. The traditional attitude of the designer, that the system will function reliably for a long period of time if the design is "good enough", is slowly giving way to the realization that many systems will be operated in conditions unforeseen by the designer, and that the cost of faults in such systems can sometimes be greater than the system itself. The successful implementation of FDD and condition monitoring systems is vital in such circumstances, to ensure that the system will continue to function well in the face of considerable operative uncertainty.

The research presented here investigates and develops FDD methods for residential air-conditioners using electrical measurements. Such an approach inherently has a variety of both benefits and challenges, both of which will be discussed in this introductory chapter. In order to provide context for this development of the FDD method, many of the factors and possible approaches which must be considered when designing FDD systems will be reviewed in the following section. After presenting this overview of FDD system architecture, previous research into the FDD systems for air-conditioning equipment will be discussed in §1.2, and this will be followed by an overview and executive summary of

the FDD techniques investigated over the course of this research in §1.3 and §1.4.

1.1 Fault Detection and Diagnostic Systems

In developing an FDD method, it is useful to identify the constituent tasks which the method performs. A useful and intuitive general framework for FDD methods is outlined by Isermann [72,73] in which the FDD process is broken down into four steps: fault detection, fault diagnosis, fault evaluation, and decision, as seen in Figure 1-1.

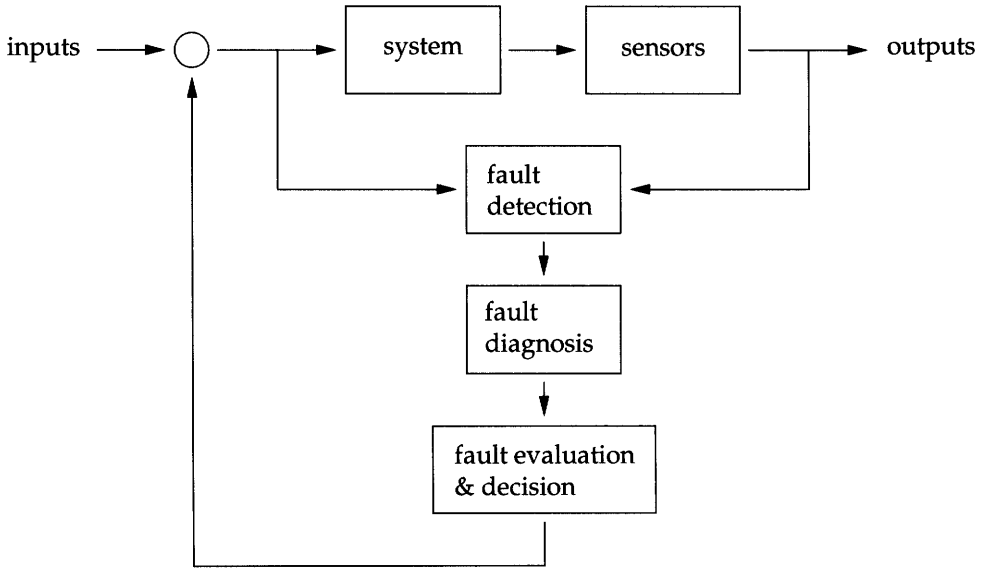


Figure 1-1: The basic structure of an FDD system (adapted from [73]).

In the fault detection block, measurements of the system are made and then processed in order to determine if a fault has occurred. If no fault is present, the output of the fault detector notes that status and the system continues operating normally until the fault detector checks again for faults. In the case that faults are reported, the fault diagnosis block processes the information from the fault detector and determines the cause of the fault due to the fact that the reported faults could potentially be caused by a variety of circumstances. Once this diagnosis has been performed, the fault evaluation and decision block assesses the significance of the fault and determines the proper course of action, whether it is to continue operating in a faulty state, shut down, change operation so that the effect of the fault is mitigated or eliminated, or any other option.

While this is a useful general framework, the construction of the particular blocks will vary widely in any specific implementation of an FDD method. For example, many such methods only implement the fault detection block in hardware and software, and rely upon the expert knowledge of a system user to perform the fault diagnosis and evaluation steps. Moreover, the performance of FDD system often depends on both the performance of the individual blocks as well as the interactions between the blocks. Each of the constituent components of the overall FDD system must therefore be designed with due consideration given to the operation of the other components.

A variety of issues which can affect the performance of the FDD method should also be considered during the design stage. For example, the reliability of the FDD system itself is critical to its performance, as situations in which the FDD system erroneously reports a fault can often be as problematic as those in which faults are reported correctly. The FDD method must correspondingly be designed and specified so that its reliability and performance in the field environment can be assessed and understood. The cost of developing and implementing an FDD method is also an important consideration. These costs may be justified by examining the savings collected by keeping the system running efficiently, servicing faults before they increase in magnitude, or by the convenience of scheduling service visits on a non-emergency basis. They might also be justified because of the consequences of not detecting the faults; even if the faults are inexpensive and easy to repair, situations in which system failure could have significant ramifications for the health of individuals or a business can be a powerful incentive for installing an FDD system. As many of these issues shaped the design and implementation of the FDD methods discussed in this research, each block of the general FDD method will be discussed briefly in the following sections in order to better provide context for the design decisions which were made.

1.1.1 Fault Detection

The structure of a fault detector can generally be broken down into three different pieces [73], as is visible in the structures that Figures 1-2 and 1-3 have in common. The first subcomponent of the fault detector takes as its input a set of data or measurements of the system obtained from a set of sensors, and uses these measurements to formulate an appropriate description of the observed system. The output of this model is then processed

to generate a set of fault signatures, which make it easier to observe the effect of the fault and which are known to be directly related to the fault or faults of interest. These fault signatures effectively project the observations onto a space which enhances the detectability of the fault. Once these fault signatures have been obtained, they are processed by a change detector which compares the observed fault signatures to the expected signatures that are generated by normal behavior, and generates an output which indicates the existence and/or size of the mismatch, also called a symptom. These symptoms are output to the fault diagnosis block, which assesses the presence or the severity of the fault.

While a number of different types of models may be used in the fault detector, they can generally be grouped into one of two different types: signal-based models, and system-based models. Signal model-based fault detection techniques do not incorporate any direct knowledge about the monitored system itself; instead, the sensor output is characterized solely by a model of the signal, and the presence of a fault is identified by looking at attributes of the signal model. This approach is illustrated in Figure 1-2. In comparison, system model-based fault detection techniques incorporate models of the underlying system that generate the observed sensor data, and use this data to characterize the proposed system model and to examine model characteristics which are indicative of a fault. This alternative strategy for fault detection is illustrated in Figure 1-3. As each of these approaches has relative strengths and weaknesses, both will be used in this research depending on the needs of the particular application.

Signal model-based fault detection methods are often the most straightforward fault detection methods to implement, since little additional information about the current state of the system is needed for the method's operation. Two common approaches look strictly at the time-series data obtained from the sensors; the first of these analyzes the measurements as they are obtained and uses a simple thresholding or limit-checking algorithm for fault detection. Observed changes in the signal which cross an established limit are deemed to be indicative of a fault. Such simplicity in a fault detector is attractive because it minimizes the fault detector's complexity, which has benefits for both the FDD system reliability and cost. An alternative approach is to analyze trends in the time-series history of the observed data, rather than looking solely at the instantaneous values of the data. This approach is often useful when the observations are expected to change over time in a

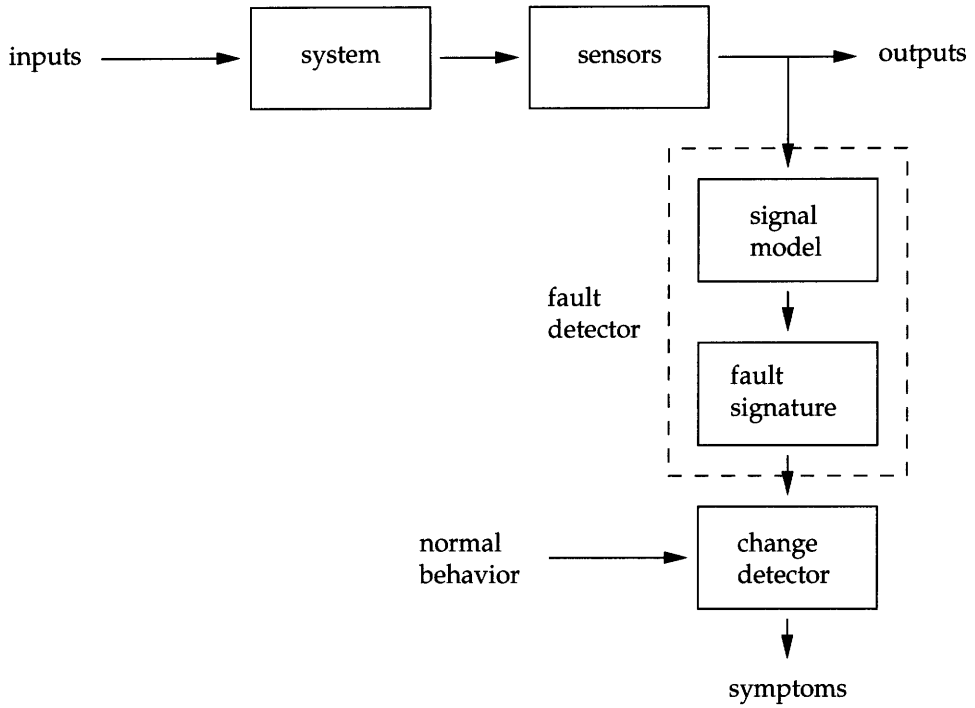


Figure 1-2: Fault detection using a signal model-based approach (adapted from [73]).

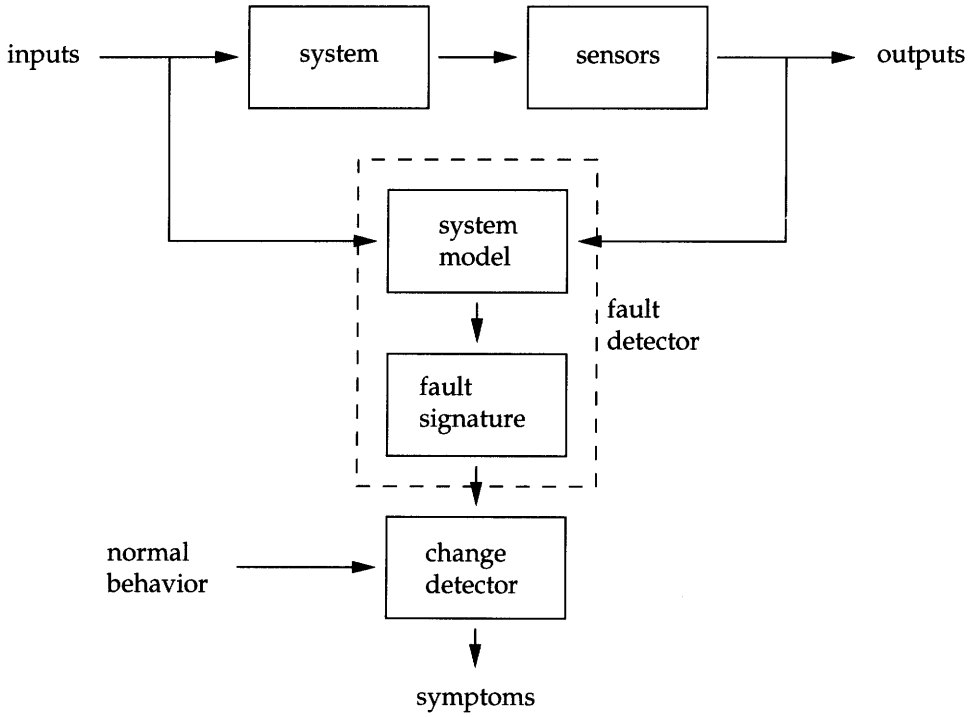


Figure 1-3: Fault detection using a system model-based approach (adapted from [73]).

well-understood manner.

While the simplicity of this type of fault detector is attractive, it also imposes limitations on the performance of the detection method. These fault detectors are typically sensitive to noise and bias, as there is no apparatus for differentiating between a spurious event which has no appreciable effect on the system and a fault condition. These detectors are also highly dependent on the equipment installer and FDD method designer to ensure that the sensor accurately characterizes the system performance over its lifetime, and that the threshold is set appropriately for the particular features of the system, with its attendant variations in performance.

An additional limitation of threshold-checking fault detectors is that faults may not appear in an abrupt manner, making it necessary to distinguish changes in the system as it "wears in" over time from slowly appearing faults. In drawing a distinction between faults which appear over a long period of time and faults which occur abruptly, it is helpful to borrow terminology from [22] in which the slowly developing faults will be referred to as "soft" faults, while the abrupt faults will be labeled "hard" faults. Using this terminology, hard faults tend to be conducive to detection with a threshold detector because they represent an abrupt change in the variable of interest, while soft faults tend to be more difficult to identify with a threshold detector because they are difficult to differentiate from characteristics of the system which gradually change with continuing use.

Alternative methods for processing the sensor data can also increase the amount of fault information obtained from the sensor array. While many extant fault detection systems rely solely on data obtained when the system is operating in quasi-steady-state, data acquired during transient changes in the system's state can provide a great deal of information about its behavior. Analyzing the sensor data by using other signal representations can also help to identify changes in the signal which would be difficult to observe in the time-series data. Common examples of such techniques include analyzing the signal with the Fourier transform or a multi-resolution basis such as wavelets.

In contrast to the signal modeling approach, the system modeling approach to fault detection involves constructing a mathematical or computational model of the system being monitored. Models are generally constructed with a given structure and a set of parameters, which are either set when the model has been chosen, or are initialized with a set

of training data which is known to accurately represent the behavior of the nonfaulty system. After the model has been initialized, faults can be identified by driving the model with inputs that are related to the inputs of the real system and comparing the output of the simulated system model with the output of the real system. A principal advantage of this approach is that it makes it possible to draw inferences about the system which are not directly visible, but are instead tied to the relationships between a number of different changes related by an unobservable variable or fault condition.

Naturally, these models can take a wide variety of different forms. One common class of models is referred to as black-box models, which require no information about the system being modeled. Perhaps the simplest type of black-box model is a simple polynomial, which uses sensor observations as the independent variable and the coefficients of the sensor observation terms as the model parameters. Other common types of black-box models are based upon machine learning techniques, such as neural nets, radial basis functions, and fuzzy logic-based models. The parameters for these models are generally obtained by a learning process that collects training data which is known to characterize the normal operating behavior; this step either occurs at the time that the model is constructed with a verified set of data, or after installing the system in the field, when the system is assumed to be fault-free.

One limitation of these types of models is that they do not take advantage of any of the physical understanding of the system which is based upon fundamental constitutive laws. There are many benefits to using these physically-motivated, or grey-box, models, such as the ability to incorporate intuitive knowledge about the behavior of the system into the mathematical dynamics of the model, as well as the ability to set the parameters of the model ahead of time, rather than relying solely upon the measured data. Linear and nonlinear Kalman filtering methods [146, ch. 4] represent perhaps the best known application of this approach to fault identification. These types of models can also be applied to processes, rather than physical systems, allowing fault detection methods to be implemented on systems which are driven by random inputs [32].

The system-modelling approach is also useful when used to track system parameters as the system performance changes over time. Rather than driving the model with the same input as the physical system and then analyzing the differences between the model output

and the system output, this alternative approach drives the model with the same input, but instead adjusts the parameters of the model to successfully reproduce the system output, so that the model input and output match the system input and output. The resulting changes in the model parameters will reflect the underlying changes in the system, and these changes can be analyzed to identify faults. The appeal of such an approach is that fault signatures can be generated for unmeasured parameters of the model which are of interest, rather than the measured variables which may not provide enough information to identify a fault by themselves. Physical models are particularly useful in this context, as they allow the designer to detect faults based upon his intuition regarding the changes in the system's physical model.

In designing the fault detector, due consideration must also be given to the possibility that the FDD method will not function as expected, either by not reporting faults which are present, or by reporting faults which are not present. Both of these scenarios are problematic because the service requirements of the system will not be addressed correctly, and because they may cause the user to lose confidence in the accuracy of the FDD method. It is therefore imperative that the reliability of the FDD method be well understood.

Failure of the FDD method can occur for a few different reasons. Malfunctioning sensors will clearly have a significant impact on the FDD method, both in the case that the sensors stop providing data to the fault detector altogether, or in the case that the output of the sensors acquires noise or bias which prevents the sensor output from accurately representing the true behavior of the system. Model inaccuracies also represent a significant challenge; for example, if the behavior of the fault-free system cannot be described by the model due to unplanned changes in the system, discrepancies between the expected and observed data could be reported as a fault. Moreover, many fault detection methods rely upon baseline data collected after the system is installed and its functionality is tested and verified (also referred to as commissioning) by a technician. If the technician does not install or commission the system accurately, the baseline data for the FDD system will represent faulty, rather than fault-free, behavior of the system. The possibility of emergent behavior which arises in complex systems must also be considered, as unforeseen circumstances might interact with highly complex FDD systems to produce unexpected results. These phenomena can only be mitigated or avoided through extensive testing.

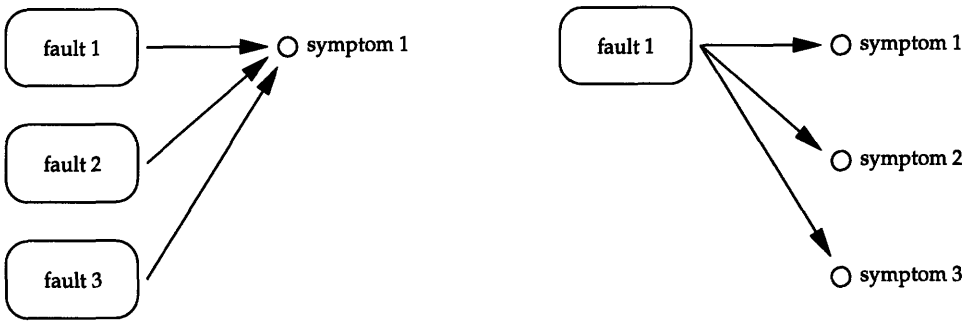


Figure 1-4: Multiple faults can exhibit the same fault symptom, and a single fault can exhibit many different fault symptoms.

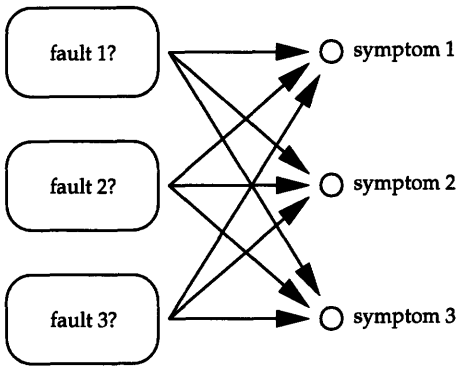


Figure 1-5: Schematic illustration of the multiple-simultaneous fault problem.

1.1.2 Fault Diagnosis

Historically speaking, most FDD systems are designed solely to monitor a small number of variables and alert the system user to the presence of a fault when a fault symptom is generated by the fault detection module, leaving the diagnosis and evaluation steps to the user. This can be an effective tool in some types of systems, where the user is interested only in knowing the presence of a fault, rather than knowing what the actual cause of the fault is. For many types of complex systems in use today, however, such systems are insufficient. Fault diagnosis modules can therefore be used to determine the cause of faults in such systems by analyzing the observed set of fault symptoms.

The central challenge which must be addressed by any fault diagnosis module is that faults are often not related to fault symptoms in a one-to-one relationship, as is illustrated in Figure 1-4. As can be seen on the right hand side of this figure, a given fault may be manifested via a number of different symptoms. While some of these symptoms may

be easier to distinguish from normal operating behavior than the others, each fault may manifest itself in multiple ways. The diagram on the left hand side of this figure illustrates the other complicating factor for fault diagnosis, which is the fact that multiple different faults can manifest themselves with the same set of fault symptoms. This can make it difficult to identify the occurrence of a particular fault if only one set of fault symptoms is monitored.

The extent of the potential difficulties facing the fault diagnosis module can be seen in Figure 1-5, in which multiple faults could be occurring simultaneously. Often referred to as the multiple-simultaneous fault problem, the FDD method must ideally be able to analyze a given set of fault symptoms and identify which faults, of all possible options, are affecting the system at any given point in time. This is particularly challenging because the fault symptoms are not necessarily binary quantities, so that the diagnostic module must infer the relative contributions of different faults to the observed magnitude of a given symptom. A pseudo-mathematical analogy to this situation suggested by [92,95] takes the following form:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \text{fault 1} \\ \text{fault 2} \\ \text{fault 3} \end{bmatrix} = \begin{bmatrix} \text{symptom 1} \\ \text{symptom 2} \\ \text{symptom 3} \end{bmatrix}. \tag{1.1}$$

In general, the technique used by FDD methods to solve this problem is called fault isolation. In the context of (1.1), this can be understood as diagonalizing the fault diagnostic matrix containing the terms a_{ii}, though it is important to note that this diagonalization is not carried out computationally, but rather is only performed analogically. This step can be greatly assisted by properly designing the sensor network for the FDD system; by carefully selecting the type and location of these sensors, it is often possible to capture fault signatures which are strongly correlated to the fault of interest and only weakly correlated with other faults. Similarly, the proper sequencing of the fault diagnoses can be assisted by using the process of elimination to reduce the number of conflicting fault symptoms.

A number of different approaches can be used to implement this fault isolation and diagnostic strategy. These approaches generally fall under the umbrella of classification

problems, since the problem is essentially one of classifying a set of observed fault symptoms as corresponding to one or multiple faults. There is a rich body of literature on the different types of classification techniques, including Bayes classification, geometric classifiers, decision trees, neural networks, and clustering algorithms. Each of these methods has its own strengths and weaknesses, and the "right" technique for a given FDD method can only be chosen by evaluating them in the context of the system of interest.

Two other factors must also be considered when designing an FDD method. One potential situation can occur when the faulty behavior of one system component causes faults in other system components, so that the faults cascade. An FDD method must therefore be able to diagnose both the proximate faults as well as the source fault; if only the proximate faults are diagnosed and repaired, the system will continue to operate in a suboptimal manner because undiagnosed fault will cause the repaired components to fail as well. The design of the FDD system should also account for the fact that the complete set of all possible system faults may not be known. In this case, the attribution of a fault symptom to a particular cause will not be accurate, because of model inaccuracies. While the only information which can be given to the system user in such a circumstance is that there is a fault, but that it is not one of the faults identified in the method, this information can still be of assistance to a service technician.

1.1.3 Fault Evaluation and Decision

Once the existence of a fault has been detected and the cause of the fault has been identified, the proper course of action must be determined. In many situations, this is chosen by the system user or service technician. As an assessment of the severity of the fault is essential to this decision process, the principal function of the fault evaluation system is making this assessment, which is commonly called the "fault level". The type of fault level can vary, as some faults are binary in nature, while other vary in a continuous manner. The fault evaluation module must therefore analyze the information provided by the fault detection module, which tells it that there is a fault, as well as the information provided by the fault diagnostic module, which tells it what the fault is caused by, in order to estimate how bad the fault is.

Once the fault level has been evaluated, the FDD method must select of the proper

course of action. In the described FDD framework, this step is designated as the fault decision module. The following options provide a sense of the possible choices which could be made by this module, depending on the exigencies of the particular situation.

1. Notify the user of the existence of a faulty condition, as well as the particular fault which has been diagnosed. Information about the fault detection signature and/or other potential faults may also be communicated to the user to properly convey information about the relative uncertainty of the diagnostic, in order to properly characterize the chance that the fault detection or the fault diagnostic modules did not correctly identify the fault. This fault evaluation procedure might be useful if the equipment owner or user has direct control over the system, and is able or desires to make his own evaluation of the status of the equipment as well as how to proceed with repairing it.

2. Record the existence of the fault, as well its diagnosis, in a logfile. This logfile might simply reside in the onboard computational framework of the FDD method, or it could be located in a remote location. Information about the confidence of the fault diagnosis could also be recorded. This procedure might be useful if the fault level is not high, or if it is either unnecessary or not practical to inform the user of the fault; when the service technician or the manufacturer examines the output of the logfile, they will be able to use this information most effectively.

3. Contact a service company directly, via a remote connection. This approach could dovetail with the previous approach, in that the service company could receive the information otherwise logged in the datafile regarding the nature and severity of the identified fault. Under such a framework, an FDD method which contacted a maintenance company directly for service could ensure that the system being monitored would be operating in a faulty state for a minimal amount of time. Obviously, such an approach would require a high degree of confidence in the FDD method's ability to correctly detect and diagnose faults, so as to prevent unneeded service calls.

4. Directly change the operation of the system in response to the presence of the diagnosed fault. Such a course of action might only be necessary or justifiable under the

most extreme of fault conditions, e.g. if the system will undergo irreparable damage if allowed to continue operating in its present state. The FDD method could also modify the system performance if it was integrated with the control system, so that the system performance is modified to compensate for the effects of the fault.

A variety of other courses of action exist, depending on the system being monitored, the particular application of the system, and the structure and implementation of the FDD method. Nevertheless, consideration of the particular approach taken to evaluate the effects of the fault and provide the appropriate response is an important element in the job facing the designer of the FDD method.

1.2 Previous Work

While the previously discussed framework is useful in developing a general understanding of FDD methods, it will be primarily applied to FDD methods for air-conditioning systems in this research. It is helpful to survey this body of work to gain a sense of the advantages and disadvantages of these approaches, as a wide variety of FDD methods have been implemented for air-conditioning and refrigeration systems. To provide context for these studies, the general operation of air-conditioning systems will first be reviewed, and will be followed by a summary of three surveys which examined the frequency and types of faults which occur in field-installed air-conditioning equipment.

1.2.1 Air Conditioning System Overview

A schematic diagram of a vapor-compression air-conditioner is illustrated in Figure 1-6. The overall objective of the ideal vapor compression refrigeration cycle is the transfer of thermal energy from the air passing through the evaporator to the air passing through the condenser via the refrigerant. To see how this occurs, consider the refrigerant leaving the compressor, travelling toward the condenser in the direction of the boldface arrow. Having just been compressed, this vapor is at both a high temperature and a high pressure. As the vapor travels through the condenser and is cooled by the air flowing over the coils, it gradually gives up enough of its thermal energy to cause it to change from 100% vapor at the top of the condenser to 100% liquid by the bottom of the condenser.

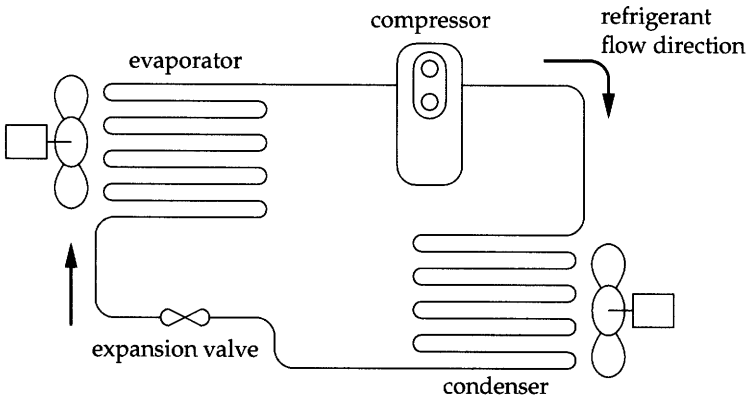


Figure 1-6: Schematic diagram of the air-conditioner.

Once the refrigerant has fully condensed, it has the capacity to absorb more thermal energy via conversion back to a gas. This process is accomplished in two stages; first, the the refrigerant passes through an expansion valve into the evaporator coil. The expansion valve serves the dual purposes of regulating the flow and reducing the pressure of the refrigerant. This drop in pressure causes a corresponding reduction in the saturation temperature of the refrigerant, or the temperature at which the liquid refrigerant evaporates. As this low pressure refrigerant flows through the evaporator coil, the process of its evaporation causes it to absorb thermal energy from the air traveling past the evaporator coil. This process causes the air flowing over the coil to become cooler. After leaving the evaporator as 100% low pressure vapor, the refrigerant is compressed and returned into the condenser, completing the refrigeration cycle [15, 98].

Due to the wide variety of requirements and applications which exist, many different types of air-conditioning systems have been constructed which implement this basic refrigeration cycle or variations thereof. Window air-conditioners satisfy the requirements for small room-size residential applications. Air-conditioning systems which serve the entire house are often constructed with either centralized air-conditioning systems, which locate an evaporator at a central location and distribute the cool air through a ventilation system, or ductless air-conditioning systems, which distribute refrigerant lines throughout the house and cool the air locally. A variety of types of compressors are used in these units, including reciprocating compressors, scroll compressors, and rolling piston compressors; the particular compressor used depends on the size of the cooling load and the applica-

tion. Still other types of mechanical cooling systems exist for larger buildings, including systems which incorporate single and multi-stage chillers and cooling towers [10].

While the function of real refrigeration equipment is theoretically equivalent to that of the ideal refrigeration cycle described above, the behavior of the physical realization of such a system differs from that of the idealized system in a number of aspects which can have a significant effect upon the unit's performance. Some of these differences are expected, and their effects can be minimized or otherwise compensated for in the design of the unit. Other differences may be unforeseen by the design engineer, and are the result of aberrant behavior due to atypical environmental conditions or malfunctioning equipment. These undesirable behaviors are precisely what FDD methods seek to characterize and eliminate.

1.2.2 Common Faults in Air-Conditioners

Faults typically occur in air-conditioners via three different mechanisms: improper commissioning, premature component failure due to unforeseen operational conditions or component defects, and wear due to normal usage. Each of these fault mechanisms results in a degradation of the air-conditioner's performance, either because the air-conditioner ceases to function (hard faults), because of a reduction in the efficiency of the air-conditioner, or because the increased rate of wear for the air-conditioner components (soft faults). These faults can be manifested in the electrical, mechanical, or thermal interactions of the air-conditioner.

In order to better understand the consequences of malfunctioning and broken air conditioning equipment, a number of studies have been performed to enumerate the different types of faults to which packaged air-conditioners are susceptible. These studies classify the faults according to a range of different criteria, such as cost to repair, importance to the overall function of the unit, and the tendency of a given fault to accelerate the onset of other faults. While their methods of comparison might differ, all of these studies are useful when attempting to understand the scope of faults which occur in package air conditioning units. Given the near ubiquity of air-conditioners, it may be surprising to find that only three major studies have been published in enumerating these faults. One possible reason for this fact is that failure information is often quite valuable, and this proprietary

information is therefore not made public by equipment manufacturers. These studies were authored by Cunniffe, James, and Dunn in 1986 [36], Stouppe and Lau in 1989 [142], and Breuker and Braun in 1998 [22]. An additional study which examined both heating and air-conditioning equipment was published by Hasan in 1974 [65]; as this study focuses on larger types of equipment, such as cooling towers and chillers, it will not be discussed here.

In [36], Cunniffe, James and Dunn conducted a survey of vapor compression refrigeration plants in a variety of different applications, such as low temperature refrigeration applications, air conditioning systems, and vehicle refrigeration systems, in order to quantify and enumerate the common faults which occur. Rather than quantify the statistics of particular faults, the ultimate causes of the faults were classified; of the 851 systems which were studied, 32% of the mechanical faults and 29% of the electrical faults were caused by unforeseen operating conditions. In comparison, 38% of the mechanical faults and 44% of the electrical faults were caused by faulty materials, manufacture, design, installation, service, commissioning, or maintenance. The remainder of the faults were either due to normal deterioration or remained unclassified.

As these statistics were collected, the authors observed that a number of operational faults overloaded the compressor, which consequently failed and was diagnosed as the faulty component. This type of failure is problematic because the compressor is often repaired without fixing the underlying fault. In order to address this concern, the authors identify a number of fault classes which cause overloading. High discharge temperatures represent one class of faults, which account for 9-15% of the failures in reciprocating compressors, and which are in turn caused by faults such as high intermittent plant operation, fouled condenser surfaces, non-condensible gases, or other unforeseen operating conditions. Liquid return to the compressor housing and refrigerant migration, in which liquid refrigerant accumulates in the compressor shell and then boils up into the compressor cylinders, can also cause a host of problems, ranging from increased discharge temperatures, to major mechanical damage, to the degradation of winding insulation. Poorly controlled expansion valve or evaporator dynamics, excessive equipment cycling, and inadequate system cleanliness also caused compressors to fail prematurely.

Another set of researchers [142] who systematically investigated the faults which are

responsible for malfunctioning air conditioning equipment were Stouppe and Lau [142]. They examined 15,760 failures in units with a cooling capacity up to 50 tons (600,000 BTU/hr) over the course of eight years (1980-1987) by scrutinizing insurance claims which were filed for the repair of air conditioning equipment. Many different types of refrigeration systems were examined, incorporating both reciprocating and centrifugal compressors. While some compressor technology has changed in the intervening fifteen years, their basic findings are still quite useful in considering the range of failures which occur in refrigeration systems.

Of the 15,760 failures examined, 12,518 had their specific cause of failure denoted on the claim. The authors found that these units generally failed when they were approximately 10 years old, and that such units had no major inspections or overhauls over that entire period of time. Furthermore, 11,349 of the failures were electrical (comprising motors, controls, and electrical apparatus), while 4,411 were mechanical (compressor bodies, system piping, or vessels). It is important to reëmphasize the fact that compressors were separated into their two different functional halves, being the mechanical section of the compressor, including the pistons, valves, and associated hardware, and the compressor motor, which drives the mechanical apparatus that performs the compression.

The authors specifically examined the causes of failure for hermetically sealed air conditioning and refrigeration units: 76.6% of the failures were electrical (motor windings, control equipment, and other associated issues), 18.9% were mechanical (compressor valves, springs, bearings, connecting rods, pistons, crankshafts, lubrication), and 4.5% of the failures were caused by malfunctions in the refrigerant circuit. A few particular notes were also made regarding trends which were observed in the causes of compressor motor failure. Insulation deterioration on the motor windings due to age and/or service, as well as unbalanced voltage and single phase operation were some of the main causes of the failure; degradation in motor stator windings was by far the most prevalent cause of motor failure, accounting for 84% of failures in hermetic motors and 74% in the non-hermetic motors. The remaining causes of the failures were generally broken rotor bars, bearings, and motor control equipment. Of the motor control failures, 168 failures were directly attributed to short cycling, or the repeated starting and stopping of the motor many times in rapid succession.

A more detailed study of mechanical compressor failures for reciprocating freon-based units showed that the bulk of the failures could be attributed to two main causes. The first of these causes is that of metal fatigue in the internal suction and discharge valves and springs, due to the hundreds of millions of operating cycles that these parts see over the average lifetime. Liquid slugging, or the unintentional injection of liquid refrigerant into the cylinder, was also the direct cause of 20% of the mechanical failures, due to the hydraulic forces from the attempt to compress the incompressible liquid refrigerant which acts on the valves, valve plates, pistons, and connecting rods. Under normal operating conditions, gaseous refrigerant is pulled into the cylinder during the intake cycle, but when liquid refrigerant enters, these hydraulic forces can cause broken valves and valve plates, as well as cascading damage to the remaining mechanical components of the compressor.

While the information collected in [142] is valuable, it does not quantify the incidence of faults which would allow the air-conditioner to continue operating in a less efficient manner, nor does it attempt to classify failures on the basis of their repair cost. This is a significant shortcoming in their report, as it only identifies the components which ultimately failed, not the proximate cause of those failures. Though many different faults can occur in an air-conditioning system, a significant portion of them can cause the compressor to fail; the identification of the compressor as the component which failed most often does not help to determine the type or frequency of problems in the system which caused that failure. The cost of repair is also an important figure-of-merit in examining failures, as failures which are expensive to repair should have priority when choosing which faults to include in an FDD system.

Braun and Breuker [22] attempted to compensate for these shortcomings by examining two different types of faults in package rooftop air conditioners: those which caused the air conditioner to run inefficiently, but did not cause the equipment to cease operation, and those in which the air-conditioner could no longer operate mechanically or electrically. By analyzing a statistically representative subset of a database containing 6000 separate faults observed from 1989 to 1995, the authors found that, of the fault classifications which resulted in "no air conditioning," or faults in which the unit was running inefficiently but was still electrically and mechanically operational, 60% were caused by electrical problems (such as motors and control problems), while the remaining 40% were caused by mechan-

ical faults. On the basis of cost, however, compressors represented the most significant portion of the total unit repair cost (24%), even though they break much less frequently.

In examining the compressor failures, the authors found that approximately 70% of the compressor faults were due to internal motor problems, such as shorted windings, open windings, or locked rotors. One interesting fact which they point out is that, although the immediate cause of compressor failure was usually attributed to motor problems, the proximate cause was often a mechanical fault which overloaded the motor. Furthermore, they reiterate one of the conclusions from [142], that the prevalent cause of this mechanical overloading condition was again liquid refrigerant in the compressor cylinder. This speaks directly to the difficulty of the fault diagnostic problem, as the diagnosis of the faults in the compressor motor is difficult to directly ascribe to the presence of liquid slugging.

The authors also specify the main contributors to the costs for faults in the condenser, evaporator, and general air handling systems. In analyzing repairs to the condenser unit, defective fan motors accounted for 50% of the cost, while fouled coils represented 30% of the cost. In comparison, fouling represented 61% of the cost of repairing an evaporator unit, and coil damage represents 25% of the cost. One interesting sidenote is that only 18% of the evaporator fouling and 14% of the condenser fouling faults resulted in a loss of comfort. Other general costs required to fix miscellaneous electrical components were dominated by contactor failure, which represented 40% of the cost, followed by 27% for general damaged components, and 13% for wiring errors and short circuits.

Though much of the stated motivation in developing FDD methods for air-conditioning systems focuses on the effect on system owner and user, equipment manufacturers are also affected by the incidence of faults in equipment. As documented in [25, 84], the difficulty of properly identifying and diagnosing faults in field-installed equipment results in air-conditioning technicians generating a large number of misdiagnoses, either because the technician does not know that a given fault can be fixed in the field or because the failure is attributed to an incorrect cause. According to Copeland Corp., half of the compressors which are returned to the factory under warranty are victim of this tendency to misdiagnose; when these diagnoses are checked by factory technicians, no defect is found in nearly half of them. This is often referred to as the NDF (no defect found) problem, and is another strong motivation for the development of accurate and cost-effective FDD methods for air-

conditioning systems.

1.2.3 FDD Methods for Air-Conditioners

While it is clear from §1.2.2 that the incidence of faults in HVAC equipment is sufficiently high to warrant the development and implementation of FDD systems, there are a number of other factors which can have important consequences for the design of such systems. First among these is the fact that many buildings are not owner-occupied. This makes the market for air-conditioning system very first-cost sensitive, as the building owners have little incentive to provide more efficient and expensive equipment for the occupants when less expensive equipment will suffice [83]. The air-conditioning industry has responded to this tendency, producing little equipment which exceeds the federal standard; as of 2000, less then 10% of the manufactured packaged air-conditioners with a cooling capacity between 5.5 and 11.25 tons[1] exceeded the federal standard for energy consumption by 20% and only 21% of the packaged air-conditioners with a cooling capacity between 11.25 and 20 tons exceeded the federal standard by the same amount [132].

When buildings are owner-occupied, higher-efficiency air-conditioners are more frequently used, but they are still adopted more slowly than might be expected (only 6.7% of residential air-conditioners exceed federal standards by 40%). A variety of reasons contribute to this phenomenon; for example, the air-conditioning system is often one of the last systems to be installed in a house, and typical cost overruns make less expensive air-conditioning units attractive, even despite knowledge of higher operating costs [156]. In addition, as the monthly cost of energy is much lower than most other commercial expenses, such as salaries or manufacturing costs, little attention is paid to the cost of operating an inefficient air-conditioning system unless energy prices are extremely high [83].

Motivated both by the need for reliability in air-conditioning equipment and the potential for increased interest in high-efficiency units, there has been a commensurate surge over the last 10 years in the investigation and development of FDD methods for air-conditioning units. Much of this work has focused on using temperature and pressure measurements in order to evaluate the relative health of the unit as well as forecasting any incipient problems which have yet to degrade the performance seriously, as can be seen by surveying

[1] 1 refrigeration ton = 12,000 BTU/hr = 3.52 kW

the published literature on such systems, some of which is treated in detail in [27].

A number of papers have been published that address the problem of casting FDD methods into the general framework for FDD systems as discussed in §1.1. Haves presents a particularly thorough perspective on some of the challenges inherent in constructing these methods [66]. The authors of [49, 116] discuss the variety of different approaches for implementing FDD methods on air-conditioners, and address concerns and potential strategies for achieving higher market penetration with such systems. Stylianou [143] also discusses ways that these FDD methods might be integrated into building energy management systems. Thybo, et al [148] provides some interesting perspectives on implementing FDD methods in air-conditioning units after completing their research in developing FDD methods for refrigerated display cases for supermarkets. Some important considerations identified in designing an FDD method include the method's scalability for different types of equipment, the reduction of the number of sensors in the system due to their cost and added complexity, and minimization of the number of procedures required for a technician to commission the FDD system. The authors also emphasize the fact that FDD methods are generally more effective when incorporated into the system during the design stage. Faults which were important to building managers are identified via a survey in [164], which determined that many of the faults of importance are caused by of improper design and commissioning. The authors of this paper also surmise that fault detection in air-conditioners is increasingly difficult because the high complexity of the systems can obscure the effects of the faults and the large number of components can make it difficult to locate the source of a fault.

Many signal-based FDD methods have been developed for air-conditioning systems. One popular approach involves identifying a set of relationships between measurements of steady-state quantities (typically temperatures and pressures) which are indicative of particular faults. Such methods are typically referred to as rule-based methods, as the relationships between variables are expressed in the form of rules derived from physical knowledge of the system. These methods typically rely on a fairly large set of measurements; the methods used in [95, p. 170] incorporate eleven temperature measurements, two pressures, two air velocities, and one relative humidity measurement. These methods also generally require training on a set of known nonfaulty data, in order to establish a

baseline against which operational changes can be compared for fault diagnosis.

Jim Braun and his graduate students at Purdue University have been using and refining this rule-based approach for FDD methods since 1995, and have experimentally demonstrated a system which can diagnose six common faults: loss of volumetric efficiency of the compressor, fouled evaporator coils and filters, fouled condenser coils and filters, liquid line restriction, incorrect refrigerant charge, and the presence of noncondensible gases in the refrigerant loop. This system was originally developed for air-conditioners with fixed orifice expansion (FOX) devices [23,115], and was also investigated in systems with thermal expansion valves (TXVs) [26]. Improvements to this FDD method resulted in the ability to successfully detect and diagnose multiple simultaneous faults [89,90,92–95], and this method was experimentally demonstrated to identify faults and increase the operating efficiency of air-conditioners at a number of locations in California [88,91]. The detector used to ensure that the air-conditioner is operating in steady-state was subsequently studied in [75], where it was proposed that the two standard measurements often used to determine steady-state (the evaporator superheat T_{sh} and condenser subcooling T_{sc}) were not sufficient to determine the steady-state in all conditions, and that all of the measured variables should be analyzed before steady-state operation can be declared.

Other research also incorporates similar rule-based approaches. For example, a system similar to that developed in [95] is evaluated in [59], albeit with a larger number of measurements, and is found to perform well experimentally. An implementation of an object-oriented FDD method is also developed in [63], which uses rule-based methods to evaluate the performance of the air-conditioning unit, and also evaluate the performance of the sensors by implementing redundancy checks on the sensors. Rule-based FDD methods are also used in [76] to identify faults in a variable speed vapor compression system.

A variety of black-box system-modeling FDD methods have also been tested. These methods do not require any physical knowledge of the air-conditioning system, but instead use training data to help the model characterize the non-faulty behavior of the system. The authors of [114] develop a method which combines polynomial regression and locally-weighted regression techniques with statistical machine learning techniques to identify faults in the air-conditioning system, and experimentally test this method on detecting incorrect refrigerant charge. Genetic algorithms are used in [5] to identify faults in boilers,

while the performance of rule-based methods is experimentally compared with the performance of a method which uses fuzzy models and classifiers to detect faults in a three zone HVAC system in [19]. A similar fuzzy-modeling approach is also studied in [165], along with the use of vibration analysis to identify faults in pumps and motors. Rule-based knowledge is also used with artificial neural networks in [62] to identify fouling in an air-handling unit.

Grey-box system-modeling techniques, in which information about the physical processes governing the system are incorporated into the model, have also been extensively used. While these methods typically also require some level of initial training to distinguish faulty behavior from nonfaulty behavior, much of this training only requires the specification of thresholds, and does not require the full characterization of the field-installed equipment. Physical models of the HVAC system are formulated in [158], which uses recursive least squares to identify the model parameters from training data, while physical models are also used in tandem with nonlinear observers and threshold-checking methods to detect condenser fan failure and capillary tube blockage in [155]. State-space models of an air-handling unit are integrated into a Kalman filter to identify and update the model parameters in [152], and airflow and sensor faults are experimentally identified based upon features in the residual waveforms generated by the Kalman filter. A state machine-like approach is also used in [139] to identify when valves and actuators get into faulty states, as when they are stuck open.

Hybrid grey-box approaches, in which machine learning techniques are used to identify parameters of physically-based models, have also been used extensively. Neural network and bond-graph techniques are used to identify parameters of a physical model of a refrigeration system in [148, 149], allowing faults to be diagnosed via fixed and trending thresholds. This method also integrates the fault detection system with a controller for the equipment, to mitigate the effect of the faults. The challenging problem of distinguishing the slowly changing characteristics of the air-conditioning system from soft faults is discussed in [67], in which radial basis functions and direct-search algorithms are used to identify the physical parameters of a system and identify the presence of fouling and valve leakage faults in an air-handling unit.

The identification of sensor faults, as distinct from system faults, is addressed in [157]

by analyzing discrepancies between sensor readings which are related through a model. The model outputs are compared with the sensor outputs, and genetic algorithms are used to determine the existence of sensor faults in the air-conditioning unit.

Research into non-mechanical sensor based FDD methods is another ongoing area of research. A preliminary system is developed in [43] which detected changes in the system due to a variety of faults from measurements of the motor current, but this system was not developed into a full FDD system which could diagnose separate faults. FDD methods based solely upon electrical measurements are tested in [6,7] for the purposes of identifying a variety of types of faults, such as incorrect refrigerant charge and fan imbalance. Copeland Corporation has also developed an FDD system for compressors based upon measurements of electrical power, named the ComfortAlert system [25]. This system is designed to identify basic compressor faults, such as long run time, short cycling, system pressure trip, locked rotor, open circuit, or low control voltage. This system has also been demonstrated to be effective and has been marketed and distributed since 2002. Other relevant literature regarding the detection of motor faults includes work published in [17,18,118,128], among numerous other sources.

1.3 Electrically-based FDD techniques

As is evident from the prior section, previous work in developing FDD systems sets the bar high for making new contributions to these systems for packaged and unitary air-conditioning units. Existing systems have detection thresholds which are sufficiently low that they can detect faults before a 5% drop in either the capacity or coefficient of performance (COP). As these systems have also been developed with an eye toward commercial implementation, FDD systems like that which was developed at Purdue University are projected to cost as little as $250-300 per unit [91].

One notable characteristic of many of the FDD methods discussed in §1.2.3 is that they rely upon a relatively extensive network of sensors. Many of the methods have five or more temperature sensors, as well as pressure sensors, mass-flow sensors, and relative humidity sensors. While these mechanical sensors can provide valuable information about the state of an air-conditioning system, their use in an FDD method must be considered in

the context of their cost and rate of failure. For example, mass flow sensors are typically very expensive, with prices exceeding a few thousand dollars per unit. Other sensors, such as temperature sensors, are prone to bias if they are not properly mounted to the measured surface, or if a radiation shield is not installed properly. Much like air conditioners, it is clear that sensors are vulnerable not only to hard faults, but also to soft faults, such as sensor bias due to environmental or unforeseen manufacturing defects. Though the ability of the FDD system to successfully identify faults certainly may outweigh the sensor reliability concerns, the effects of both cost and reliability must be considered in the design of FDD methods for air-conditioners.

Another interesting observation that pertains to most of the extant FDD methods is that many of the common faults identified in the surveys of §1.2.2 are not identified by the FDD methods. In the case of many of the electrical faults, a variety of FDD methods have already been developed by the community of engineers who study motor faults which could be easily adapted to the case of identifying faults in the compressor or fan motors; however, these methods have not been integrated into FDD systems for air-conditioners. Other faults simply are not conducive to the types of measurements made by the majority of the FDD systems. For example, even though liquid slugging is identified by all three fault surveys as a primary cause of damage in compressors, the first published FDD method proposing a method for identifying this fault was not published until 2004 [7]. This is probably due to the fact that a substantial amount of sensitive mechanical instrumentation must be installed in order to directly detect this phenomenon [96, 136, 137].

Mechanical diagnostic techniques based upon the output of electrical sensors represent an interesting and potentially viable alternative fault detection approach for air-conditioning systems which addresses both of the above concerns. In effect, these methods measure the currents and the voltages at the terminals of the electromechanical device, and then identify faulty mechanical behavior on the basis of these observations. These techniques are fundamentally rooted in the first law of thermodynamics: in many situations, the electrical power flowing into an electromechanical device is directly related to the mechanical power flowing out of it, so that changes in the mechanical load of the motor which are related to fault behavior can be detected by observing corresponding changes in the electrical power flowing into the motor. FDD methods using this approach either use a signal-based

approach, in which *a priori* knowledge is used to relate mechanical faults to particular features observed in the electrical sensor output, or a model-based approach, in which the observed inputs and outputs of the system are used to characterize the behavior of the system and determine the existence of any faults.

This electrically-based approach to mechanical fault detection, suggested by [6,7], has a variety of benefits, not the least of which is the fact that electrical diagnostics, such as broken rotor bars and shorted windings, can also be performed without using any additional sensors. The electrical information can also be used to measure the energy consumption of the air-conditioning system for end-use load information purposes. Perhaps the greatest benefit, however, is that electrical sensors are generally easier to install, less expensive, and more reliable than the equivalent mechanical sensors. This is due in part to the fact that electrical sensors do not need to be in close physical proximity to the unit to function well, allowing them to be placed indoors or in a similarly protected location, while at least part of the physical air-conditioning apparatus usually resides outside. These features have not gone unnoticed by industry, as the ComfortAlert product manufactured by Copeland Corp. relies upon electrical measurements to perform its diagnostic procedures.

Another appealing characteristic of electrically-based FDD methods is that some diagnostics can be performed on an aggregated electrical signal which contains electrical power information for multiple loads. Generally referred to as non-intrusive load monitoring (NILM) [38,82,87,124,129,130], this technique is useful because it can further reduce the number of sensors needed to identify the operation of loads and perform control and diagnostic operations for those loads, since multiple loads can be monitored simultaneously. There are limits to the number of loads which can be effectively monitored with a NILM, as well as the types of diagnostics which can be successfully implemented, due to bandwidth and noise limitations; nevertheless, the synergy between electrically-based FDD methods and potential NILM applications makes the development of such FDD methods that much more attractive.

Naturally, this alternative approach to mechanical fault detection and diagnostics also has its limitations. One such limitation is associated with the fact that, in such applications, the motor is functioning both as a transducer and as an energy conversion device. While motors are typically designed to be good energy conversion systems, they are not usually

intended to be used as transducers, so that variations in the motor's operating characteristics, while not problematic for their role as energy conversion systems, make it much harder to develop reliable fault detection methods. Moreover, there are mechanical faults which have no observable effect on the electrical variables because of effects like noise or scaling problems, such as arise for loads coupled through gearboxes. Electrically-based methods can also require substantial computational support, although the decreasing cost of computation and the prevalence of similar requirements among other FDD methods for air-conditioners suggest that this is less of a barrier than might otherwise be perceived. A final disadvantage of this approach is suggested by the fact that the wide variety of mechanical changes, which can be measured via temperatures, pressures, forces, and so forth, are being mapped into a relatively small number of electrical variables, i.e. currents and voltages. This mapping can make fault isolation more difficult, since different types of mechanical faults sometimes have identical effects on the electrical variables.

The observation that different mechanical faults may not be individually distinguishable because they have identical effects on the electrical variables has a direct impact on the design of the FDD method. It is therefore incumbent upon the FDD method designer to carefully model the physical behavior underlying the different mechanical faults, so that the distinguishing characteristics of each fault might be identified and utilized for fault isolation. One useful attribute of electrical measurements is that the high bandwidth of electrical sensors makes it possible to identify fault signatures which contain relatively high-frequency behavior. Fault signatures which have timescales ranging from fractions of a second to many hours can therefore be used in these electrically-based FDD techniques, allowing faults to be isolated not only by looking for different types of behavior on the same interval of time in the signal or model, but also by analyzing the input data on a number of very different timescales. Such an approach differs markedly from that used by many other FDD methods, which rely upon steady-state features of the measured variables partly because of the low bandwidth of many mechanical sensors, e.g. most temperature sensors.

1.4 Research Overview

The main objective of this research is the development of electrically-based FDD methods for a set of common faults in air-conditioners. Though most extant FDD methods for air-conditioners rely upon mechanical sensors, the opportunity to identify a range of mechanical faults with a small number of reliable electrical sensors is quite compelling, as is the fact that such an FDD system would be able to identify both electrical and mechanical faults with a single sensor array. Moreover, the relatively low cost for the sensors and processing system makes such an approach attractive in the cost-sensitive air-conditioning market. This research represents a continuation and substantial expansion of the early efforts initiated by Peter Armstrong and published in [6–8], which suggested the potential opportunities inherent in these approaches. Three particularly common and problematic classes of faults will be studied in this research: airflow faults, liquid slugging, and refrigerant leakage. Each of these faults is theoretically and experimentally investigated, and this document will address both the fundamental characteristics of these faults as well as the many experimental considerations which arose in inducing and detecting these faults in commercially-available equipment.

With regards to the structure of the remainder of this document, the airflow diagnostic methods will be discussed first in Chapter 2, due to the fact that many of the issues which surround the simulation and parameter estimation of induction machine models are treated extensively in this chapter. The liquid slugging FDD methods, which also rely upon the induction machine model of Chapter 2, will then be addressed in Chapter 3. Chapter 4 will address the diagnostic techniques used to identify the presence of refrigerant leaks in an air-conditioning system with electrical measurements and a small set of temperature measurements, and then the range of FDD methods developed over the course of this research will be summarized and placed in the larger context of potential air-conditioning FDD systems in Chapter 5. Each of these chapters will begin by motivating the development of FDD methods for the particular fault of interest and reviewing the previous work done in identifying these faults. The theoretical background used to develop the electrically-based approach to identifying these faults will then be discussed; this will be followed by a survey of the experimental apparatus used to simulate and de-

velop the particular diagnostic techniques. Finally, the experimental results for the class of faults studied will be presented and discussed at the end of each chapter.

Chapter 2

Airflow Diagnostics

In the first set of faults studied in this research, the volumetric airflow through the ventilation system is outside of previously established specifications. These faults are particularly interesting to study because they can either occur as a result of the failure of system components, or because of normal use and wear. Due to the variety of circumstances in which these faults can arise, the chapter will begin by presenting an overview of the types of faults that cause the airflow to deviate from expected values. After surveying alternative methods that have been developed to identify these types of faults, the structure of the fault diagnostic method will be described and the operation of each of its components will be studied. Finally, experimental results will be presented for the purposes of evaluating the fault diagnostic method's effectiveness.

2.1 Motivation

While a general description of the class of faults studied in this chapter is easy to formulate in that they consist of undesired deviations in the volumetric flow rate, they can arise in a variety of different circumstances. The design of an effective fault detection system must take this range of faults into account to accommodate the requirements for independently identifying each fault and assessing the degree of confidence with which any particular fault can be diagnosed. The range of different faults that modify the airflow through ventilation systems will be discussed in this section, as well as their relative incidence in field-installed equipment as evaluated in published fault surveys. Furthermore, a

survey of previous methods used to identify some or all of these different fault conditions will also be presented, as the study of their relative advantages and disadvantages in field-installed equipment was helpful in designing the fault detection method described in this research.

2.1.1 Background

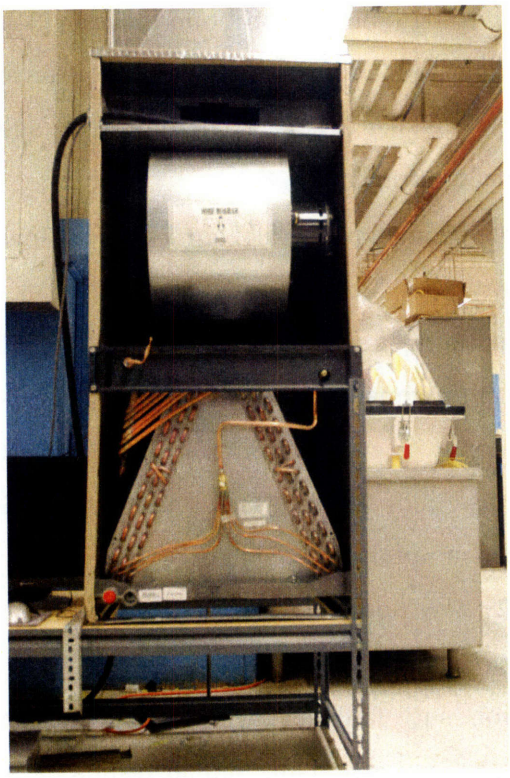
Ducted ventilation systems are an important component of buildings today, and their operation can serve a variety of different functions. Perhaps the effect that most readily comes to mind is that of increasing the level of thermal comfort in buildings, as they distribute cool, dehumidified air during warm summer conditions, and heated air during cold winter months. In this capacity, they serve the dual purposes of keeping the occupants of a building comfortable and increasing the occupants' productivity in businesses. Ventilation systems also ensure that contaminants, whether natural (CO_2) or artificial (building odors and so forth) are removed from the occupied spaces. Some particular types of buildings are especially reliant upon their ventilation systems; for example, buildings that house food service and storage operations are dependent upon the ventilation system to supply air at the proper conditions to maintain the quality of their products. Manufacturing processes that are sensitive to environmental conditions are also similarly dependent upon properly functioning ventilation systems. These two concerns can also be combined in such applications as hospitals, which are dependent upon ventilation systems for maintaining the comfort and well-being of their patients, as well as ensuring that contaminated air is treated appropriately before leaving the building.

Unfortunately, not all ventilation systems function as intended, resulting in airflow through the system that varies with respect to the design specifications. A variety of problems can cause these systems to malfunction, including blockage or leakage in the ducting system, malfunctioning fans that cannot supply the proper amount of airflow to the occupied spaces, and malfunctioning control systems for the fans and ducting systems, causing the airflow to be poorly directed or inadequate for the demand. These faults can have a myriad range of effects that can be inferred from the loss of the previously mentioned benefits, not the least of which is the loss of thermal comfort for the building occupants. Duct leakage is particularly notable for its effect, not only in its deprivation of airflow and po-

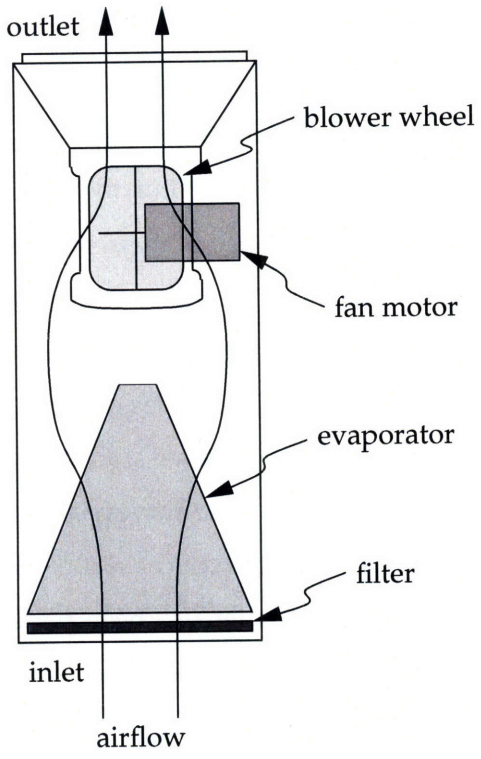
tential distribution of exhaust air, but also in that the energy expended in conditioning the air is wasted if that air gradually leaks out over the length of the distribution system [34]. Moreover, it is often difficult for a building manager to identify these problems, because they are hard to locate physically, and because problems such as severe blockage or leakage are often the cumulative result of a number of smaller problems.

The range of these problems can be seen by reviewing a number of surveys of airflow faults in buildings. One compendium of fault surveys, which examined 503 rooftop air-conditioning units in 181 buildings in five states in the Western U.S. from 2001-2004, found that the airflow was out of the specified range in approximately 42% of the units [31] surveyed. A separate study of 4,168 commercial air-conditioners in California [100] reported that 44% of the surveyed units had airflow that was out of specifications. Due to the dual concerns of reduced airflow and wasted energy consumption, a number of studies of duct leakage have also been performed. Studies of 29 new homes in Washington State [61] found that average duct leakage rates to the exterior ranged from 687 to 140 CFM. Similar results are also reported in [35]. Extrapolating from such fault surveys, one estimate for the total energy consumed by duct leakage is $ 5 billion/year [140]. Similar results have also been reported for a field study of buildings in Belgium and France [24]; the authors also estimate the cumulative energy savings potential over a period of 10 years to be approximately 10 TWh, if substantial improvements were made to the tightness of the ducts. In examining the specific causes of these leaks, it has also been found that that the majority of leaks were due to poorly fitting joints in the ducts, while a smaller portion of leaks could be attributed to poorly sealed seams along the length of the ducts [12].

A system that is able to monitor the state of airflow and detect faults in ventilation systems would thus fulfill a significant need in contemporary buildings, due to the prevalence of the faults. Since split systems represent a very large portion of the air-conditioning systems installed in buildings, this research focuses on identifying airflow faults for this particular class of ventilation systems. The air-side distribution systems for these air-conditioning units are typically called air handlers, air handling units, or AHUs. To examine the causes of particular faults for these units, it is helpful to refer to a visual representation of an AHU; a picture of the air handler used in this research is displayed in Figure 2-1a and is also illustrated schematically in Figure 2-1b.



(a) Picture of air handling unit.



(b) Schematic diagram of air handling unit.

Figure 2-1: Schematic diagram and picture of air handler.

The fan used in a typical residential air handling unit is a centrifugal fan, or blower. The fan in this picture is a particular kind of blower, known as a double-duct blower. These fans work by pulling air in through the inlet at the center of the wheel, and then relying upon centrifugal force to accelerate the air through the blades of the wheel and out into the fan housing. This particular configuration is called a double-duct blower wheel because air can enter the blower wheel on both sides, as illustrated by the arrows. Other blower wheels are configured with the dividing plate located on one side of the wheel, so that the air can enter only into one side.

An example of one type of fault that will result in reduced airflow is that in which the fan motor is either malfunctioning or nonoperational. Clearly, this will cause problems in the ventilation system that need to be addressed; unfortunately, the severity of such faults is not directly proportional to the ability of the building maintenance team to fix them. Fig-

Figure 2-2: Picture of broken mount for fan motor in a field-installed AHU, from [31].

ure 2-2 illustrates one particularly egregious example, as recounted in [31], in which a team of engineers performing a fault survey of rooftop air-conditioning units (RTUs) struck up a conversation with the occupants of a particular building that they were studying prior to opening up the unit on the roof. According to the occupants, the air-conditioning system had been malfunctioning for some time, but complaints to to the building management had only resulted in a preliminary diagnosis of thermostat failure. When the fault survey team reached the unit on the roof, however, they found that the fan motor (visible on top of the fan cowling) had completely fallen off of its mount, dislodging the belt from the blower wheel, so that the fan had been wholly inoperational for the entire period of time. The identification of such faults, as simple as the diagnostic procedures may be, is therefore essential to ensuring that the ventilation system is functioning correctly.

One very common, and less extreme, fault is that in which the filter to the air handler, or the evaporator itself, is clogged, causing the airflow through the fan to be reduced. While the most notable effect of such a fault will be on the reduction in the air delivered to the

building occupants, this fault can also potentially chill the volume of air flowing through the AHU further than is intended. A dramatically reduced flow rate could also affect the system health of the overall air-conditioning system, and of the compressor more specifically; if little air is travelling through the evaporator, the cooling load on the evaporator could be substantially reduced, causing the amount of refrigerant being evaporated in the evaporator to be much smaller than required by the design specifications. This could potentially result in liquid refrigerant entering the compressor through the suction line, causing the ingestion of liquid refrigerant by the compressor. Finally, the accumulation of material on the filter or the evaporator can also effect the health of building occupants, as the accumulation of bacteria or mold begin to accumulate on these surfaces can cause the people breathing the air to become ill.

Another type of fault that affects air handlers and their ducting systems are leaks in the ducts. These leaks can arise for a variety of reasons, such as cracks or holes in the ducts themselves, poorly assembled fittings, or broken and stuck dampers in diffusers and ventilation ports. Leaks typically have the effect of changing the distribution of the airflow, as the pressure drop across given sections of duct differs from the design value. Duct systems are often designed by controlling the pressure drop across each section of duct, so that the airflow through each section of duct can be controlled accordingly. Unfortunately, when these unplanned pressure drops caused by leaks are introduced into the ducting system, the airflow through the system will differ from its design specifications, introducing more airflow into some areas and depriving other areas of needed airflow. As previously mentioned, these leaky ducts also represent a substantial loss of energy, since it will be necessary to increase the flow to the conditioned spaces beyond what it strictly needed to compensate for the leaks in the system.

While all of these fault conditions can arise over the course of the equipment lifetime for either the AHU or the attached duct system, these faults can also be introduced to the system if the ducts are not installed or commissioned correctly. In the terminology of Chapter 1, the previously discussed faults can be described as operational faults, while faults that are introduced when the system is installed are commissioning faults. These faults can be introduced via a variety of means; if the installer does not seal the ducts adequately, leaks will remain in the duct system. Similarly, the airflow will not meet the designer's

specifications if the installer does not balance the ducts correctly. This type of fault is often difficult to identify, since the FDD method will not be able to accurately measure the baseline airflow through the system for the purposes of comparing later data. A fault detection method that could directly estimate the actual airflow in the system would avoid this difficulty, ensuring that the installation and commissioning of a system met established design specifications.

An additional consideration of importance is that different types of faults will tend to occur in the ventilation system over different timescales, and at different points in the lifecycle of the air handler. For example, some types of faults, such as commissioning faults, will occur when the unit is first installed, and will affect the behavior of the AHU in a similar manner over the course of the lifecycle of the equipment. In comparison, other operational faults, such as a malfunctioning fan, will occur after the unit has been operating for a while, and will be manifested by an abrupt change in the system behavior. Other types of operational faults, such as filter blockage, will gradually occur over time, causing a correspondingly gradual change in the system behavior. While the particular patterns of fault behavior might not be uniform across all ventilation systems, study of the time at which different faults are generally introduced might provide additional information into the means of distinguishing between these various types of faults.

2.1.2 Previous Work

A wide range of methods for estimating airflow exist, ranging from licking one's finger and holding it in the flow field of interest, to complex fault detection systems that implement sophisticated airflow measurement techniques. To better understand the context for the airflow estimation method developed in this research, this section presents an overview of some of the airflow measurement techniques that have been used in ventilation systems that are supplied by air handling units.

The ASHRAE Fundamentals handbook [9, ch. 18] discusses a number of traditional approaches to measuring airflow in ducts. One class of approaches to measuring airflow uses a type of device called anemometers[1]. Two main types of anemometers are commonly used to measure flow in ducts. Propeller anemometers function by measuring the speed of

[1] from the Greek $\alpha\nu\epsilon\mu\omega\varsigma-$, for wind.

rotation of a set of propeller blades after calibrating the rotational speed to the the airflow through the blades. Another type of anemometer that has found widespread use is often called a hot-wire anemometer. This device is constructed with a heating element and a RTD or thermocouple at the tip of a long probe that is inserted into the airflow of interest. Since different velocities of air past the probe tip change the heat balance at this point, the temperature of the tip can be used to calculate the velocity of the airstream at the measurement location. This is a very popular and accurate method for estimating air velocity, but its two drawbacks are that it is very sensitive to the probe orientation and it only measures the velocity at one point in a duct, making an array of measurements necessary to identify an average velocity of the airflow.

A different set of approaches to measuring the flow in ducts entails relating the measurement of the pressure at points in the duct to the flow through the duct. One common approach is to use a pitot-static tube to measure the dynamic and static pressures at a point in a duct. The velocity in the duct can be determined from the resulting pressure measurement by using the relation

$$V = \sqrt{\frac{2p_w}{\rho}}, \tag{2.1}$$

where V is the air velocity, p_w is the measured dynamic pressure, and ρ is the air density at the point of measurement. This is also a commonly-used technique to measure the flow in a duct, but a number of measurements also have to be taken at different points in the airstream to measure the average flow. Due to the limitations of this single-point measurement, a measurement device, sometimes called a flow plate or a true-flow grid[2], has been developed [51] in which a series of tubes with holes drilled in them at specific locations are interconnected to form a manifold that is inserted into the duct at the location of interest. By having a number of holes located at points across the diameter of the duct, a spatially averaged pressure measurement can be obtained that can be related to the average flow in the duct. Other high-accuracy methods for measuring the flow using pressure measurements include the use of venturi, nozzle, or orifice flowmeters, such as that discussed in [140], which force the air through a channel with a known pressure-flow relationship.

[2]Manufactured and sold by the Energy Conservatory (`http://www.energyconservatory.com`).

The mechanical instrumentation required to obtain these measurements is sensitive and subject to bias, however, and the instrumentation is relatively expensive and difficult to install, limiting the appeal of such systems when they are not absolutely necessary.

Less common flow measurement techniques have also been developed. One such technique [159] involves injecting CO_2 into the duct at one point, and measuring the change in its concentration at a point in the duct downstream; by using models to characterize the change in concentration over time, the flow of air can be calculated. A similar approach is used in [32, 33], using ozone as the detection agent, rather than CO_2. Another method that has been tested involves performing an acoustic sweep of the duct and measuring the duct's acoustic transfer function [37]. The location of the resonances and anti-resonances in the resulting spectrum, along with information about the expected structure of the ducting system, makes it possible to identify the amount and location of blockage in the duct.

Many of these methods are not used by air-conditioning fault detection systems, though they can accurately identify the airflow in a duct system. The first reason for this fact is that fault detection systems, such as those discussed in in [53, 54, 69, 85, 86, 95], usually make the assumption that the system can be commissioned well, and that baseline data obtained from the air-conditioner after installation represents a fault-free state. Under this assumption, it is only necessary to measure changes in the state of the airflow, rather than obtain a numerical estimate of the airflow itself. The second reason for this fact is that changes in the airflow can often be inferred from the behavior of the system as measured by the sensor network of the fault detection system. The relatively high incremental cost of adding an additional sensor to a network of seven sensors provides a considerable disincentive to adding a sensor that will only clarify one fault diagnostic. Changes in the state of the airflow through the evaporator are therefore identified as one of a number of other faults, such as low refrigerant charge or blocked expansion valves.

Changes in the airflow in such systems are usually detected by constructing fault models. One type of fault model does not use any physical information about the system, but instead tracks changes in all of the steady-state measured quantities and identifies correlations between the variables that change during the fault condition. Another type of fault model uses physical knowledge of the effect of blockage on the measured variables to develop an indicator of the blockage. For example, Li and Braun [95] use the following

equation to estimate a reduction in the flow through the evaporator:

$$\dot{V}_e = \frac{v_e \dot{m}_{ref} \left[h_{suc}(P_{suc}, T_{suc}) - h_{ll}(P_{ll}, T_{ll})\right]}{\left[h_{air,in}(T_{air,in}, \phi_{air,in}) - h_{air,out}(T_{air,out}, \phi_{air,out})\right]}, \tag{2.2}$$

where $\dot{V}_e$ is the volumetric flow rate through the evaporator, v_e is the specific volume of the air in the evaporator, $\dot{m}_{ref}$ is the mass flow rate of refrigerant through the evaporator, $h_{suc}(P_{suc}, T_{suc})$ is the enthalpy of the refrigerant in the suction line at the outlet of the evaporator as a function of the pressure and temperature at this location, $h_{ll}(P_{ll}, T_{ll})$ is the enthalpy of the refrigerant in the liquid line at the inlet of the evaporator, $h_{air,in}(T_{air,in}, \phi_{air,in})$ is the enthalpy of the air entering the evaporator as a function of the temperature and relative humidity, and $h_{air,out}(T_{air,out}, \phi_{air,out})$ is the enthalpy of the air exiting the evaporator. Not all of these quantities are directly measured by the system; for example, the mass flow rate through the compressor is estimated from a compressor map. Similar estimators are proposed [93–95] for the various terms in this expression that are not directly measured.

In-situ methods are necessary to validate equipment performance and evaluate annual energy consumption and account for part-load operation that is affected by system controls [111]. Moreover, as is clear from the frequency of faults found in field-installed equipment discussed in §2.1.1, a system that can identify airflow fault conditions could have broad applicability in contemporary air-conditioning systems. Many of the fault diagnostic systems that have been developed thus far have limitations on their performance, however; the instrumentation that is needed to identify faults accurately is often costly and delicate, and the methods that do not incorporate this instrumentation generally rely upon baseline data which will not describe the true "fault-free" behavior of the system if the system is not commissioned properly. Moreover, many of the fault diagnostic systems that have been implemented in RTUs rely upon a set of assumptions which might not be valid when multiple faults occur simultaneously. For example, the evaporator blockage fault detector discussed in [95] relies upon a compressor map to get the mass flow rate of the refrigerant. If faults, such as valve leakage, are present in the , the compressor map will not describe the behavior of the compressor and consequently the diagnostic indicator for evaporator blockage may not accurately identify the presence of this fault.

2.2 Structure of the Diagnostic Method

One alternative method for identifying this fault can be developed by observing that the volumetric airflow through the fan can be expressed via the relation

$$W_f = P_f Q_f, \tag{2.3}$$

where W_f is the mechanical power supplied to the fan, P_f is the pressure across the fan, and Q_f is the volumetric flow rate through the fan. Since changes in any of these quantities will be reflected in the other two terms of the expression, more information is needed to characterize the changes in these variables. This additional information can be obtained by empirically characterizing the relationship between these three variables for a given fan. This characterization is typically referred to as a fan curve. Fan curves are generally measured for most manufactured fans, as they are crucial to predicting the fan performance over a wide range of conditions. They can be used to predict the pressure across the fan as a function of the airflow and the speed of the fan, so that

$$P_f = f_1(Q_f, \omega_f), \tag{2.4}$$

where ω_f is the speed of rotation of the fan and f_1 represents the relationship established by the fan curve. Simple algebraic manipulation makes it clear that a direct relationship between Q_f and W_f can also be developed, since

$$\begin{aligned} W_f &= f_1(Q_f, \omega_f) Q_f & (2.5) \\ &= f_2(Q_f, \omega_f). & (2.6) \end{aligned}$$

When the fan is in its stable region of operation, this relationship is single valued, meaning that a given airflow and speed can only produce one value of mechanical power flowing into the fan. Because of this property, this expression can be inverted. This results in the statement

$$Q_f = f_2^{-1}(W_f, \omega_f). \tag{2.7}$$

This relationship can further be simplified by noting that the mechanical power supplied to the fan can be expressed as

$$W_f = \tau_f \omega_f. \tag{2.8}$$

This results in the following expression,

$$Q_f = f_3(\tau_f, \omega_f) \tag{2.9}$$

By using this relationship, one can obtain a numerical estimate of volumetric airflow through the fan by measuring the torque applied to the fan τ_f and the speed of the fan rotor ω_f. It is also important to note that the electrical power W_e entering the motor terminals cannot be substituted for the mechanical power W_f because the motor efficiency ν has an unknown and strong dependence on temperature, due to the resistive heating of the motor windings. As this temperature dependence is not known, a method of directly estimating the torque developed by the motor at a particular speed will be developed that will compensate for these temperature dependent effects.

The fault diagnostic method developed from this relationship differs from many of the previous fault detection methods in that it will generate numerical estimates of the airflow, rather than estimates solely of the change in the airflow. The two attributes that recommend such an approach are that it is much less susceptible to commissioning faults, and that the airflow estimates produced represent spatial averages. One of the characteristics common to many of the airflow measurement methods discussed in §2.1.2 is that they generate point estimates of the airflow, and that relating the point estimates of the airflow to the average airflow requires either a number of estimates or additional knowledge of the velocity profile in the duct. This fault diagnostic method, by its nature, generates estimates of the average airflow, circumventing the complications of the other mechanical method.

The second particularly propitious aspect of this method is that it is much less susceptible to commissioning faults than other fault diagnostic approaches that require the characterization of the "fault-free" unit to identify faults. Such methods cannot, by their very nature, identify faults that are present in the system when the set of baseline data is acquired. If leaks are present after installation in the ducting system, the fault diagnostic method will characterize the leaky behavior of the system as fault-free. In comparison,

a system that obtains numerical estimates of the airflow which are not dependent upon this baseline data could potentially identify the presence of such leaks. Moreover, one related feature of such a method is that the fault detection method itself does not need to be commissioned along with the air-conditioner. When installing the other FDD systems, it is necessary not only to ensure that the air-conditioner is running in a fault free state when it is installed, but also that the fault-diagnostic system is commissioned properly. As an example, suppose that the baseline data for the fault diagnostic system is acquired without installing the filter in the air handling unit. Such a mistake will cause the fault diagnostic system to identify a fault when a much smaller amount of blockage is present in the duct, as the filter will already represent a considerable amount of blockage over its baseline state. This type of difficulty can also be avoided by using numerical estimates of airflow to identify faults.

One of the other appealing characteristics of this method is that it is possible to obtain the measurements needed to estimate the airflow (i.e. W_f and ω_f) solely from electrical sensors, rather than from any mechanical sensors. This is particularly useful because of the relative robustness of electrical sensors in comparison to the equivalent mechanical sensors; where the output of mechanical sensors such as temperature or pressure sensors are susceptible to bias because of variations in the local environment in which they are installed, electrical sensors can typically be installed in relatively shielded locations, and their performance will not vary substantially with changes in their conditions. While the addition of some mechanical sensors could potentially aid the fault detection method by enabling the use of more complex models for the mechanical loss mechanisms, such as windage and bearing friction, the performance of this fault detection method will be evaluated solely using observations of the electrical data to evaluate the quality of the predictions that are made in the absence of any additional mechanical information.

This technique for estimating the airflow through an air handler using only electrical measurements will be developed over the remainder of this chapter. In the context of the previous work that has been presented, the objectives of this method are twofold: that the technique be able to identify the occurrence of changes in the airflow, and that these changes are quantified via the volumetric airflow through the unit. By developing a method that is able to numerically estimate the volumetric airflow through the fan at any

point in time, the method will be able to identify both leaks and blockage in the ducting system.

The architecture of this airflow estimation method is dependent upon the estimation of three related quantities: the mechanical torque applied to the fan τ_f, the speed of the fan blades ω_f, and the fan curve at the operating point of the fan. Since the fan curve is measured empirically by the manufacturer, it is necessary to develop a method to determine ω_f and τ_f from the motor electrical variables V_m and I_m. With this in mind, consider the following block diagram description of the estimation method, as illustrated in Figure 2-3. To identify the airflow, the speed ω_f and the torque-speed curve $\tau_f(\omega_r)$ are first

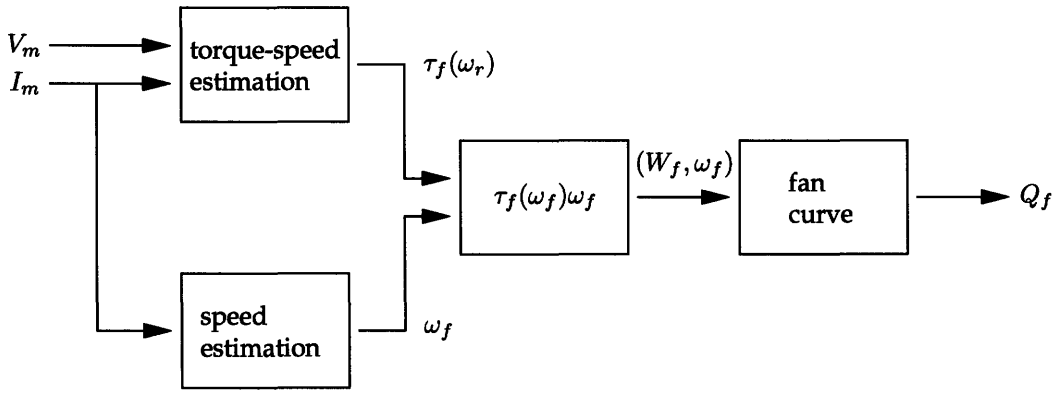


Figure 2-3: Structure of the airflow estimation method.

estimated independently from the electrical variables. The torque at the motor's present operating point is then identified by evaluating the $\tau_f(\omega_r)$ at the operating speed ω_f. With these estimates of ω_f and τ_f, the operating mechanical power W_f can be identified. The mechanical power and the present operating speed are then used to identify the point on the fan curve that describes the fan's current state, thereby generating an estimate of the volumetric airflow Q_f through the fan.

While this approach to identifying airflow faults has much to recommend it, it is not without its own shortcomings. In particular, it is important to note that this fault detection method is susceptible to diagnostic features indicative of airflow faults that are actually caused by other unrelated changes in the system. Since the diagnostic method evaluates the state of the airflow on the basis of the electrical terminal variables of the motor, it will be difficult to distinguish between electrical behavior related to the airflow fault and electri-

cal behavior which is unrelated to the airflow fault but which has a similar manifestation. Many common motor faults are unlikely to be problematic, however, as such faults (broken rotor bars, shorted windings) affect one phase winding more than the others, while airflow faults will affect all three phases identically. Other motor faults, such as bearing failures, could qualitatively be similar to airflow faults, but even these often have distinguishing frequency characteristics that have been modeled extensively, allowing them to be identified separately [106].

It is also important to note that this method of estimating airflow is largely based upon the correlation of changes in P_f to changes in Q_f via the fan curve. This method therefore fundamentally cannot tell whether or not the change in P_f is caused by an additional pressure drop upstream of the fan, or downstream of the fan. This makes it difficult to identify whether or a change in the fan performance is caused by a change in state of the filter, or by additional duct leakage. In addition, if two simultaneous faults occur, one of which has the effect of reducing P_f while the other has the effect of increasing P_f, the diagnostic indicator will remain unchanged, preventing the fault diagnostic method from identifying either fault.

Fortunately, other information may be available in many of the potential circumstances in which these difficulties arise. For example, while simultaneous leakage and blockage are not independently distinguishable, ventilation systems may be more susceptible to leakage during the commissioning phase and more susceptible to blockage that accumulates slowly over time. Information about changes in the system performance over these different timescales could thus be used to differentiate between these types of faults. Such concerns would have to be addressed when testing this fault diagnostic method in field installations to better characterize its performance in residential or commercial settings.

This diagnostic method was developed using a three-phase induction machine, since this type of machine is generally quite common due to their relatively low manufacturing cost. These types of motors are often used in larger building ventilation situations, where three-phase power is available, while single-phase induction motors are more typically used in residential locations. Due to the wealth of diagnostic techniques and research into models of this type of motor, this research will be focused on developing diagnostic methods for three-phase motors, rather than single-phase motors. While the fact that these

motors are not as popular in residential applications may appear to limit the utility of this diagnostic method, three-phase motors were studied because the modeling techniques used are better understood for three-phase than single-phase motors. These motors were also studied because it was desired to the feasibility of the overall diagnostic method in as straightforward a manner as possible, without becoming mired in the technical details of the method's implementation for a single-phase motor. Such an implementation can be developed analogously after the method has been proven to work.

Each constituent component, or module, in the overall diagnostic method of Figure 2-3 will be discussed in detail to more precisely describe this research. The module comprising the method of estimating the speed of the rotor will be explained first, and this will be followed by an explication of the method for estimating the mechanical torque generated by the motor. After reviewing the methods to estimate the torque and the speed, the fan module will be examined, illustrating the means by which the rotational energy is converted to air velocity. Finally, the experimental setup and results will be reviewed and the overall success of this method will be demonstrated and evaluated.

2.3 *Rotor Speed Estimation*

The first module of this fault detection technique consists of a method to identify the rotor speed. A wide variety of methods for obtaining this measurement exist, which include instruments like optical or magnetic tachometers, or mounting additional coils on the rotor and measuring the voltage induced in a magnetically-coupled coil. Since one of the main objectives of this module involves minimizing the number of additional sensors installed in the system, the approach taken avoids the use of these extra transducers, and instead infers information about the rotor speed from measurements of the motor current. An additional criterion for the speed estimation method is that the method for determining speed must not be dependent on the temperature of the motor. This additional criterion is included because these motors get quite warm when used in air handlers (ranging from 20°F to 100°F above the ambient temperature, in the experience of the author) as the motor windings heat up during continuous operation.

In light of these considerations, the method used to identify the rotor speed analyzes

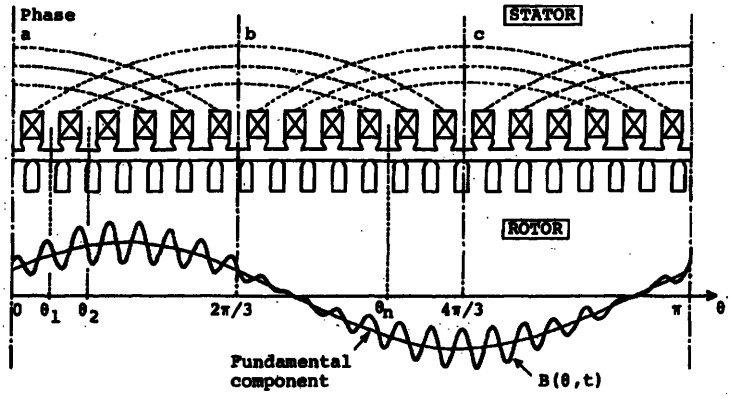


Figure 2-4: Picture of effect of rotor slot harmonic, from [74].

the frequency content of the motor currents and identifies harmonics that are related to the interaction of the rotor slots with the magnetomotive force (MMF) wave in the air gap of the motor. The underlying phenomenon that makes this method possible is that the variations in the air gap due to the rotor slots cause corresponding variations in the spatial permeance wave and magnetic flux density in the air gap. As these variations thus show up in the motor currents, knowledge of the number of rotor slots, the electrical drive frequency, and a few other pieces of information make it possible to identify the rotor speed by tracking the harmonic content of the stator currents. This method is developed in [74] by examining the effect of the rotor slots on the induced stator voltages; an alternate derivation is also developed in [3, ch. 9] that analyzes the effect on the permeance waves as well as the components of the MMF wave. In addition, static and dynamic eccentricities also have an effect on the permeance wave, introducing additional harmonic content in the magnetic flux wave that can also be related to the rotor speed. Figure 2-4 qualitatively illustrates the interaction of the rotor slots with the magnetic flux density wave in a developed diagram of the rotor.

$$\Phi_{gs}(\phi_m, \theta_e) = \mathrm{MMF}_{gs}(\phi_m, \theta_e) P_{gs}(\phi_m, \theta_e) \tag{2.10}$$

where Φ_{gs} is the flux in the air gap as referred to the stator, P_{gs} is the permeance of the air-gap as referred to the stator, ϕ_m is the mechanical angle of the rotor, and θ_e is the electrical angle of the stator.

The MMF wave can be written as having the following components: principal harmonics, stator slot harmonics, and rotor slot harmonics, as discussed in [3, p. 330], and [104]. The treatment in [105] illustrates this in a particularly straightforward manner, and is the basis for the following discussion. The expression for the rotor slot harmonics can be derived without considering the effect of the rotor slots by first considering only the principal harmonics in the MMF wave,

$$\text{MMF}_{gs} = A\cos(pnx \pm \omega t), \qquad n = 6k \pm 1,\ k = 1, 2, 3 \ldots \tag{2.11}$$

where p is the number of pole-pairs, n is the order of the harmonic, ω is the radian frequency of the electrical excitation, and x is the linear distance from $\theta = 0$ in the developed diagram, or linear model of the machine. In this initial case, no eccentricity is assumed, so that the permeance can be described by

$$P_{gs} \approx P_0. \tag{2.12}$$

The flux in the air gap therefore takes the form

$$\Phi_{gs} = AP_0\cos(pnx \pm \omega t) \tag{2.13}$$

with respect to the stator. This expression can be modified by transforming the rotor position [41] into a reference frame that rotates synchronously with the rotor via $x = \omega_r t + x'$, where x is in the stator frame of reference, x' is in the rotor frame of reference, and ω_r is the rotational frequency of the rotor in radians, so that

$$\Phi_{gr} = AP_0\cos[pn(x' + \omega_r t) \pm \omega t] \tag{2.14}$$

with respect to the rotor.

The effect of the rotor slots can be seen quite clearly by looking at a representative MMF wave for a 44 rotor bar, 4 pole motor as seen in Figure 2-5. It is clear that, while this waveform does have harmonic content at the frequency of the fundamental MMF wave, it will also have a large component at the frequency kR, where k is the number

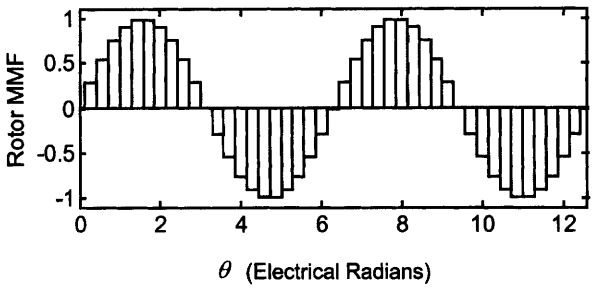


Figure 2-5: MMF pattern of a representative four-pole, 44-slot rotor, from [105].

of the harmonic and R is the number of rotor slots. These harmonics can also be seen in the lower part of Figure 2-4. The effect of these rotor slots can be viewed as introducing an additional permeance wave $\alpha P_0 \cos(Rx')$, which will multiply the magnetic flux Φ_{gr}, generating additional harmonic content in this waveform. By including this additional effect, the magnetic flux in the air gap can now be written as

$$\Phi_{grh} = \alpha P_0 \cos(Rx') A P_0 \cos[pn(x' + \omega_r t) \pm \omega t], \tag{2.15}$$

which can be simplified to

$$\Phi_{grh} = A_1 P_0^2 \cos[Rx' \pm pn(x' + \omega_r t) \pm \omega t - \phi_1], \tag{2.16}$$

where A_1 represents the set of multiplicative constants in the front of the expression and ϕ_1 represents the offset in the spatial phase of the rotor slots. Equation 2.16 expresses the magnetic flux in the rotor frame of reference, but this flux can also be formulated in the stator frame of reference, as

$$\Phi_{gsh} = A_1 P_0^2 \cos[(R \pm pn)(x - \omega_r t) \pm pn\omega_r t \pm \omega t - \phi_1]. \tag{2.17}$$

By making the substitution

$$\omega_r = \frac{(1-s)}{p}\omega, \tag{2.18}$$

where s is the fractional slip of the machine [50], this can finally be rewritten as

$$\Phi_{gsh} = A_1 P_0^2 \cos\left[(R \pm pn)x - \left(R\frac{(1-s)}{p} \pm 1\right)\omega t - \phi_1\right]. \tag{2.19}$$

The principal slot harmonics (PSH) can thus be found by analyzing the stator current and looking at the frequency corresponding to

$$\left(R\frac{(1-s)}{p} \pm 1\right)\omega t. \tag{2.20}$$

Though this derivation assumes that the air gap is uniform and straight, loading can introduce both static and dynamic eccentricity to the system, as discussed in [154, p. 307-9]. Static eccentricity is defined as an eccentricity in which the location of the minimum air-gap distance is fixed in space. This could take place, for example, if the location of the rotor were translated to the side of the true center of the stator due to poor construction of the rotor or stator. This eccentricity is only space dependent, and can be modeled by

$$P \approx P_0 + P_1 \cos x. \tag{2.21}$$

Dynamic eccentricity, on the other hand, is defined as an eccentricity in which the minimum air gap location rotates with the rotor, and can be caused by such phenomena as a bent or misaligned rotor shaft, worn bearings, and so forth. In comparison to the static eccentricity, this eccentricity is both time and space dependent, and can be modeled by

$$P \approx P_0 + P_2 \cos(x - \omega_r t). \tag{2.22}$$

The effect of both of these eccentricities can be seen in the corresponding term for the rotor slot harmonics. The effect of the variation in the permeances is worked out in detail in [104], showing that the location of the slot harmonics resulting from the rotor slots and the two types of eccentricity is given by

$$f_{sh} = f_e\left[(kR + n_w)\frac{(1-s)}{P/2} + n_d\right] \tag{2.23}$$

where f_{sh} is the frequency of the slot harmonics, f_e is the electrical supply frequency,

$n_d = 0, 1, 2, 3 \ldots$ is the order of the eccentricity (0 for static eccentricity, and $1, 2, 3, \ldots$ for dynamic eccentricity), k is any integer, and $n_w = \pm 1, \pm 3, \ldots$ is the order of the stator MMF time harmonic. By using this equation, it is possible to locate harmonics in the current waveform that are related to the rotor speed at a given point in time.

It is worth noting that, as illustrated in [105], not all of these rotor slot harmonics will be visible in the current waveform, due to the overlap with other MMF harmonics, the harmonic content of the voltage waveforms driving the motor, and other related phenomena. As long as some of these harmonics can be observed in the current waveform, however, they can be used to identify the rotor speed. Other research has developed methods of tracking these harmonics in real-time from observations of the current for a single motor [70, 71], but as the objective of this research is the demonstration of the feasibility of this technique to identify the rotor speed for the purpose of estimating airflow, the focus of this research is limited to that scope.

2.4 Torque Estimation

The purpose of the second module of the airflow diagnostic method is the generation of estimates of the torque produced by the motor during its steady-state operation. As the considerations that motivated the use of electrically-based methods for identifying rotor speed also apply to this module, this method was designed to generate these torque estimates solely on the basis of electrical observations of the motor. These estimates of torque will change as the airflow through the fan changes, so this method must be able to identify the torque-speed relation for the motor over the range of operating speeds.

Variations in motor temperature over the operating cycle, as discussed in §2.3, have a substantial effect on the torque-speed characteristic of the motor. Since the resistance of the stator windings is dependent upon their temperature, large changes in the motor temperature will cause corresponding changes in the winding impedances and the motor's torque-speed characteristic. While some applications that require knowledge of the torque-speed curve are able to use torque-speed data identified with specialized instrumentation (such as a dynamometer) before the motor is installed in the air handler, the site-specific nature of the temperature changes, which are dependent on the run time, the

airflow conditions, and so forth, require the torque-speed characteristic of the motor to be identified *in situ*.

A motor model needed to be selected to relate the set of electrical observations of the motor to its torque-speed characteristic. While a variety of types of system models could potentially be applied to this system, a model based upon the physical interactions governing the behavior of the system was used because of the range of behavior that could be accurately captured by a small set of parameters. This physically-based model was also appealing because the set of physical parameters characterizing the electrical response of the system can also be used to describe the mechanical behavior of the system. The use of physical knowledge to construct a mapping between a set of electrical behavior and mechanical behavior without observations of both types of behavior is an extremely powerful attribute that is difficult to replicate with other types of models. The approach taken in this research therefore involves the identification of a set of parameters that describe the observed motor voltages and currents using this model. These parameters will then be used to predict the torque-speed curve describing the mechanical behavior of the motor.

Because of this structure, the development of the torque estimation module will be broken into three constituent parts. In the first of these parts, the electromechanical model of the induction machine will be developed, so that the relationship between the electrical and the mechanical behavior can be clearly described. Since the technique used to determine the parameters of this model for a given set of experimental observations involves both a simulation component and a parameter identification component, tools that were developed for simulating the time-domain behavior of this induction motor model in response to experimental inputs will be discussed, and then the parameter identification method itself will be explained.

2.4.1 Induction Machine Model

As is fitting the status of induction motors as one of the "workhorses of industry", which they have held since shortly after their invention by Nikola Tesla in 1888, they have been extensively studied and modeled. The basic equations describing the behavior of the induction machine as described in this section are well established [3,79]. One of the major advances in the understanding and analysis of these machines was formulated by R.H.

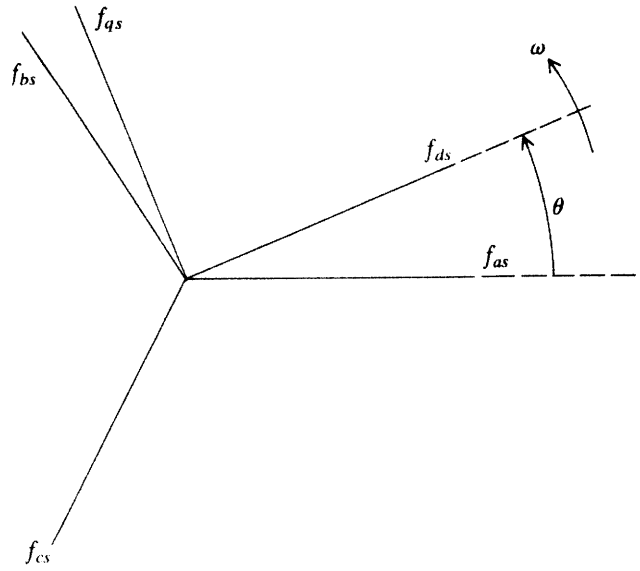


Figure 2-6: Vector representation of the input and output of the Park transformation for an arbitrary variable f, adapted from [79, p. 136].

Park in 1929 [110], in which he simplified the time-varying inductances of the motor by transforming the motor equations into a reference frame that is rotating synchronously with the fields in the machine. This transformation, often referred to as the Park transformation, can be written as follows:

$$\begin{bmatrix} f_d \\ f_q \\ f_0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin(\theta) & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} \tag{2.24}$$

or, in a more compact form,

$$\mathbf{f_{dq0}} = \mathbf{T}(\theta)\mathbf{f_{abc}} \tag{2.25}$$

where f stands for the variable to be transformed, such as voltage, current, or magnetic flux. The angle θ of the transformation can theoretically be chosen arbitrarily, but one useful choice of this angle is the electrical angle of the voltages driving the stator windings. The effect of this transformation can be observed in Figure 2-6, in which θ is the argument of the transformation $\mathbf{T}$, and ω is the rate of change of that angle, i.e. $\omega = d\theta/dt$.

Interpreting each of these vectors to be a voltage, for example, this picture makes it

clear that the Park transformation changes the three-phase voltages that are 120° degrees apart into two equivalent d-axis and q-axis voltages that are 90° degrees apart. This simplification results because of the balanced nature of the three-phase set being transformed; were that not the case, a 0-axis voltage would also result. As the utility is usually balanced when driving small motors such as are used in an air handler, however, this is not a concern. While this transformation makes the equations describing the behavior of the induction machine much more tractable than they would otherwise be, it is also useful because it can be applied directly to a set of three-phase electrical observations, reducing the three sets of voltages into 2 orthogonal components. This use of the Park transformation will be exploited in §2.4.2 to transform the set of observed three-phase stator voltages driving the induction machine into the equivalent d- and q-axis stator voltages, and it will also be used in Chapter 3 to facilitate the detection of compressor faults.

After applying the Park transformation to the equations describing the dynamics of the induction machine, the machine can be described by the following equations:

$$\frac{d\lambda_{qs}}{dt} = v_{qs} - R_s i_{qs} - \omega_e \lambda_{ds} \tag{2.26}$$

$$\frac{d\lambda_{ds}}{dt} = v_{ds} - R_s i_{ds} + \omega_e \lambda_{qs} \tag{2.27}$$

$$\frac{d\lambda_{qr}}{dt} = v_{qr} - R_r i_{qr} - (\omega_e - p\omega_r)\lambda_{dr} \tag{2.28}$$

$$\frac{d\lambda_{dr}}{dt} = v_{dr} - R_r i_{dr} + (\omega_e - p\omega_r)\lambda_{qr}. \tag{2.29}$$

where λ denotes the flux linkages with the rotor variables and parameters reflected to the stator, ω_e is the frequency of the stator excitation (i.e. the frequency of the drive voltage) in rad/s, and ω_r is the rotor speed in rad/s, and p is the number of pole pairs. The voltages v_{ds} and v_{qs} represent the driving voltages in most experimental applications, as v_{dr} and v_{qr} are set to zero due to the fact that the rotor bars are shorted together on a squirrel-cage

machine. The flux linkages and the currents are related by the following equations:

$$\lambda_{qs} = L_{ls}i_{qs} + L_m(i_{qs} + i_{qr}) \tag{2.30}$$

$$\lambda_{ds} = L_{ls}i_{ds} + L_m(i_{ds} + i_{dr}) \tag{2.31}$$

$$\lambda_{qr} = L_{lr}i_{qr} + L_m(i_{qs} + i_{qr}) \tag{2.32}$$

$$\lambda_{ds} = L_{lr}i_{dr} + L_m(i_{ds} + i_{dr}). \tag{2.33}$$

The torque of electrical origin produced by the motor is given by

$$\tau_e = \frac{3}{2}p(\lambda_{qr}i_{dr} - \lambda_{dr}i_{qr}). \tag{2.34}$$

This torque τ_e is related to the mechanical load of the fan by the usual force balance equation,

$$\frac{d\omega_r}{dt} = \frac{1}{J}(\tau_e - \beta\omega_r^2). \tag{2.35}$$

The $1/J$ term will also be referred to as K (as it is in the code in Appendix A). After (2.26)-(2.29) and (2.35) are integrated over the time interval of interest, the d- and q-axis currents need to be transformed back into the lab frame. This is performed by applying the inverse Park transform,

$$\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 1 \\ \cos(\theta - 2\pi/3) & -\sin(\theta - 2\pi/3) & 1 \\ \cos(\theta + 2\pi/3) & -\sin(\theta + 2\pi/3) & 1 \end{bmatrix} \begin{bmatrix} f_d \\ f_q \\ f_0 \end{bmatrix} \tag{2.36}$$

or

$$\mathbf{f_{abc}} = \mathbf{T}^{-1}(\theta)\mathbf{f_{dq0}} \tag{2.37}$$

The angle θ that is used in the inverse transformation must be the same as that used in the forward transformation to ensure the transformation's invertibility.

The steady-state behavior of the induction motor can also be described by an equivalent circuit model, as shown in Figure 2-7. This circuit model does not describe the dynamics of the system in *dq*-space, but rather describes the behavior of the induction motor in steady-state operation in the lab frame. This model is used because it can provide intu-

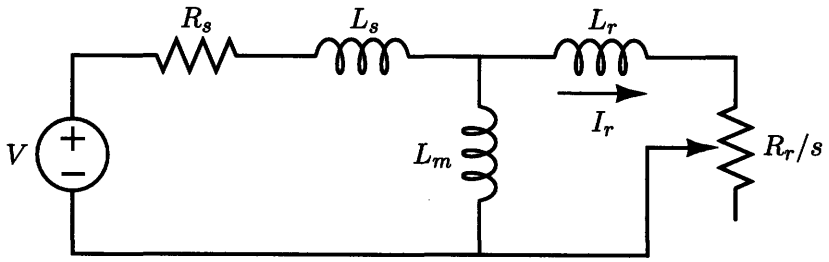


Figure 2-7: Equivalent circuit model for one phase of an induction machine.

itively appealing expressions for the steady-state behavior of the induction machine.

As described in [50, p. 317], the mechanical torque developed by the motor is given by

$$T_{mech} = \frac{n_{ph} I_r^2 (R_r/s)}{\omega_s}, \tag{2.38}$$

where n_{ph} is the number of phases of the motor, I_r is the current flowing through the rotor, R_r is the rotor resistance, ω_s is the synchronous speed of the motor, and s is the value of the fractional slip of the motor. The fractional slip is calculated by

$$s = \frac{n_s - n}{n_s}. \tag{2.39}$$

where n_s is the synchronous speed of the motor, and n is the present operating speed. It is important to again note that the parameters of this steady-state model are the same as the parameters of the transient model described earlier, and that the parameters that describe the transient electrical behavior of the electrical machine will also describe its torque-speed characteristics.

2.4.2 Motor Simulation

While the model discussed in §2.4.1 describes the dynamic behavior of the motor with a set of ordinary differential equations, these ODEs need to be integrated over the time interval of interest to describe the time evolution of the model variables, such as the currents or the magnetic fluxes. Though the implementation of such a simulation is fairly straightforward, a few particular issues do arise in this particular case because of the need to simulate the response to a set of the measured stator voltages. Motor simulations often operate

under the assumption that the stator voltages are constant and therefore fix them in the simulation, but the experimental measurements of the stator voltages demonstrated that this assumption was not completely valid in the laboratory setting. The simulation method therefore had to compensate for these small variations in the stator voltages, so that the motor simulations could be driven with the same input as the actual motor.

A standard Runge-Kutta variable-stepsize, fourth-order embedded method was used to integrate the system of ODEs; this and related methods are described in [113, ch. 16], [39,40,121,122]. The method chosen has high accuracy due to its variable stepsize control, and also has proven to be relatively fast. A dedicated solver was written in Matlab and C to test the performance of the method, but the Runge-Kutta-Fehlberg (`rkf45`) algorithm included with the GNU Scientific Library for C (GSL) was used in the final implementation, due to its good performance and the ease with which it could be integrated into the rest of the estimation program.

It was necessary to apply the Park transformation to the three-phase set of lab-frame voltages V_{an}, V_{bn} and V_{cn} to obtain the 2-phase set of voltages V_{ds} and V_{qs} in the stator reference frame to run the motor simulation with the set of measured electrical inputs. An estimate of the electrical angle θ_e of the utility voltage is therefore needed to compute the coefficients in the time-varying transformation matrix. Unfortunately, this angle cannot be modeled as a simple affine function of time, e.g., $\theta_e = \omega_e t + \phi$, due to the nonidealities present in the system; a more accurate functional description of the electrical angle is $\theta_e = \omega_e t + \phi(t)$. This representation is useful because it both captures changes in the phase angle that might occur because of other loads on the same voltage bus, as well as measurement error in the electrical frequency ω_e. The time-varying phase angle $\phi(t)$ will change the relative magnitudes of V_{ds} and V_{qs}, but as long as the transformation $\mathbf{T}(\theta_\mathbf{e})$ is the same as $\mathbf{T}^{-1}(\theta_\mathbf{e})$, the transformation will be invertible, and the resulting V_{ds} and V_{qs} will characterize the three-phase voltages driving the motor.

The mechanics of transforming the stator voltages can be broken down into two steps. In the first step, the delta-referenced voltages driving the motor are transformed into an equivalent set of wye-referenced voltages. Though delta-referenced set of voltages can be transformed into the wye-referenced set through the complex transformation $V_{wye} = \sqrt{3}e^{\frac{i\pi}{6}}V_{delta}$, an easier means of transformation involves the simple solution of the linear

system that converts from one set of voltages to the other. This set of relations can be written

$$v_{an}[k] - v_{bn}[k] = v_{ab}[k] \tag{2.40}$$

$$v_{bn}[k] - v_{cn}[k] = v_{bc}[k] \tag{2.41}$$

$$v_{cn}[k] - v_{an}[k] = v_{ca}[k] \tag{2.42}$$

$$v_{an}[k] + v_{bn}[k] + v_{cn}[k] = 0 \tag{2.43}$$

or in matrix notation,

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} v_{an} \\ v_{bn} \\ v_{cn} \end{bmatrix}_{[k]} = \begin{bmatrix} v_{ab} \\ v_{bc} \\ v_{ca} \\ 0 \end{bmatrix}_{[k]} . \tag{2.44}$$

where these equations are solved at each sample point k, as indicated by the indices at the bottom right-hand corner of the vectors. This is a simple overdetermined set of equations, and can be solved by the standard techniques for solving the normal equations [55, ch. 5] to determine V_{wye} from V_{delta}.

The second step in implementing this transformation involves the estimation of the electrical frequency ω_e. This was performed by finding the parameters V, ω_e and ϕ of $V\sin(\omega_e t + \phi)$ that best fit the first two line cycles of the voltage waveform V_{an}. A modified Gauss-Newton method of solving nonlinear least squares problems was used to find these parameters (discussed in more detail in §2.4.3, §3.2, and [123,126,128]); since the waveform being fitted was only two periods long, this method worked very well. With this estimate of ω_e the Park transformation matrix could be constructed for each time instant $t[k]$, and the three-phase set of utility voltages could be transformed into the stator reference frame.

The above method of estimating the characteristics of the voltage waveform was implemented and found to work quite well, as can be seen in plots shown in Figures 2-8 and 2-9. Figure 2-8 shows that the fit is fairly good and captures most of the behavior of the voltage of the utility. Figure 2-9 illustrates the size and characteristics of the resid-

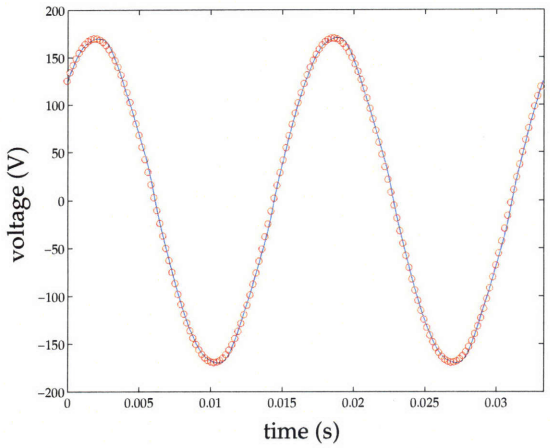


Figure 2-8: Plot of the observed and fitted voltage waveform. The observed waveform is in blue, and the fitted waveform is in red.

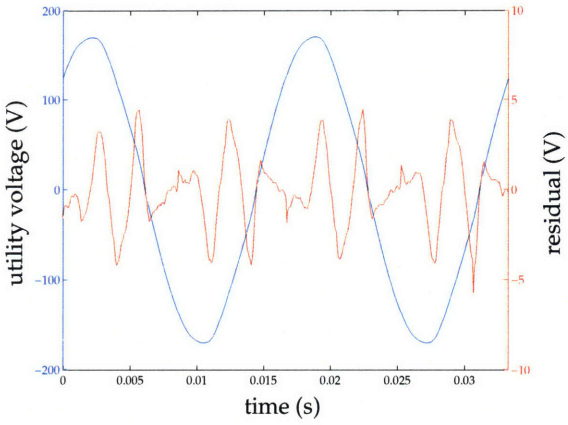


Figure 2-9: Plot of the observed voltage waveform superimposed on a plot of the residual $y_{obs} - y_{fit}$ between the observed voltage and the correponding fitted sinusoidal voltage. The observed voltage is in blue, while the residual is plotted in red.

ual, which is dominated by the nonsinusoidal components of the utility voltage, rather than failures in the fitting of the sine wave to the data. The data illustrated in Figure 2-9 support this observation, as the peaks of the voltage waveform are much flatter than an ideal sine wave, and there is much less curvature around the zero crossings than would otherwise be expected.

Some of the effects of these nonsinusoidal characteristics and noise in the waveform can cause errors in the estimates of the parameters in the model of the voltage, which is $V\sin(\omega t + \phi(t))$. The need for this model can be seen by comparing the predictions made by estimating the parameters for the model $\hat{v}(t) = V\sin(\omega_e t + \phi)$ from the first two cycles of utility voltage to the set of observations of that same voltage v_{ab} after incrementing the time index by 35 seconds. The resulting simulated data $\hat{v}_{ab}(t)$ and the set of observed data $v_{ab}(t)$ are illustrated in Figure 2-10.

It is clear from Figure 2-10 that the model for the utility voltage in the motor simulation must assume that the phase angle of the voltage is time-varying; while the estimate of the phase angle appears to be accurate at the beginning of the observed voltage waveform, it is off by approximately $\pi/4$ after 35 seconds. This problem is particularly noticeable when it is applied to the parameter estimation process on the motor, as the residual $\hat{i}_a(\mu) - i_a$ will

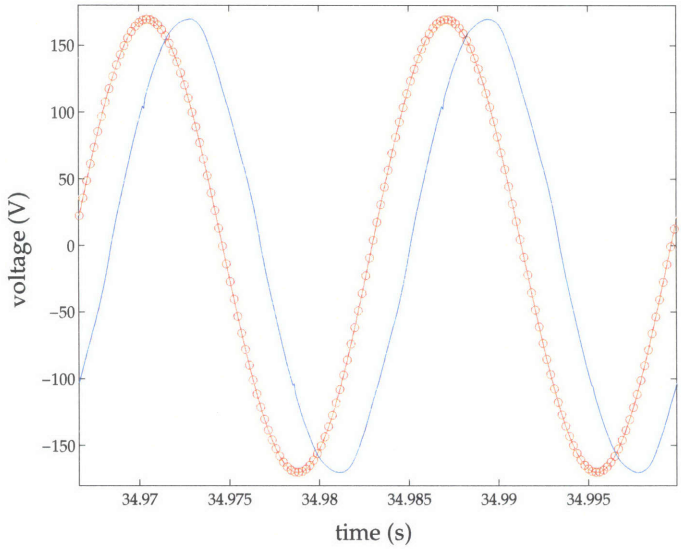


Figure 2-10: Plot of the observed and fitted voltage waveforms running the simulation for 35 seconds. The observed waveform is in blue, and the fitted waveform is in red.

be dominated by this phase difference, rather than differences due to the incorrect parameters. By obtaining an initial estimate of the parameters for the voltage, and then using this estimate to transform the observed lab-frame voltages in to the synchronously rotating reference frame, the transformed voltages will reflect the time-varying behavior of the utility voltage while still providing the input in the right form for the motor simulation. Figure 2-11 illustrates the effect of transforming the utility voltage into the rotating reference frame. It is apparent from this plot that the phase angle of the voltages (assuming a balanced set) are not constant, but instead vary as the effective phase offset $\phi(t)$ varies. By transforming the voltages into this rotating coordinate system, the characteristics of the utility can be precisely described and cast into a form that is useful for the simulation method.

One additional modification to the standard motor simulation method, needed to accomodate the set of measured voltages, was due to the variable stepsize method used to solve the system of ODEs. Since the stator voltages are only sampled at fixed time intervals, the variable stepsize method will not be able to solve the system at the arbitrary time points specified by the stepsize control algorithm. Rather than use a fixed step solver, which could result in reduced accuracy, an interpolation method was used to approximate the value of the inputs voltage waveforms between adjacent samples. The interpolation method used was a cubic spline method, which matches the first and second derivatives

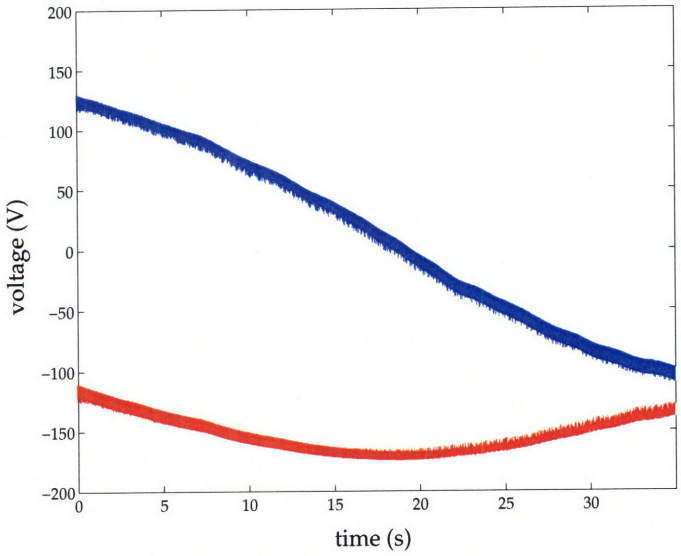


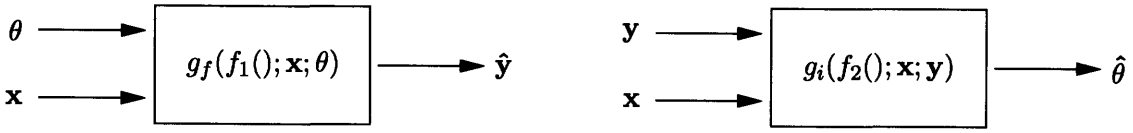
Figure 2-11: Plot of the three-phase set of voltages transformed into the rotating reference frame, referred to the first two cycles of the observed phase A voltage. The d-axis voltage is plotted in blue, and the q-axis voltage is plotted in red.

at the end of each interval. The particular implementation of this method used, which also incorporated a table lookup strategy to accelerate its performance, came from the standard distribution of the GSL libraries. Additional information on this interpolation method can be found in [151, ch. 9].

The overall method for simulating the motor for a given set of parameters was developed in Matlab and in C, and the final set of simulation code that was applied in this method was implemented in C using the GSL libraries. This code is available in Appendix A.

2.4.3 *Parameter Identification*

A parameter identification method is needed to estimate the set of parameters that accurately describe the observed behavior of the machine to estimate the torque produced by the induction motor at a given speed. Many of the particularly salient features of the parameter identification process can be illustrated clearly by comparing the underlying behavior of both the parameter estimation method and the simulation method for the induction machine. The simulation and parameter estimation problems can be described by the two illustrations in Figures 2-12a and 2-12b. The figure on the left, Figure 2-12a, sum-



(a) The basic structure of the simulation process.

(b) The basic structure of the estimation process.

Figure 2-12: Block diagram representations of simulation and estimation processes.

marizes the the problem of simulation, in which the known set of information includes the model of the system $f_1()$, the set of inputs $\mathbf{x}$, and the set of model parameters θ. This simulation g_f then takes the set of inputs and tracks the evolution of the system as determined by the constituent relationships of the model $f_1(\mathbf{x};\theta)$. This is also referred to as solving a forward problem. The output of the simulation is referred to here as $\hat{\mathbf{y}}$, since it is related only to the inputs and the model of the system, which may not describe the output $\mathbf{y}$ of the actual system, depending on the accuracy of the description of the model f, the parameters θ, or the inputs $\mathbf{x}$.

In comparison, the type of problem illustrated in Figure 2-12b is often referred to as an inverse problem; the objective in solving this type of problem is the identification of a set of parameters that best map the observed input $\mathbf{x}$ to the observed output $\mathbf{y}$, subject to the constraints imposed by the structure of constituent relationships $f_2(\mathbf{x};\mathbf{y})$. The inverse problem is solved by a function or an algorithm g_i that takes this input-output mapping $f_2(\mathbf{x};\mathbf{y})$ and produces an estimate of the parameters $\hat{\theta}$ that ideally is close to the actual set of parameters θ which accurately describes the observed system. Inverse problems tend to be challenging in two regards: first, while the model f_1 for calculating the output $\hat{\mathbf{y}}$ that is based upon the input $\mathbf{x}$ and the model parameters θ tend to be relatively well understood, a direct relationship between the input-output pair $(\mathbf{x};\mathbf{y})$ and the set of parameters $\hat{\theta}$ tends to either be either intractible for the purposes of implementation or difficult to formulate theoretically. The second challenge typically posed by inverse problems is that it is often difficult to characterize the sensitivity of this input-output mapping to the types of changes in the system parameters.

Using the nomenclature of Figure 2-12b, the function g_i captures the structure of the estimation method. Closed-form representations of this function are often not available or are poorly conditioned, so that practical considerations such as noise in $\mathbf{x}$ or $\mathbf{y}$, uncertainty,

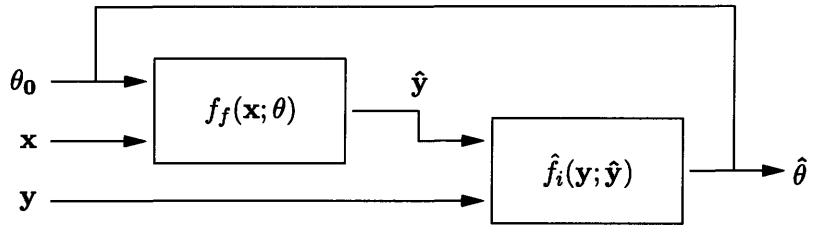


Figure 2-13: Structure of an estimation method that incorporates a forward model.

or variability in the structure of g_i can make the use of such functions problematic. An alternative approach to solving these problem used in this research involves embedding the forward problem solver in the overall inverse problem loop, so that the internal function f_1 in the simulation block and f_2 in the estimation block are the same, e.g., $f_1 = f_2 = f_f$. This approach is illustrated in Figure 2-13.

This diagram illustrates the means by which the simulation method is incorporated into the overall parameter estimation method. An initial guess for parameters θ_0 and the inputs to the system $\mathbf{x}$ are used as inputs to the simulation routine f_f, the simulation is run, and then the resulting output of this simulation $\hat{\mathbf{y}}$ is fed into the estimation routine f_i. This estimation routine analyzes the residual between the two vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$, and computes a new parameter estimate θ. This new estimate of the parameters is then fed back into the simulation routine. This cycle is repeated until an exit condition is reached, the intended one being that the residual between the observations $\mathbf{y}$ and the predictions $\hat{\mathbf{y}}$ is reduced below an established threshold. Other potential exit conditions that are implemented to prevent the estimation routine from running indefinitely include exceeding a set number of iterations, no change in the parameters, and no change in the evaluated residual, despite the fact that it remains above a set threshold.

It is important to note the distinction between the functions f_i and g_i. The estimation block of Figure 2-12b describes the overall parameter estimation method, since it takes $\mathbf{y}$ and $\mathbf{x}$ as inputs and generates an estimate $\hat{\theta}$ of the system parameters. The block $\hat{f}_i$, in comparison, generates an estimate of parameters $\hat{\theta}$ based upon the residual $\hat{\mathbf{y}} - \mathbf{y}$, which is not directly dependent on the set of inputs $\mathbf{x}$.

Applying the structure of this general parameter estimation method to the problem of identifying the induction motor parameters, the set of inputs $\mathbf{x}$ are the d- and q-axis stator

voltages, and the output of the motor simulation $\mathbf{y}$ is the A-phase stator winding current $\hat{i}_a$. While all three phases of the current could be used for the minimization instead of one phase, each of the phases will be the the same (albeit 120° out of phase) with the motor in balanced operation, and will not contribute additional information to the minimization method that is used to generate a new set of parameter estimates $\hat{\theta}$ based upon the residual $\mathbf{r}(\theta) = \hat{\mathbf{i}}_\mathbf{a}(\theta) - \mathbf{i}_\mathbf{a}$.

A variety of minimization techniques could potentially be used to find the argument θ which minimizes $\mathbf{r}(\theta)$. These techniques can generally be classified into two categories: gradient-based minimization and non-gradient-based minimization. While non-gradient-based techniques, such as simulated annealing or global search methods, are quite popular and used in a variety of applications, a gradient-based technique (nonlinear least squares) was used because the information embedded in the physically-based construction of the model can be exploited with this approach. Nonlinear least squares is also a useful minimization method because it does not rely solely upon a grid search of the solution space to locate the minimum of the loss function.

The objective of the nonlinear least squares algorithm [120] that was used to generate the new set of parameter estimates $\hat{\theta}$ was to minimize the "sum-of-squared errors" loss function $V(\theta)$, where

$$V(\theta) = \sum_{k=1}^{M} r_k^2 = r^T r, \tag{2.45}$$

and there are M parameters of the system. Most implementations of nonlinear least squares methods are essentially modifications of Newton's method, which iteratively updates the set of parameters θ to solve for $\mathbf{G}(\theta^{(i)}) = 0$, where $\mathbf{G}(\theta)$ is the gradient of $V(\theta)$. This update procedure can be written as

$$\theta^{(i+1)} = \theta^{(i)} - \left[\nabla \mathbf{G}(\theta^{(i)})\right]^{-1} \mathbf{G}(\theta^{(i)}). \tag{2.46}$$

While nonlinear least squares is an attractive minimization method, it does share some attributes with general gradient search algorithms that can be problematic. Since gradient-based methods are designed to locate the minimum of the loss function by moving in the direction of the negative gradient $\mathbf{G}(\theta)$ until the gradient is zero, any other location

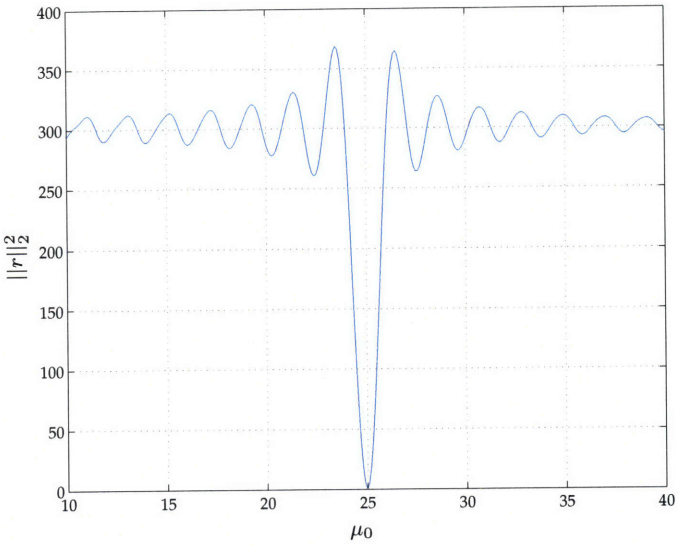


Figure 2-14: Loss function $V(\mu)$ for the example problem, where μ is varied between 10 and 40.

of $V(\theta)$ where the gradient is also equal to zero (called local minima) will represent a stopping point for the underlying gradient-search method which may occur before the set of parameters is located that best characterizes the observations.

This behavior can be best illustrated via an example: consider a set of observations of the system $y_{obs} = \sin(25t)$. The dynamic response of the system to the input t is modeled as $\hat{y} = \sin(\mu t)$, so that the objective of the nonlinear least squares function is the determination of μ such that it minimizes the residual $r(\mu) = \hat{y}(\mu; t) - y_{obs}(t)$. The operation of the nonlinear least squares algorithm begins with an initial guess μ_0 and successfully refines the estimate of the parameter μ until the nonlinear least squares algorithm either converges or reaches another exit condition. One can see the potential difficulties with this by looking at the loss function $V(\mu)$ as the parameter μ is varied over a range of values, as seen in Figure 2-14. Recalling that basic gradient-based methods typically stop once they reach a point in the loss function where the gradient is equal to zero, it is clear from this plot that the initial guess μ_0 must be very close to the actual parameter (e.g., $\mu = 25$) for the method to converge on this solution. For example, an initial guess of 19.5 will gradually converge to a final parameter estimate of 20.3, as the gradient is zero at this point. For this reason, many practical implementations of nonlinear least squares incorporate modifications to the basic Newton iteration to reduce the susceptibility to this limitation.

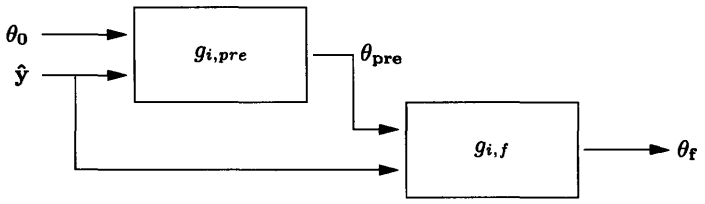


Figure 2-15: Block diagram illustrating the structure of the pre-estimation method.

This example makes it clear that the loss function is very sensitive to errors in the parameters that are manifested in the sinusoidal response of the system. The estimation method was therefore broken into two consecutive steps to reduce this sensitivity, as illustrated in Figure 2-15. In the first of these steps, a set of preliminary parameter estimates, or pre-estimates, is generated by processing the data in a manner that is less susceptible to the effects of the local minima. These pre-estimates, while they do not describe the full behavior of the system, represent a set of parameters that are much closer to the final set of the parameters of the system than most initial guesses that are generated by a user. After obtaining these pre-estimates, the final set of parameter estimates are generated by using the pre-estimates for an initial guess and finding a set of parameters that describe the full set of observations. A general overview of this pre-estimation technique, as well as examples of other applications, is given in [125].

It is important to note that both the pre-estimates $\theta_{\mathbf{pre}}$ and the final parameter estimates $\theta_{\mathbf{f}}$ are generated via a parameter estimation process. The main distinction between these two parameter estimates is that these pre-estimates are not designed to describe the full set of observations, but rather to facilitate the eventual identification of parameters that best describe the full set of observations and are not trapped in local minima. Such an approach could theoretically estimate the parameters in a number of consecutive steps, forming a continuation approach to parameter estimation, allowing the parameter estimates to be gradually refined in a controlled way. In comparison to such a multi-step continuation approach, the motor parameter estimation problem only requires a two-step approach to parameter identification.

The method used to obtain pre-estimates of the motor parameters was developed after observing that the shape of the envelope of the motor current is sensitive to changes in many of the motor parameters, as suggested by [141]. By filtering the observed current

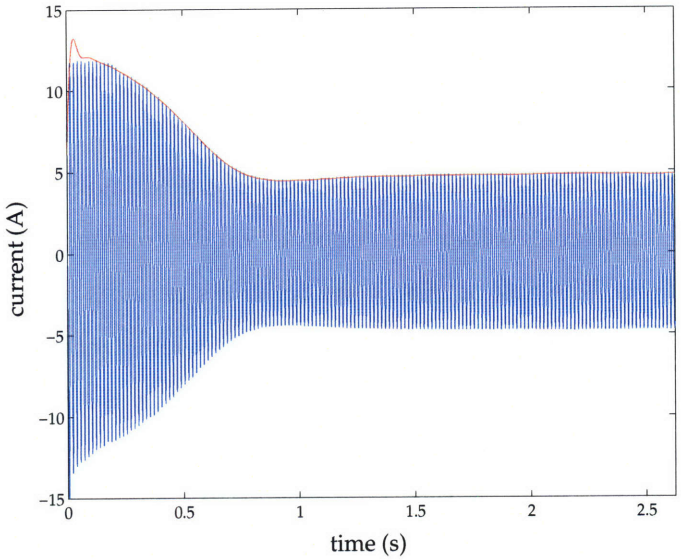


Figure 2-16: Plot of the observed phase A motor current along with the preprocessed current, which extracts the envelope of the current. The current is illustrated in blue, while the envelope is shown in red.

data with a low-pass filter, it was possible to preserve the shape of the envelope while eliminating the problematic 60 Hz component. The method used to filter the data is a standard demodulation technique described in [144, 145] and illustrated in Figure 2-16. A third-order Butterworth filter [109, ch. 7] was used for this purpose, and forward-backward filtering [60] was used so that the phase of the output data would not be distorted. One can see the effect of this filtering method on the motor current in Figure 2-16.

After filtering the observed current data, nonlinear least squares was used to iteratively determine the set of parameters that best fit the filtered motor current. The parameter estimation approach illustrated in Figure 2-13 was used to find these pre-estimates. In addition to generating a prediction of the motor currents for a given set of motor parameters, the motor simulation block (f_f) also filters the predicted current, so that nonlinear least squares can compare the filtered observed data to the filtered simulated data.

Once the set of pre-estimates is obtained, it is used as the initial guess for the minimization for the full (non-filtered) set of observations. The Levenberg-Marquardt method [120] for solving nonlinear least squares problems was used to determine the set of parameters located at the minimum of the loss function. The version of this algorithm included with the GSL libraries [52] was used in the C implementation of the method. This method was

tested on the fan motor, as well as the compressor motor, and it was found to identify parameters that successfully reproduced the observed data for both of these motors despite variations in their temperature and load.

While the above discussion describes the scope of the estimation method, the performance of the minimization algorithm also depends on the treatment of a few specific numerical considerations. The first of these considerations pertains to the scaling of the parameters: widely spaced parameters, whose size varies over several orders of magnitude, can be very sensitive to the parameter corrections applied by the nonlinear least squares algorithm. For example, if the gradient of the loss function in the direction of one parameter K is quite steep, the corresponding adjustment to the parameters could cause a large change in another very small parameter, such as β. These large adjustments in β could consequently cause problems for the convergence of the method as other parameters are adjusted according to the new position on the loss function.

This problem can be mitigated by scaling the parameters in the motor simulation according to their expected magnitudes, which are relatively easy for a user to estimate. This will allow the minimization algorithm to change all of the parameters by the same order of magnitude; for example, if the true parameters of the system are $\beta = 5 \times 10^{-6}$ and $K = 140$, scale factors of 10^{-6} and 100 would multiply the parameters β and K, respectively, inside of the simulation function, resulting in final parameter estimates of 5 and 1.4. This approach effectively stretches the residual space so that it makes all of the scales of the parameter gradients comparable.

Parameter scaling is also important for the determination of the parameter pre-estimates. By incorporating the expected scales for the parameters into the simulation function, the initial guesses for the first estimation step can all be set to one. The only exception to this is the estimate of the stator resistance R_s; the value of this parameter needs to be estimated to provide an adequate constraint for the other parameters. A simple measurement of R_s using a multimeter when the motor is cold has been found to suffice for an initial guess for this parameter. In the implementation of the torque-speed estimation method, the measured pre-estimate of R_s was fixed during the pre-estimation step and the pre-estimates of all other parameters were determined using the nonlinear least squares.

The second constraint that was imposed to improve the performance of the minimiza-

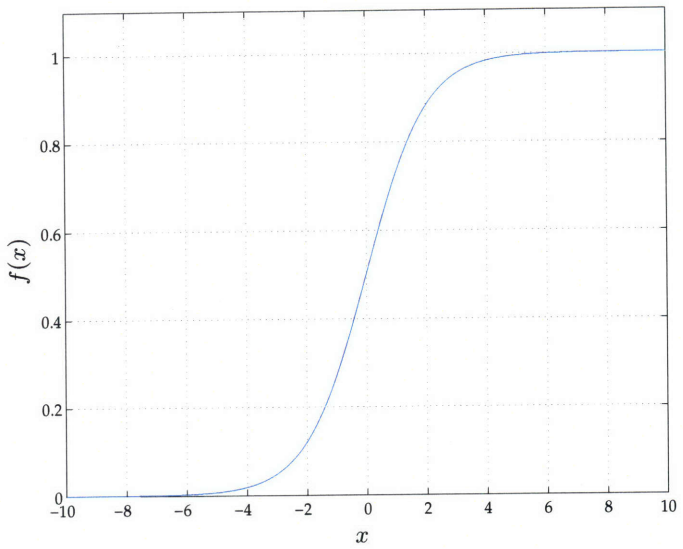


Figure 2-17: Plot of the response of the sigmoid function.

tion algorithm incorporates the information that the parameters of the model cannot be negative. The addition of a hard constraint that enforces nonnegativity on the parameters can be problematic because many such constraints, such as the application of the absolute value function, do not preserve derivative information and introduce other spurious behavior into the operation of the minimization algorithm. This constraint was implemented in this research by applying a sigmoid, or logistic, function to the parameters. The behavior of this function is given by

$$f(x) = \frac{1}{1 + e^{-\alpha x}}. \tag{2.47}$$

and can be seen in Figure 2-17, where α is set to 1.

This constraint was implemented in much the same way as the scaling; the parameters generated by nonlinear least squares were transformed using this sigmoid function, and the resulting transformed parameters were used in the motor simulation. In this way, the operation of nonlinear least squares was not explicitly constrained, but the transformed parameter estimates obtained from nonlinear least squares that were used as the parameters to drive the embedded motor simulation were guaranteed to be positive. Unlike the parameter scaling discussed previously, this constraint was only imposed during the pre-estimation step, as the parameter pre-estimates were sufficiently close to the final pa-

rameter estimates that this constraint was not needed during the final step.

The final numerical constraint imposed on the minimization method placed limitations on the size of the adjustment $\delta^{(i+1)} = \theta^{(i+1)} - \theta^{(i)}$ applied to the parameters after every iteration. Even after applying the appropriate scaling to the parameters, large gradients in the loss function can cause the parameter step taken to be very large. If the loss function is globally convex as is the case in linear problems, such an adjustment in the stepsize is useful since the gradient will be bigger as the distance from the minimum increases. This is not the case in nonlinear problems, however, and the resulting movement far away from the previous initial guess will not take advantage of the additional information that is often implicit in the initial guess for the parameters selected by the user.

This information can be incorporated into the minimization algorithm by imposing a set of linear constraints on the parameters, effectively placing a "rubber band" on the parameters so that they do not move far in consecutive iterations. This can be accomplished by adding an additional set of equations to the system of linearized equations that is solved at each step. If the system of equations that represents the linearized behavior of the nonlinear system at the current speed is written by $\nabla \mathbf{G}(\theta^{(i)})\delta^{(i+1)} = \mathbf{G}(\theta^{(i)})$, then the additional constraint can be implemented by solving the simultaneous set of equations

$$\begin{bmatrix} \nabla \mathbf{G}(\theta^{(i)}) \\ \mathbf{I} \end{bmatrix} \delta^{(i+1)} = \begin{bmatrix} \mathbf{G}(\theta^{(i)}) \\ \gamma \frac{\delta^{(i)}}{\theta_0} \end{bmatrix} \tag{2.48}$$

This technique is commonly known as regularization [47], and can be controlled by the size of the constant γ that multiplies the parameters. This parameter is often set by trial and error; values of γ which are too large prevent nonlinear least squares from adjusting the parameters at all, while values that are too small allow the parameters to change more than is desired. In the research, the implementation of regularization in the nonlinear least squares algorithm proved to be essential to obtaining useful parameters from the method.

While the parameter estimation method was primarily developed to identify the torque-speed curve of the motor solely on the basis of electrical measurements, the method was also modified so that torque-speed data measured on a dynamometer could be also incorporated into the parameter estimation method. This data was included using an approach

that was similar to the addition of the regularization constraints as discussed above; the torque-speed data was concatenated to the set of constraints that describe the system as given by

$$\begin{bmatrix} \nabla \mathbf{G_1}(\theta^{(i)}) \\ \nabla \mathbf{G_2}(\theta^{(i)}) \\ \mathbf{I} \end{bmatrix} \delta^{(i+1)} = \begin{bmatrix} \mathbf{G_1}(\theta^{(i)}) \\ \mathbf{G_2}(\theta^{(i)}) \\ \gamma \frac{\delta^{(i)}}{\theta_0} \end{bmatrix} \tag{2.49}$$

where $\mathbf{G_1}(\theta^{(i)})$ represents the gradient of the current residual with respect to each of the parameters θ_i, and $\mathbf{G_2}(\theta^{(i)})$ represents the gradient of the torque-speed residual with respect to each of the parameters θ_i. This approach, in which the two different sets of outputs of the same system are simultaneously minimized, generally follows the approach used in [123, p. 112]. Both approaches to the parameter estimation problem were tested, and will be discussed in the results section of this chapter.

2.5 *Fan Dynamics*

As the modules for identifying the rotor speed and the mechanical power produced by the motor have been covered, this discussion must turn to the mechanism by which the fan converts mechanical power into air flow. The following section is therefore intended to develop such a mechanism from first principles, with the necessary restrictions provided when considering the non-idealities of flow in real systems, as opposed to such a theoretical treatment. The discussion that follows is fashioned along the lines of the clear and insightful expositions written by Goodfellow [56, pp. 742-790] and Eck [46, ch. 1]. This discussion of fans will focus solely on centrifugal fans, as residential air handlers are almost universally equipped with this type of fan. Other helpful treatments of this material include [45,78,161]; in particular, [78] provides a helpful discussion of some of the complex flow patterns that can be experimentally observed in blowers and squirrel-cage fans.

Ultimately, diagnostic methods for identifying blocked airflow conditions and leaky ducts can both be understood as specific applications of a general method for detecting changes in the pressure difference across the fan, since one of the simplest explanations of how fans work is that they create a pressure gradient that causes air to flow. Accordingly,

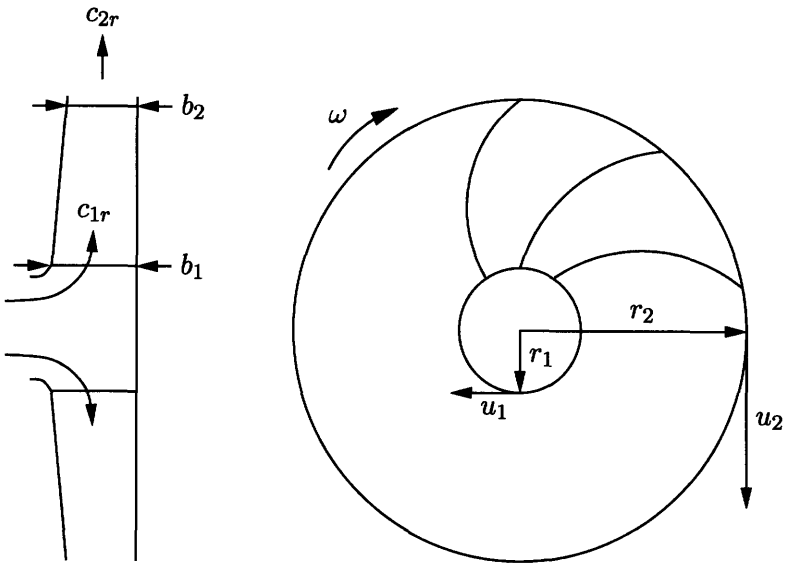


Figure 2-18: Diagram of mass flow through fan.

this development of a set of relations that govern fan behavior will seek to develop an expression for that pressure difference in terms of other variables of interest. Figure 2-18 illustrates a representative centrifugal fan with forward curved blades to establish a clear frame of reference for the following section. Applying mass conservation to steady-state flow through the fan,

$$\begin{aligned} \dot{m} &= \rho c_1 A_1 = \rho c_2 A_2 \\ &= 2\rho\pi r_1 b_1 c_{1r} = 2\rho\pi r_2 b_2 c_{2r} \end{aligned} \tag{2.50}$$

where ρ is the density of air in which the fan is operating, A_1 and A_2 are the cross-sectional areas of the inlet and outlet, c_{1r} and c_{2r} are the radial components of air velocities c_1 and c_2 at the inlet and outlet, r_1 and r_2 are the radii of the inner and outer edges of the fan blades, and b_1 and b_2 are the widths of fan blades at the inlet and the outlet of the impeller.

Referring again to Figure 2-18, the velocity of the inner and outer edges of the impeller blades (referred to as the inner and outer circumference velocities) can also be calculated by using the relation

$$u = 2\pi r n, \tag{2.51}$$

where u is the circumference velocity at the edge of the blade in consideration, r is the

radius of the fan impeller at the same edge of the blade, and n is the rotational speed of the fan. This relationship between the fan's rotational speed and the circumference velocity of the impeller blades is important, as it makes explicit the fact that the ratio of rotational speed to the circumference velocity at a given location on a fan blade is constant; the circumference velocity u_v^{II} can be determined from the speeds n^I, n^{II}, and the reference circumference velocity at the same point u_v^I,

$$\frac{u_v^I}{n^I} = \frac{u_v^{II}}{n^{II}} = 2\pi r \tag{2.52}$$

This is clearly a powerful tool, in that it makes it possible to scale variables related to circumference velocity with the speed of the fan. The objective of the remainder of this discussion is thus focused on developing a series of relationships that connect the circumference velocity to the air velocity and the mass flow rate. These relationships can be obtained from computing both a momentum balance and an energy balance on the air flowing through the fan.

In formulating an energy balance for this system, it is important to note that the change in potential energy is negligible for a fan in an air handler where the suction and discharge outlets are at approximately the same height. In addition, most of this discussion will focus on the development of the fan's characteristics assuming reversible (isentropic) behavior; the effects of irreversibilities will be reviewed after the overall process has been discussed. Given that isentropic behavior is being studied, the relation $Tds = dh - vdp$ can be used (where s, h, and v are the specific entropy, enthalpy, and volume respectively) with s set to 0 to indicate an isentropic process, to express the total change in specific enthalpy Δh_{tot} generated by the fan as

$$\begin{aligned}\Delta h_{tot} &= h_2 - h_1 + \frac{1}{2}c_2^2 - \frac{1}{2}c_1^2 \\ &= v(p_2 - p_1) + \frac{1}{2}(c_2^2 - c_1^2) \\ &= \frac{1}{\rho}(p_2 - p_1) + \frac{1}{2}(c_2^2 - c_1^2),\end{aligned} \tag{2.53}$$

assuming constant air density ρ. This expression can also be used to find the total isen-

tropic pressure difference across the fan, e.g.,

$$\Delta p_{tot} = \rho \Delta h_{tot} = \Delta p + \frac{1}{2}\rho(c_2^2 - c_1^2) \tag{2.54}$$

This pressure difference can be decomposed into two components, the static pressure p and the dynamic pressure $\frac{1}{2}\rho c^2$.

Another relationship between the various quantities under study can be computed by applying a momentum balance to the system. In order for air to flow through the blades of the fan impeller, a tangential force must be applied to the air. This occurs because of the basic assumption that the impeller is rotating around an axis, while the air entering the impeller passages does so in a direction which is radial with respect to that axis, causing the air to enter the blade passages tangential to their direction of movement. Applying a momentum balance for these tangential forces,

$$F_t = \frac{d}{dt}(mc_{2t} - mc_{1t}) = \dot{m}(c_{2t} - c_{1t}), \tag{2.55}$$

where c_{1t} is the tangential component of the air velocity at the inlet to the impeller blades, c_{2t} is the tangential component of the air velocity exiting the impeller blades, and q_m is the mass flow of air through the impeller apparatus. The mechanical power can be calculated by multiplying this tangential force by the velocity of the impeller blades,

$$P_u = \dot{m}(u_2 c_{2t} - u_1 c_{1t}), \tag{2.56}$$

where u_1 is the impeller velocity at the inner edge of the blades (referred to as the inner circumference velocity), and u_2 is the circumference velocity at the outer edge, and the use of the subscript u for the term P_u make the reference to impeller power explicit. The mechanical power required to move the air can also be calculated from the expression for total enthalpy, i.e.

$$P_a = \dot{m}\Delta h_{tot} = \dot{m}\left(\frac{1}{\rho}\Delta p_{tot}\right). \tag{2.57}$$

As these two different expressions for power are equivalent, the following relationship is

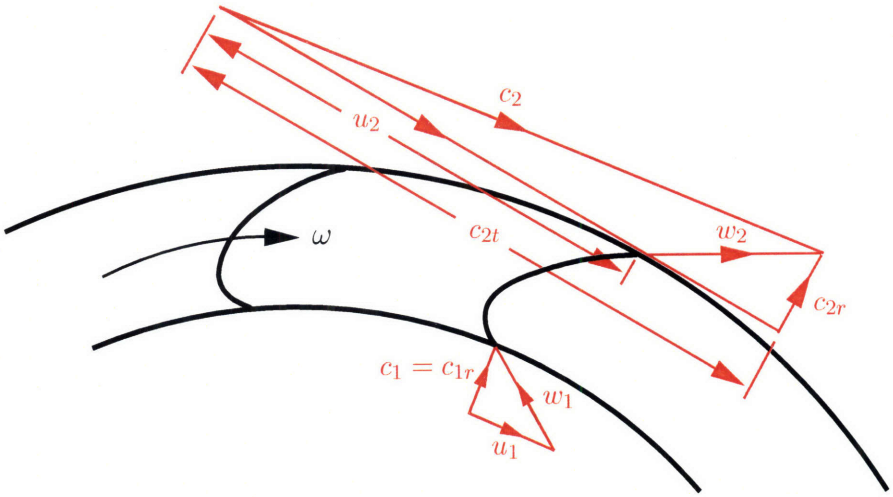


Figure 2-19: Vector diagram for forward-curved fans.

also valid for isentropic fan behavior:

$$u_2 c_{2t} - u_1 c_{1t} = \frac{1}{\rho} \Delta p_{tot}. \tag{2.58}$$

Unfortunately, the proliferation of different components of the inlet and outlet air velocities, as well as the use of both the circumference velocity and air velocity, tend to make it difficult to keep all of these various terms straight. A vector diagram illustrating the circumference and air velocities is provided in Figure 2-19 to better visualize the relationships between these different components. In addition to the air velocities c and the circumference velocities u, a third term is evident: the relative velocity w_i. This relative velocity is simply the vector difference between the air velocity and the circumference velocity, so that $\bar{w} = (\bar{c} - \bar{u})$. This relative velocity can be used to simplify the terms involving the product of the tangential velocity and the circumference velocity in (2.56). For example, the vector expression for $\bar{w_2}$ from Figure 2-19 may be written

$$\begin{aligned} w_2^2 &= \bar{w}_2 \cdot \bar{w}_2 = (\bar{c}_2 - \bar{u}_2) \cdot (\bar{c}_2 - \bar{u}_2) \\ &= c_2^2 + u_2^2 - 2\bar{u}_2 \cdot \bar{c}_2 \\ &= c_2^2 + u_2^2 - 2u_2 c_{2t}. \end{aligned} \tag{2.59}$$

This has the effect of dramatically simplifying the nature of expression of the left hand side

of(2.58),

$$u_2 c_{2t} - u_1 c_{1t} = \frac{1}{2}(c_2^2 - c_1^2) + \frac{1}{2}(u_2^2 - u_1^2) - \frac{1}{2}(w_2^2 - w_1^2) \tag{2.60}$$

which, after substituting into (2.58), an expression for static pressure can be found to be

$$\Delta p = p_2 - p_1 = \frac{1}{2}\rho(u_2^2 - u_1^2) + \frac{1}{2}\rho(w_1^2 - w_2^2). \tag{2.61}$$

It is also notable that, for most centrifugal fans as installed in air handler, the airflow into the impeller is mostly radial, so the approximation can be made that

$$c_1 = c_{1r} = c_{2r}, \tag{2.62}$$

which is consistent with the original expression for mass conservation in the system (2.50).

One can see from the expression (2.61) that the pressure difference across the fan can be formulated in terms of a contribution from the change in the circumference velocities, as well as a contribution from change in the relative velocities. Since both of these terms may be directly related to the circumference velocity, they may be scaled according to changes in speed by (2.52), since similar velocity triangles for the inlet and the outlet are found at different speeds ([46, p. 343] and [56, p. 762]). This similarity is the basis for the ubiquitous fan laws that are used in HVAC computations [10, ch. 18]. This results in the equivalence of the different components of the air and circumference velocities at two different speeds n^I and n^{II},

$$\frac{n^{II}}{n^I} = \frac{u_2^{II}}{u_2^I} = \frac{u_1^{II}}{u_1^I} = \frac{c_{2t}^{II}}{c_{2t}^{II}} = \frac{c_{1t}^{II}}{c_{1t}^I} = \frac{c_{2r}^{II}}{c_{2r}^I} = k. \tag{2.63}$$

The first fan law states that the mass and volumetric flows scale directly with speed. This can be seen by simply applying the velocity-speed relation (2.52) to the mass conservation expression(2.50),

$$\frac{\dot{m}^{II}}{\dot{m}^I} = \frac{\rho^{II}}{\rho^I} \cdot k, \tag{2.64}$$

where under normal operating conditions, $\rho^I = \rho^{II}$. The second fan law expresses the

difference in static pressure for two different speeds,

$$\begin{aligned}\frac{p_{tot}^{II}}{p_{tot}^{I}} &= \frac{\rho^{II}}{\rho^{I}} \frac{u_2^{II} c_{2t}^{II} - u_1^{II} c_{1t}^{II}}{u_2^{I} c_{2t}^{I} - u_1^{I} c_{1t}^{I}} \\ &= \frac{\rho^{II}}{\rho^{I}} \frac{(ku_2^{I})(kc_{2t}^{I}) - (ku_1^{I})(kc_{1t}^{I})}{u_2^{I} c_{2t}^{I} - u_1^{I} c_{1t}^{I}} \\ &= \frac{\rho^{II}}{\rho^{I}} \cdot k^2. \end{aligned} \tag{2.65}$$

The third law follows in an analogous manner, using (2.57),

$$\frac{P_u^{II}}{P_u^{I}} = \frac{\rho^{II}}{\rho^{I}} \cdot k^3. \tag{2.66}$$

This analysis has demonstrated the feasibility of using the fan laws to predict the operating characteristics of the fan for a range of behavior by using a reference set of measurements relating flow to either the static pressure across the fan or to the rotational power required to move the fan. It is important to acknowledge that this descriptive model is not suitable for the theoretical development of a fan curve that can be validated by experiment. While this analysis has developed a model of the behavior of the fan from a mass balance, a momentum balance, and an energy balance, it has not incorporated a host of higher-order behavior that undoubtedly affects the fan; for example, the thermodynamic energy balance of Equation (2.53) assumed isentropic behavior and set $s = 0$. Moreover, a variety of other flow characteristics were also not modelled, either isentropically or otherwise. Such effects include the pre-rotation of the flow before the inlet, which would result in a tangential component of the inlet velocity, and vortex formation in the fan passages, which would have the effect of reducing the outlet air velocity.

Bearing this in mind, then, the utility of this model is apparent when one considers the application of this model to characterize and predict the behavior of the fan near a given operating point. Models of fan behavior are not usually developed from first principles arguments, for the reasons mentioned previously, but are instead either determined empirically or from the application of computational fluid dynamics techniques. Such methods can effectively characterize fan behavior and take into account the variety of complex phenomena that can occur experimentally. Once that behavior has been characterized at

a particular operating point, however, the system model that has been developed in this chapter can be used to model the deviations in the fan behavior around that operating point. This model is valid to describe these small deviations around the operating point because the shape of the velocity triangles at different operating points will be similar to one another, as will the momentum and thermodynamic relationships that have been established.

The particular approach taken in this research is that of using fan curves from the manufacturer to characterize the behavior of the particular fan under study. These curves were obtained from the manufacturer of the blower wheel, and were made on a test apparatus which is able to precisely measure the variables of interest. During such a characterization test, the rotational speed is fixed (in this case, at 1000 rpm), and the volumetric flow is gradually increased from 0 ft^3/min (cfm) to some maximum quantity that has been also been set. As the volumetric flow increases, the static pressure across the fan and the shaft power supplied to the fan are measured. Once the full range of airflow has been surveyed, the resulting data relating the volumetric flow to either the static pressure or the shaft power is fit to a polynomial; these coefficients can then be used find the static pressure or the shaft power at the new operating points via the fan laws.

The best demonstration of how fan curves are used can be quickly seen through the use of an example. Figure 2-20 illustrates these fan curves in which the power in brake horsepower is plotted against the airflow in cfm for three different speeds. Each of these curves was generated by scaling the reference fan curve, which was obtained at a speed of 1000 rpm, by using the fan law from Equation (2.66). As mentioned previously, this reference fan curve was obtained by empirically measuring the volumetric flow and mechanical power over a range of flow rates, and then fitting a sixth order polynomial to the fan curve in the region of interest, e.g.,

$$P = a_0 + a_1 Q + a_2 Q^2 + a_3 Q^3 + a_4 Q^4 + a_5 Q^5 + a_6 Q^6, \tag{2.67}$$

where P is given in bhp and Q is given in cfm. The coefficients for the fan under study were provided by Tim May from Lau, Inc. This fan curve is used in the airflow estimation method after the power and the speed for a given set of test conditions are computed; the

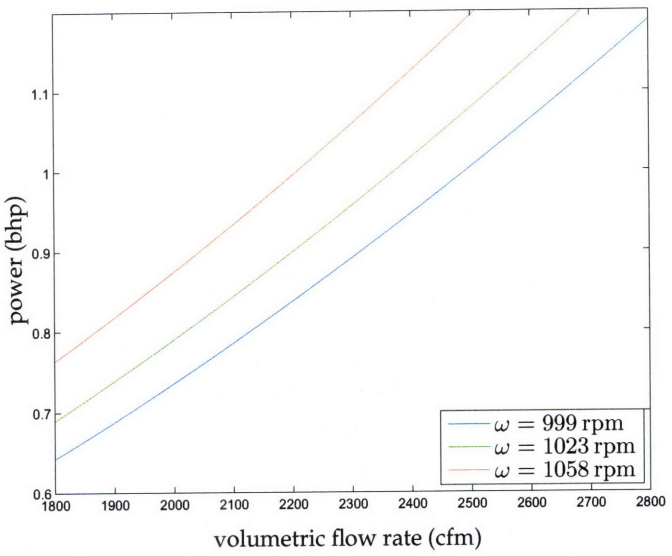


Figure 2-20: Fan curves for three different speeds.

reference fan curve is scaled by Equation (2.66), the estimated mechanical power is located on the y-axis of the appropriate plot, and then the corresponding airflow is identified. One can see that these fan curves are relatively smooth in the region of interest, so that the appropriate value of flow can be estimated by interpolating between nearby values for which the fan curve has been computed. It is also possible to generate a speed-flow-power surface on which each of these curves lie; as the fan's operating conditions change, the fan will move around to different points on this surface. Such a surface for the fan of interest is shown in Figure 2-21, and makes the relationship between all of the variables easier to visualize. It is important to note that this fan curve is not valid over all of the regions of operation. We will mostly operate in the region of operation for which these parameters are valid. The coefficients of the fan curve are only designed to characterize the efficient region of operation.

2.6 Experimental Apparatus

The architecture of the overall experimental apparatus used, including the air handler, data acquisition system, and length of duct, is illustrated in Figure 2-22. The air handling unit was purchased from ABCO Refrigeration Supply Corp. in Somerville, MA, and was

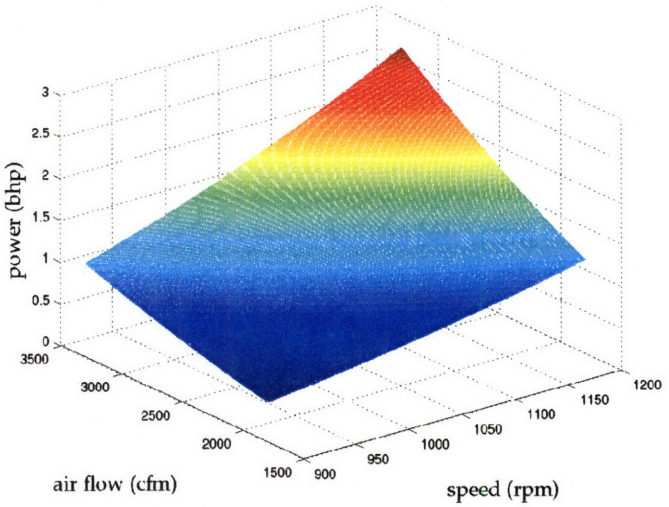


Figure 2-21: Speed-Flow-Power surface for fan in region of interest.

Figure 2-22: Picture of the overall air handler apparatus and attached duct system.

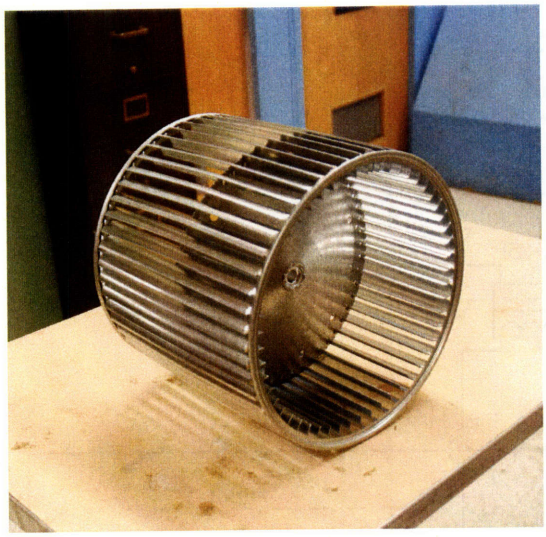


Figure 2-23: Blower wheel.

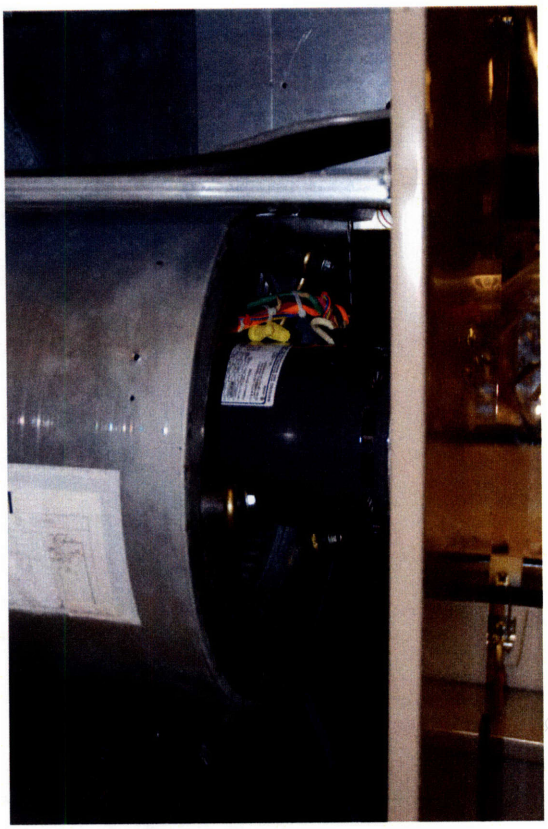


Figure 2-24: Mounted motor.

manufactured by International Comfort Products, LLC. Pictures of this unit were included previously in §2.1.1, in Figure 2-1a and Figure 2-1b. This unit is designed to be part of a split air-conditioning system, with an evaporator and filter holder located at one end, and was configured for the duration of the experiments in an "upflow" configuration, meaning that the air intake is located below the air handler, and the exhaust is located at the top of the air handler. It is equipped with a forward-curved double-duct centrifugal fan manufactured by Lau Industries, part no. DD 11-10AT, which is 10 inches in diameter and 11 inches wide, and has 54 blades. This fan is a direct drive unit, with the blower wheel attached directly to the motor shaft via a set screw. The blower wheel by itself and mounted in the air handler can be seen in Figures 2-23 and 2-24.

This air handler is sold with a single-phase capacitor-run induction motor driving the fan, but owing to the experimental nature of this work and the fact that effects such as saturation and hysteresis of the magnetic material tend to have a large effect on commercially

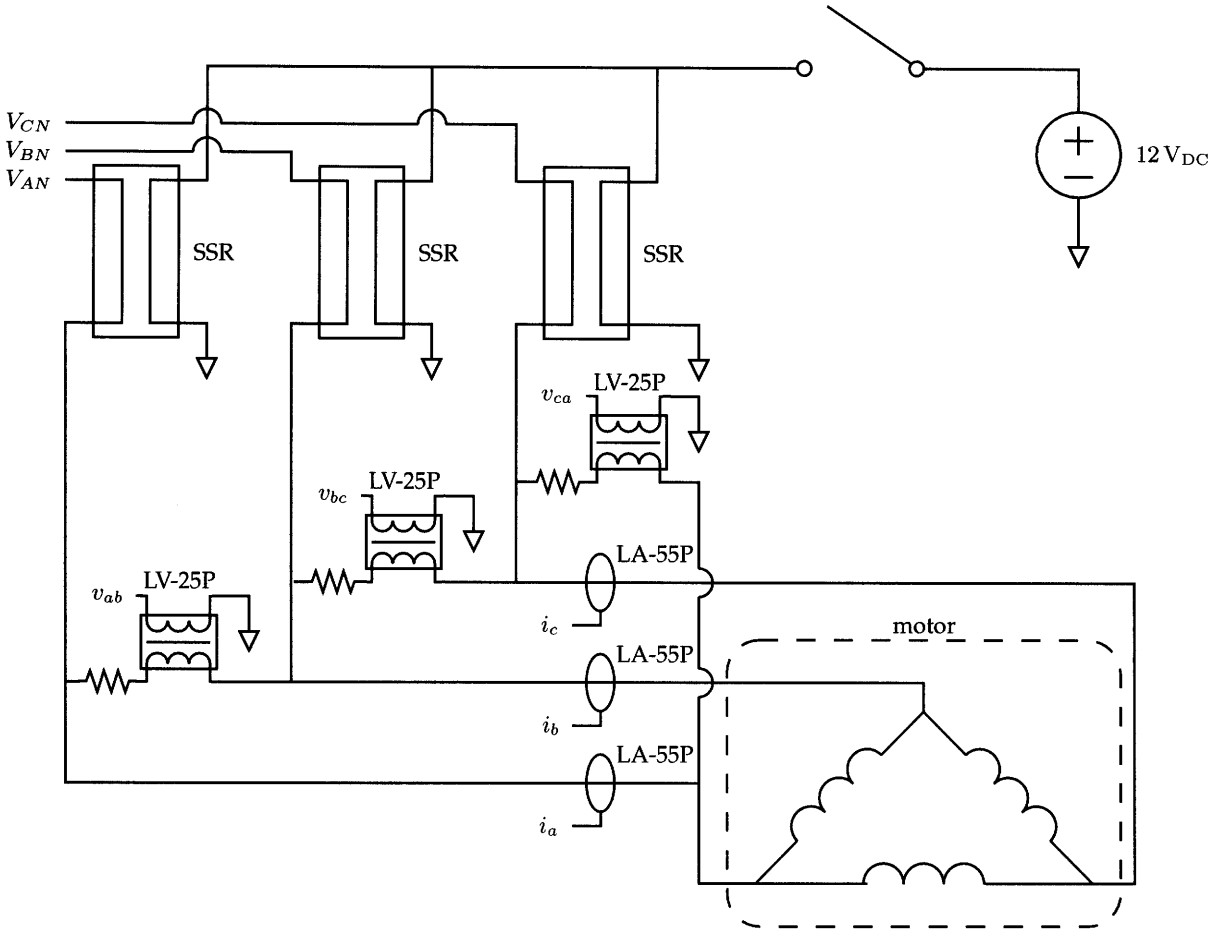


Figure 2-25: Schematic diagram of the control and data acquisition system.

manufactured single-phase motors, a three-phase induction machine was installed in its place so that the effectiveness of the airflow estimation method could be tested without being obscured by the variety of issues inherent in estimating the parameters of a single-phase machine. This motor was manufactured by Emerson Corp., part no. 1326, and is a delta-connected 3-pole pair induction machine with 48 rotor slots.

A control and data acquisition system was also installed to acquire the variety of signals necessary to implement the fault diagnostic method. This instrumentation necessary to acquire these signals is shown in Figure 2-25. One can see from this figure that each phase of the electric utility is connected to the motor through a solid-state relay (SSR), and the voltage across each pair of stator windings, as well as the line currents, are input to the data acquisition system. These solid-state relays were used to precisely control when

each phase was connected to the electric utility; in traditional mechanical contactors, there can be a substantial delay (perhaps as much as a line cycle) between the time that the first phase of the motor is connected to the utility and the time that the last phase is connected. These arbitrary turn-on relays (manufactured by Crydom, part no. D2450-10) were used to ensure that all three phases of the electric utility were connected to the motor simultaneously to avoid these unusual electrical transients. The fault diagnostic method used would also work if the mechanical contactor were used by monitoring the voltages and currents directly at the motor terminals, but solid-state relays were used to keep the parameter estimation method as simple as possible.

The line-to-line phase voltages V_{AB}, V_{BC}, and V_{CA} were monitored via three voltage transducers (LEM LV-25P), which are effectively high-accuracy transformers that reduce the phase voltages on the stator windings from 208 V_{pp} to approximately 3 V_{pp}. Similarly, three Hall effect current transducers (LEM LA-55P) were used to monitor and properly condition the phase currents I_A, I_B, and I_C so that they could be input to the data acquisition system. The resulting three voltage signals (v_{ab}, v_{bc}, v_{ca}), as well as the three current signals (i_a, i_b, and i_c) were input to an Advantech PCI-1710 card, which interfaced directly with a computer running Debian Linux. The seventh signal input to the data acquisition card is the tachometer signal v_{tach}, which is the $0-5$ V square-wave output of a US Digital E6S 64 count optical encoder, which was mounted directly to the shaft of the motor. The resulting signal from the encoder had the unfortunate effect of causing noise spikes on the nearby channels, due to its high frequency content; this had no appreciable effect on the torque-speed estimation method, but it dramatically affected the spectrum of the motor currents. Whenever it was necessary to acquire current data for the purposes of analyzing the harmonic content, it was therefore necessary to disable the optical speed encoder.

A length of duct was purchased and installed in the laboratory so that accurate airflow measurements could be taken, and so that the behavior of the air handler would more closely approximate behavior in field-installed equipment. Two views of this ducting system can be seen in Figures 2-26 and 2-27. The bulk of the ducting system is 30 feet of 16 inch round sheet metal duct, obtained from McMaster-Carr in 3 foot long sections that are taped together with metallized duct tape and suspended from pipes in the ceiling by steel strapping. The transition between the exhaust of the air handler and the 16 inch round

Figure 2-26: View of ducting apparatus.

Figure 2-27: Additional view of ducting apparatus.

duct was accomplished by installing a custom fabricated 90 degree elbow and a 78 inch long converging section that also transitioned from the square cross-section of the elbow to the round cross-section of the remainder of the duct. This particular duct fitting was fabricated by the Medford-Wellington Service Company at a cost of approximately $250.

The first reason that this length of duct was constructed, i.e. increasing the accuracy of the airflow measurements, stems from the fact that the forward curved fan blades cause the air velocities in the duct to be much higher on the side of the duct that is in the direction of rotation. This fact, in concert with the particular configuration of the ducting system due to space constraints in the laboratory, caused the air velocities at the top of the duct to be much greater than those at the bottom of the duct. A set of adjustable double thickness turning vanes was therefore installed in the elbow so that the velocity profile could be adjusted. These turning vanes are visible through the access door in Figure 2-28. This access door was also used to test the leaky duct; when the leaky duct test was conducted, the fan was run with the access door open. The airflow was measured via a set of 4 traverses, each of which included 6 sample points, according to the guidelines specified by ASHRAE [9, p. 14.17]. These traverses were made at a location 77 inches from the end of

Figure 2-28: Turning vanes.

the duct, or about 5 duct diameters from the end of the duct, to minimize the effects of the diverging flow at the end of the duct. Each of the 4 holes that had to be drilled in the duct was covered with 2 layers of electrical tape and then slit open, in order that the 3 unused holes would mostly be covered when a given traverse was being performed. One of these pieces of duct tape can be seen in Figure 2-26 at the right end of the duct.

The results will be presented in the order in which they were discussed in this research; the results of estimating the speed via the slot harmonics will presented first. In the second section, the results of estimating the motor parameters and the torque-speed curve will be discussed. After the separate presentations of the speed and torque estimation methods, the concluding results section will examine airflow predictions generated from these methods and compare these predictions to the measured airflow in the duct. These results will then be considered and the effectiveness of the overall system will be discussed.

2.7 Speed Estimation Results

The estimation module, as discussed in §2.3, was the first of the modules to be experimentally evaluated. This data was collected with the fan running in four states of interest: unblocked flow, 30% blocked flow, 50% blocked flow, and leaky flow. The speed estimates were collected and verified by collecting sequential data sets; due to the distortion caused by the high-frequency content of the tachometer signals, tachometer data was collected for 30 seconds while the motor was running, the data collection was stopped and the tachometer channel was disconnected quickly, and then measurements of the motor currents were collected for 30 seconds while the motor continued running. Speed estimates were generated from the tachometer data by using the simple strategy of counting 128 edges in the tachometer waveform, which correspond to the 64 notches in the tachometer dial, and then using the fact that the time in which the 128 edges pass by the detector corresponds to one revolution of the wheel. This algorithm was implemented in C, the source of which is listed in Appendix A. These speed estimates obtained from the tachometer data have a time-varying component, since the number of samples between the edges is not constant. If there happen to be seven samples before an edge of the tachometer waveform, rather than six, the resulting speed estimate will accordingly change by a few rpm. Accordingly, the speed estimates from the tachometer have a standard deviation around 2 rpm.

These estimates from the tachometer were used to validate the estimates of the speed obtained from the observations of the phase A current. A window of 2^{16} datapoints (about 4.5 sec of data at the sample rate of 14.29 kHz) were selected from the datafile, and then a Hanning window was applied to this segment of data to minimize spectral leakage. This data was then filtered twice, first by a third-order elliptical notch filter to remove the 60 Hz component of the signal, and then by a sixth-order elliptical bandpass filter that selected the frequencies between 600 Hz and 1600 Hz where the expected slot harmonics were expected to be. While an algorithm could be designed which locates the peaks in the peaks in the frequency spectrum by using methods such as those suggested in [71], this speed estimation step was carried out by hand in this research to expedite the collection of the speed estimates. The resulting spectrum of the phase A motor current collected when the fan was running in an unblocked state can be seen in Figure 2-29.

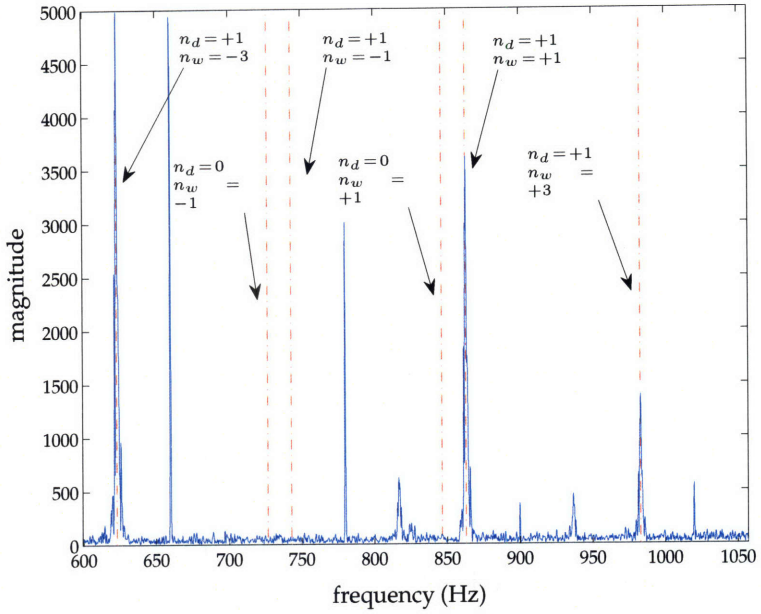


Figure 2-29: Spectrum of the motor current for the unblocked fan illustrating the locations of the theoretical and experimentally observed slot harmonics.

The results of processing the tachometer data that corresponds to this current data yield an average speed over the observed window of 988 rpm with a standard deviation of 2.4 rpm, while the speed predicted from the location of the slot harmonics seen in this figure is 984 rpm. This agreement is reasonably good and indicates the usefulness of the speed prediction algorithm. The discrepancy in the two estimates of the speed can also partially be attributed to the fact that the estimates were not obtained concurrently due to the limitations of the experimental apparatus, so that the fan speed might have changed by a small amount between collecting the two datasets.

The analysis of this current spectrum must also consider the fact that the fan speed in an experimental apparatus typically has short-time variations around a slowly time-varying average speed. The peaks in the Fourier transform of the current spectrum will therefore be "smeared" or wider than they would otherwise be if the motor was only run at a single speed. One can see in Figure 2-29 that the peaks corresponding to slot harmonics are actually about 5 Hz wide, which corresponds to the small variations present in the fan speed over the measurement window. This variation could be minimized by analyzing a smaller window of data, but this would also reduce the frequency resolution of the spectrum. In

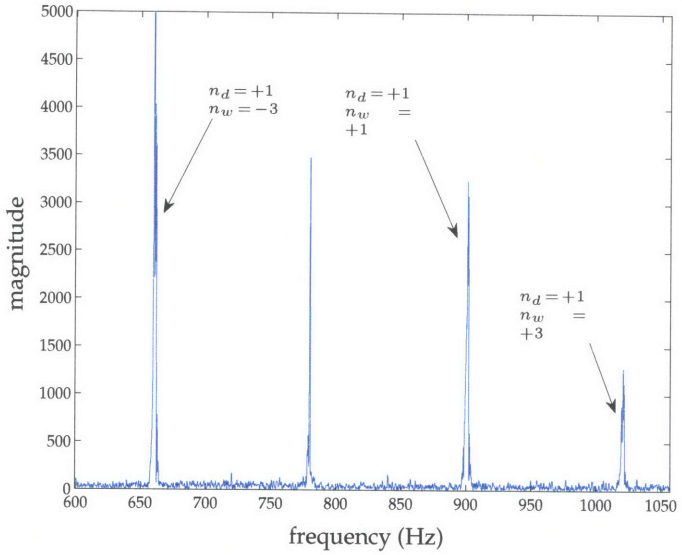


Figure 2-30: Spectrum of the motor current for the 30% blocked fan illustrating the locations of the experimentally observed slot harmonics.

developing an implementation of this system for field use, these two considerations would have to be traded off against one another.

Recalling Equation (2.23), which identified the expected location of the series of slot harmonics, it is interesting to note that while some of the slot harmonics are clearly present, others are much harder to locate. In particular, the harmonics that correspond to the (k, n_w, n_d) triplets (1, 1, -3), (1, 1, 1), and (1, 1, 3), are clearly visible at the frequencies 627 Hz, 866 Hz, and 987 Hz. On the other hand, the harmonics such as those predicted by the triplets (1, 0, -1), (1, 1, -1), and (1, 0, 1), which would be located at 730 Hz, 747 Hz, and 850 Hz, are not at all discernible from the background noise present in the spectrum of the signal. One can also easily identify the presence of the main MMF harmonics, notably those at 660 Hz, 780 Hz, 900 Hz, and 1020 Hz. Nevertheless, this method is evidently able to identify the motor speed reasonably well.

The spectrum from the second dataset, in which the fan was blocked by 30%, is illustrated in Figure 2-30. The predictions from this experiment are similar in accuracy to those of the unblocked experiment, as the speed predicted from the tachometer output is 1034 rpm with a standard deviation of 1.8 rpm, while the speed predicted from the motor slot harmonics is 1029 rpm. This plot only illustrates the same harmonics that were experimentally identified in the unblocked case, with the harmonics corresponding to the triplets (1,

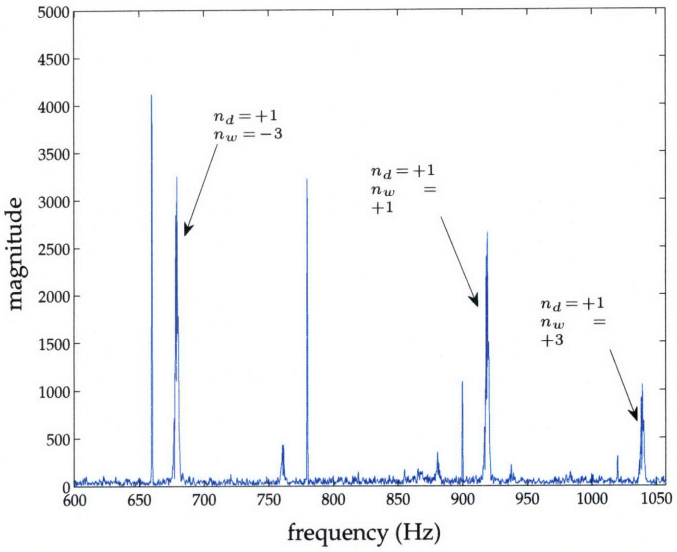


Figure 2-31: Spectrum of the motor current for the 50% blocked fan illustrating the locations of the experimentally observed slot harmonics.

1, -3), (1, 1, 1), and (1, 1, 3) being located at 664 Hz, 904 Hz, and 1024 Hz.

One of the challenges in identifying motor speed by looking for the slot harmonics with the biggest magnitude can be seen in considering the spectrum illustrated in this figure. While the harmonics identified in the plot can be seen clearly, it is difficult to discern whether they are caused by the slot harmonics of interest, or whether they are instead caused by the principal MMF harmonics, which are found at 660 Hz, 900 Hz, and 1024 Hz. One can see clear differences in the harmonics at these frequencies by comparing the spectra in Figures 2-29 and 2-30; the spikes in Figure 2-30 are both greater in amplitude and width than those that are found in Figure 2-29. The increased width is presumably caused by the variation in the fan speed over the window of analysis, and the increased amplitudes are presumably caused by the fact that both the MMF harmonics and the slot harmonics are occurring at the same frequency, and therefore add. These characteristics could be used to construct an automatic speed detector that can identify the slot harmonics when they are found at the principal MMF frequencies. A variety of other detection methods could also be used to identify other smaller slot harmonics that do not line up with the MMF harmonics, such as are suggested in [70,71].

The spectrum of the motor current that results from blocking the fan inlet by 50% is illustrated in Figure 2-31. This plot shows that the speed prediction method also works well

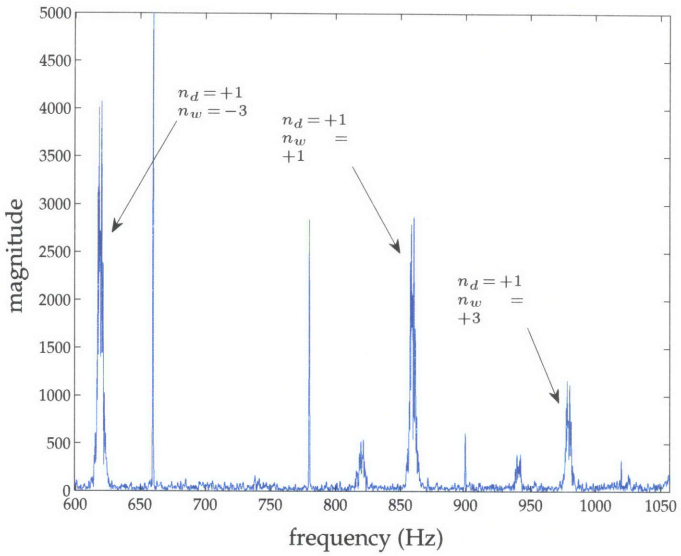


Figure 2-32: Spectrum of the motor current for the leaky fan illustrating the locations of the experimentally observed slot harmonics.

when the fan is blocked by as much as 50%. In this case, the average speed estimate from tachometer data is 1057 rpm, with a standard deviation of 1.5 rpm, while the predicted speed from the slot harmonics is 1052 rpm. As was the case when the fan was running unblocked, the slot harmonics are clearly distinguishable from the MMF harmonics, due to both their location and the fact that they are much wider at their base than the MMF harmonics.

The final tests of interest involved the detection of leaks in the air handling system by running the fan when the access door was open. The results of conducting this test are shown in Figure 2-32. Not surprisingly, the correlation between the speed measurements made by the tachometer and those predicted via the slot harmonics are similar to those observed in the previous experiments. The speed measured by the tachometer is 985 rpm, with a standard deviation of 2.5 rpm, while the prediction from the slot harmonics is 979 rpm. This suggests that this airflow identification method will be able to identify scenarios in which the duct is either blocked or leaking.

In the process of conducting these experiments, it also became apparent that the speed varies over a wide range of timescales, rather than just the timescale of seconds due to turbulence in the air handling unit. Figure 2-33 illustrates this variation; this data was collected when the fan was in an unblocked state over the course of five days in February

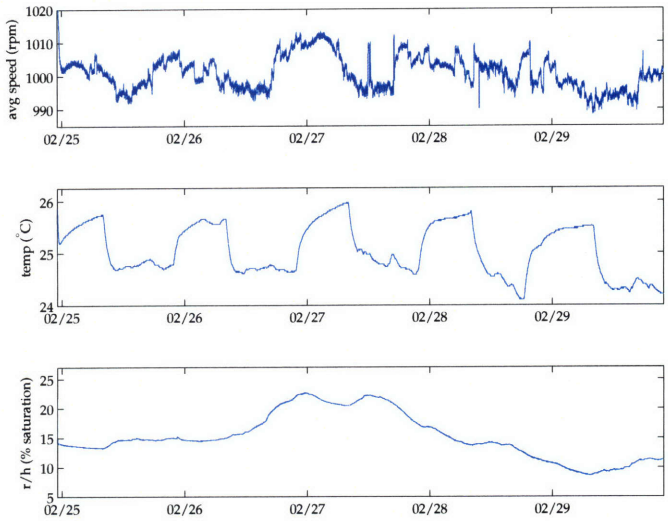


Figure 2-33: Plot of unblocked fan speed as fan operated continuously over approximately 5 days.

2008. The changes in the fan speed that are visible in this plot are not sudden, but they do cover a range of about 15 rpm on top of the average speed of 1000 rpm. These speed variations could be caused by a number of phenomena; as one possibility, the temperature and relative humidity in the laboratory space where the air handler is located are also plotted in Figure 2-33 so that they can be compared to the fan speed and correlations can be spotted. These variations may also be correlated with small changes in the amplitude of the utility voltage. This figure suggests that there is a correlation between the fan speed and the air temperature, as the fan speed is somewhat higher when the air temperature is higher, but this does not appear to be a 1:1 relationship. It is likely that the changes in the fan speed can also be correlated to changes in the air density, but this will also depend on the barometric pressure. Rather than develop a correlation that would require the measurement of temperature, humidity, and barometric pressure, consequently requiring a number of undesired mechanical sensors, it is important to consider how the effect this has on the airflow prediction system can be reduced or eliminated.

These speed variations have two effects on this method of predicting airflow, the first being the fact that it is more difficult to experimentally validate the performance of the method. The method by which the predictions of airflow are validated is somewhat laborious, as it takes approximately 45 minutes for the fan speed to reach steady-state after the

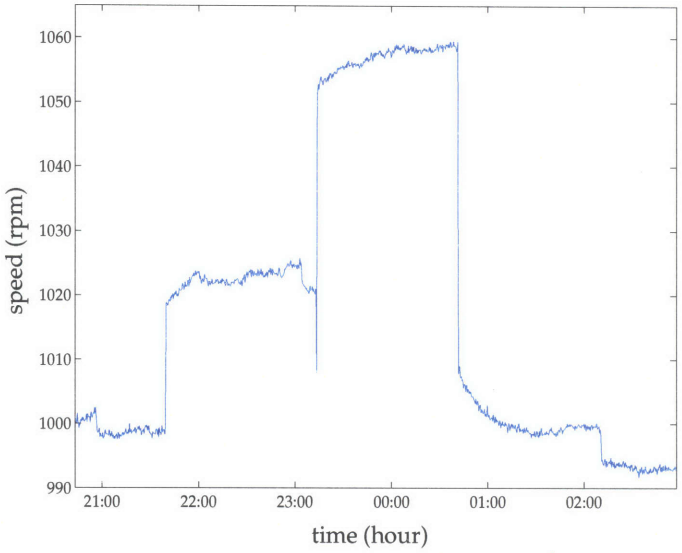


Figure 2-34: Speed over the course of the airflow measurement period.

amount of blockage is changed, and another 45 minutes to sample the air velocity of the duct at a sufficient number of points to estimate the average duct velocity. As four different states of flow are being tested in this thesis, a full set of validation experiments takes about six hours. If the unblocked fan speed changes over the course of these experiments due to environmental phenomena, this will add bias and inaccuracy to the measurements. Since the speed of the unblocked fan was observed to be relatively constant between 9 pm and 3 am, the airflow was validated during this period of time. The speed during these validation tests can be seen in Figure 2-34. During this interval, the experimental sequence is that the fan was started in the unblocked state, followed by 30% blockage, 50% blockage, no blockage, and then a leakage test. Each of these states can be seen in Figure 2-34 by a corresponding change in the speed. This plot suggests that the speed is relatively constant over the interval, as the speed of the first unblocked test is nearly the same as the speed of the second unblocked test.

Once the airflow prediction method is validated, the above concerns do not affect its functionality, since the process of identifying the speed can be done nearly instantaneously on the computer, and, with the set of motor parameters, the calculation of the airflow can be accomplished by running the prediction algorithm. Thus, once the method is validated and the baseline airflow ('unblocked' airflow as referred to in these tests) is established,

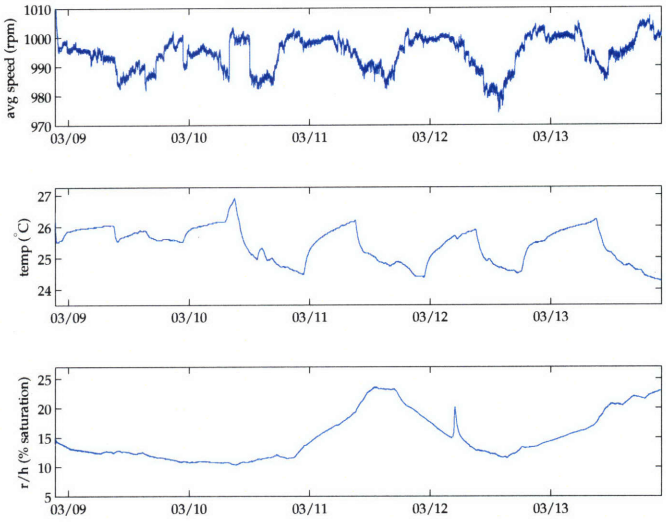


Figure 2-35: Plot of leaky fan speed as fan operated continuously over approximately 5 days.

the changes in the flow through the duct can be tracked as the speed varies.

While these variations in the airflow will not compromise the performance of airflow prediction method, they do make it more difficult to measure the baseline airflow through the duct system. For example, the design of a given air supply system will generally include a specification of volumetric flow through the duct, and one would expect that flow in the installed system would be close to this specification. Unless these airflow specifications take the factors affecting the fan speed into account, however, the speed variations will cause the airflow measurements to deviate from the specifications. This will make it difficult to determine whether or not the flow through the duct is out of spec, or whether it is just deviating due to normal operational fluctuations. Naturally, large changes that are beyond the range of these deviations can be detected, but these smaller changes are also important and should be detected as well.

Figure 2-35 suggests one method for solving the problem of identifying the average fan speed in its illustration of the fan speed variation for a leaky duct, in comparison with the variation in fan speed for the unblocked duct shown in Figure 2-33. While the highest values of the fan speed for the leaky fan are higher than the lowest values of the fan speed for the unblocked fan, one might note that the average value of the fan speed for the leaky fan appears to be lower than the average value of the fan speed for the unblocked fan. These

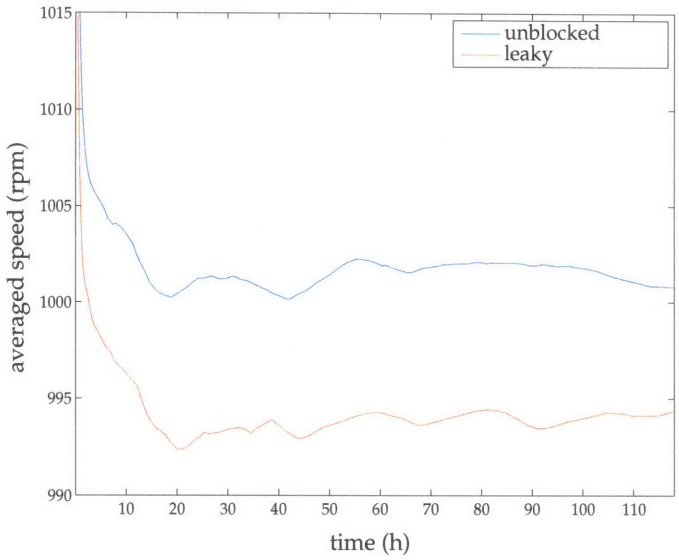


Figure 2-36: Plot of mean of the fan speed for each of the conditions given in the legend over windows of increasing size.

average values of the fan speed for both the unblocked and the leaky duct experiments are computed over successively larger windows are plotted in Figure 2-36. Each variable illustrated in this plot was calculated by the following:

$$s_n = \frac{s_1 + s_2 + \cdots + s_n}{n} \tag{2.68}$$

where s is the speed being averaged, and n in the index of the datapoint in the output. One can see that, as the size of the averaging interval gets larger, the fan speed becomes relatively stable, and that it is relatively easy to distinguish between the unblocked duct and the leaky duct on the basis of these averaged measurements. One possible method of commissioning the system would therefore be to collect an average speed of the fan over two to three days, and then to look at the prediction of the airflow through the duct with this average speed; if the predicted airflow is higher than the expected airflow, then there are most likely leaks in the system.

2.8 Torque Estimation Results

Due to the somewhat complex nature of the torque estimation method, the discussion of the experimental results for this module is split into two sections. In the first section, the operation of the motor parameter estimation algorithm is analyzed, so that its robustness and effectiveness can be assessed independently of any variations in the operating conditions of the motor. After the basic performance of the parameter estimation algorithm has been established, the application of this method to the estimation of the fan motor parameters in the experimental unit under a number of different possible conditions will be tested.

2.8.1 Parameter Estimation

In order to test the performance of the parameter estimation method on data that most closely resembles the data of interest, the fan was started and run for approximately two hours so that it could reach thermal equilibrium under load. After two hours, a startup transient that represented the torque speed behavior for the fan in continuous operation was obtained by briefly turning the fan off, allowing it to spin down to a standstill, and then recording all three voltages and currents during the following startup transient.

As discussed in §2.4.3, the parameter estimation of the motor proceeded in two steps. In the first step, the parameters are pre-estimated by constraining the stator resistance to be a measured value, constraining the remaining motor parameters to be strictly positive, and finding the parameters of the motor which best fit the low-passed envelope of the motor current. The initial guess of the stator current was obtained by measuring the resistance of the A-B stator windings with a Fluke model 87 III multimeter when the motor was cold. While the resistance of the stator windings do change with temperature, as will be seen, this measurement of the stator resistance was used to evaluate the robustness of the parameter estimation method. Since the measured terminal resistance measured was 8.2 Ω, the wye-connected model of the induction machine implied that the equivalent resistance R_s of each stator winding is 4.1 Ω. Results using this stator resistance on the pre-estimation fit, as well as estimated scales for the other parameters, are illustrated in Figure 2-37. It is apparent from this plot that the pre-estimation routine worked fairly well, as the estimated

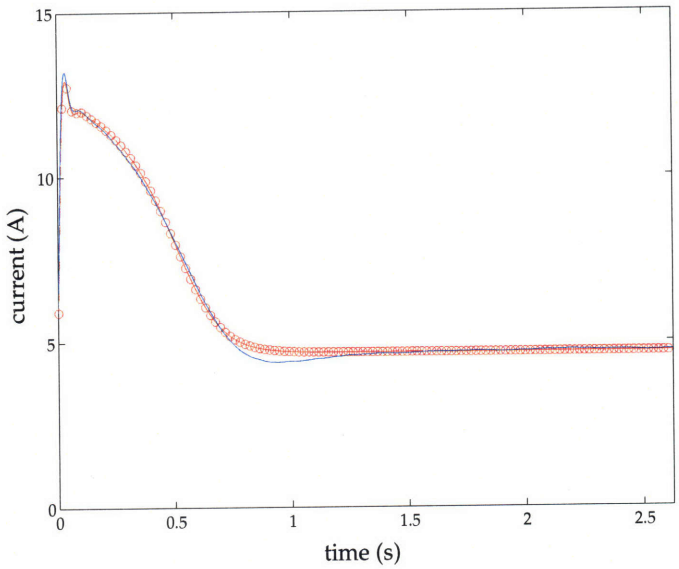


Figure 2-37: Plot of the preprocessed observed phase A motor current envelope along with the estimated current envelope. The observed current envelope is illustrated in blue, while the estimated envelope is shown in red.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Value	4.10	4.19	90.72	3.78	8.26	5.69e-04	27.57

Table 2.1: Pre-estimated motor parameters.

dynamics are captured resonably well in the estimated envelope. This can also be seen in the generally reasonable parameters that result from this pre-estimation step, as listed in Table 2.1.

Once the pre-estimates of the motor parameters were generated, they were used as initial guesses for the Levenberg-Marquardt minimization of the set of observations of data. This minimization was performed according to the method described in §2.4.3, with all of the parameters being minimized in the second step of the algorithm. As with the pre-estimation, the full minimization was tested and validated minimizing only against the observations of the terminal current i_a. The results of this minimization are illustrated in Figures 2-38- 2-41.

The plots in Figures 2-38 and 2-39 are zoomed into the first second of operation of the motor so that the quality of fit can be visually evaluated. The simulation of the induction motor with the final parameter estimates is within 10% of the observed current at all points,

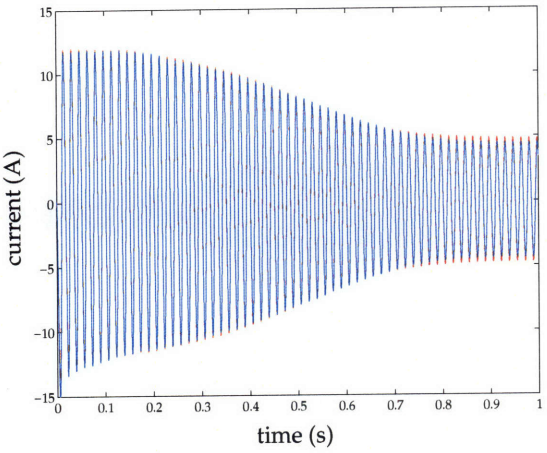


Figure 2-38: Observed and fitted motor currents for the hot motor; observed data are in blue, and fitted data are in red. This waveform is zoomed in so the agreement between the observations and the predictions can be seen.

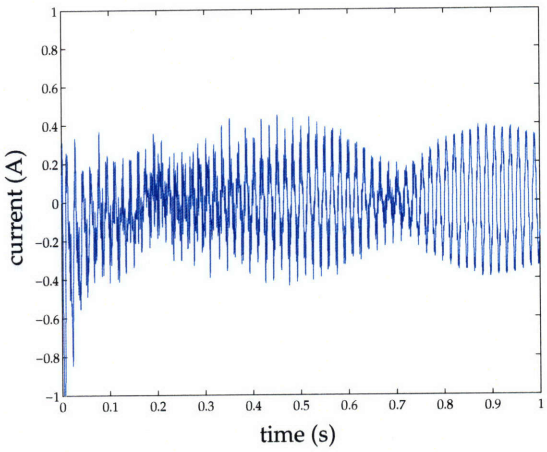


Figure 2-39: Residual from $i_{a,fit} - i_{a,obs}$.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Value	6.25	4.03	57.75	3.14	7.71	4.59e-04	31.00

Table 2.2: Final hot motor parameters minimizing solely against the current.

and within 5% for most. The size of the residual to the amplitude of the transient is larger in the last 0.2 seconds of the data, which may be caused by unmodeled behavior of the motor, but this amount of misfit remains below 10%. Moreover, the quality of fit improves over time, as one can see from the plots in Figures 2-40 and 2-41, which show the same data as in the previous two figures, but on a longer time scale.

These figures show that the quality of fit improves after the initial transient, suggesting that while the model is able to represent the behavior of the system in the transient state fairly well, its description of the system is even better when the system is in steady-state. The final values of the parameters, after this second stage of parameter estimation, are listed in Table 2.2.

The validity of these parameters was evaluated by measuring the impedance of one set of motor terminals before and after running the motor for two hoursand comparing the the measured impedances with the estimates of the impedances. Such a test provides an

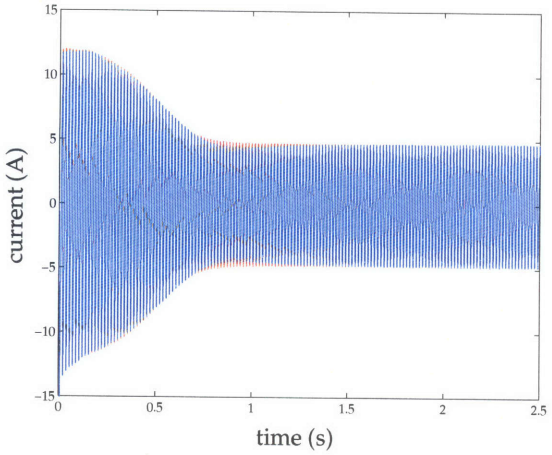


Figure 2-40: Observed and fitted motor currents for the hot motor; observed data are in blue, and fitted data are in red. This waveform is zoomed out so that the quality of fit for a larger segment of time can be seen.

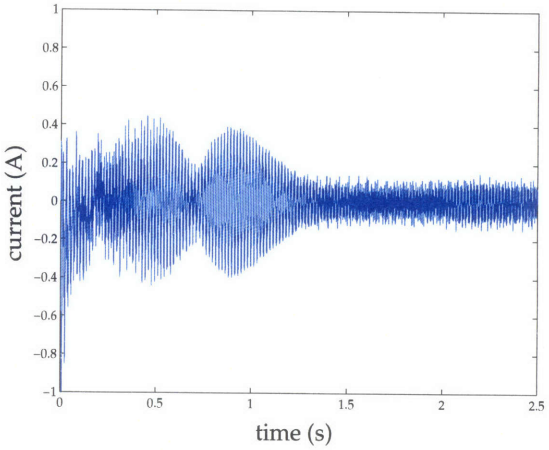


Figure 2-41: Residual from $y_{fit} - y_{obs}$.

state	f [Hz]	R_{meas} [Ω]	X_{meas} [Ω]
hot	5	11.37	5.00
hot	60	16.17	30.16
hot	100	17.72	49.22
cold	0	8.2	-
cold	60	12.58	26.012

Table 2.3: Measured motor parameters, as measured on 11/30/2007.

independent evaluation of the accuracy of the parameters estimated by the preceding routines. These parameters are reproduced in Table 2.3. It is important to note that these are the measured terminal variables, rather than the model parameters as stated in Table 2.2. These parameters could be converted to the electrical parameters, e.g., $R_{meas} = 2R_s$.

One important consideration regards the dependence of the parameter estimation method on the quality of the initial guess for the motor parameters. As the only parameter explicitly set before the pre-estimation step is R_s, a number of tests were run in which the value of R_s was changed over a relatively wide range, from $0.8\,\Omega$ to $20.1\,\Omega$, to ascertain the corresponding impact on the final motor parameters. The set of parameters for each of these initial guesses are collected in Table 2.4 for comparison.

The first fact to note is that all of the parameters that are obtained by setting $R_{s,init}$

$R_{s,init}$	$R_{s,final}$	R_r	X_m	X_{ls}	X_{lr}	K	β	$\|\|i_a - \hat{i}_a\|\|_2$	$\|\|\tau - \hat{\tau}\|\|_2$
0.8	6.24	3.95	54.2	3.72	6.97	4.59e-04	31.0	29.3	21.0
1.1	6.25	4.12	55.3	2.55	8.48	4.59e-04	31.0	29.3	1.09
2.1	6.25	4.03	54.7	3.19	7.65	4.59e-04	31.0	29.3	0.414
4.1	6.25	4.03	54.7	3.14	7.71	4.59e-04	31.0	29.3	0.0
6.1	6.25	4.03	54.7	3.15	7.70	4.59e-04	31.0	29.3	0.420
8.1	6.25	4.48	57.7	0.215	11.6	4.59e-04	31.0	29.3	0.0952
10.1	6.25	4.51	57.9	0.0168	11.9	4.59e-04	31.0	29.2	0.171
20.1	6.25	4.44	57.4	0.458	11.3	4.59e-04	31.0	29.3	8.39

Table 2.4: Various parameters resulting from starting the simulation with different initial guesses of R_s.

between 2.1 Ω and 6.1 Ω are nearly identical, which is encouraging in that it suggests that a relatively wide range of initial guesses for R_s, be they obtained when the motor is either hot or cold, will suffice to identify nearly the same set of parameters. Moreover, it is also apparent that some of the parameters are identical in all of the cases; for example, the mechanical parameters β and K are the same regardless of the initial guess. This is encouraging, as it indicates that the estimation method is able to characterize the mechanical phenomena that govern much of the behavior of the system. The same logic extends to the stator resistance R_s, and to R_r and X_m to a lesser extent. X_{ls} and X_{lr} appear to have a more substantial amount of variation upon first glance, but a closer look shows that the sum of the two parameters is nearly the same for all of the initial guesses. This makes some intuitive sense, as the impedance of the rotor leg of the model in Figure 2-7 is dominated by the rotor leakage reactance X_{lr} and this term in parallel with the mutual reactance X_m will be approximately equal to the X_{lr} as well.

In considering these parameters, however, it is important to remember that they are not necessarily directly indicative of the state of the motor; these parameters suggest that this model is overdetermined with respect to the observable data, since a number of different sets of parameters effectively yield the same observed currents and torque-speed curves (within measurement noise and numerical noise from the process of estimation). While these parameters were all obtained from the same set of initial guesses, the numerical issues surrounding this problem limit their uniqueness to a certain extent. Some of the parameters, such as R_s, β, and K, appear to be well-determined, while this does not appear

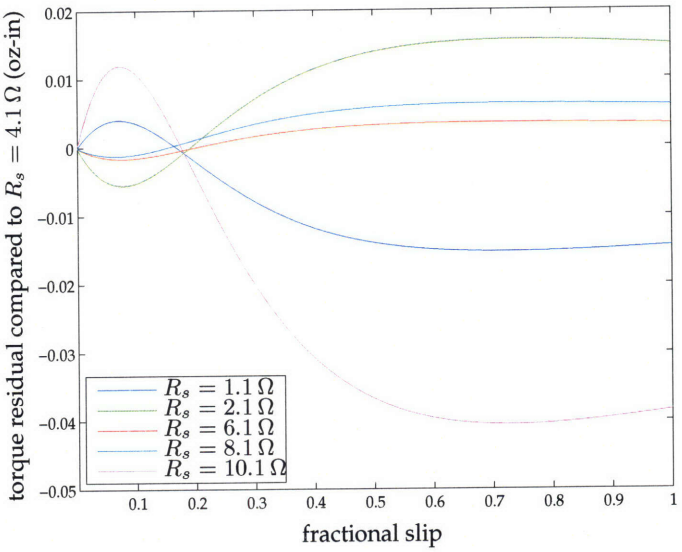


Figure 2-42: Plot of the estimated torque-speed curves for the motor when the parameters of the motor are estimated with different initial guesses for the stator resistance R_s. The value of the initial guess for R_s correponding to each curve is given in the legend.

to be the case for some of the other parameters.

As is also apparent from Table 2.4, the residual $\hat{i}_a - i$ was identical for all of the different values of R_s, but the torque speed curves were relatively different. Visualizing the deviation of these torque-speed curves is instructive to lend additional context to the residuals $||\tau - \hat{\tau}||_2$ tabulated in the previous table, so all of them were compared to the torque-speed curve obtained from an initial guess of $R_s = 4.1\,\Omega$ and plotted in Figure 2-42. It is clear that that these deviations, while growing larger for the values of R_s further away from $R_s = 4.1\,\Omega$, are still relatively small with respect to the system's torque-speed curve, and any of these curves will describe the torque-speed curve with sufficient accuracy for most experimental purposes. Nevertheless, the set of initial guesses close to the measured initial guess for R_s all yield nearly identical results; this fact is encouraging in that it indicates that the method has some robustness to comparatively poor initial guesses. These estimated torque-speed curves will be compared with torque-speed curves measured on the dynamometer in the following section, providing another means for evaluating the accuracy with which the estimated parameters describe the behavior of the induction motor.

2.8.2 Torque-Speed Characteristics

After validating the method for identifying the parameters of the system, it was necessary to test the method on a range of data that could be observed in the experimental air handler. Moreover, it is useful to compare the parameter estimation method's performance when when minimizing solely against the phase A motor current to the method's performance when minimizing against both the current and a set of observed torque-speed data points, so that the variation in the torque-speed curves in the absence of dynamometer data can be better understood.

This section will therefore explore the range of parameters obtained from the fan system while varying one or both of two different variables: the temperature of the motor, and the type of minimization used. The temperature of the motor is clearly an important variable, as this changes dramatically between the time that the motor is first turned on and times observed after the motor has been running for a long period. Results will therefore be presented that explore the correlations between the variation in the motor temperature with the variation in the performance of the parameter estimation method as well as motor performance.

An additional area of study regards the *in situ* motor performance in the air handler during a fault. In particular, it is important to understand the changes in the behavior of the motor when the air handler inlet is in a blocked state, rather than in its normal unblocked state. Such changes may result from the effective load on the fan decreasing as a result of an increasing amount of blockage, or from increased winding temperature due to a decrease in the amount of air flowing over the motor when the fan is blocked. After characterizing the performance of the estimation method, the sensitivity of the parameters to the air handler faults will be studied.

The mechanical losses of the motor, such as bearing friction and windage, are another important concern, especially as they cannot be predicted by the steady-state model of the induction motor as given in Figure 2-7 that is used to generate the estimated torque-speed curve. The parameter estimation method therefore incorporates observations of both the motor current and observed torque-speed datapoints to account for these losses. These experimental observations of the torque-speed characteristic for the motor will capture the

mechanical losses of the motor that are not described by the electrical model. This process will effectively introduce some misfit into the model parameters for the motor current to fit the observed torque-speed parameters, but both sets of observations will ensure that the method's performance will capture the observed physical phenomena as well as possible within the limitations of the model for the system.

The torque-speed datapoints were measured by mounting a motor of exactly the same make and model (Emerson Motors, part no. 1326) on a test stand with a Magtrol dial weight dynamometer, and collecting a set of torque-speed datapoints that resembled the analogous tests on the fan motor as closely as possible. Torque-speed datapoints for the cold motor were collected by starting the motor while running the dynamometer at a number of different torques, with a substantial (approximately 20 minutes) period of time between each datapoint, so that the resistive heating of the motor windings might be minimized. Torque-speed datapoints for the hot motor were collected by wrapping the test motor with a conformal rubber resistive heating blanket and heating the motor up to the temperature of the fan motor observed after a long period of running (about 170° F). The motor was subjected to this condition until it reached thermal equilibrium, and then a set of torque-speed datapoints were obtained by setting the torque to a number of different levels on the dynamometer and observing the resulting speed. Clearly, these methods for simulating the temperature of the system are not identical to the behavior of the real system, but they are useful in evaluating the efficacy of the two approaches to motor parameter identification for the purposes of airflow measurement, and will provide important information for understanding what data are necessary to accurately measure the airflow.

The following sections, in which the temperature of the motor and the minimization strategy are varied, are all presented in a parallel manner. For each condition of interest, the observed and predicted phase A motor currents are illustrated, along with the residual $i_{a,pred} - i_{a,obs}$ so that the extent and time-dependent characteristics of the misfit can be seen. The parameters for each of these cases are then presented, for the sake of comparing to the remaining tests, and then the predicted torque-speed curve is plotted with the observed torque-speed curve for the purposes of comparison. Finally, the variation of the parameters for a given set of conditions is explored by collecting a startup transient for the same condition on different days, and then comparing the torque-speed curves obtained

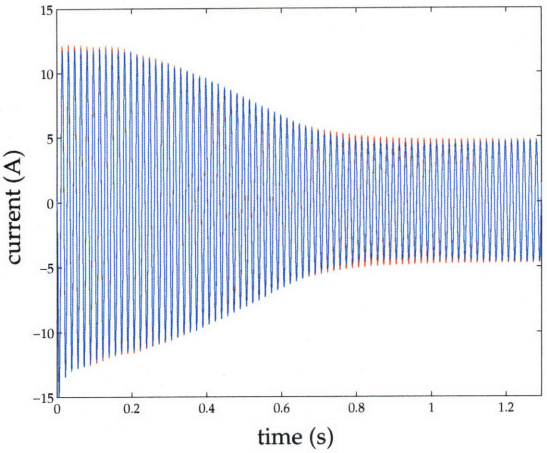


Figure 2-43: Observed and fitted motor currents for the hot motor; the observed data are shown in blue, and fitted data are in red. The minimization was performed with only observations of the motor current.

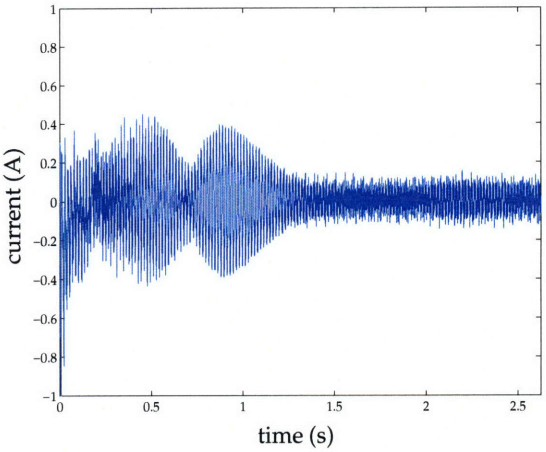


Figure 2-44: The residual obtained via $i_{a,fit} - i_{a,obs}$.

from the parameter estimates generated from these different datasets. By conducting the same test in an environment with minimal variations, such as the laboratory at MIT, the variation inherent in the both estimation method and the system can be studied.

2.8.2.1 *Hot motor: current-based estimation*

The first set of data presented is that which was discussed in the implementation section; in this experiment, the motor was first run for approximately three hours to reach thermal equilibrium. The motor was then turned off, the shaft was allowed to stop spinning, and then the motor was immediately turned on again so that the resulting hot motor transient could be obtained. This data was sampled at approximately 14.29 kHz. The parameter estimation method was then used to determine the set of parameters for the model which best fit the observed current; the set of current observations and the resulting set of predictions, as well as the residual between the two, can be seen in Figures 2-43 and 2-44. Note that the scale on the time axis on each of these plots is somewhat different; this was done to ensure that the quality of fit and regions of misfit are visible in Figure 2-43, while the size of the overall residual, which is difficult to visually evaluate because of the extent to which the two waveforms are the same after the first 1.5 seconds, is illustrated in Figure 2-44.

The motor parameters estimated from this transient are listed in Table 2.5, while the

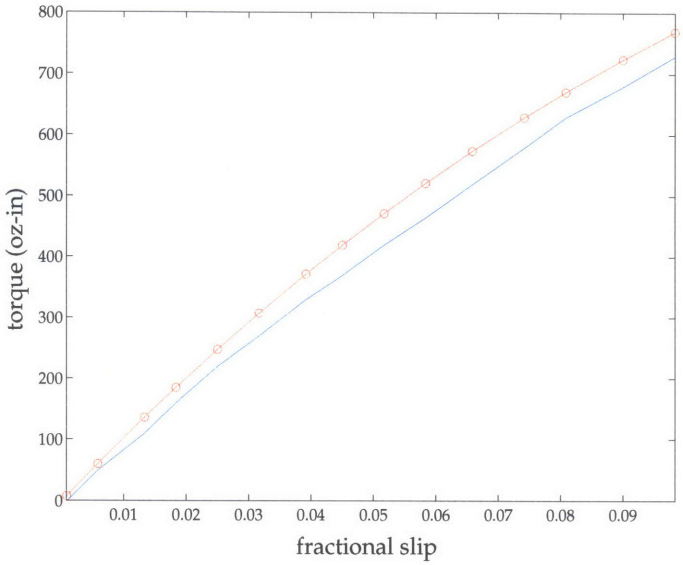


Figure 2-45: Plot of the estimated and measured torque-speed curves for the hot motor; the measured curve is in blue, and the estimated curve is in red.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Value	6.25	4.03	57.75	3.14	7.71	4.59e-04	31.00

Table 2.5: Hot motor parameters obtained by minimizing only against the observations of the motor current.

torque-speed curve obtained from these parameters over the region of slip measured by the dynamometer is illustrated in Figure 2-45. For the purposes of comparison, the set of torque-speed points measured on the heated motor that is mounted on the dynamometer test stand is also shown.

Comparison of the predicted torque-speed curve, obtained from the fitted motor parameters, with the observed torque-speed datapoints shows that the observed torque-speed curve falls below the line of the torque-speed curve as predicted from the motor parameters. This is expected because the observed torque-speed points also include the mechanical losses, which are not modeled by the electrically-based model. The exclusion of such behavior from the model would cause the actual motor to to produce less torque at given speeds than is otherwise indicated by the predicted torque-speed curve.

The final set of results shown are those in which a few identical experiments were performed with the motor in the same condition. For each of these tests, the motor was run

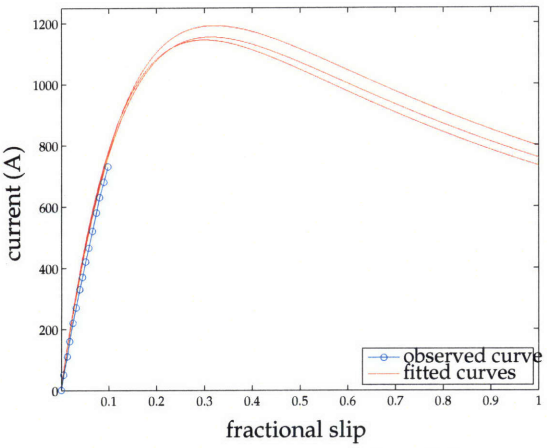


Figure 2-46: Observed and fitted hot motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data are in red, and observed torque-speed data from the motor is in blue.

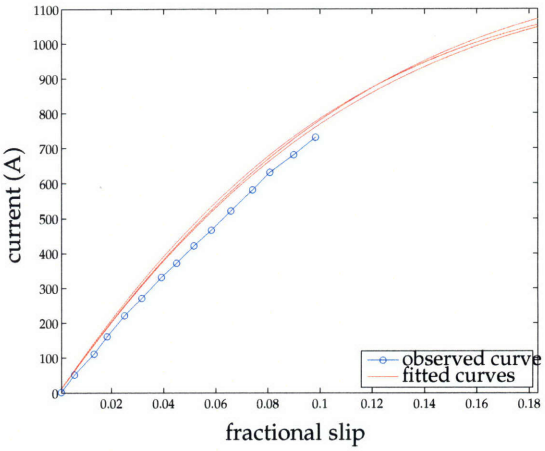


Figure 2-47: Zoomed torque-speed curves.

for a number of hours, turned off and spun down, and then the resulting startup transient was collected. Each of these experiments was conducted on a different day, to ensure that independent observations of the system in similar situations were collected. The torque-speed curves that are generated by the parameter estimates obtained from each of these datasets are illustrated in Figures 2-46 and 2-47. It is interesting to note from this plot that the three estimated torque-speed curves are all quite similar to each other, especially as compared to the observed set of torque-speed points. This fact provides additional encouragement that the estimates of the torque-speed curve are accurate, and that the deviation is dominated by the unmodeled losses.

2.8.2.2 Cold motor: current-based estimation

The second dataset studied is that in which the motor current is observed when the motor is at room temperature ("cold"[3]). This test was conducted by observing the startup current transient when the motor was turned on after a period of at least 12 hours during which it was at rest. The motor parameters were estimated using only these observations of the motor current; the observed motor current, as well as the predicted current that

[3]In comparison to the motor that has been running continuously for over an hour, which is most best described as "hot," in the absence of any other descriptive designation.

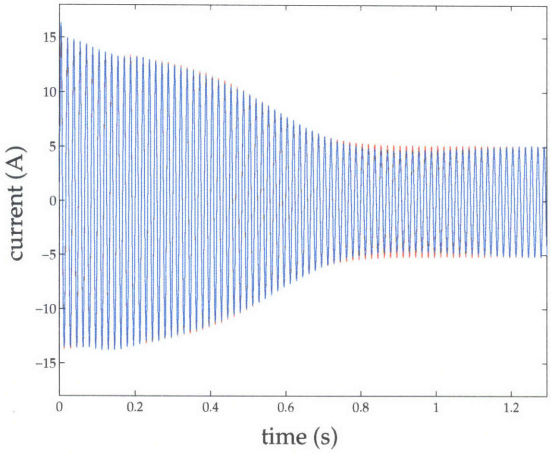


Figure 2-48: Observed and fitted motor currents for the cold motor; observed data are in blue, and fitted data are in red. The minimization was performed with only observations of the motor current.

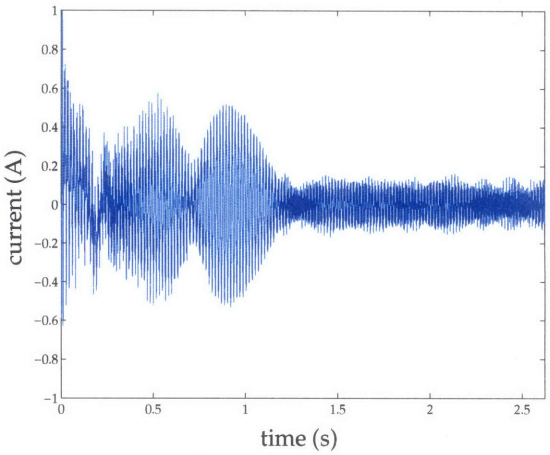


Figure 2-49: Residual from $y_{fit} - y_{obs}$.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Value	4.96	3.24	53.67	1.64	9.23	4.91e-04	29.25

Table 2.6: Cold motor parameters obtained by minimizing solely against the current.

results from a simulation using the estimated parameters, are shown in Figure 2-48, with the corresponding residual illustrated in Figure 2-49. Qualitatively, these plots are quite similar to those of the hot motor with the current-only estimation. The predicted motor current fits the observed motor current quite well, and the residuals are of comparable size. While the airflow estimation method is largely reliant upon the parameters of the hot motor, the performance of the motor parameter estimation for the cold motor suggests that the method can accurately characterize the performance of the motor regardless of its final temperature.

The parameters of the cold motor identified by the parameter estimation method are listed in Table 2.6. The torque-speed curve generated from these parameter estimates, along with the set of observed torque-speed parameters for the cold motor as obtained from the dynamometer test stand, are plotted in Figure 2-50. The first observation that might be made by comparing these parameter estimates with the parameter estimates obtained from the hot motor data is that R_s and R_r are lower for the cold motor than for the

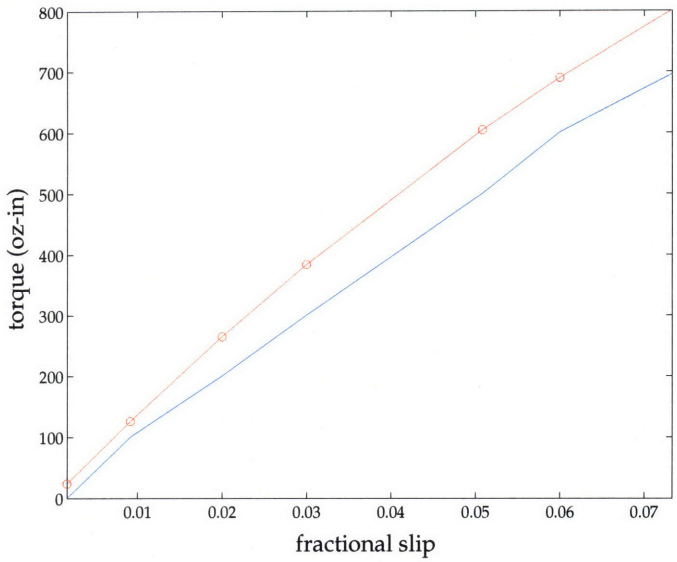


Figure 2-50: Plot of the estimated and measured torque-speed curves for motor; the measured curve is in blue, and the estimated curve is in red.

hot motor, as would be expected. One might also note that expectations are also met by the fact that the mechanical parameters β and K are very similar to those estimated for the hot motor. In addition, the relationship between the predicted and observed torque-speed curves for the cold motor is similar to the same relationship for the hot motor; the predicted torque-speed curve is higher than the observed torque-speed curve.

A set of predicted torque-speed curves for a number of cold motor starts that were collected on different days is illustrated in Figure 2-65. These plots are also qualitatively similar to the analogous plots for the hot motor, providing additional evidence for the consistency of the estimated motor parameters for a given experimental configuration.

2.8.2.3 Hot motor: torque-speed and current-based estimation

The third dataset present used observations of both the startup current transient discussed in §2.8.2.1 and a set of observations of the hot motor's torque-speed characteristic to estimate the parameters of the hot induction motor. The observations and predictions of the motor current and the corresponding residual appear in Figures 2-53 and 2-54. Not surprisingly, it is clear from looking at the plots of both the motor currents and the residual that the fit of the predicted motor current to the observed current is not quite as good when minimizing against observations of both the current and the torque-speed characteristic as

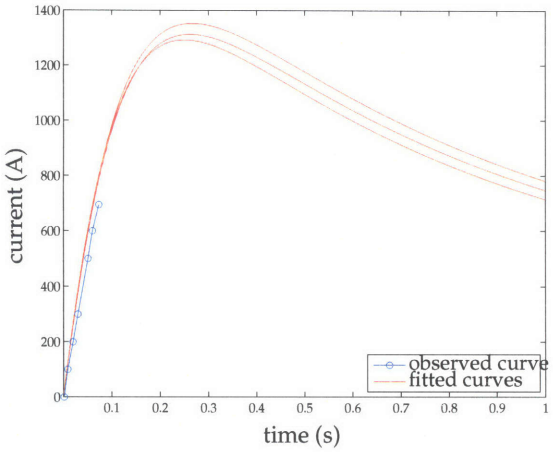


Figure 2-51: Observed and fitted cold motor torque-speed curves for a number of motor starts on different days; fitted data are in red, and the torque-speed data from the motor is in blue.

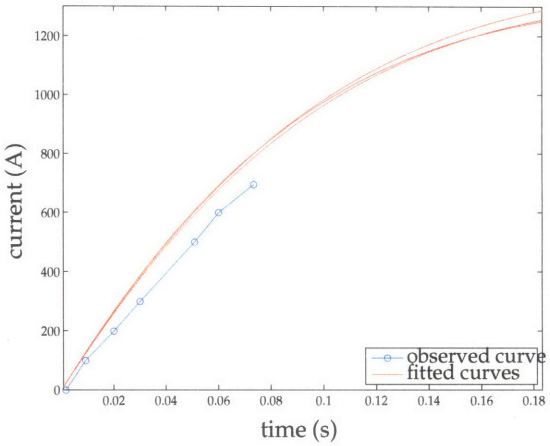


Figure 2-52: Zoomed torque-speed curves.

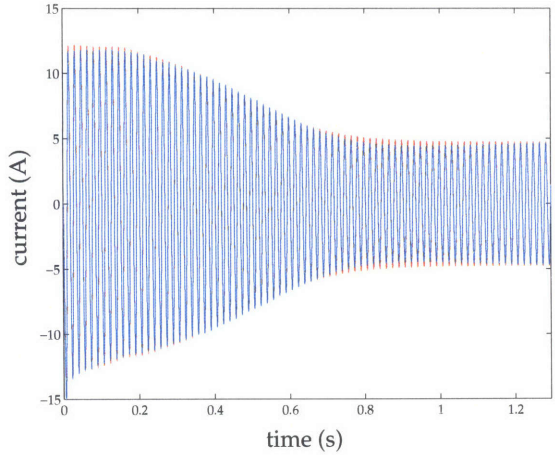


Figure 2-53: Observed and fitted currents for the hot motor; observed data are in blue, and fitted data are in red. The minimization was performed with observations of both the motor current and torque-speed characteristic.

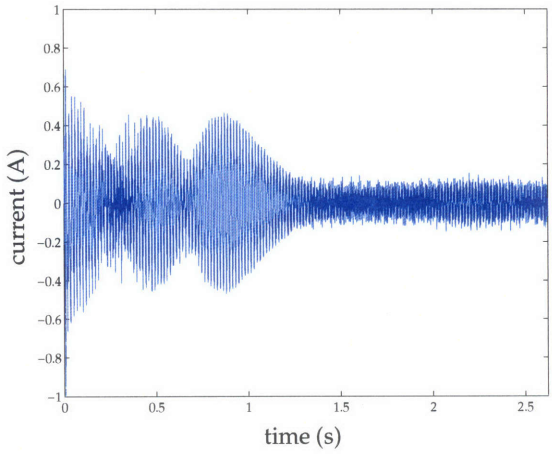


Figure 2-54: Residual from $y_{fit} - y_{obs}$.

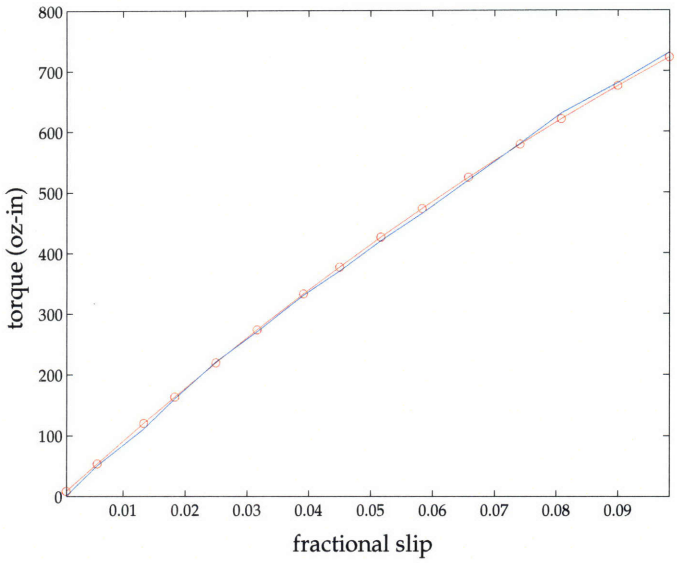


Figure 2-55: Plot of the estimated and measured torque-speed curves for hot motor; the measured curve is in blue, and the estimated curve is in red.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Value	5.24	4.76	57.84	2.86	7.74	4.98e-04	25.03

Table 2.7: Hot motor parameters obtained by minimizing against both the current and the torque-speed curve.

when the minimization is performed solely against the current. In this respect, it is perhaps more remarkable that the fit is as good as it is, since the residual is only slightly larger in Figure 2-54 than it it is in Figure 2-44.

The parameters estimated from the observed current and torque-speed characteristic for the hot motor are listed in Table 2.7, and the torque-speed curve generated by these estimated parameters is plotted along with the observed torque-speed datapoints in Figure 2-55. One caveat must be made in comparing these parameters to the previous set of hot motor parameters as listed in 2.5: while the previous set of parameters is biased only from the measurement error in the current and and the inaccuracy present in the model, the set of parameters presented in this section are also fitting the inaccuracies due to neglecting the mechanical losses of the motor as well as the errors made in assuming that the behavior of the hot motor on the dynamometer test stand is the same as the behavior of the fan motor after a few hours' operation. This is clearly a fairly bold set of assump-

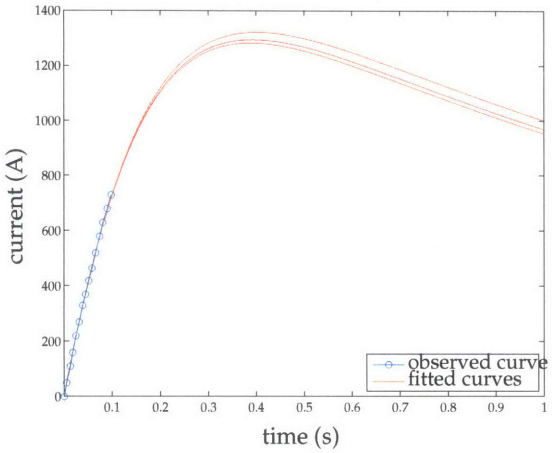


Figure 2-56: Observed and fitted hot motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data is in red, and the torque-speed data from the motor is in blue. These motor parameters were obtained by fitting to the motor current and the torque-speed curve.

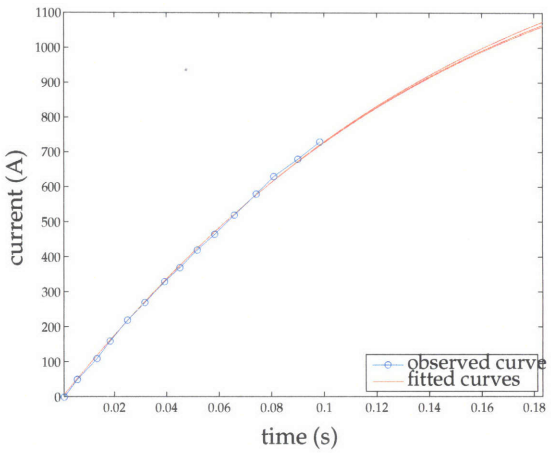


Figure 2-57: Zoomed torque-speed curves.

tions to make, but it is generally reasonable for the sake of testing this method. Looking at the set of parameters, it is notable that all of the parameters listed in the set given in Table 2.7 are fairly close to those of the set given in Table 2.5. Due to the fact that the observed torque-speed datapoints were used to estimate the parameters of the model, it is perhaps unsurprising that the agreement between the predicted torque-speed curve and the observed data are as good as they are; nevertheless, it is encouraging that the set of observed data and the predicted data should be in such close agreement. The closeness of the parameters of these two datasets suggests that the motor is not extremely sensitive to any particular parameter in getting a final estimate of parameters. This allows the torque-speed characteristic to be matched closely, as would be expected from the overdetermined nature of the model.

As the parameter estimates were obtained by using observations of the torque-speed characteristic, the close agreement between the predicted torque-speed curves and the observed torque-speed datapoints for a set of different experiments, as illustrated in Figure 2-56, is to be expected. Nevertheless, it is encouraging to see that all of the predicted torque-speed curves match as closely as seen here. This suggests that this would

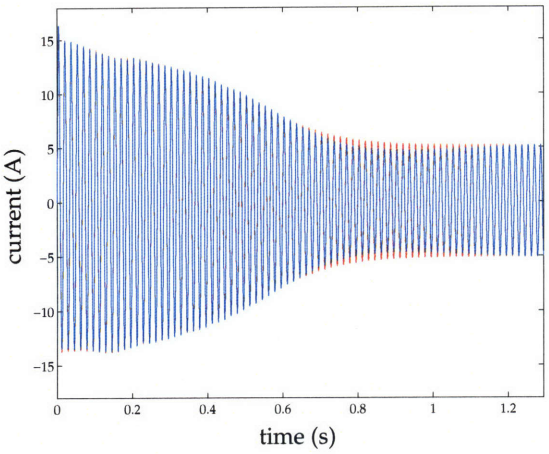


Figure 2-58: Observed and fitted currents for the cold motor; observed data is in blue, and fitted data is in red. The minimization was performed with observations of both the motor current and torque-speed characteristic.

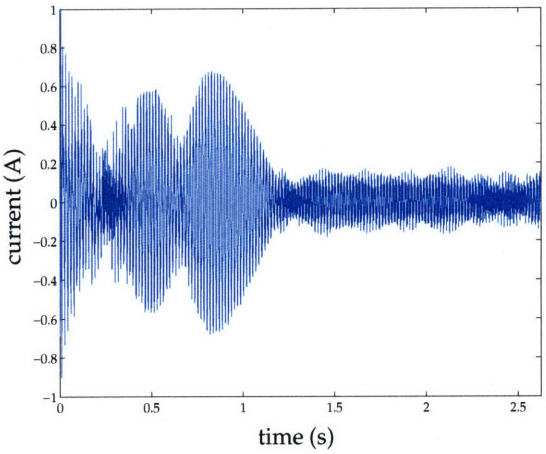


Figure 2-59: Residual from $y_{fit} - y_{obs}$.

be an extremely effective method to estimate the motor parameters if observations of the torque-speed characteristic of the fan motor could be obtained that accurately described the behavior of the loaded fan motor in operation.

2.8.2.4 Cold motor: torque-speed and current-based estimation

The final set of motor data examined is that in which the parameters of the cold motor are estimated using both the current transient and the set of observed torque-speed datapoints for the cold motor. The prediction of the motor current from the motor parameters and the observed motor current are shown in Figure 2-58, and the respective residual is also illustrated in Figure 2-59. As was the case in the estimation of the hot motor parameters from both observations of the current transient and the torque-speed characteristic, these plots show that the prediction of the motor current does not fit the observed motor current as well when performing this type of minimization. The size of the residual in this case is even larger than before; while the fit still appears to be relatively good, the addition of the torque-speed constraints has appeared to have an adverse effect on the fit of the current transient.

The parameters estimated from this data are given in Table 2.8, and the torque-speed

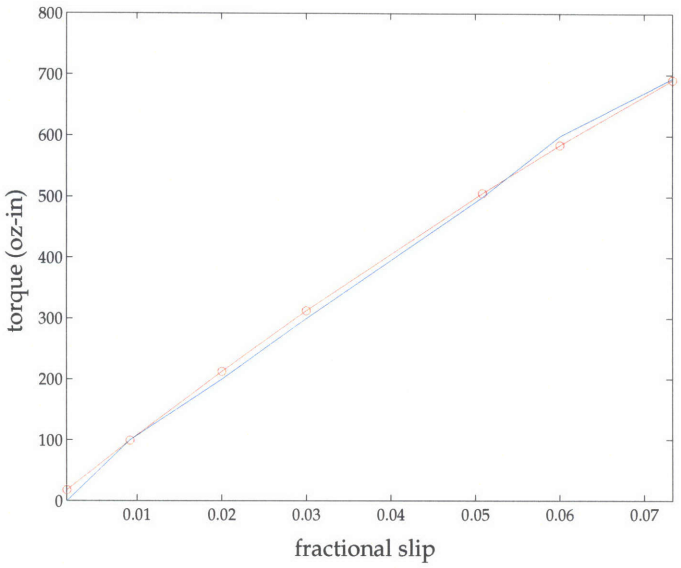


Figure 2-60: Plot of the estimated and measured torque-speed curves for cold motor; the measured curve is in blue, and the estimated curve is in red.

Parameter	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β $[\mathrm{N{\cdot}m/(rad/s)^2}]$	K $[1/\mathrm{kg{\cdot}m^2}]$
Value	3.94	4.12	56.32	1.71	8.86	5.39e-04	21.71

Table 2.8: Final cold motor parameters minimizing against the current and the torque-speed curve.

curve generated by these parameters is illustrated in Figure 2-60. As was the case for the two types of estimates for the hot motor, the sets of parameters given in Tables 2.6 and 2.8 are relatively close; their differences can be ascribed to the adjustments required to fit the observed torque-speed datapoints. While the observed torque-speed curve is fit very well by the predicted parameters, it is unclear as to the overall effectiveness of such a parameter estimation method for this case, as the process of obtaining the measured torque-speed datapoints doubtlessly caused some resistive heating of the motor windings, thereby changing the motor's torque-speed characteristic. While the effects of such a process were minimized in the way that the datapoints were obtained, it is nearly impossible to eliminate such effects.

The torque-speed predictions generated by the sets of parameters that are obtained from estimates of three separate and unrelated datasets are shown in Figures 2-61 and 2-62, to study the variation existing in the method for a number of unrelated datasets. As would

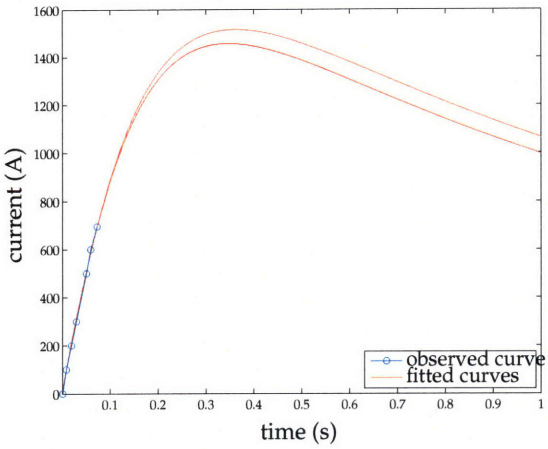


Figure 2-61: Observed and fitted cold motor torque-speed curves for a number of unblocked fan motor starts on different days; fitted data is in red, and the torque-speed data from the motor is in blue. These motor parameters were obtained by fitting to the motor current and the torque-speed curve.

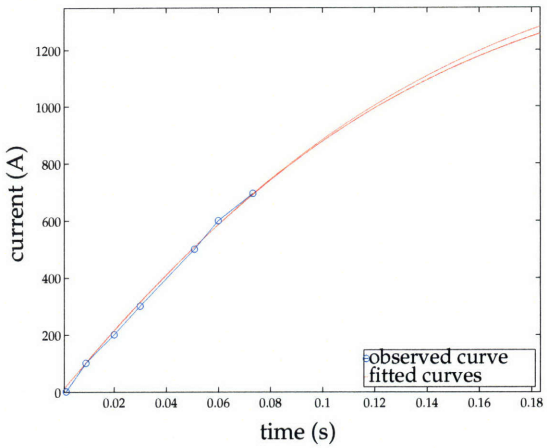


Figure 2-62: Zoomed torque-speed curves.

be expected, the resulting torque-speed curves are quite similar, since they are all obtained by minimizing against the same set of torque-speed observations. The predicted torque-speed curves are all very similar to much higher values of slip as well, suggesting that there is some value in extrapolating from the observed torque-speed datapoints, as long as the observed torque-speed datapoints accurately represent the behavior of the system under the desired operating conditions.

2.8.2.5 *Summary of estimation for the four different conditions*

In general, the model of the induction motor proposed in §2.4.1 is able to accurately describe the observed behavior of the fan motor. In addition, the method used to estimate the parameters of this model for the motor *in situ* can successfully identify the corresponding parameters that characterize the operation of the motor in a given situation. For the ease of comparison, each set of motor parameters for the four conditions discussed in previous sections has been collected in Table 2.9. This side by side comparison makes many of the parameter changes very clear; for example, the increased motor temperature causes R_s and R_r to increase in both of the estimation schemes, as would be expected. An additional

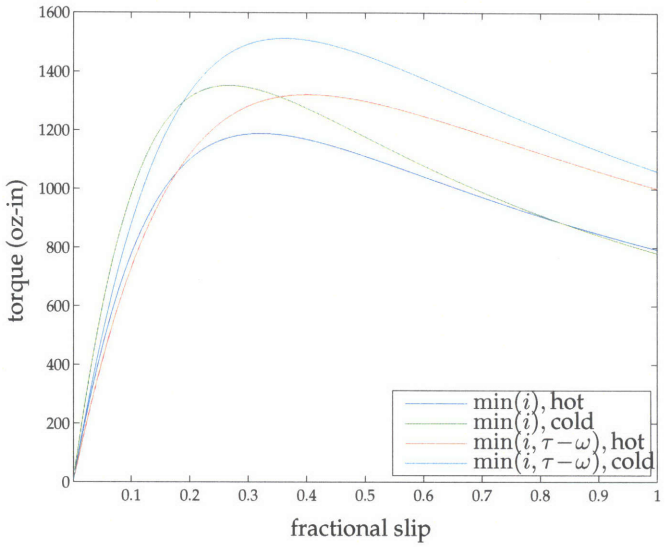


Figure 2-63: Plot of the estimated torque-speed curves for the motor under the set of different conditions. The particular conditions correponding to each curve are given in the legend.

Method	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
i, cold	4.96	3.24	53.67	1.64	9.23	4.91e-04	29.25
i, hot	6.25	4.03	57.75	3.14	7.71	4.59e-04	31.00
i, $\tau-\omega$, cold	3.94	4.12	56.32	1.71	8.86	5.39e-04	21.71
i, $\tau-\omega$, hot	5.24	4.76	57.84	2.86	7.74	4.98e-04	25.03

Table 2.9: Aggregate table of all estimated motor parameters.

observation that becomes apparent upon looking at this data is that the changes in all of the parameters between the datasets of differing temperature are comparable for both types of minimization; for example, the R_s increases by 26% for the current-only minimization and by 32% for the torque-speed and current minimization. This trend suggests that, even given the relative adjustment in the motor parameters caused by the constraints imposed by the observed torque-speed characteristic, the parameters will reflect the changes caused by changes in temperature uniformly.

The effect of the changes in temperature and minimization method can also be seen by comparing four representative torque-speed curves, as generated from the four different estimation conditions. These curves can be seen in Figure 2-63. The torque-speed characteristics as measured on the dynamometer test stand were not included in this plot

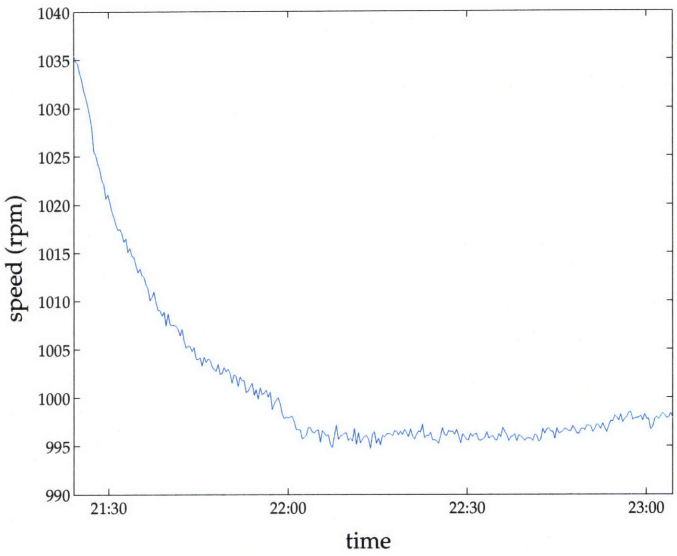


Figure 2-64: Plot of the unblocked motor speed as a function of time as the motor warms up, starting from room temperature.

because the curves for the torque-speed and current-based minimization effectively interpolate those same observations. It is clear from examining these curves that the effect of the motor temperature is approximately the same for both types of minimization; that is, the increasing motor temperature causes the low-slip (up to approximately $s = 0.2$) to decrease the amount of torque produced for the given amount of slip. This does not imply that the amount of torque produced by the machine decreases as the it warms up, however. It is important to note that the motor speed changes dramatically as its temperature increases, as can be seen in Figure 2-64. This increase in slip, accompanied by the change in the shape of the torque-speed characteristic, will effectively reduce the changes in the torque caused by the winding heating. Nevertheless, while these features of the transient behavior of the motor provide some interesting insights into its operation, the main area of concern in this research is with the steady-state operation of the air handler, so that such analysis is largely beyond the scope of this work.

2.8.3 *Effects of different operating conditions on parameter estimation*

The final set of variation of operating characteristics of the motor that can be evaluated is that in which the air handling unit is operated in a variety of fault conditions. As men-

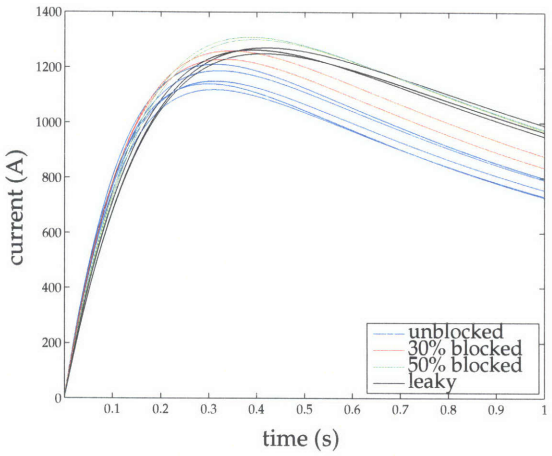


Figure 2-65: Fitted hot motor torque-speed curves for a number of motor starts in different conditions; these conditions are listed in the legend. These motor parameters were obtained by fitting to the motor current only.

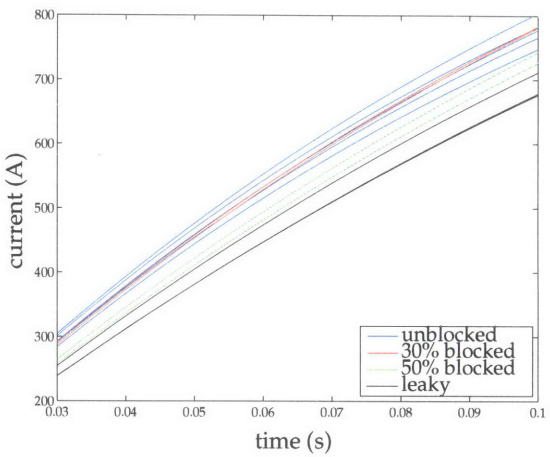


Figure 2-66: Zoomed torque-speed curves.

tioned previously, each of the faulty conditions is associated with a change in both the load on the fan motor and a change in the airflow cooling the motor; both of these types of changes could conceivably alter the operating torque-speed characteristics of the machine. These predicted torque-speed curves for a number of different air handler fault conditions are shown in Figure 2-65 in their entirety, and then zoomed in on the particular region in which the fan motor usually operates in Figure 2-66. Only the torque-speed curves for the current-based estimation are described below, as the parameter estimation with the torque-speed constrains exhibits almost no variation in the region of interest.

It is apparent from these different characteristics that the torque-speed curve of the motor is markedly affected by changes in the condition of the air handler. These sources of variation in the torque-speed characteristics can be better understood by considering one representative set of the model parameters for each these different fault conditions, as listed in Table 2.10. It is important to remember the caveat that some of the parameters appear to be underconstrained, as can also be observed in Table 2.4. That table showed that the parameters R_s, β, K, and R_r (to a lesser extent) are relatively consistent for a given set of data in the face of different initial guesses, while other parameters exhibited less consistency. There are a few plausible explanations for these changes in parameters. For example, the change in the load, due to the different amounts of air being coupled to the

State	R_s [Ω]	R_r [Ω]	X_m [Ω]	X_{ls} [Ω]	X_{lr} [Ω]	β [N·m/(rad/s)^2]	K [1/kg·m^2]
Unblocked	6.25	4.03	57.75	3.14	7.71	4.59e-04	31.00
30% blocked	5.73	4.39	57.39	1.65	9.36	4.26e-04	27.22
50% blocked	5.23	4.72	60.34	2.51	8.39	3.84e-04	22.41
Leaky	5.64	4.84	56.53	2.72	8.10	5.06e-04	24.72

Table 2.10: Table of all estimated motor parameters for different fault conditions.

fan, could potentially cause the observed changes in the parameters. Another explanation is that the change in the airflow could cause the fractional slip to increase, thereby causing more resistive heating. The unusual curve and datapoint in this respect is that of the leaky duct; one would normally expect that the increased airflow and approximately equivalent slip would equate to a torque-speed characteristic that is comparable to the unblocked duct. Yet another source of variation could simply be changes in the environmental conditions of the laboratory. Nevertheless, while the reasoning behind these explanations is based upon some sense of physical intuition, the precise cause of these changes would ideally be characterized in a more controlled environment to understand the sources of variation in the system.

2.9 Airflow estimation

As the speed and torque estimation modules have been explored and demonstrated fully in previous sections, the only module that remains to be studied is the flow prediction module. It is helpful at this point to recall the overall structure of the airflow estimation method: for a given condition of the air handler, the speed is first estimated from the slot harmonics present in the current, then this speed is used to identify the corresponding torque on the torque-speed curve that describing the motor's present state. With these estimates of the torque and the speed, the mechanical power into the fan can be estimated, and the airflow that corresponds to this measured power and speed can then be determined via a fan curve. Since the torque-speed curve is clearly dependent upon the temperature of the motor, the startup transient was obtained in the same manner as discussed in §2.8.1.

The airflow predictions generated by the above method were validated by measuring the airflow through the duct in an unblocked state and with the imposed faulty condi-

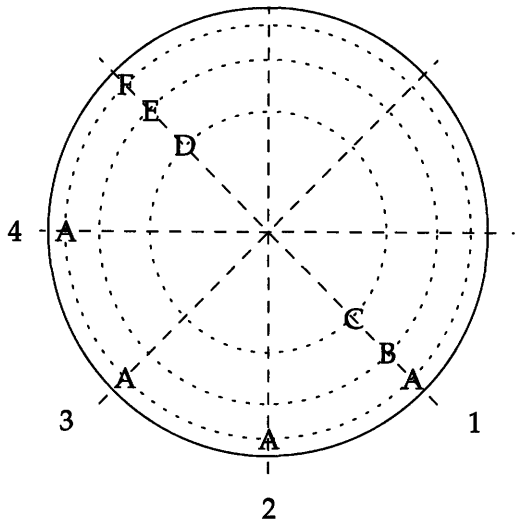


Figure 2-67: Cross-section of the duct illustrating the points where the measurements were made.

tions. These validation tests were performed by measuring the airflow with a hot wire anemometer, as described in §2.6; for each specified fault condition, four traverses were measured with six samples of the airflow per traverse, so that a total of 24 points were sampled for each fault condition. A general schematic of this measurement process, not drawn to scale to enhance visual clarity, is illustrated in Figure 2-67. In this figure, the traverses are numbered one through four, and the sample points at which the anemometer measured the airflow are labeled A-F. These measurements were made at a set of points along the given duct diameters that are specified to produce an accurate average of the flow, as enumerated in [10, ch. 19].

As mentioned previously in §2.7, the speed of the fan varies somewhat substantially over the course of the day, so these airflow measurements were conducted over a period of relative speed stability in the evening. Before conducting these tests, the fan was turned on and allowed to reach thermal equilibrum, so that the airflow would not be affected by changes in the motor impedances. Due to the need for the fan motor to reach a new thermal equilibrium after changing the blockage, the test sequence ran as follows after the motor had been running in its previous state and reached equilibrium in that state: first, the inlet was blocked down by the specified amount (e.g., 30%). The fan was then allowed

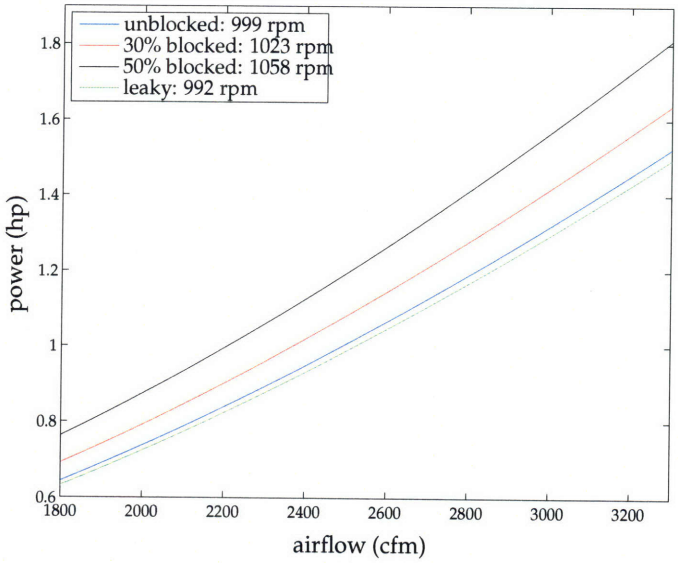


Figure 2-68: Fan curves illustrating the power-flow relation at the four speeds of interest.

to run for 50 minutes to an hour to reach thermal equilibrium. After this period of time, the set of traverses were conducted to measure the airflow at this level of blockage. After the traverses were complete, the inlet of the air handler was changed to reflect the next faulty condition under test.

The power-flow fan curves for the different speeds of operation as generated by the fan coefficients are shown in Figure 2-68 to compare the effects of the different speeds on the corresponding power-flow characteristic. This plot makes it clear that the different blockage conditions will have a significant effect on the flow through the duct.

It was necessary to identify both the mean and the standard deviation of the speed and torque estimates to estimate the flow to an acceptable degree of confidence. These quantities were therefore measured and used to run a series of simulations that could generate distributions of the airflow predictions. These distributions could then be used to quantify the probability that a given airflow measurement represented a given blockage condition. These statistics for the measured speed and torque, as well as the corresponding derived distributions for the power and airflow, are given in Table 2.11a.

This table also includes the numerical values of the predicted airflow. These values were calculated using the following procedure: first, the measured mean and standard deviation for the speed were used to generate a set of 1000 simulated values of the speed

that statistically matched the measured speed. Each of these speed values were then used to generate a distribution of torque points according to the statistical distribution of the predicted torque-speed curves. This resulted in a 1000x1000 matrix of torque estimates, in which each row corresponded to a speed estimate. Each element of this matrix was then multiplied by the corresponding speed estimate, which generated a similarly sized matrix of simulated power datapoints. These power datapoints were then used to estimate the corresponding airflow via the fan curve; this set of 10^6 simulated airflows could then be used to identify the mean and standard deviation of the airflow predictions, and evaluate them in comparison with the the corresponding velocity and airflow measurements for the unblocked, 30% blocked, and 50% blocked ducts, as given in Table 2.11. The air velocities in the leaky duct experiment are not included in the table of validation measurements, as they could not be measured upstream of the leak in the experimental apparatus.

As is illustrated in Table 2.11a, both of the motor parameter estimation methods were used to generate airflow predictions to assess the performance of each method. In the upper half of the table, the torque-speed curves obtained by performing the current-based parameter estimation were used to make the airflow predictions, while the airflow predictions in the bottom half of the table were generated by the torque-speed characteristics from the torque-speed and current-based parameter estimation methods. To facilitate the interpretation of the data in these tables, the histograms of the airflow predictions are also illustrated in three related plots, representing three possible approaches to the use of this FDD method. In the first of these plots, Figure 2-69, the airflow predictions are generated from four different torque-speed estimates which represent the system in its four faulty states. For each condition of the air handler (i.e. unblocked, 30% blocked, 50% blocked, and leaky), the system was run for two hours, then turned off and immediately turned back on to obtain a startup transient, and then the torque-speed curve used to generate the airflow prediction was obtained from the motor parameters identified by the current-only parameter estimation method. In the second plot of the airflow prediction histograms, Figure 2-70, all of the airflow predictions are generated from a single torque-speed curve. This torque-speed curve was obtained from a set of parameter estimates identified by the current-only estimation method operating on a current transient collected for the fan running in an unblocked state. In the final plot of the airflow prediction histograms, Figure 2-

state; estimation method	μ, speed [rpm]	σ, speed [rpm]	μ, torque [oz-in]	σ, torque [oz-in]	μ, power [hp]	σ, power [hp]	μ, airflow [cfm]	σ, airflow [cfm]
unblocked; i	999.19	2.47	1020.6	10.93	1.01	1.08e-02	2511.4	25.05
30% blocked; i	1023.38	2.21	989.64	2.89	1.00	2.93e-03	2378.8	17.77
50% blocked; i	1058.40	1.91	827.62	12.451	0.87	1.31e-02	1990.9	29.79
leaky; i	992.96	2.40	972.47	4.78	0.96	4.71e-03	2450.8	20.52
unblocked; $\tau-\omega, i$	999.19	2.47	1017.66	5.64	1.01	5.59e-03	2506.5	21.84
30% blocked; $\tau-\omega, i$	1023.38	2.21	959.90	2.77	0.974	2.81e-03	2328.0	20.38
50% blocked; $\tau-\omega, i$	1058.40	1.91	827.63	2.03	0.869	2.14e-03	1991.2	20.37
leaky; $\tau-\omega, i$	992.96	2.40	1020.68	3.43	1.01	3.38e-03	2533.5	19.36

(a) Predicted airflow values.

state	μ, velocity [m/s]	σ, velocity [m/s]	μ, airflow [cfm]	σ, airflow [cfm]
unblocked	8.13	0.115	2304.92	33.0
30% blocked	7.65	0.115	2168.84	33.0
50% blocked	6.72	0.115	1905.17	33.0

(b) Measured airflow values.

Table 2.11: Predicted and measured airflow values.

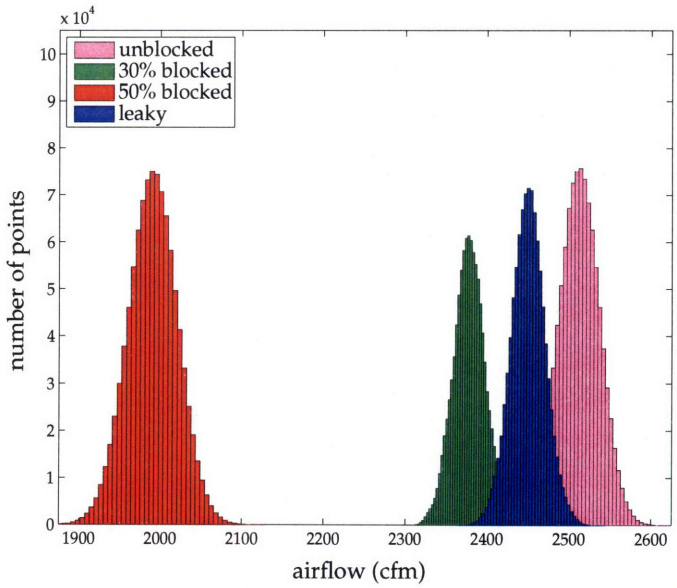


Figure 2-69: Illustration of airflow detectability using torque-speed curves that are generated from minimization against only the motor current, as collected for each blockage condition.

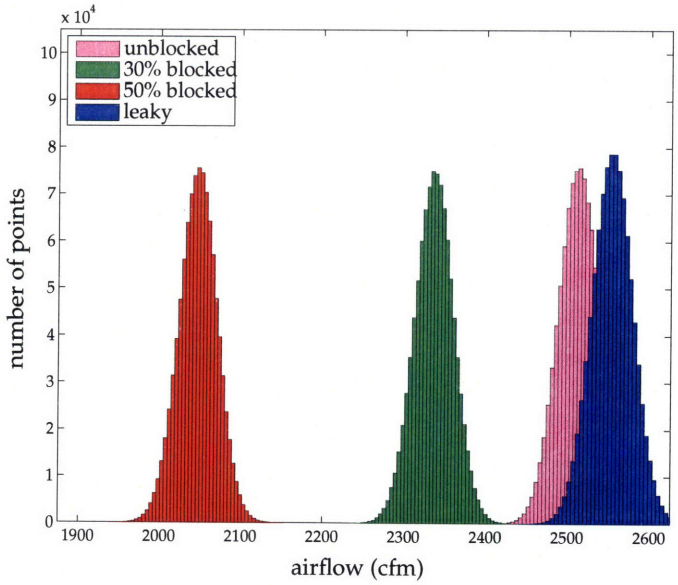


Figure 2-70: Illustration of airflow detectability using a torque-speed curve that was generated from minimization solely against the unblocked motor current.

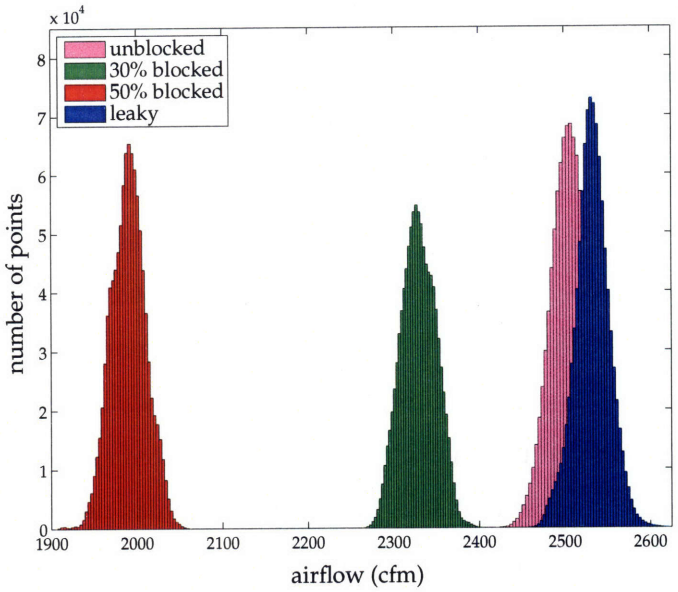


Figure 2-71: Illustration of airflow detectability using torque-speed curves that are generated from minimization against the motor current and the torque-speed curve, as collected for each blockage condition.

71, the airflow predictions were generated in a similar manner to those in Figure 2-69, with the essential difference that the torque-speed and current-based parameter estimation method was used to identify the parameters, rather than the current-only parameter estimation method.

All of these figures illustrate encouraging results. Specifically, it is clear from these figures that the task of differentiating the unblocked flow from either the 30% blocked flow or the 50% blocked flow appears to be relatively simple using either of the motor parameter estimation methods. In particular, the 50% blocked flow is extremely different from any of the other inlet conditions, and should be easily detectable. The 30% flow is also relatively different, and should also be easily detectable; it appears that the tails of the 30% blocked distribution and the unblocked distribution may have some minor overlap.

While the ability to qualitatively identify the presence of blockage is certainly useful, it is even more useful to quantitatively identify the particular airflow in each situation. Examining the unblocked airflow measurement and predictions from Table 2-69, both the current-based prediction (2511 cfm) and the torque-speed and current-based prediction (2507 cfm) are somewhat higher than the measurement (2305 cfm). The close agreement

of the unblocked flows is immediately noticeable, suggesting that little is gained from the minimizing against the torque-speed curves in this scenario. In an absolute sense, these predictions are quite good, as the means of the predictions and measured value differ by about 9%; moreover, the standard deviation $\sigma_{a,m}$ of the set of airflow measurements made by hand is 33 cfm, while $\sigma_{a,p}$ for the predictions is approximately 25 cfm. In addition, the datasheet with the fan coefficients for the particular fan used specify that the values of airflow are accurate to within $\pm 1.25\%$, while the values of power are accurate to within $\pm 2.5\%$. While the prediction and the measurement would ideally be closer, they are sufficiently close to recommend this method for the prediction of airflow, and for the further characterization of the sources of inaccuracy, such as the bias possibly introduced in the process of making the anemometer measurements.

Turning to the values of the 30% blocked airflow, similar results are found: the predicted airflow using the current-based method is 2379 cfm, and the predicted airflow using the torque-speed and current-based method is 2328 cfm, while the measured airflow is 2168 cfm. The two different parameter estimation techniques yield slightly different results for this level of fault, but the two standard deviations of the sets of measurements ($\sigma_{a,p}$ = 18 cfm and $\sigma_{a,p}$ = 20 cfm for the current-based and torque-speed and current-based minimization, respectively) suggest that these differences might be attributed in part to measurement noise and bias. One particularly interesting observation is that the difference between the predicted and measured airflows for the no-fault condition for the air handler (206 cfm) is almost the same as the difference between the predicted and and measured airflows for the 30% blocked air handler (211 cfm). While this may be partially attributed to a particularly fortuitous set of measurements, it also suggests that the source of bias has a relatively constant value. If this is the case, this bias may be able to be easily calibrated out of the system, thereby increasing the accuracy of the airflow measurements.

The analysis of the 50% blocked fault level is quite similar to that for the 30% blocked fault. It might be particularly noted that both of the airflow predictions are identical (1991 cfm). One change from the 30% condition is that the difference between the predicted and measured flows is only 86 cfm, rather than 210 cfm. One potential cause of this change is the loss of model accuracy due to the fact that the present operating condition of the fan lies slightly outside the region of operation used to generate the fan coefficients, causing

a larger residual between the empirical data and the predictions obtained from the fan coefficients in this region. A comparison of the values calculated using the empirical data given in Figure 2-72 illustrates that the power predicted by the fan coefficients at a given speed and airflow is quite close to the power predicted by the empirically obtained set of curves. Private correspondance with Tim May from Lau Fan Corp. suggests that the prediction from the fan curves at this operating point has a lower accuracy than predictions generated by the fan curves in other regions of operation. Nevertheless, this data indicates that the airflow at this level of blockage can be known with similar accuracy as the other fault and no-fault conditions.

Yet another reason that these differences in the torque-speed curves between the two estimation methods diminish as the fractional slip decreases is that the sensitivity of the model that generates the torque-speed curve decreases as the induction motor runs closer to synchronous speed. This fact is apparent from noticing that the curves tend to converge as the slip gets closer to zero in Figure 2-63. It is thus reasonable to expect the discrepancy between the predictions from the two methods to get smaller as the fractional slip decreases. It is also striking that the torque-speed curves do not apparently yield significantly better estimates of the airflow; by eliminating the need to obtain *a priori* observations of the torque-speed characteristic, the process of identifying the torque produced by the electric motor is much more straightforward.

Analysis of the leaky duct fault condition for the two different motor parameter estimation methods provides perhaps the most surprising result. If the operating conditions of the motor were identical for the fan in both the leaky fault condition and the unblocked condition, one would expect that the flow through the fan would be greater in the leaky condition because there would be a smaller pressure drop across the fan due to the open access door. This intuitive expectation is reinforced by the airflow predictions made with the torque-speed and current-based model; since the torque-speed characteristics of the leaky and unblocked motors obtained by this method are nearly identical, the increase in fractional slip results in a higher amount of torque produced, as well as a higher quantity of mechanical power delivered. In comparison, the torque-speed curve for the current-based estimation is sufficiently different from the torque-speed curve for the leaky duct that the the airflow through the leaky duct is lower than the airflow through the unblocked duct.

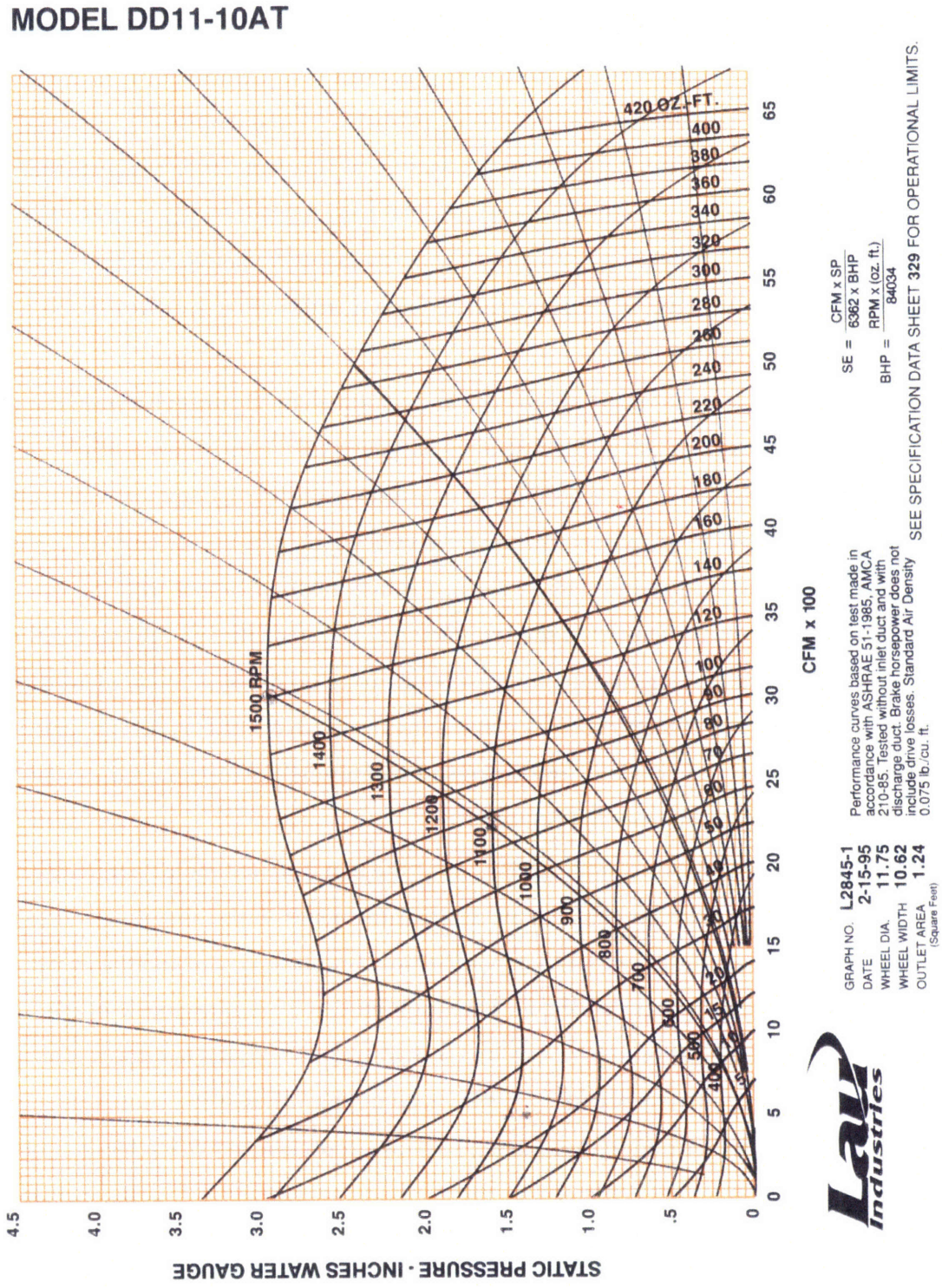


Figure 2-72: Empirically measured fan curve for the blower wheel used in experimental air handler.

This begs the question: what causes the torque-speed curve to change this much? One possible cause is that the torque-speed characteristic is dramatically affected by the change in the load due to the leaks in the ducts, as the increased load on the fan could have thermal effects causing changes in the winding impedances, which in turn could cause the motor to produce less torque at a given speed.

Unfortunately, the particular experimental setup used precludes the empirical measurement of the airflow upstream of the leak and direct determination of the airflow through the fan, as the high degree of turbulence upstream of the access door makes spatial averaging with anemometer measurements in the short upstream length of duct all but impossible. In addition, experimental limitations preventing direct torque measurements being performed in the fan *in situ* make it difficult to explore and identify potential causes for the apparent changes in the torque-speed curve.

Nevertheless, consideration of these results suggests a number of effects upon the implementation of this type of system in a field installation. There are two distinct scenarios in which this airflow estimation method might be used: for validation and commissioning of the air handler when it is installed, and for tracking the airflow delivered to the ducts in normal operation; in the terminology of Chapter 1, the system could be used for detecting either commissioning faults or operational faults. As these results indicate, this airflow prediction method can identify the changes in the flow quite well and can thus identify operational faults. While the assumption that the system is commissioned correctly is not always accurate, many FDD systems are constructed on the basis of such assumptions, and it is useful to consider the capabilities of an airflow prediction system that only needs to monitor ongoing faults. Moreover, such an assumption is often quite useful from the standpoint of implementing a FDD system, as it requires a minimum of *a priori* information for the airflow prediction system, since no correction factor or measurement of a torque-speed characteristic is needed to identify operational faults which result in a change in the volumetric airflow.

The identification of the commissioning faults is more difficult, however, and requires more information than do the operational faults. The structure of possible field implementations of an airflow prediction system which is able to identify commissioning faults depends upon the desired accuracy of the method and the validation of the torque-speed

behavior observed in the fan motor from the current-only parameter estimation. Comparison of the results illustrated in Figures 2-69 and 2-71 emphasizes the important function served by the torque-speed curve, and the method is largely dependent upon the accuracy of these torque-speed estimates.

One possible instance of a field implementation of an FDD method is that additional information about the torque-speed curve is either unavailable or to expensive to attain in more detail; however, the results presented previously suggest that the airflow prediction method could still be extremely useful in the absence of any such correction factor. For example, operational faults can be detected by looking for changes in the airflow predictions as in comparison to the system behavior observed in the initial unblocked state, which is established when the air handler is installed and commissioned. While the absence of a correction factor prevents this commissioning data from being completely accurate, the fact that the predictions are within 10% suggests that such estimates still have considerable utility. Different blockage conditions could then be identified and the reduction in airflow determined by the use of the prediction method to identify reduced flow and higher fan speeds, as is clear from the results given in Table 2.11a. One possible application of such a system would be for detecting relatively large commissioning faults. This could be accomplished by comparing the airflow predictions to the results of a traverse taken downstream, but before any branches leave the duct from the air handler.

Leaky ducts could also be detected for a properly commissioned system, though the particular torque-speed curve used changes the way in which the leak is detected. As is clear from Figure 2-69, if new torque-speed curves are obtained after every period of operation, the presence of a leak will result in a prediction of reduced airflow and a reduced fan speed. Two other possibilities for commissioning and operating the FDD method could also be used, both of which result in the diagnostic indicator for a leak consisting of an increased airflow and a reduced speed: either a single torque-speed curve is obtained from a current-only estimation routing for the unblocked system when it is commissioned (illustrated in Figure 2-70, or the torque-speed curve is obtained for each operational cycle by using the torque-speed and current-based estimation routine.

While the above discussion suggests that additional accuracy in airflow predictions is not required for the construction of a highly useful FDD method, the application of

a correction factor which can compensate for the apparent variation in the torque-speed curves could measurably improve the performance of the prediction method. Such a correction factor would not depend on the particular installation, as it would be largely dependent upon the motor dynamics. One particularly suggestive use of the airflow prediction method could be for the independent verification of commissioning accuracy, as results from the prediction method could be compared to the expected airflow after the installation of the air handler, and any divergence between the two could be identified and repaired.

Such diagnostic tools could have a variety of uses for diagnostics in buildings. For example, the airflow through a ducting system changes as adjustments are made to dampers and so forth; with prior knowledge of how these changes should be affecting the system, the airflow detection method would be able to use knowledge obtained only by the use of electrical measurements of the system response to determine whether or not those responses are within specifications. Even without the added functionality of detecting commissioning faults, this could be very useful. A variety of different operational faults detection methods and energy monitoring applications could be constructed with such tools, such as methods for identifying time-varying energy consumption of a residential building, and so forth.

2.10 Summary

A method for identifying faults in which the airflow deviates from previously established specifications was presented in this chapter. This method exclusively used information available from the fan manufacturer and electrical data, and did not rely upon any mechanical measurements made in the experimental unit. Experimental results were presented in which the efficacy of the method was demonstrated, and two different methods of identifying the airflow were studied, depending on the information available.

Chapter 3

Liquid Slugging Diagnostics

The second fault studied in this research, generally referred to as liquid slugging, involves the heart of any air-conditioning system: the compressor. This chapter will begin by investigating the phenomenon of liquid slugging as it arises in a variety of conditions, which have similar but not identical effects on the behavior of the compressor. Some previous research has also investigated this fault, and this research will be used to motivate and provide some initial direction for the approaches for studying liquid slugging in this research. As this fault manifests itself in a variety of ways in the behavior of the compressor, the suite of instrumentation used to detect this fault, as well as validate the detection method, will be discussed. This chapter will conclude by presenting a variety of data illustrating the effectiveness of this fault detection method and looking at future directions for this research.

3.1 Motivation

In Chapter 1, the basic characteristics of liquid slugging were described, as well as some of its effects on the air-conditioning system. Though this description suffices for a general understanding of this fault, a number of different effects can occur which will influence the structure of the fault detection strategy. In order to properly set up the context for the development of the fault detection method that is proposed in this research, the circumstances in which liquid slugging can arise, as well as other methods used to detect this fault, are discussed in the following section.

3.1.1 Background

While the fault surveys of packaged air-conditioning equipment discussed in Chapter 1 ostensibly classified the range of observed faults into electrical and mechanical categories, the enumeration of the types of faults made it clear that such distinctions are largely artificial. In particular, some of the mechanical faults which afflict compressors can affect both mechanical and electrical subsystems. One such fault that was mentioned in all of the extant surveys occurs when liquid refrigerant enters the compression chamber. Since the compressor is designed to compress refrigerant vapor, this difference in the state of refrigerant in the compression chamber gives rise to a number of phenomena which have a malign effect on the compressor, consequently causing a number of faults. This general fault condition, usually referred to as liquid slugging, was extensively studied in the course of this research. In order to develop and motivate methods of detecting this fault, it is necessary to understand the reasons for its incidence, as well as its myriad effects on the behavior and components of the air-conditioning system.

While liquid slugging can arise for a number of relatively different reasons and in a number of different circumstances, many of the effects of this fault take place regardless of the particular cause of the fault. In general, the presence of liquid in the compression chamber causes higher pressures in the chamber, due to the flashing of the liquid and the relative incompressibility of the liquid as compared to the vapor. These higher pressures result in comparatively larger forces on the mechanical compressor components, such as the valves, the pistons, and the connecting rods. If relatively small quantities of liquid are ingested over long periods of time, the cumulative effect of the increased stresses on the compressor internals can cause those components to wear at an accelerated rate and fail much sooner than would otherwise be expected. Naturally, if very large amounts of liquid are ingested, resulting in very high cylinder pressures, the compressor components can be destroyed almost instantaneously; HVAC repair technicians have found broken valves and shattered connecting rods in compressors in which severe liquid slugging has occurred. Moreover, the liquid refrigerant tends to degrade lubrication by washing away the oil film that normally coats the ring slots and the walls of the cylinders and pistons, causing a variety of problems related to the change in the lubrication conditions.

These higher pressures also make it necessary for the compressor motor to produce more torque to drive the piston so that the liquid can be ejected from the compression chamber. This results in a greater amount of electrical power flowing into the compressor during the slugging event. This fault can thus have the effect of overloading the compressor motor beyond design parameters, giving rise to such faults which are indicative of overloading. In addition, liquid refrigerant which is dissolved in the compressor oil when the compressor starts will begin to boil as the compressor heats up; this oil-refrigerant mixture can be extremely corrosive to motor windings and can cause faults. This can also result in increased quantities of oil being carried out of the compressor shell, starving the compressor of needed lubricant.

To better understand the conditions which cause liquid slugging to occur, it is useful to consider the operation of the compressor as divided into two different periods of time: the period of transient operation during which the fields in the air-gap are being established and the drive train is being accelerated, and the period of steady-state operation, in which the motor is running at roughly constant speed and current. This is an oversimplification of the motor behavior, naturally, as there are a variety of other transients which occur in the induction machines which are used in compressors, such as the long-time transients which affect the motor windings, and thus the motor currents, as they come up to a steady-state operating temperature. The mechanical transients of the air-conditioning system also have a significant effect on the motor behavior, as the load on the compressor motor changes as the quantity of refrigerant in the condenser and evaporator transitions from their values when the system is off to their values when the system is on. Nevertheless, these transients occur sufficiently slowly that the operation of the compressor can be effectively divided into a transient regime and a quasi-steady-state regime for the purposes of this research.

In considering the causes of liquid slugging, the first time interval considered is that of the motor's starting transient; liquid slugging events which occur during this interval will be referred to as transient liquid slugging. Transient liquid slugging can occur when liquid either enters the compressor during transient operation, or when liquid is present in the compressor chamber before the compressor starts operating. These circumstances can arise for a few different reasons. The most common of these reasons can be visualized by considering the following scenario: on a typical summer day, the compressor for an air-

conditioner runs for a large part of the day, but after the sun goes down and the outside air gets cooler, the air-conditioner typically does not need to run as often, if at all. Assuming that the compressor does not run during the evening, the refrigerant vapor will tend to accumulate in the lowest and coolest place in the system. As early mornings in such conditions are relatively cool, the coolest part of the system is often the suction line leading to the compressor. Depending on the level of the refrigerant charge and the refrigerant used, the temperature and pressure in the suction line can be such that the refrigerant present in the suction line, in the compressor head, or in the compression chamber itself will condense, causing liquid refrigerant to be present in some or all of these locations. When the compressor starts for the first time that morning, it will ingest the entire quantity of liquid present in the head and subsequently experience liquid slugging faults. This problem can also happen in a variety of other similar circumstances; one application in which this is particularly common is that of refrigeration trucks, where the typically low temperature of the truck can cause large amounts of liquid to condense in the suction line prior to the compressor start [30, p. 5], [20]. These problems can also be exacerbated by the failure or malfunction of other components, such as malfunctioning expansion valves or refrigerant overcharge.

The phenomenon described above, which is sometimes also referred to as refrigerant migration, can result in the refrigerant travelling into the compressor shell and mixing with the compressor oil. During subsequent compressor starts, the rapid pressure drop in the crankcase pressure will cause the refrigerant in the oil/refrigerant mixture to boil and start to foam. This foaming oil can become entrained in the normal refrigerant flow and cause the lubrication oil to be carried out of the compressor shell, depriving the compressor of needed lubricant. As this oil/refrigerant mixture is ingested into the compressor cylinder, it may also cause slugging problems as well [163].

Many of the causes of transient liquid slugging have analogous counterparts in the case of liquid slugging during the period of time that the compressor is in quasi-steady-state operation. Due to this common and apt description of the operating state of the compressor, the fault which occurs when liquid is ingested by the compressor during steady-state operation will be referred to as steady-state liquid slugging, to distinguish it from transient liquid slugging. Steady-state liquid slugging is also sometimes referred to as liquid

floodback. The causes of liquid slugging in this case can generally be traced to problems with one or a combination of the following: refrigerant overcharge, off-design conditions for the evaporator, or a malfunctioning expansion device. One of the most straightforward causes of steady-state liquid slugging is the condition in which there is simply an over-abundance of refrigerant in the system, resulting in an overfilled evaporator. Problems with the evaporator, such as a dirty or iced coil, a low load, or a non-functional evaporator fan, can also cause liquid slugging because the expected heat transfer in the evaporator coil is less than is needed to evaporate the refrigerant as it travels through the tubes in the coil. Finally, faults in the expansion device, be it a capillary tube or a thermostatic expansion valve (TXV), can also result in a larger supply of liquid refrigerant to the evaporator than will evaporate in the heat exchanger, again resulting in the presence of liquid refrigerant at the exit of the evaporator. Naturally, more than one of these faults could easily take place, especially in an air-conditioning unit which was not well maintained, increasing the probability liquid slugging faults [150, p. 50], [160, p. 392].

Before investigating this fault, it is useful to note the relative frequency of this fault in order to ascertain the necessity of developing a corresponding fault detection method. The fault surveys [22,36,142] all independently emphasize the prevalence of liquid slugging faults in the scope of the identified faults. Moreover, Cunniffe [36] suggests that approximately 20% of the mechanical faults are directly attributable to liquid slugging. Unfortunately, these fault surveys only examined the ultimate cause of the service visit; the proximate cause is difficult to discern, so the effect of liquid slugging on electrical faults, such as winding failure, is not documented. Nevertheless, the detection of a class of faults which are responsible for at least 20% of mechanical compressor failures would certainly be useful.

Before developing such fault detection methods, it is also useful to put their application into context; what could one hope to be accomplished by using this fault detection tool? Perhaps the most relevant answer to such a question is that, given the difficulty of identifying liquid slugging in field-installed units, such a tool would permit service technicians to identify the incidence of liquid slugging in operational equipment without having to disassemble the compressor to look for the tell-tale signs of slugging damage. Such an indicator would give the technician sufficient information to determine whether liquid slugging is

happening often enough to justify the cost of installing the components needed to mitigate its effects, such as a crankcase heater or a suction line accumulator to trap the liquid. A slugging detection method could also interface with automated fault detection methods administered by a remote service company. Such an application could potentially allow a monitoring company to schedule a service visit before the equipment was damaged. It would also enable more accurate fault diagnostic methods and reduce the no defect found (NDF) problem[1], as a service technician could be prevented from sending a compressor back for repair under warranty if liquid slugging was not detected. Finally, such a system, coupled with a remote monitoring system, could enable compressor manufacturers to better understand the incidence of liquid slugging in field-installed equipment, giving them more information with which they can design their compressors and the associated refrigerant and lubricant handling components and controls.

3.1.2 Previous Work

As fault detection methods which were investigated in this research focused on liquid slugging in reciprocating compressors, it is useful to briefly review the construction of the reciprocating compressor, in order to establish terminology and the roles of the various system components involved in the compression process. The relevant parts of the reciprocating compressor are illustrated in Figure 3-1, which is a cross-section drawn from the point of view of the end of the compressor furthest away from the motor. Under normal operating conditions, the superheated refrigerant vapor leaves the evaporator and enters the suction chamber through the port in the left side of the compressor. When the pressure in the cylinder is lower than the pressure in the suction chamber, the reed valve under the valve plate opens and the gases from the suction chamber enter the cylinder. As the piston reaches bottom dead center and starts traveling upward, the suction reed valve closes and the refrigerant vapor in the cylinder is compressed as long as the discharge reed valve remains closed. When the pressure across the discharge valve is great enough to open it, the pressurized refrigerant vapor leaves the cylinder through the discharge chamber and travels through the piping to the condenser.

A side view of the compressor, illustrating the linkage between the motor and the com-

[1] discussed at the end of §1.2.2.

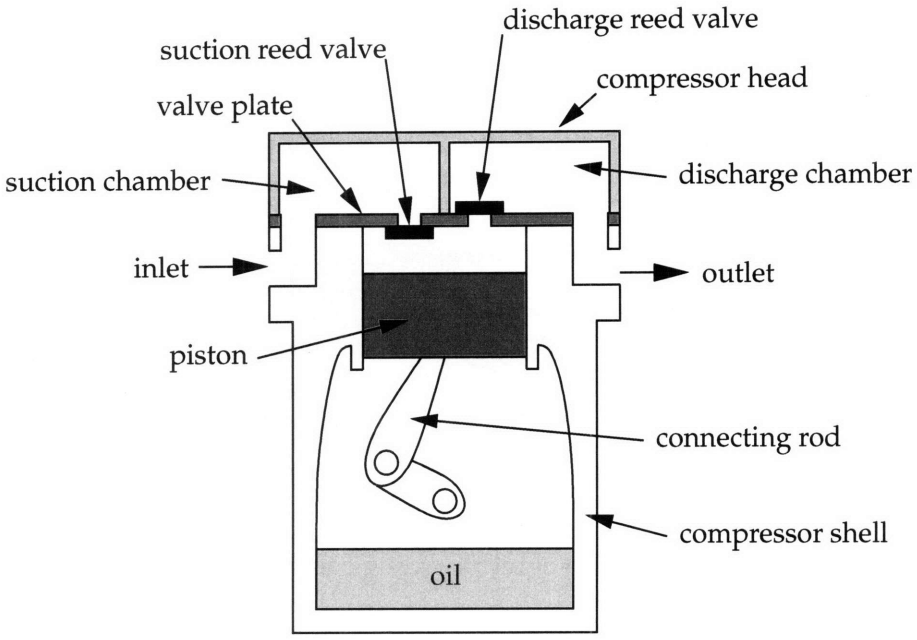


Figure 3-1: End cross-section of compression chamber.

pressor body, is illustrated in Figure 3-2. This diagram emphasizes the flow of oil in the compressor, showing how it flows down the center of the crankshaft and out under the pistons. This picture also shows another, more accurate representation of the discharge valves, in which a set of springs holding the discharge reed valve down until the force applied by the pressure in the cylinder is greater than the force applied by the spring.

Visible in Figure 3-3 are four essential features which will be referred in the following section. One can see the overall configuration of the compressor from a top view, in the same orientation as it is in many of the photos of the compressor itself. The physical orientation of the compressor motor as related to the compressor internals (pistons, crankshaft) can be visualized by comparison with Figure 3-2. The two valves on the left and right hand sides of Figure 3-3 are referred to as the suction and discharge service valves; these service valves are particularly useful because they provide a port for measuring the pressure without interfering with the connection between the refrigerant lines and the compressor. A more detailed picture of the operation of these valves is presented in [160, p. 440]. The other features of note in this picture are the letters **L** and **H** on the top of the compressor head. The bolt heads which are located adjacent to these letters correspond to holes located

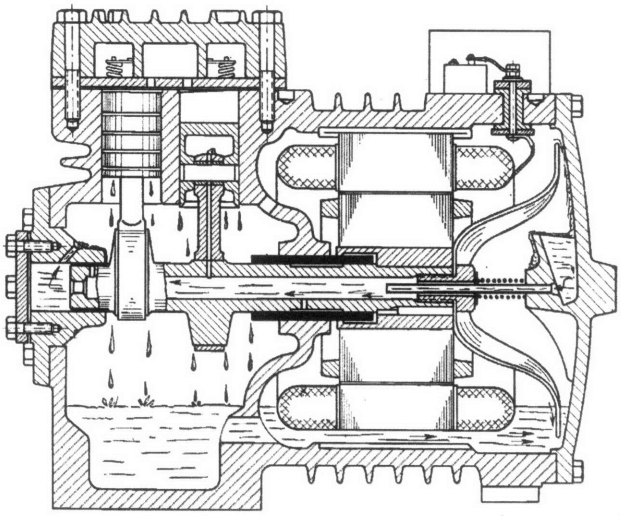


Figure 3-2: Side cross-section of reciprocating compressor, from [14, p. 145].

in the head for the purpose of monitoring the suction and discharge pressure directly inside of the head. This is useful because it can provide a direct assessment of the pressure inside the head, as opposed to the next nearest measurements, which are located on the service valves.

Other photographs of the compressor internals are provided in Figures 3-4 and 3-5 to more clearly illustrate the physical layout of the system. Figure 3-4 illustrates the interface between the valve plate and the compressor head, showing both the suction and discharge plenums in the head and the top of the discharge valves. The holes in the center of the valve plate along the top and bottom edges are where the suction gas enters and the discharge gas exits the compressor. The discharge reed valves are not visible in this picture, as they sit directly on top of the valve plate under the backstops, which are located on top of the valve plate. The next picture, Figure 3-5, shows the interface between the bottom of the valve plate and the top of the compressor. The pistons are clearly visible in this picture, as is the location of the suction valves. The gasket between the valve plate is visible in this photograph, while the gasket that creates a seal between the valve plate and the head was removed from the photograph shown in Figure 3-4 for the sake of visual clarity.

One can get a sense of the type of damage which could be caused by liquid slugging by examining these pictures. Due to the fact that the liquid is largely incompressible under these conditions, the piston must force all of the liquid from the cylinder through the dis-

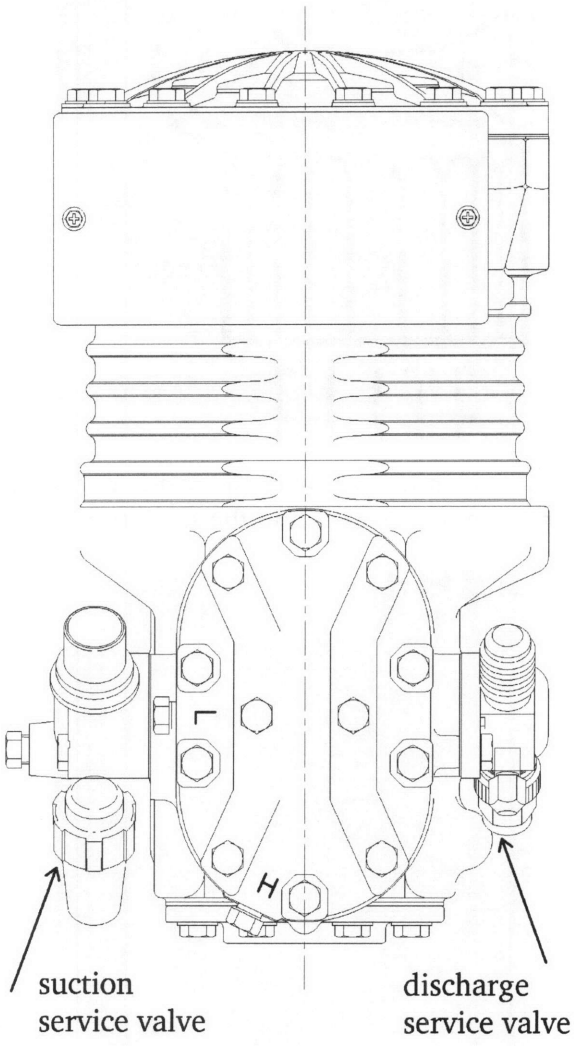


Figure 3-3: Drawing of external compressor appearance.

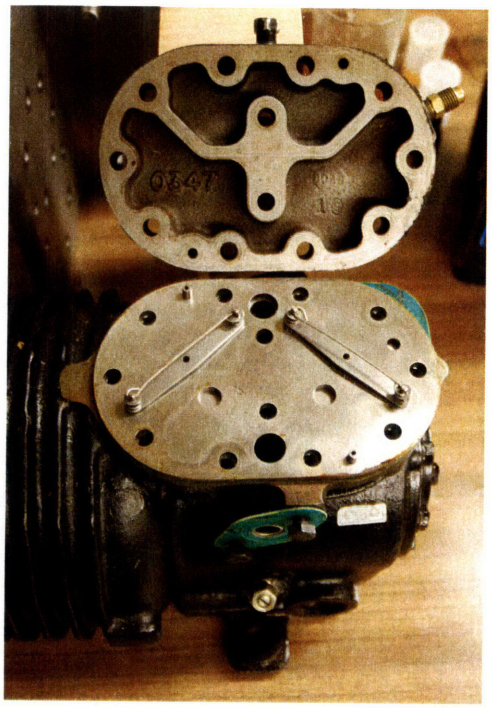


Figure 3-4: View of compressor internals, with the inside of the head and the top of the valve plate illustrating the discharge valves and the suction ports.

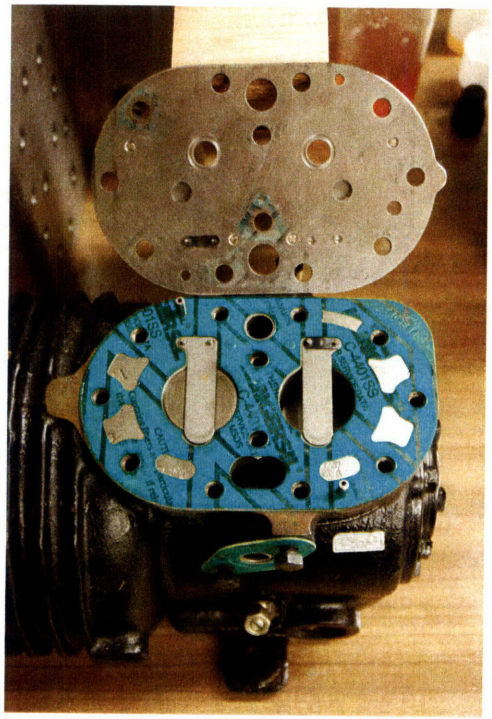


Figure 3-5: View of compressor internals, with the bottom of the valve plate and the suction valves.

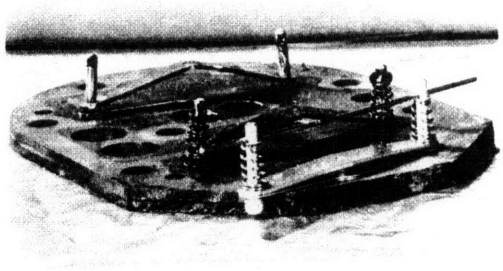


Figure 3-6: Picture of valve plate with damaged discharge valves, from [29, p. 26-25]. A healthy discharge valve is shown at the front of the valve plate.

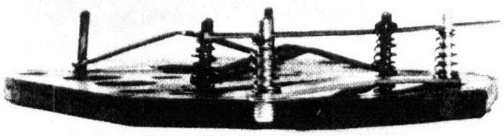


Figure 3-7: Picture of valve plate with damaged discharge valves, from [29, p. 26-25]. A healthy discharge valve is shown at the front of the valve plate.

charge valve. If the forces in this circumstance are sufficiently large, the force of the liquid on the suction valve can distort and destroy the suction reed valve as the liquid attempts to leave through that orifice; alternatively, if the liquid cannot leave through either of the ports, the force on the piston and connecting rod can be large enough to bend or shatter the connecting rod. In many circumstances, the forces applied will merely increase the rate of wear for these components, but the valves and connecting rods can be immediately destroyed if liquid slugs of sufficient size are ingested. Figures 3-6 and 3-7 illustrate the effect that large liquid slugs can have on discharge valves. These pictures are for the same set of valves taken at two different angles; on the two valves attached to the valve plate, one can see that the valve stop is bent at an extreme angle, and that the pin controlling the tension on the reed valve has been snapped. In order to properly understand the extent of the damage, a healthy reed valve has been placed at the front of the valve plate for the purposes of comparison.

Similarly, Figure 3-8 illustrates the effect that extremely large liquid slugging events can have on connecting rods. Under normal circumstances, the connecting rods in this photo would be straight, but the large forces applied to them during the liquid slugging event have severely bent and distorted them. It is important to note that these pictures were taken from a Copeland service manual published in 1970; modern compressors use aluminum connecting rods rather than the steel connecting rods illustrated here, and these connecting rods would shatter instead of deforming in the manner shown in the picture.

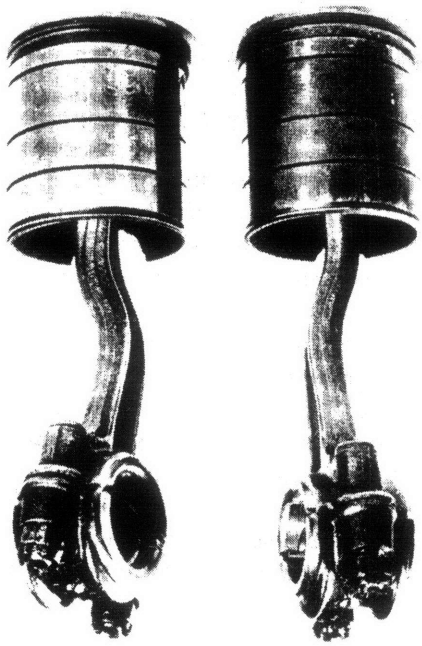


Figure 3-8: Picture of compressor pistons which have been damaged as a result of liquid slugging, from [29, p. 26-29]. The connecting rods should be straight, not bent.

A modicum of research has been performed in detecting liquid slugging in reciprocating compressors. The first extant paper on the slugging phenomenon [133] explores some of the common causes of liquid slugging outlined above, and investigates the slugging phenomenon via the construction of an experimental apparatus qualitatively similar to that which was used in the research conducted in this thesis; unfortunately, the authors do not describe the transducers used to measure the cylinder pressure. The authors find that peak pressures from slugging range from 1500 to 2500 psi for their compressor, with the first audible effects of slugging being observed around 1600 psi. Furthermore, it was found that the peak pressures were only developed after 30 to 60 revolutions, as there is not sufficient inertia before to sustain the high loads causes by the slugging overpressures. The authors also document a number of component failures due to slugging, and note that these failures are attributable to both the overpressures in the cylinder (e.g. snapped discharge valve seat rivet) and to the reduction in lubrication (e.g. failed main crankshaft bearings.)

The next set of liquid slugging-related publications are not found until 1986, with the publication of [136–138]. The authors of these publications developed a test stand similar to that of [133], and included measurements of the cylinder pressure, discharge plenum pressure, an accelerometer on the discharge head, and the measurement of the crankshaft angle. All of these quantities were measured during the starts in which liquid was ingested; they found that there was no substantial difference in the cylinder pressures if the machine was operated at either 1200 rpm or 3600 rpm, and that the maximum cylinder pressure was about 1200 psi for their apparatus, with the peak cylinder pressure gradually decreasing as the liquid is ejected. Two different analytical models for the liquid and vapor configuration in the cylinder (where the liquid is either above or below the vapor) were tested in [137,138], and the compressor behavior was simulated with either the ideal gas law or the real equation of state for the refrigerant R-22. They found that both of the simulations in which the vapor was above the liquid in the cylinder, while characterizing the refrigerant as either an ideal gas or via the equation of state, agreed fairly well with their experimental observations.

In a similar paper, Simpson and Lis [134] conducted an experimental investigation in which they attempted to characterize the effects of liquid slugging. The majority of this

publication focuses on the physical method of measuring the cylinder pressure. After trying a number of different approaches, the authors find that the most reliable approach available consists of mounting a strain gauge under the crankshaft; during a slugging event, the increased forces on the connecting rods are transferred to the crankshaft, where they can be identified by the strain gauge. The authors also investigate the change in the peak slugging pressure due to changing refrigerant charge, finding that these pressures increase dramatically as the system is overcharged. Finally, an empirical method based upon their experimental observations for predicting the peak slugging pressure as a function of the compressor bore, the stroke, and the nominal motor horsepower is determined.

Shiva Prasad [131] investigated similar slugging phenomena more recently in natural gas compressors. He develops a detailed physically based simulation of the cylinder pressure, and simulated results for his compressor (for which he provides parameters, such as cylinder bore and stroke) correlate well with the results published previously; cylinder pressures four to five times the nominal value are predicted with his method. Research has also been performed [119] into developing relief plate mechanisms to relieve the high pressures caused by liquid slugging, although it is difficult to assess the practicality of the relief plate design from this paper, as none of the plots include units.

While these previous investigations into the phenomenon of liquid slugging have focused on reciprocating compressors, slugging can also affect other compressors used in air-conditioning and refrigeration applications, such as scroll compressors, rolling piston compressors, and screw compressors. Liu and Soedel [96, 97], in their theoretical study of two-phase compression processes, have demonstrated that reciprocating compressors are particularly susceptible to liquid slugging because of the steepness of the volume compression gradient. This can be seen by considering the fact that while scroll compressors typically compress the vapor refrigerant in 540° of rotation and rolling piston compressors compress the refrigerant in 360° of rotation, reciprocating compressors complete their entire compression stroke as the piston travels from bottom dead center to top dead center, or in 180°. The rapidity of this compression process is what causes the overpressures to be extreme. However, since the displacement of a scroll compressor is smaller than that of a reciprocating compressor, a smaller amount of liquid is needed to cause a substantial fault in the scroll compressor motor.

Because of the reduced susceptibility to liquid slugging, one of the tangential areas of compressor research relating to this area is the study of liquid refrigerant injection into rotary and scroll compressors [2,13,44,162]. By injecting a small volume of liquid into the compression cylinder, some of the thermodynamic inefficiency due to the superheating of the suction gas can be minimized while little risk to the mechanical health of the compression apparatus is accrued because of the types of compressor used. These papers do not address the question of liquid slugging, and assume that the volume of liquid injected can be tightly controlled; nevertheless, this appears to be a very effective way to increase the compressor's operating efficiency.

One of the notable observations that might be made after surveying this range of publications is that direct measurements of liquid slugging are relatively difficult to obtain; so much so that one paper is entirely devoted to the issue of accurately measuring the cylinder during liquid slugging events. It is also interesting to note that, while there is much circumstantial and indirect evidence that liquid slugging is responsible for many compressor failures in the field, there are no concrete figures illustrating the relative incidence of liquid slugging in surveys of field-installed equipment. Part of the reason for this is surely that compressor manufacturers regard such data as proprietary and are unlikely to share it with other sources, but this is probably also due to the difficulty of directly identifying the occurrence of liquid slugging events.

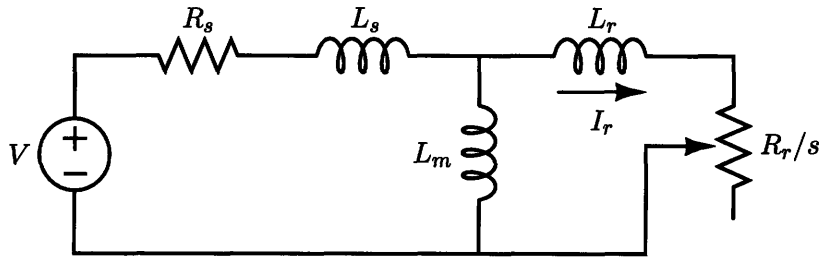
Motivated both by the difficulty of identifying liquid slugging in field-installed equipment and the apparent prevalence of liquid slugging-induced compressor failures, an alternative approach to fault identification can be developed by using the observation that the increased forces required to eject the liquid from the cylinder during a slugging event will cause the load on the motor to temporarily increase for the period of time that the liquid is present in the cylinder. Since the mechanical power provided by the motor is related to the electrical power entering the motor terminals, it should therefore be possible to identify liquid slugging by analyzing the stator currents for the compressor motor.

Preliminary research based upon this observation was conducted by Armstrong, et al [8] and is more fully explored in his doctoral thesis [6]. He conducted a set of experiments in which small quantities of liquid refrigerant (R-30) were directly injected into the suction port either while the compressor was either in transient or steady-state operation.

To determine whether or not the change in performance could be identified via electrical observations, he then analyzed filtered versions of the compressor current collected during the same interval of time, and found that the filtered current waveforms during these slugging events did deviate measurably from the comparable currents measured without any injected liquid.

As the results collected in [6] were very encouraging, the present research was undertaken to further refine and develop a fault diagnostic method with increased sensitivity to liquid slugging, and to characterize the performance of such a method on a compressor installed in a complete air-conditioning unit. The objective of this research is therefore to identify the presence of liquid slugging in the compressor by only analyzing the electrical terminal variables of the motor; to validate the performance of this fault detection method, however, it will also be necessary to install a suite of mechanical instrumentation so that the predictions made by the electrical method can be independently verified. It is also essential that the detection methods for both transient and steady-state slugging be evaluated to fully characterize slugging in compressors.

This research is necessarily experimentally oriented, as the main goal of this research is the development of a fault detection system that will work on field-installed equipment. The construction of an experimental apparatus which can simulate a variety of conditions found in the field, while simultaneously allowing the user to control the relevant variables affecting liquid slugging, is therefore a major focus of this research. This experimental apparatus has gone through a number of iterations, some of which are briefly described in [80,81]. As no other extant publications describe such an apparatus, it will be explored in some detail, in order to record one possible approach to the acquisition of these measurements, as well as the resulting data. As the thermodynamic mechanism of liquid slugging has been elaborated in previous publications [97, 131, 136, 137], the reader is referred to those for more information. In comparison, the methods of preprocessing the electrical observations will be extensively discussed in the following section, since they have developed in this research.



3.2 Structure of the Liquid Slugging Diagnostic Method

In developing the preprocessing method for the electrical measurements, it is useful to examine the effects of liquid slugging on the electromechanical dynamics of the motor via a circuit model of the motor. Unlike the development of the fan diagnostic method, this circuit model will not be used directly to identify the occurrence of slugging, but it will rather be used as a motivation for the method and to describe a variety of the phenomena which are observed. One particularly useful electrical model of the induction motor, given in Chapter 2 and repeated here for convenience, is illustrated schematically in Figure 3.2. The mechanical power, as opposed to the torque, produced by the motor is given by

$$P_{mech} = n_{ph} I_r^2 R_r \left(\frac{1-s}{s} \right), \tag{3.1}$$

where s is the value of the fractional slip of the motor and the other electrical variables are defined in Chapter 2.

One can see from this expression that the mechanical power developed by the motor is related to the rotor current, and therefore to the stator current, as well as to the fractional slip of the machine. Since, under most operating conditions, the voltage distribution system is relatively stiff (low effective source resistance), the stator currents and the slip during a liquid slugging event will change as the speed of the rotor changes when the compressor ejects the liquid. It is also useful to consider the system dynamics for a moment, to ensure that the motor will continue operating if the speed changes; the representative torque-speed illustrated in Figure 3-9 shows that, for a fractional slip up to approximately 0.3, the torque increases as the slip increases. In the context of liquid slugging, a reduction

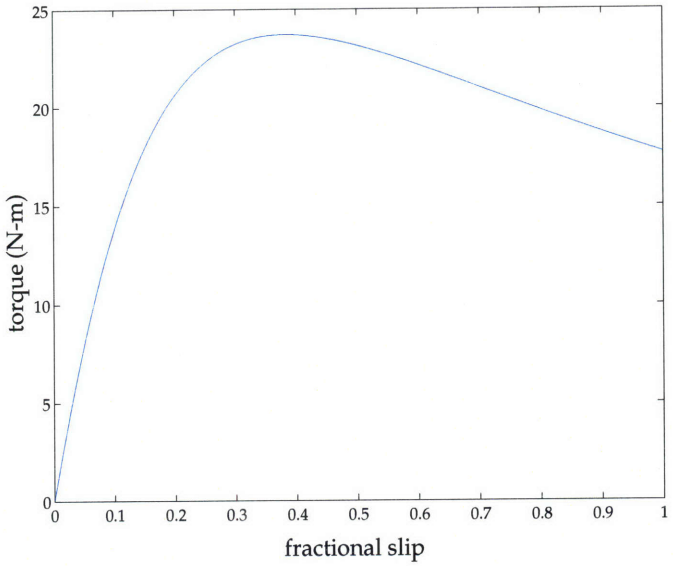


Figure 3-9: Representative torque-speed curve for a compressor motor.

in speed as the piston hits a slug of liquid will cause the torque developed by the motor to increase proportionately for that period of time, and as the slip returns to the normal value, the torque developed will also return to a normal value.

In considering the effects of slugging on the electrical terminal variables, it important to note that this diagnostic method is designed to identify elevated peak pressures in the cylinder, rather than solely detect the presence of liquid refrigerant. Since the damage of concern is caused by overpressures in the cylinder, rather than requiring liquid refrigerant specifically, this diagnostic method can successfully identify the fault condition of interest. In order to distinguish between liquid migration and foaming oil, oil pressure measurements would also be necessary, but this would be relatively straightforward to implement if diagnostic requirements necessitated the acquisition of such information.

While it is clear from the preceding discussion that liquid slugging events will cause changes in both the stator currents and the slip, there are also other phenomena which can cause variations in either the mechanical power or the motor currents. In order to properly develop the fault diagnostic method, it is essential that these phenomena be identified and characterized, so that their effect on the liquid slugging diagnostic metric can either be minimized or characterized. By performing this step, it will be possible to ensure that other faults are not misdiagnosed as liquid slugging faults, or the complement of this sce-

nario, in which the fault diagnostic system raises a false alarm for liquid slugging when the system is changing in some other way which may or many not be attributable to a fault. Through consideration of these factors, the fault diagnostic technique can be improved by developing the proper preprocessing technique and by simultaneously testing for other faults so that the fault diagnostic system can identify liquid slugging as distinct from any other changes in operating condition. Some of the phenomena which may cause such variations in either the mechanical power or the winding currents are described in the following list.

Utility transmission frequency: While one might not otherwise consider the 60 Hz line frequency as a source of variation due to the common practice of thinking about voltages and currents, changes in the motor currents will necessarily be modulated on top of this carrier frequency. This fault detection method must therefore effectively demodulate the variations in the stator currents caused by liquid slugging phenomena from the 60 Hz transmission frequency.

Initial utility angle: The transient current in each motor winding is dependent upon the initial phase angle of voltage driving the winding when the winding is first connected to the utility, making it necessary for a fault diagnostic metric to compensate for the changes in the stator currents which occur due to this variation.

Motor winding temperature: As the compressor motor operates, the resistance of the stator and rotor windings will change as they heat up. This resistive heating effect, which happens very gradually, will cause the motor currents to change for a given power supplied to the mechanical load. Even more pertinently, the change in the winding resistances can potentially have a large impact on the shape of the initial starting transient, which will in turn affect the transient slugging diagnostic metric. The temperature of the windings must therefore be the same when characterizing the slugging metric, in order to separate the effect of slugging from the effect of temperature.

Initial piston position: The load on the motor will depend in upon the angle of the crankshaft when the motor is started, especially when considering the liquid slugging problem.

If the cylinder into which the liquid is injected has its piston located at top dead center (TDC), no liquid will be able to collect in the piston, and no slugging will be observed. Similarly, the load on the motor may differ if the two pistons are at the same height, as opposed to one being at the top of its stroke, while the other piston is at the bottom of its stroke. Characterization of the transient liquid slugging fault diagnostic metric thus requires that the initial starting position of the piston be the same.

Initial suction and discharge pressure: The load on the compressor motor is closely tied to the pressures on either side of the valves; as the pressure gradient increases, the torque developed by the motor must increase proportionally. In order to characterize the performance of the liquid slugging fault diagnostic metric, it is therefore necessary to characterize the motor behavior when the suction and discharge pressures are nearly identical.

Motor faults: Induction motors can sustain a variety of faults over time, such as broken rotor bars, shorted windings, and faulty bearings. Many of these faults predominantly affect one of the windings of the motor, and the unbalanced nature of these faults will make them easy to distinguish from liquid slugging, but the manifestation of other faults may be similar to that of liquid slugging, making it necessary to characterize these faults using diagnostic features which differ from those used for liquid slugging.

Other unmodeled phenomena: While this list covers many of the phenomena which will affect the compressor in normal operation, it is important to consider when developing an FDD method that other unforeseen behavior might arise due to off-design or faulty operation which has not been incorporated into the design of the fault detection method, either due to the difficulty of simulation or to their unexpected nature. While these cannot be explicitly tested for by the fault detection method, it is essential to consider their presence when encountering otherwise unexpected system behavior.

In considering the above list of phenomena which will affect either the load on the

compressor motor or the currents in the motor, two objectives in the design of the preprocessing method become apparent. The first of these objectives is that the FDD method should preprocess the electrical measurements so that the effect of the time-varying components of the electrical signals that are not related to the slugging fault are minimized. The second objective is that the preprocessing method would ideally relate the electrical variables as closely as possible to the torque and speed, as the physically expected changes in the behavior of those variables will be most closely related to the power of the motor. This will increase the sensitivity to changes in the load on the motor, and decrease the sensitivity to other ancillary changes which are not directly related to the torques created by liquid slugging. We will examine the sensitivity of the method to these mechanical changes, such as different initial piston positions, by careful experimentation, and then use the information gleaned from these experiments to assess the effectiveness and sensitivity of this method as it might be implemented in the field to the potentially wide range of sources of mechanical variation which might be found. As the effect of the mechanical variations will be discussed in the next section of this chapter, which is devoted to the particular issues involved in conducting the experimental investigations, the remainder of this section will focus on mitigating and minimizing the electrical variations of the signal.

One of the other techniques which can be used to distinguish between the liquid slugging fault diagnostic metric and other variations is that of timescale; many of the other phenomena occur on a very different timescale than liquid slugging and can thus be distinguished from this fault. For example, transient liquid slugs will typically occur over a period of time ranging from a fraction of a second to a few seconds, while variations in the winding impedance due to thermal phenomena will occur over a period of many minutes or hours.

The preprocessing method used to identify the effect of liquid slugging involves applying a linear transformation with time-varying coefficients, commonly referred to as the Park transformation, to the set of observed motor currents. This transformation is useful in that it eliminates the effect of the 60 Hz component of the currents and is invariant to the initial phase angle of the wall, and the output of the method is related to the torque developed by the motor. It has been used for decades in the analysis and design of induction machines, and was briefly discussed in Chapter 2. The equation describing the

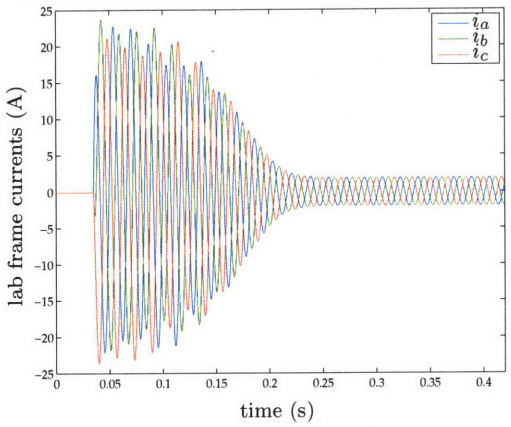


Figure 3-10: Lab-frame currents.

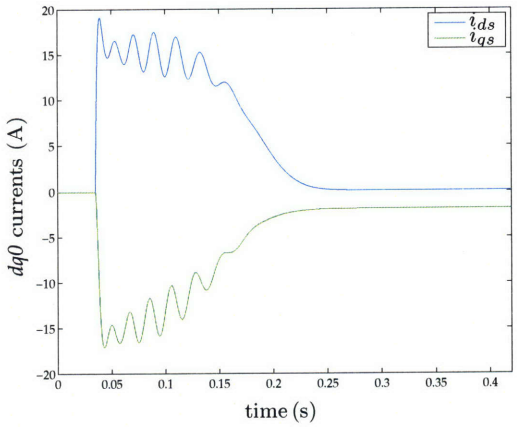


Figure 3-11: Stator currents in stator reference frame.

transformation is repeated again here, for convenience:

$$\begin{bmatrix} i_{ds} \\ i_{qs} \\ i_0 \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin(\theta) & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix} \tag{3.2}$$

or, written in a more concise form, as

$$\mathbf{i_{dq0}} = \mathbf{T}(\theta)\mathbf{i_{abc}}. \tag{3.3}$$

This transformation was mainly used in Chapter 2 because of the simplifications which result from the transformation of the model of the induction machine. In this context of the liquid slugging diagnostics, however, a different set of effects from the transformation is useful. Figures 3-10 and 3-11 illustrate the effect of applying the Park transformation to stator winding currents during the period in which the motor is starting up. One can see that the bulk of the 60 Hz component of the motor current has been eliminated, leaving only the variation in d-axis and q-axis currents which is present while the fields in the machine are established. Of particular note is the fact that these currents have no time-varying components once the startup transient has subsided; this will enable a fault detection method to identify small changes in these steady-state currents caused by steady-state slugging.

This transformation is also useful because the variation in the d- and q-axis current waveforms due to the initial phase angle of the electric utility is minimized. By using the estimate of the electrical angle of the voltage as the argument θ of the transformation, the effect of the transformation on the observed phase currents will produce identical results regardless of the starting angle. This use of the transformation differs slightly from its application in Chapter 2, in that the angle estimates are changed with time according to the variation of the utility's electrical angle $\theta_e = \omega_e t + \phi_e(t)$. While the previous application of the Park transformation made the assumption $\phi_e(t) = K$, where K is a fixed constant, the output of the transformation will slowly vary as $\phi_e(t)$ varies. By updating the estimate of $\theta_e(t)$ with the new estimates of $\phi_e(t)$, the transformation will eliminate variation in the startup waveforms which would otherwise be caused by the changes in the electrical angle.

The last useful characteristic of this preprocessing method is that it is directly related to the torque developed by the compressor motor. As discussed in [79], the torque developed by the induction motor is given by

$$\tau_e = \frac{3}{2} p(\lambda_{ds} i_{qs} - \lambda_{qs} i_{ds}) \tag{3.4}$$

where τ is the torque developed by the motor, p is the number of poles pairs of the motor, and λ_{ds} and λ_{qs} are the stator d- and q-axis fluxes. This equation makes the relationship between i_{ds}, i_{qs} and τ_e explicit, in that the torque is composed of components which are proportional to each of these d- and q-axis currents. Since liquid slugging is expected to affect the torque developed by the motor, changes in τ_e will be reflected in i_{ds} and i_{qs}, making the analysis of these signals for liquid slugging fault detection a natural approach.

The implementation of the Park transformation as a preprocessor for the set of measured currents requires two consecutive steps. The first of these is required because the data acquisition system samples the inputs sequentially, rather than simultaneously. Since this transformation requires that the three stator currents be synchronized with the set of stator voltages, the data acquisition device must collect six channels of data. Rather than sampling all of these channels at the same time instant t, the delay T_d between sampling consecutive channels causes the underlying current and voltage signals to be sampled ac-

cording to

$$V_{ab}[k] = V_{ab}(kT_s) \tag{3.5}$$

$$I_a[k] = I_a(kT_s + T_d) \tag{3.6}$$

$$V_{bc}[k] = V_{bc}(kT_s + 2T_d) \tag{3.7}$$

$$I_b[k] = I_b(kT_s + 3T_d) \tag{3.8}$$

$$V_{ca}[k] = V_{ca}(kT_s + 4T_d) \tag{3.9}$$

$$I_c[k] = I_c(kT_s + 5T_d), \tag{3.10}$$

where k is the sample index. Were these delays to be neglected, a phase error would be introduced when multiplying the phase currents with the components of the Park transform matrix $\mathbf{T}(\theta)$, where $\theta = \omega_e k T_s + \phi$. This error was eliminated by interpolating all of the sets of observations using a spline interpolation algorithm provided in Matlab (`interp1`) so that they were all effectively sampled at the same time instant kT_s.

Since the compressor motor was wound in a delta configuration, it was also necessary to convert the line-to-line stator voltages V_{ab}, V_{bc}, and V_{ca} to the equivalent line-to-neutral voltages V_{an}, V_{bn}, and V_{cn} which would have been driving an equivalent wye-wound motor. This was performed in the same manner as described for the fan motor in Chapter 2.

The second step required to implementing the Park transform for the set of observed data is that the parameters of the utility voltage must be estimated. A simple model of the 3-phase balanced set of voltages depends on 3 parameters, V, ω_e, and ϕ, where the set can be described by

$$V_{an} = V\cos(\omega_e t + \phi) \tag{3.11}$$

$$V_{bn} = V\cos(\omega_e t - 2\pi/3 + \phi) \tag{3.12}$$

$$V_{cn} = V\cos(\omega_e t + 2\pi/3 + \phi). \tag{3.13}$$

These expressions are obviously very simple and do not capture the full harmonic content of the signal; for example, the prevalence of switching power supplies and other loads means that the utility voltage also has a a substantial third-harmonic component, which

is not captured by this model. Due to such issues, one of the challenges of this parameter estimation problem is that the parameters which describe a portion of the set (e.g. a few line cycles) may not describe the set of observations in a best-fit sense. Estimates of the parameters that are obtained from a subset of these observations may therefore contain some bias which will change the behavior of the d- and q-axis currents and complicate the fault detection process. Unfortunately, the computational expense of finding the parameters which describe the entire dataset in a best-fit sense may be prohibitively high, as a dataset which is even a few minutes long will require a sufficiently large amount of storage that other methods which do not have such storage requirements may be greatly preferable.

A similar approach to that used in Chapter 2 was also employed to solve this problem, though the particular application is somewhat different. As illustrated in §2.4.3, the residual between a set of observations of a sine wave $\hat{y}$ and a model with an unknown parameter ω_e has a large number of local minima, which can make a gradient-based minimization method, such as the Gauss-Newton or Levenberg-Marquardt techniques, converge to a parameter which is far away from the global minimum.

The means by which the time varying parameters of the utility voltage were estimated is briefly described as follows. In general, the residual for a parameter identification problem can be written

$$r_k = \hat{y}(\mu, t_k) - y_{obs}(t_k), \tag{3.14}$$

where $\hat{y}$ represents the output of a model for the system with parameters μ, and y_{obs} represents the set of observations of the system output. For many types of system identification problems, such as those based upon sinusoids or sums of sinusoids, the residual is often nearly linear in the parameters for a small subset of observations [123,128]. One technique for solving these nonlinear least squares problems that uses this observation identifies the number of datapoints for which the residual is expected to be linear, and then perform the minimization over this subset of datapoints. By solving a series of minimization problems in which the size of this subset of observations is gradually increased, the convergence of nonlinear least squares is greatly improved.

For example, suppose that one had a set of observations of a sinusoid generated by $y_{obs} = \cos(2\pi t_k)$, and that the model for this signal is $\cos(\mu t_k)$. A cursory examination

of the data would show that the data was nearly linear for $t_k < 1/8$, so the procedure would first identify the number of samples K for which $t_k < 1/8$. Once this length K of the subset was identified, the nonlinear least squares problem would be solved using only the first K datapoints of y_{obs}, thus taking advantage of the fact that the residual of the problem is nearly linear in this region, ensuring that no local minima are present. Once the final parameter estimate $\mu^{(1)}$ for this subproblem was obtained, it would then be used as the initial guess to solve the next subproblem, in which the first $2K$ datapoints of y_{obs} are used to find the next parameter estimate $\mu^{(2)}$. This process would continue until the number of datapoints over which the minimization took place was equal to the number of datapoints in y_{obs}, and the parameter estimate $\mu^{\lceil (N/K) \rceil}$, which is calculated for the length of the dataset, would represent the final least-squares estimate for the problem.

The advantages of this method can be seen by considering the loss function described in §2.4.3. The normalized residual $r(\hat{\mu})/N$ to be minimized in this case is

$$\frac{r(\hat{\mu})}{N} = \frac{1}{N}\sum_{k=1}^{N}(\sin(\hat{\mu}kT_s) - \sin(\mu kT_s))^2 \tag{3.15}$$

where T_s is the sample period for the waveform, k is the sample index which runs from 1 to N, the true parameter of the system is μ, and the present estimate of the parameter is $\hat{\mu}$. Figure 2-14 illustrates this loss function (unscaled by the number of points N) when the number of samples of the sine wave is relatively large, so that many periods of the sine wave are represented in the data. By using the algorithm described above to find the estimate $\hat{\mu}$ by solving the series of minimization problems in which the number of samples of the observed dataset K is slowly increased from a small number to the length of the dataset N, the susceptibility of nonlinear least squares to local minima is greatly reduced. This process can be visualized with the help of Figure 3-12, which illustrates the family of residuals for this problem where the parameters for the system are $\mu = 25$, the range of initial guesses $\hat{\mu}$ ranges from 10 to 40, $T_s = 0.05$, and $N = 601$. The benefit of gradually increasing the number of samples K over which the residual is formed is apparent, as the loss function is globally convex for $K < 10$ samples, so that the estimate $\hat{\mu}$ will be very close to μ over this range. By using this estimate of $\hat{\mu}$ as an initial guess to generate the next parameter estimate $\hat{\mu}$, it is possible to quickly find the global minimum of this loss

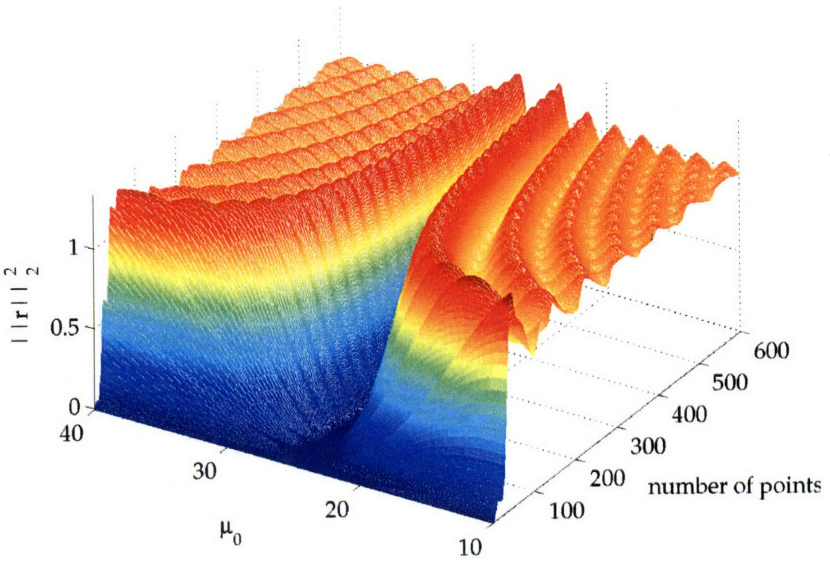


Figure 3-12: Loss function showing the effect on $r(\hat{\mu})$ of varying both the initial guess $\hat{\mu}$ and the number of points K used in forming the residual.

function (around ($\hat{\mu} = 25$) , and then stay in the region of this global minimum as the number of points is increased until it reaches N.

While this approach solving the nonlinear least squares problem has proven to be very effective, it necessarily takes a long time to converge if the region over which the problem is linear is very small in comparison to the size of the dataset. An additional observation for this problem which points to a means of improving the method's performance in this regard is that one would not expect the estimate of μ to change appreciably for $t_k > 1$, and the method could presumably move very quickly through the remainder of the dataset in the absence of measurement noise. This observation can be integrated into the method by supposing that a Taylor series exists for the model, so that the residual can be rewritten

$$r_k = \overbrace{\left(\hat{y}(\mu,0) + \frac{d}{dt}\hat{y}(\mu,0)t_k + \frac{d^2}{dt^2}\hat{y}(\mu,0)t_k^2 + \cdots\right)}^{\hat{y}(\mu,t_k)} - \overbrace{\left(a + bt_k + ct_k^2 + \cdots\right)}^{y_{obs}(t_k)}. \tag{3.16}$$

Since many problems which can be written in the above form are dominated by their DC- and first-order coefficients, it is possible to control the size of the dataset by analyzing the output of the Taylor series expansion of the residual r. The method can set the size of

the increment on the dataset by comparing the coefficient of the second-order term to an established threshold, thereby ensuring that the method will converge. The method for estimating the parameters therefore is as follows:

Method

1. Start with initial guess $\mu^{(0)}$ and the error function $\hat{y}(\mu, t_k)$; set $i = 1$ and $K^{(0)} = 1$.
2. Determine $K^{(i)}$ by finding the largest number of datapoints in the set $[K^{(i-1)}, K^{(i)}]$ for which the linear term of the residual dominates over higher-order terms by fitting a power series to the residual in the interval.
3. Find $\mu^{(i)}$ as the solution to the nonlinear least-squares problem minimizing $r_k^T r_k$ over the interval $[1, K^{(i)}]$ using the initial guess $\mu^{(i-1)}$.
4. If $K^{(i)}$ is less than the number of datapoints in y_{obs}, increment i and loop back to step 2.
5. If $K^{(i)}$ is greater or equal to the number of datapoints in the whole dataset, solve the nonlinear least squares problem over the whole dataset, using $\mu^{(i)}$ as an initial guess.

By using the Taylor series expansion of the residual, the time that the method takes to converge can be improved markedly without sacrificing the ability of the method to avoid local minima. More details regarding the development and application of this method can be found in [126].

While two parameters of the system, V and ω_e, are fixed, the parameter estimation process is further complicated by the fact that different loads turning on and off on the same electrical service effectively cause the phase ϕ to vary, so that this term must instead be expressed as $\phi(t)$. The parameter estimation method must therefore be able to identify this time-varying parameter, or an equivalent parameter which describes the variation caused by $\phi(t)$ in a best-fit sense. This time-variation was accommodated by updating the estimate of the phase after every cycle. The updates of the phase could have been made more frequently, but $\phi(t)$ changes sufficiently slowly that this update frequency adequately tracked the changes over time. These modifications to the standard Levenberg-Marquardt algorithm and their implementation in these experiments are further discussed in §3.4.1.

3.3 Experimental Platform

In developing any fault detection method, it is necessary to either theoretically or experimentally isolate the behavior of the system in response to the relevant fault from other system behavior in order to understand the fault's impact on the rest of the system. One of the particular challenges to studying faults in complex systems is that as the number of constraints on the system increase to better understand and characterize the fault's behavior, the ability of the system to represent typical systems as they are found in the field diminishes. For example, a purely theoretical study of liquid slugging behavior in the compressor would be quite useful in illuminating the subtle aspects of slugging as it affects the simulation, but the results of the simulation might not reflect noise and variations found in the system in an experimental setting. Due to the objective of characterizing the behavior of the fault detection method to some of the experimental variations in the system behavior, the performance of the fault detection method was characterized and refined experimentally, rather than theoretically. The process of inducing and measuring the faults also had the effect of operating the system in ways that might not be representative of common conditions found in the field, but the impacts arising from the particular implementation of the experimental system can be partially mitigated by careful consideration of experimental protocol.

The standard air-conditioning system had to be outfitted with a number of different transducers to detect and validate the presence of liquid slugging. A set of electrical sensors was installed to measure the electrical terminal variables so that the liquid slugging fault detection method could be tested and evaluated, and a variety of additional mechanical instrumentation, both for inducing the liquid slugging fault and for measuring the effect of the mechanical fault on the motor speed and the pressures at different points in the system, were also installed. Control and data acquisition systems were also necessary to operate the system in an experimentally repeatable manner. These systems will be discussed extensively in the following sections.

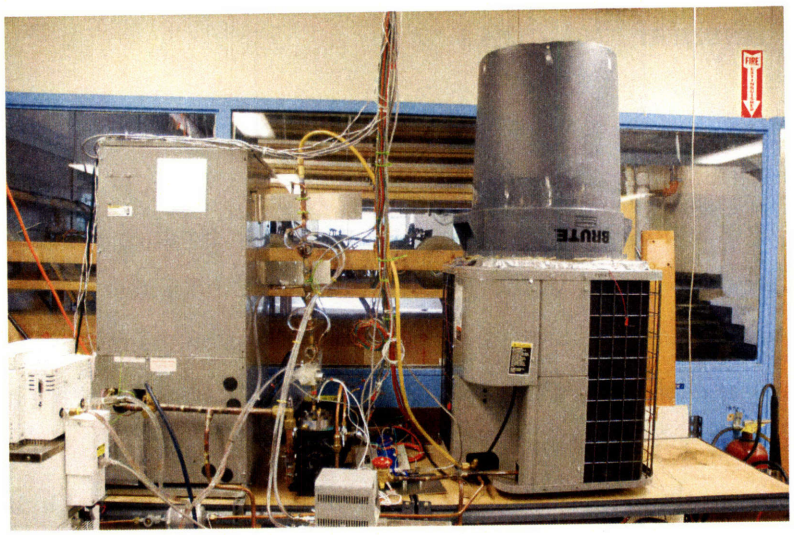


Figure 3-13: Picture of the complete experimental air-conditioning apparatus.

3.3.1 Mechanical Systems and Sensors

The fault detection method for identifying the presence of liquid slugging was tested on an experimental air-conditioning platform which was constructed in the Laboratory for Electromagnetic and Electronic Systems at MIT, a picture of which is shown in Figure 3-13. This unit, built from components manufactured by International Comfort Products, was constructed with components purchased from a local HVAC supplier. The nominal cooling capacity of this air conditioner is 3 tons, or 36,000 BTU/hr, which is comparable to such air-conditioners as might be installed in a home or small business. The condenser, compressor, and evaporator were purchased separately, and assembled in the lab after the requisite EPA certification was obtained so that the air-conditioning system could be evacuated and refilled with refrigerant without requiring the assistance of a professional HVAC service technician. The unit was shipped with 6 pounds of refrigerant, which was evacuated in the laboratory and then recharged into the unit after assembly. For the sake of clarity, the schematic diagram illustrated in §1.2.1 is reproduced in Figure 3-14. The schematic diagram is arranged in much the same way as the equipment shown in the photo; the evaporator is located inside the air handling unit on the left-hand side of the photo, the condenser is located on the right-hand side of the photo, and the compressor is located in the middle of the photo, between the two heat exchangers. It is important

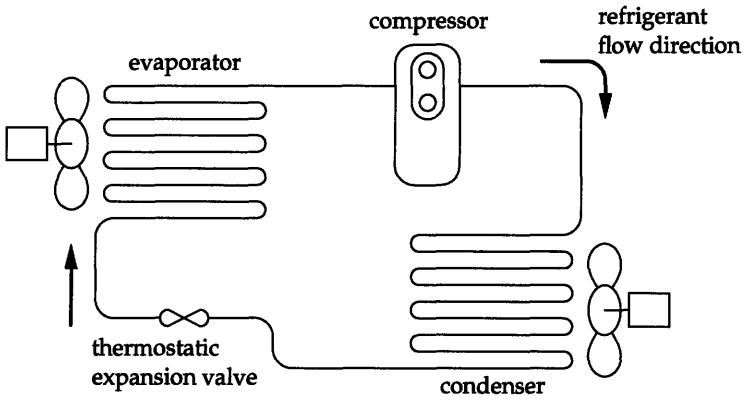


Figure 3-14: Schematic diagram of the air-conditioner.

to note that much of the apparatus seen in the picture (Figure 3-13) above the compressor is associated with the experimental apparatus required to inject liquid refrigerant into the compressor.

The compressor used was a 3/4 horsepower semi-hermetic compressor manufactured by Copeland Corp, part no. KAMA-007A-TAC-800. This compressor has two cylinders, each with a bore size of 1.37 inches and a stroke of 0.93 inches, so that the maximum volume of each compression chamber is 1.37 cubic inches, or 22.48 cubic centimeters. This type of compressor was used because it was well suited for the installation of the variety of instrumentation needed to simulate liquid slugging faults and to measure the effect of those faults. Other types of reciprocating compressors, such as the hermetic or open-frame types, could have been used; semi-hermetic reciprocating compressors were a natural choice because they are fairly common in many air-conditioning and refrigeration applications, and the fact that the head and the valve plate can be easily changed and modified made them a natural choice for experimentation.

The first set of modifications needed to study liquid slugging which were implemented involved the construction of an apparatus to inject liquid refrigerant into the cylinder. This apparatus went through four major iterations over the course of this research. Each iteration of the apparatus incorporated a set of experimental observations regarding the performance and the shortcomings of the previous slugging apparatus; each of these versions will be discussed in this section in order to illustrate aspects of the behavior of the unit which had to be considered. In order to describe these modifications most clearly, a few

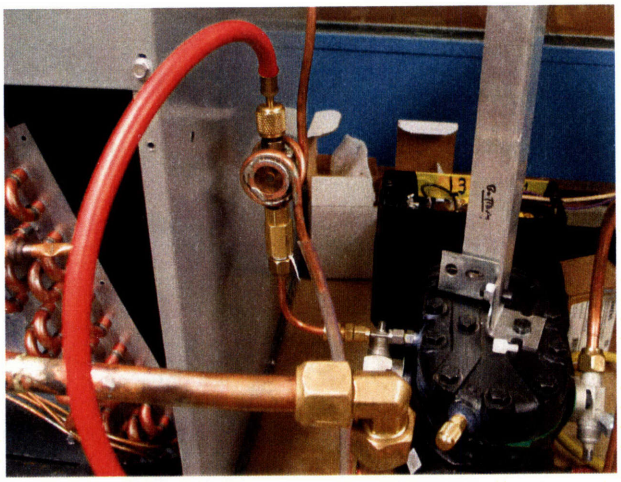


Figure 3-15: First liquid slugging injection apparatus.

pictures are reproduced below which will help the reader in picturing the external connection between the injection apparatus and the compressor.

One of the main factors considered in constructing the first version of injection apparatus was the fact that replacement compressor heads cannot be purchased separately from the compressor; since a new compressor head is only available when attached to a new compressor. Due to the high cost of compressors, permanent modifications to the compressor head were initially avoided to ensure that no mistake would require the purchase of a new compressor to continue with this research. The liquid refrigerant was therefore injected into the side of the head via a 1/8 inch o.d. stainless tube which was plumbed into the supply of liquid refrigerant. During a liquid injection event, the tube would thus deposit the supply of liquid directly over the suction reed valve of the rear compressor cylinder, allowing gravity to drain the majority of the liquid into the cylinder. Photos of this apparatus, as well as its configuration inside of a representative compressor head, are visible in Figures 3-15 and 3-16.

A second design consideration for the initial liquid injection apparatus entailed maintaining a constant mass of refrigerant in the system, as the continual injection of new refrigerant into the system could gradually change the behavior of the system as the mass of refrigerant increases. This potential effect was mitigated by drawing the liquid to be injected from the liquid line, located between the condenser and the TXV. During normal operation of the air-conditioning system, all of the refrigerant will have condensed

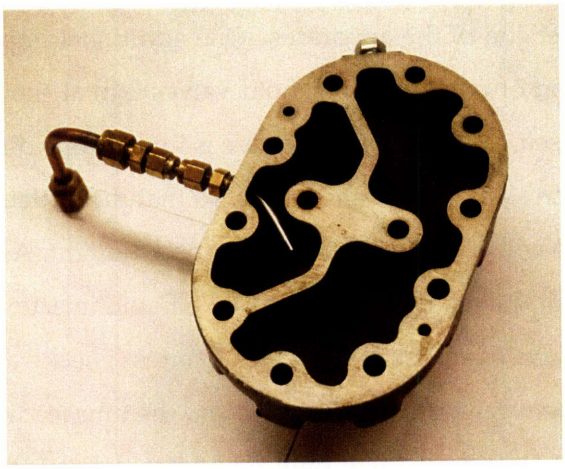


Figure 3-16: Illustration of the injection tube for the first apparatus.

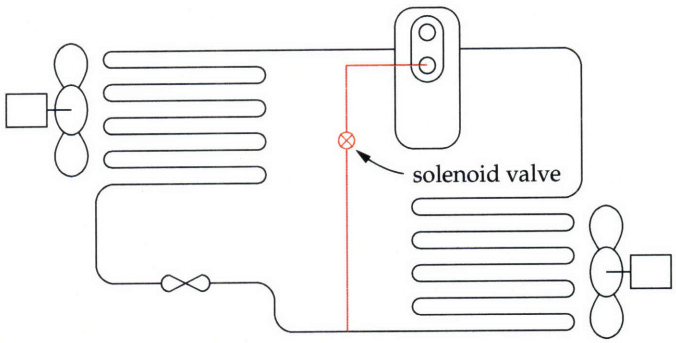


Figure 3-17: Diagram of the experimental RTU platform with the modifications to the refrigerant piping. The red line indicates the added plumbing and solenoid.

upon reaching the bottom of the condenser, making this a particularly appropriate place to obtain liquid for the purposes of injecting into the cylinder. Other research into liquid slugging [133] has also used this approach. This modification of the refrigerant loop is illustrated schematically in Figure 3-17.

A solenoid valve was also installed in the liquid injection apparatus, so that the duration of the liquid injection events could be regulated to within about 10 ms. This valve was installed between the liquid line and the tube leading into the compressor head. A few types of valves were tested as part of the setup, and it was found that valves which were driven with 120 VAC appeared to open most quickly and reliably. A Danfoss 1/4 inch solenoid valve (part no. 032F1155-00-EVR-3) with a 120VAC coil (part no. 018F7692) was

installed in the final version of the apparatus. One additional important fact to consider regarding the operation of refrigerant solenoid valves is that they are only designed to function when the pressure drop across the valve is in the same direction as the intended flow; if the pressure across the valve is reversed, so that the outlet is at a higher pressure than the inlet, the valve will open up with nary a second thought. As this situation presents substantial problems for the reliable operation of the liquid injection apparatus, as well as the operation of the compressor itself, a 1/4 inch Danfoss check valve (part no. 020-1040-NRV-6) was installed between the solenoid valve and the tube leading into the compressor head, to ensure that the refrigerant will only flow from the condenser into the compressor, and not the opposite direction. A control system was also developed to regulate the amount of time that the solenoid valve was opened, and thereby regulate the quantity of refrigerant injected into the compressor head, but this circuitry will be discussed in a later section.

It is also important to note that the limitations of the experimental apparatus prevented the liquid from being injected directly into the cylinder. By injecting the liquid into the head, one cannot directly characterize the response of the system to the presence of a specific volume of liquid in the cylinder; while some of the liquid injected will surely enter the cylinder of interest (the rear cylinder), some of the liquid will also undoubtedly enter the other cylinder, pool in the head, or drip out of the suction port of the compressor. Modifications could conceivably have been made to the head and valve plate to allow the liquid to be injected directly into the cylinder, but this would have substantially increased the clearance volume of the cylinder and affected the performance of the compressor. Nevertheless, the volume of liquid injected into the compressor can be approximately be controlled via the amount of time that the solenoid is held open, and the response of the system to the presence of liquid in the cylinder can be monitored by measuring the cylinder pressure. By using these methods to regulate and monitor the amount of liquid ingested into the cylinder, it is possible to characterize the performance of the liquid slugging diagnostic metrics.

The remaining required component of the liquid slugging apparatus is the insertion of a liquid refrigerant reservoir above the solenoid valve. Since the refrigerant exiting the condenser is almost solely liquid, this might seem unnecessary, but if the temperature of

Figure 3-18: Illustration of mounted suction pressure transducer.

the slugging apparatus is higher than will allow the refrigerant to remain in the liquid state, or conversely if the pressure is lower than will allow the refrigerant to remain in a liquid state, the refrigerant could potentially evaporate as it enters the compressor head, compromising the effect of the experiment. To protect against this possibility, a copper tube and a sight glass were inserted between the solenoid valve and the connection to the liquid line. Around both of these refrigerant reservoirs was soldered a 1/4 inch copper tube, through which cold water was circulated; once these collection points became sufficiently cool, they ensured that the refrigerant being injected into the compressor was completely condensed. The sight glass was also essential in that it permitted the individual conducting the experiments to visually verify that liquid was condensing in the apparatus; it also made it possible to visually ascertain that liquid was actually being injected during the liquid injection events.

One of the sets of mechanical measurements implemented at this point was the installation of two pressure transducers at the monitoring ports of the suction and discharge valves of the compressor. These measurements were obtained so that it would be possi-

ble to monitor the effect of liquid slugging on the pressures adjacent to, but not inside the compressor, and because these measurements would permit the calculation of the mass flow rate and other mechanical quantities. The transducers installed were manufactured by Measurement Specialties, and were selected on the basis of the maximum expected pressure at the respective ports; a 250 maximum psi transducer (part no. MSP-300-250-P-5-N-1) was installed at on the suction service valve, and a 500 maximum psi transducer (part no. MSP-300-500-P-5-N-1) was installed at the discharge service valve. The output of these transducers was a 4-20 mA signal, which is generally useful in field-installed instrumentation because only two wires are needed to both power the transducer and send the signal to the receiver. One of these pressure transducers is illustrated in Figure 3-18. A custom instrumentation amplifier had to be constructed to convert this signal to a 0-5 V input for the USB data acquisition card; this amplifier will be discussed later in this section along with the other electrical subsystems.

This first version of the liquid injection apparatus was useful in that it facilitated the completion of a series of steady-state slugging experiments in which different amounts of liquid were injected into the compressor head, and detected in the power waveforms at the corresponding instants in time. This was highly useful in that it validated the direction of this research. Unfortunately, it was discovered that transient slugging events exhibited no statistically significant change between the liquid slugging event and normal behavior. Two causes could have contributed to this problematic behavior. The first, and most straightforward, was that an insufficient quantity of liquid was being directed into the compressor cylinder, preventing the effect of liquid slugging from being observed on top of the large deviations in motor current during startup. The second of these explanations is based upon the fact that one can see in Figure 3-16 that the liquid from the 1/8 inch tube exited the tube in a jet directed across the opening of the suction port, so that gravity and turbulence in the compressor head were relied upon to direct the refrigerant into the cylinder. While this undoubtedly happened to a certain extent, the fact that the refrigerant had to slosh around inside of the head before entering the cylinder may have caused more refrigerant than expected to drip out of the suction port, rather than enter the cylinder. Nevertheless, this series of experiments validated the underlying fault detection approach in that deviations in the electrical power could be related directly to the quantity of liquid

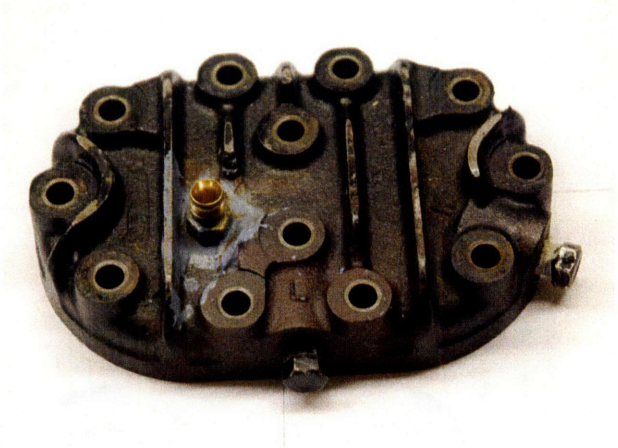


Figure 3-19: Compressor head for the second iteration of the liquid slugging apparatus.

ingested by the compressor, strengthening the case for developing a more sophisticated liquid injection apparatus.

To compensate for the shortcomings in this first design, the second iteration of the liquid injection apparatus incorporated a hole which was drilled into the compressor head directly above the suction port for the rear compressor cylinder. A compression fitting installed into this hole allowed a 1/4 inch copper tube to be plumbed directly into the head, so that a much larger quantity of liquid could be injected directly into the cylinder in the same period of time. Figure 3-19 illustrates this modified head, with the compression fitting located near its center, while Figure 3-20 shows the manner by which this new apparatus was installed on the modified compressor head. This configuration of the apparatus also had the benefit of pointing the nozzle directly down into the cylinder, minimizing the amount of liquid that splashed around in the head unnecessarily. This second version of the liquid injection apparatus actually went through two revisions; in the first, a hole was drilled directly in the stock compressor head, and the new arrangement was tested to evaluate its effectiveness. Once preliminary testing of this version of the apparatus was found to reproduce the steady-state liquid slugging data and provide an indication that transient liquid slugging could be detected, a custom head incorporating this hole for the slugging apparatus was machined at MIT's central machine shop. A photograph of this custom head is provided in Figure 3-22; the liquid slugging port is visible on the right side of the dividing wall toward the top end of the suction plenum.

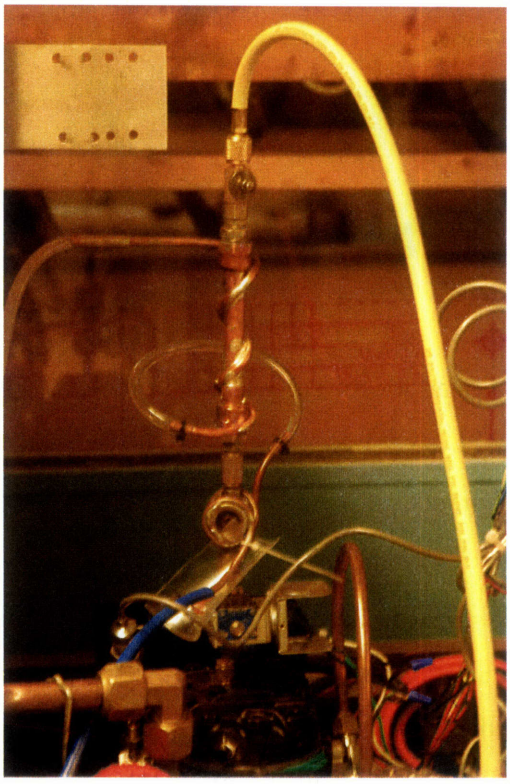


Figure 3-20: Second iteration of the liquid slugging apparatus.

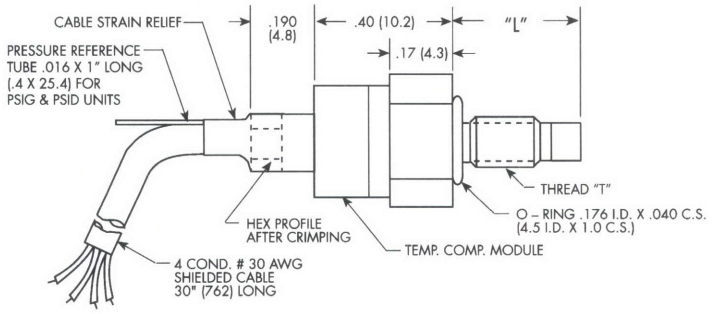


Figure 3-21: Illustration of cylinder pressure sensors (from datasheet).

In addition to the benefit of placing the liquid injection port exactly where it was desired, the custom head was also fabricated because a modified head design was necessary to house the transducers that directly measured the cylinder pressures. Two precision strain-gauge pressure sensors were purchased from Kulite (part. no. XT-190-2000A) expressly for this purpose; they were sized according to the maximum pressure expected in the cylinder during the liquid slugging transient, or 2000 psia. These pressure transducers were selected because they have a dynamic response which has a bandwidth high enough to record the response during the liquid slugging transient, and they will also be able to withstand the relatively high temperatures present in the compressor head during operation. A drawing of a representative pressure sensor is illustrated in Figure 3-21. Dave Otten, a research staff member in LEES, recommended these sensors for this application after he installed a similar range of sensors and measured the pressures in the head of a scroll compressor during operation. These are essentially constructed by correlating the deflection in a silicon diaphragm with the pressure across the diaphragm, effectively forming a silicon strain gauge, and placing this strain-gauge in a Wheatstone bridge so that the changes in resistance will be converted into an electrical signal that is proportional to the pressure. The output of this device interfaced with the data acquisition system via an instrumentation amplifier.

The principal modification to the design of the stock compressor head required for the installation of the pressure sensors was that the wall between the suction and discharge plenums in the compressor head had to be enlarged to ensure that the new sensors could be installed without introducing leakage between the two plenums. A picture of this head in which the size of the expanded wall is apparent is provided in Figure 3-22. Since the cable carrying the output of each sensors had to interface with the instrumentation amplifiers which were mounted externally to the compressor, the sensors were countersunk into the enlarged wall so that the sensor could extend down into the volume of the compressor sufficiently far that the sensor face would be located inside the cylinder. Two holes, each of which was the size of the sensor tip, also had to be drilled in the valve plate to allow the pressure sensors to pass through from the head to the cylinder itself. These holes in the valve plate are identified by arrows overlaid on the photograph of the modified valve plate shown in Figure 3-23. As the tip of the pressure sensors was recessed slightly

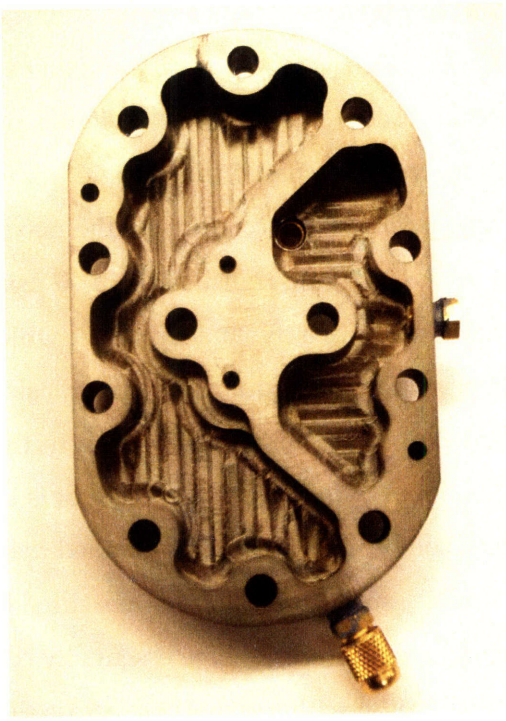


Figure 3-22: View of the bottom of the new compressor head.

(perhaps 0.01 in) from the bottom of the valve plate to ensure that the piston did not collide with the sensor face, the clearance volume of the cylinder was increased marginally, but the observed effect on the compressor performance was negligible. This particular approach to sensing the cylinder pressure also had the potential to increase the amount of leakage from the cylinder to either the suction or discharge sides, but the presence of gaskets on both sides of the valve plate, as well as the fact that the holes in the valve plate were mated to the bottom of the expanded wall in the compressor head made this effect negligible as well. Finally, the expansion of the wall decreased the size of the discharge plenum somewhat, but the fact that this plenum is coupled to the discharge line and the condenser through comparatively large piping also minimizes the effect of this decrease in the volume of the discharge plenum. The arrangement of the new head with the cylinder pressure sensors installed can be seen in Figure 3-24.

The third modification made to the experimental setup after the initial successes of the first configuration of the liquid slugging apparatus was the implementation of a method for identifying the position (and, by extension, speed) of the motor. One of the particu-

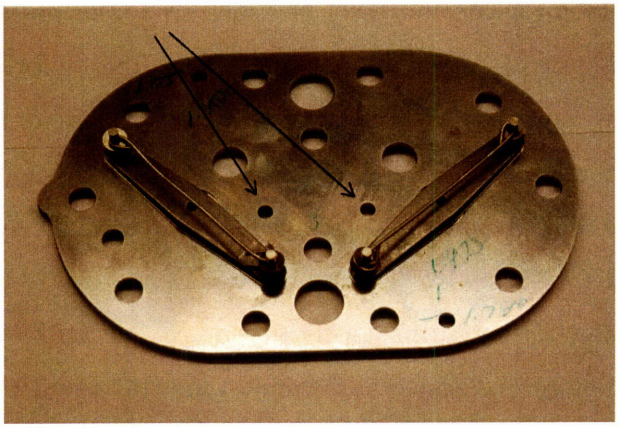


Figure 3-23: View of compressor valve plate.

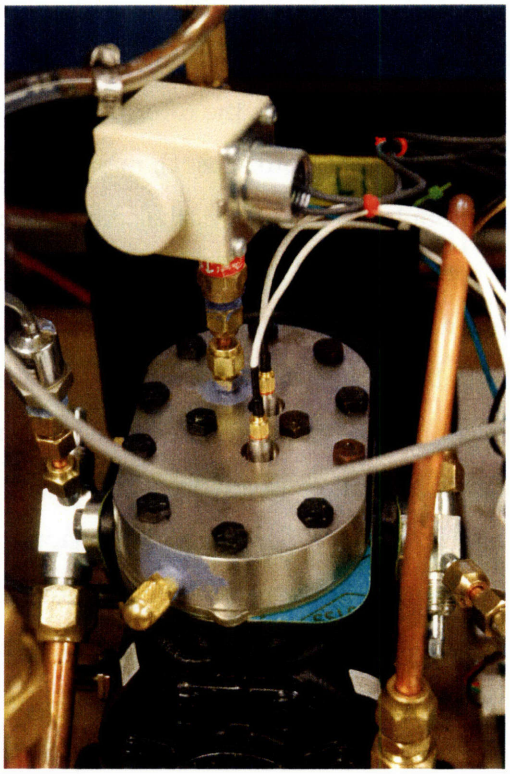


Figure 3-24: View of compressor with new head and additional sensors installed.

lar challenges in obtaining these measurements for a semi-hermetic compressor is caused by the fact that the motor and compressor internals are under pressure, as is the rest of the air conditioning system, so any method for sensing the motor position must transmit this information from the high-pressure environment inside of the compressor shell to the external environment where the data collection and processing hardware is located. Moreover, the environment inside of the compressor shell is relatively inhospitable, as the temperatures are high (perhaps 160° F) and refrigerant and oil are being splashed about inside the shell as the crankshaft turns. Lastly, the ends of the crankshaft are tightly sealed, as one main bearing mount is located at the end of the crankshaft and the oil circulation system collects and distributes oil from both ends. The collection of these factors served to preclude the use of many traditional methods of directly measuring motor speed, such as traditional optical or electromagnetic shaft encoders.

The position of the shaft was therefore identified by using the fact that the connecting rods are coupled to the motor shaft via an eccentrically located bearing, which transforms the circular motion of the motor shaft to up and down motion on the part of the piston. This eccentric motion was exploited to sense the rotor position by using a magnetic proximity sensor (also referred to as a geartooth sensor), manufactured by Cherry Corp (part no. GS100102) which was mounted in the side of the compressor shell at a fixed location; when the position of the connecting rod was sufficiently near, the sensor could register the presence of the connecting rod, but as the connecting rod continued in its rotation, it gradually moved away from the sensor. The proximity sensor was mounted in a hole precisely drilled in the side of the compressor shell directly in the plane of the connecting rod. These holes were drilled by locating the intended position of the hole on the outer surface of the compressor shell, and then constructing a mounting apparatus so that the compressor could be fastened to the bed of a mill and precisely positioned so that the hole would be drilled in the correct location. This hole was tapped to interface with the threaded proximity sensor, allowing the altitude of the proximity sensor face to be carefully adjusted so that the registration of the location of the geartooth sensor could be tightly controlled. The location of the position sensors in the side of the compressor shell can be seen in Figure 3-26, while the proximity of the sensor face to the connecting rod at its closest position can be seen in Figure 3-25.

Figure 3-25: Closeup picture of position sensors.

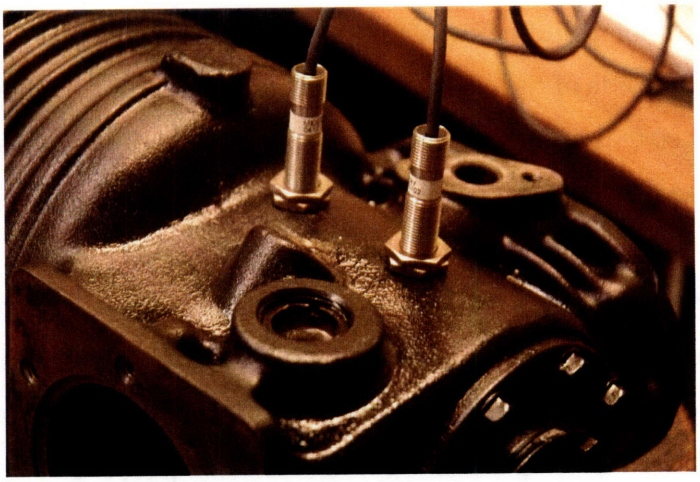


Figure 3-26: Picture of installation of position sensors.

Figure 3-27: Set screw and connecting rod viewed from outside the crankcase through the position sensor mounting hole with position sensor absent.

One of the other complicating factors in the development of this measurement was the fact that the connecting rods are made of aluminum, rather than steel, preventing the magnetic proximity sensor from sensing the presence of the connecting rod. To compensate for this, two tiny steel set screws, which could be detected by the magnetic proximity sensors, were mounted in the connecting rods. These set screws were installed and fixed with Loctite into holes which were only drilled partway into the connecting rods to ensure that the lubrication of the connecting rods would not be compromised. A photograph of one of these set screws viewed through the compressor shell can be seen in Figure 3-27. After the position of the proximity sensors with respect to the connecting rod was set so that the crankshaft position could be registered reliably, the sensors were locked in position with a nut on either side of the compressor shell and the thread was sealed with Loctite 246 Threadlocker so that the thread would be sealed for high-pressure use. The outputs of these proximity sensors were also interfaced with the data acquisition system, as will be described shortly.

After the compressor had been reassembled with this new ensemble of components, a second set of liquid slugging experiments, similar in scope to the first set, were conducted in which the effectiveness of the electrically-based liquid slugging fault detection method was evaluated and compared to the output of the newly available ensemble of mechanical measurements. The second version of the apparatus was found to effectively simulate liquid slugging as well as the first version of the apparatus, and the mechanical measurements provided data which validated the observations made from the electrical data. The

observed pressure measurements made it clear, however, that a much smaller volume of liquid was being ingested into the compressor head during transient slugging experiments than was expected. This could be seen by comparing observed starts when the compressor was operating normally to starts when the compressor was reputedly ingesting liquid; the waveforms of the cylinder pressure were nearly identical in both circumstances. One possible explanation for this, confirmed by further experimentation, is that the pressure gradient on the liquid column was not necessarily in the direction initially expected. After the air-conditioning experimental platform has been off for a sufficient period of time, the refrigerant will reach an equilibrium state in which the pressures at the bottom of the condenser are nearly the identical to the pressure in the compressor head. When the solenoid valve is opened, the liquid could therefore flow either direction before the compressor turns on, making it hard to ensure that the liquid refrigerant will flow from the slugging apparatus into the head and cylinder before the compressor is started.

One final set of modifications to the liquid slugging apparatus was therefore made in response to these experimental observations. In order to ensure that the pressure above the column of liquid is greater than the pressure in the compressor head, a vapor reservoir and sight glass were installed above the lower liquid reservoir and sight glass, and a ball valve was installed at the top of this plumbing tower of increasing height. In a manner similar to the liquid reservoir and sight glass, a length of 1/4 inch copper tube was coiled around the top reservoir and a second sight glass, and this tube was connected to another water circulation system, so that hot water could be used to circulate around this reservoir and pressurize the vapor contained in the reservoir. A photograph of the entirety of this improved injection apparatus is visible in Figure 3-28, while the vapor portion of this new injection apparatus is magnified in Figure 3-29.

As the increasing complexity of this liquid injection apparatus tends to obscure the means by which the liquid slugs are injected into the compressor head, a brief description of the experimental protocol is useful in more clearly elucidating its use. As before, the top of the slugging apparatus is connected to the bottom of the condenser, and the output of the slugging apparatus is located right above the suction valve for the rear cylinder. In order to get the refrigerant to condense, the cold water circulation system, which is connected to the liquid reservoir and sight glass, is turned on and set to a temperature low

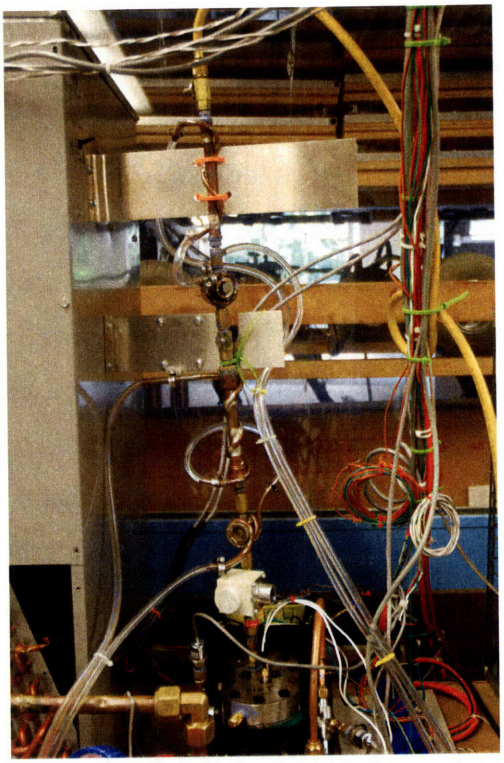


Figure 3-28: Slugging apparatus with vapor and liquid reservoirs.

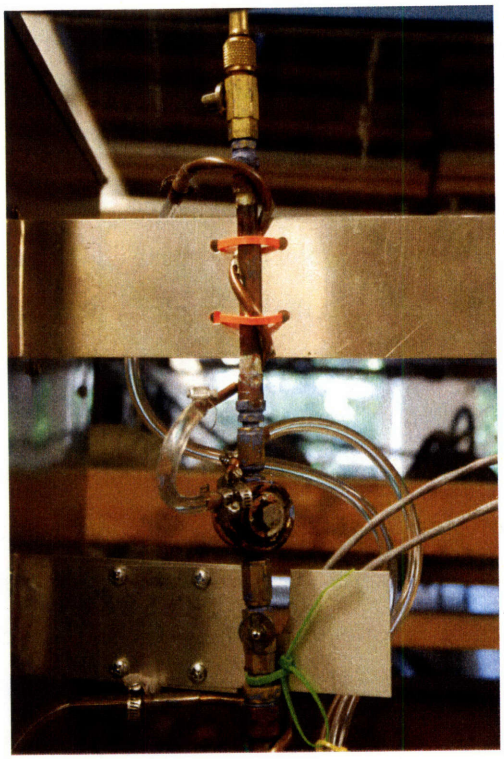


Figure 3-29: Vapor portion of slugging apparatus.

enough to make the refrigerant condense (0.1° C water was often used). The lower part of the apparatus is then chilled until the level of the condensed refrigerant reaches approximately halfway up the upper sight glass. Once this quantity of refrigerant has condensed, the top ball valve is closed, the cold water circulation system is turned off, and the hot water circulation system, connected to the copper tubing wound around the upper vapor reservoir and sight glass, is turned on. This circulation system circulates water which is hot enough to evaporate the refrigerant and pressurize the saturated vapor to a pressure greater than that of the rest of the system[2]. Once the upper reservoir has approximately reached thermal steady-state (about 15 minutes), the solenoid valve is opened for a fixed period of time, driving the liquid into the compressor cylinder and head, and the compressor is turned on soon after so that the effect of the liquid slug can be observed.

After this vapor reservoir and sight glass were installed, another set of liquid slugging experiments were conducted to test the effectiveness of this apparatus. As expected, the steady-state slugging diagnostic still functioned as it had in previous experiments; unlike the previous experiments, the effect of the transient liquid slugging on the preprocessed power was also consistently observed in this data. Over the course of these experiments, it was also discovered that the slugging apparatus tended to collect oil which was entrained with the liquid refrigerant, but the net effect of slugging this mixture of liquid oil and refrigerant was similar to the effect of the liquid on the power consumption of the compressor, so that no additional modifications to the slugging apparatus were necessary.

3.3.2 Electrical Systems and Sensors

In addition to the widely varied collection of mechanical instrumentation that was installed, perhaps the most central set of instrumentation for this research is that which was used to obtain observations of the electrical behavior of these electromechanical loads. Similarly, the control and data acquisition systems are also significant. These systems were essential to the success of this research, and will be discussed in this section.

Measurements of the voltages supplying the various loads of the currents flowing into the loads were obtained for these diagnostic methods. All of the electrical transducers

[2] 45° C water was often used, as the corresponding saturation pressure is 250 psia, which is much higher than the pressure in the system.

used were manufactured by Liaisons Electroniques Mecaniques, S. A. Geneve (LEM) due to their high bandwidth, low signal distortion, and good reliability. The voltage measurements were made with voltage transducers (part no. LV-25-P), while the current measurements were made with current transformers scaled appropriately for the expected peak transient current drawn by the load (part no. LA-205-P).

As this research was focused on identifying mechanical faults in the compressor both for the individual load and as part of an aggregate collection of loads in the air-conditioner, two sets of measurements were made. One set of transducers (comprising three measurements of the line-to-line voltages, and three measurements of the line currents) made measurements directly at the terminals of the compressor motor, while the other set of transducers measured the same electrical variables at the service entrance of the entire machine, so that the electrical behavior of the compressor, as well as the evaporator and condenser fans, was observed. The results in the later section are taken primarily from the data collected from the compressor-only data, but these should generally apply to the aggregate data as well.

Three different sets of associated interface circuitry were required for the range of mechanical instrumentation, which consisted of cylinder pressure sensors, suction and discharge pressure sensors, and magnetic proximity sensors. The magnetic proximity sensors did not require a great deal of signal conditioning, as the output of the sensors was a square wave oscillating between the supply voltage (+12 V) and ground; the amplitude of this signal needed merely to be reduced to vary between 0 and 4 V for the data acquisition system. The suction and discharge pressure sensors, on the other hand, generated a current output which varied linearly between 4 mA for atmospheric pressure to 20 mA for the maximum signal pressure range. A current-to-voltage amplifier was thus constructed to convert this current into a form suitable for the data acquisition setup, as is given in Appendix B; Burr-Brown manufactures a single-chip device called a 4-20 mA receiver expressly for such applications, and instrumentation amplifiers incorporating this device were used for this purpose. An additional revision of this amplifier was also designed which implemented the same process of conversion, in the case that the Burr-Brown part became obsolete[3].

[3]This is generally inevitable as soon as a device starts being used, and as soon as this part was installed in the board, it did indeed find its way onto an obsolescence list.

The third instrumentation amplifier needed was a bridge amplifier for the cylinder pressure sensors. This amplifier was constructed from a low-noise linear amplifier (part no. INA126PA), and the schematic for this circuit is also found in Appendix B.

A custom USB-based data acquisition system was built to collect the data from this ensemble of instrumentation. This board was built by Steven Shaw using the FTDI-based USB interface chip, a 12-bit 8-channel AD7856 analog-to-digital converter, and a SX28AC/DP microcontroller to control the operation of the components on the board. A driver allowing this system to be used in Linux was also written by Jim Paris and Steven Shaw; the computer used to collect the data was an 800 MHz Athlon PC with 512 Mb of RAM running Debian Linux. Since the data acquisition device has 8 single-ended inputs which have a dynamic range of 0 to 4 V, this board had to mated to the instrumentation by using a number of level-shifting amplifiers so that the signals would not go below the ground, as many of the input signals swing both negative and positive. The maximum sample rate of this data acquisition device was fast enough to provide sufficient resolution for the signal analysis used; the sample rate used for each card was 8 kHz per channel. An additional benefit of this system was its spatial flexibility; the data acquisition cards could be located close to the sensors to minimize the noise coupled into the sensor or instrumentation leads, and the signal out of the data acquisition card could then be sent over a long USB cable to the computer.

An especially useful aspect of this data acquisition system was the fact that, as it interfaced to the computer via the USB protocol, multiple cards could be used to simultaneously monitor the sensor outputs. This was of particular use in this research as 18 measurements in total were used to characterize the performance of the system; moreover, the capacitive coupling between adjacent channels on the data acquisition card was strong enough that the output of the proximity sensors, which was a 5 V square wave at 30 Hz, clearly caused interference and noise in adjacent channels on the data acquisition card. Because of the flexibility of the data acquisition system, this effect could be eliminated by collecting the data from the proximity sensors on a separate card, so that its noise wouldn't corrupt the other signals. Four data acquisition cards were ultimately used on the experimental air-conditioning system.

In using these cards to simultaneously collect data for long periods of time from all

four data acquisition boards, it was discovered that the drift between the clocks on the four boards caused a considerable amount of delay between the data collected on different boards. After about 6 hours of operation, the delay amounted to approximately 400 samples between the different cards. This effect was minimized by stopping all data collection and synchronizing the resumption of data acquisition via a signal sent simultaneously to all boards right before the event of interest took place. For example, these delays were avoided for the transient liquid slugging starts by starting data collection simultaneously for all four boards approximately 7 seconds before the compressor turned on; the electrical and mechanical transients which were observed were thus nearly (within a sample or two of each other) synchronized. The drift between data acquisition cards was also minimized by collecting the same number of channels of data on each data acquisition card, even for those which were not observing 6 channels of incoming data.

As discussed in §3.2, the electrical power consumed by the compressor can vary due to a number of different effects, such as thermal effects or the initial phase angle of the electric utility when the motor is turned on. To minimize the effect of the experimental variation which could not be eliminated in the preprocessing method, a control system was constructed which enabled the compressor and air-conditioner to be operated in a tightly regulated manner. This system used a PIC 16F877A microcontroller to control a set of Crydom arbitrary turn-on solid-state relays (part no. D2450-10) which connected the loads of interest to the utility. This microcontroller was driven with an interrupt which was locked via a comparator to the line-to-line voltage V_{ab}, so that the time at which the compressor was turned on could be precisely controlled with respect to the electrical angle of the wall. The solenoid valve in the liquid injection apparatus was also controlled with the microcontroller, so that the duration of time for which it was opened and injecting refrigerant into the compressor could also be regulated. Very small amounts of liquid could also be injected by using this control system, as it would be difficult for a human operator to only open the solenoid valve for 20 ms. Lastly, this control system was essential in automating the collection of large quantities of data so that the repeatability of the compressor behavior could be studied without the risk of human operator mistakes.

This automatic control system was also augmented with a mechanical control system for the electrical components of the air-conditioning system, allowing the user to turn on

loads without relying upon the microcontroller system to do so. A series of switches was devised to allow the user to select which loads were to be activated via the computer, and which loads were to be activated by hand. This system proved to be very flexible in allowing the user to run the air-conditioner in a wide range of experimental procedures.

Of the many benefits of this control system, perhaps the most useful was the fact that the behavior of the microcontroller could be modified dynamically from the same machine which was controlling the data acquisition cards. A bootloader, written by Mariano Alvira and Jim Paris, was programmed onto the PIC so that control programs could be written on the fly as the experimental protocol was changed and modified. This had two benefits, the first of which was in allowing the user to quickly change the experimental protocol, such as the amount of time the the liquid injection solenoid valve was open, by simply uploading a new procedure. The second of these benefits is that, since both the data acquisition and the control systems were integrated on the same PC, Perl and shell scripts could be written which managed both systems according to the experimental requirements. For example, the transient liquid slugging data was acquired by having the control system start the data acquisition shortly before the compressor was started, thereby eliminating drift and ensuring that the transients were collected in a repeatable way.

3.4 Experimental Results

The success of this approach for identifying liquid slugging faults relies upon the ability to distinguish between the effects of normal variations in the operating conditions on the compressor's electrical behavior and the effects on these electrical waveforms caused by liquid slugging. This section therefore begins by characterizing the performance of the preprocessor to potential types of variation in the raw electrical signals, and is followed by the study of the variation in the preprocessed electrical waveforms resulting from expected variations in the compressor's fault-free operating state. After the variation in the fault-free behavior of the compressor has been described, results from experimentally simulating steady-state liquid slugging and transient liquid slugging will be presented and discussed.

3.4.1 Preprocessor Characterization

The primary objective addressed in the design of the preprocessor was that it would convert the measured electrical signals into a form which was amenable to the creation of a fault-free signal model and the identification of fault signatures for liquid slugging. Three particular sources of variation in the compressor's electrical behavior have a noticeable effect on the initial electrical transient: the electrical angle of the utility when the compressor motor is turned on, the 60 Hz component of the utility voltage, and the varying electrical angle of the utility. All of these phenomena therefore need to be studied to validate the preprocessor's performance.

The ability of the preprocessor to compensate for the 60 Hz component of the utility and changes in the electrical angle when the compressor was turned on were tested using an induction motor simulation in Matlab. Three sets of balanced three-phase voltages were generated, in which the initial phase of the voltage was set to $\phi_e = 0$, $\pi/2$ and $3\pi/2$, and these were used to drive the induction motor simulation which output a three-phase set of currents. These simulated voltages and currents were converted to the form used by the preprocessor and input to the preprocessor so that its ability to effectively implement the Park transform on the currents and compensate for the changes in the initial starting angle could be evaluated. These outputs of the preprocessor can also be compared with the signals i_{ds} and i_{qs} which were also generated by the simulation, as seen in Figures 3-30 and 3-31.

By comparing i_{ds} and i_{qs} from the simulations with the set of outputs from the preprocessor, it is clear that the preprocessor is able to effectively compensate for the variation in the initial electrical angle when the induction motor is turned on, as will inevitably occur in the field. All three of the output waveforms are well aligned in time, despite the range of variation in ϕ_e.

The other main function of the preprocessor is its ability to track the variation in the electrical angle θ_e caused by $\phi_e(t)$, as illustrated in §2.4.2. While this variation only needed to be tracked in developing the airflow estimation methods, the function of the preprocessor depends on its ability to compensate for the changes in $\phi_e(t)$, so that the d- and q-axis currents do not vary with time. It is important to note that the preprocessor re-

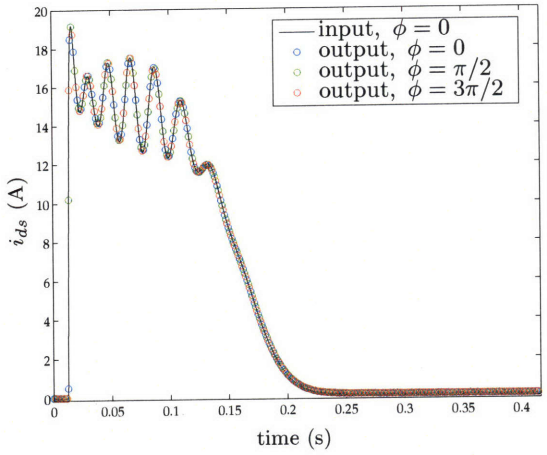


Figure 3-30: D-axis stator current input and output of preprocessor.

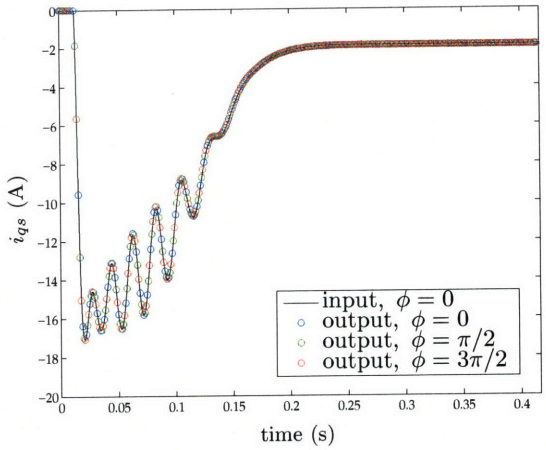


Figure 3-31: Q-axis stator current input and output of preprocessor.

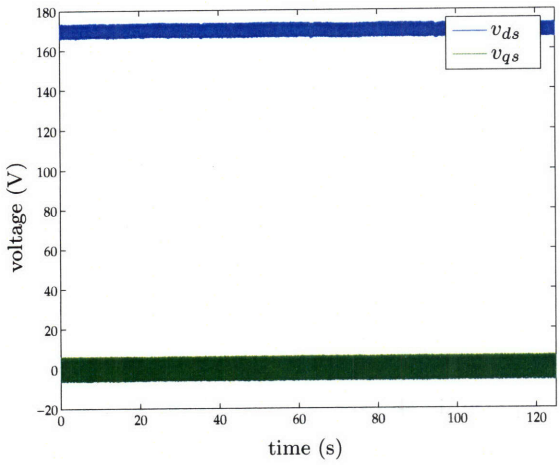


Figure 3-32: D- and Q-axis voltages transformed with phase correction.

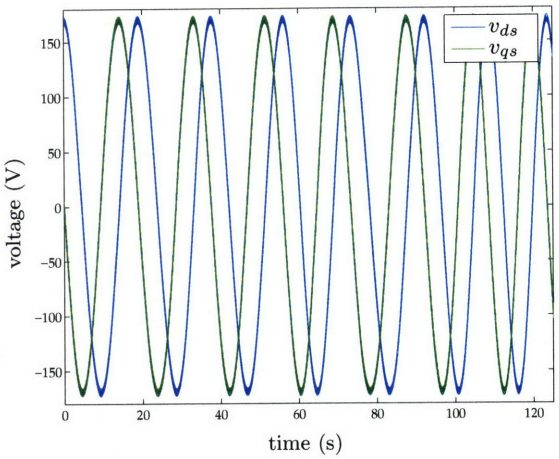


Figure 3-33: D- and Q-axis voltages transformed without phase correction.

lies on the changes in ϕ_e being relatively slow, as abrupt changes in the angle of the wall will cause convergence problems for the nonlinear least squares solution method. Many electrically-powered system also operate on the basis of this assumption, however, making this a reasonable condition to impose upon the preprocessor. The operation of the preprocessor on a set of experimentally observed 3-phase voltages, which include variations in $\phi_e(t)$, can be seen in Figures 3-32 and 3-33.

It is evident that the preprocessor is able to successfully track the variations in the electrical angle over a reasonable period of time (125 seconds) and produce d- and q-axis currents over that period of time which are well synchronized with the utility voltage. In

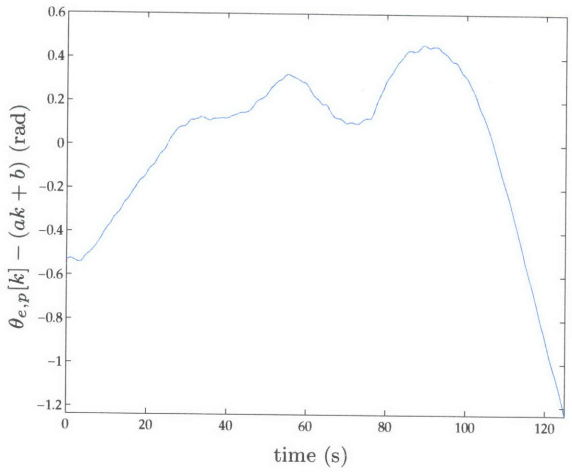


Figure 3-34: Time-varying component of utility voltage angle.

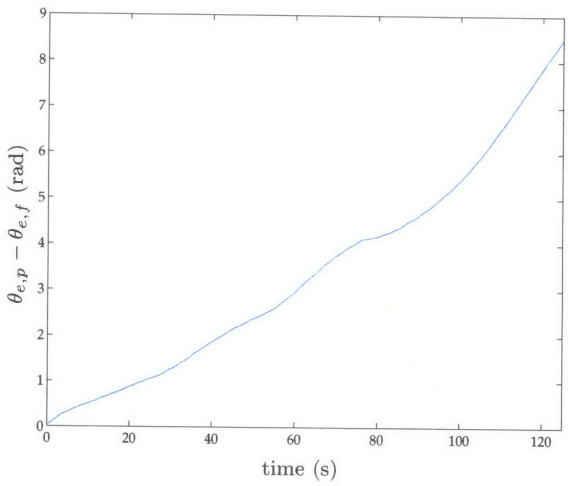


Figure 3-35: Residual between nonlinear phase estimate and linear phase estimate.

order to appreciate the range of variation in $\phi_e(t)$ tracked by the preprocessor, a comparison was made between the electrical angle estimates $\theta_{e,p}[k]$ generated by the preprocessor and a comparable set of electrical angle estimates $\theta_{e,f}[k] = \omega_e k T_s + \phi_e$, where an estimate of the electrical frequency ω_e is obtained by locating the peak in the FFT of the voltage waveform, and the constant phase angle estimate ϕ_e is obtained for the beginning of the voltage waveform by the techniques discussed in §2.4.2. The discrepancy between these two waveforms can be seen in Figure 3-35, as this plots the residual $\theta_{e,p}[k] - \theta_{e,f}[k]$. It is clear that the FFT-based estimate of the voltage angle $\theta_{e,f}$ does not accurately characterize the time-varying electrical angle of the utility.

Two different aspects of the time variation of $\theta_e[k]$ are illustrated in Figures 3-34 and 3-35. In Figure 3-34, the nonlinear time-varying component of the angle is illustrated. Since the wall is normally modeled by $\theta_e(t_k) = \omega_e t_k + \phi_e$, the waveform $\theta_{e,p}[k]$ was fitted to the model $ak + b$ using standard linear least squares techniques, allowing best-fit parameters a and b to be obtained. The nonlinear time-varying component of $\theta_{e,p}$ could therefore be obtained by computing $\theta_{e,p}[k] - (ak + b)$, as is shown in the figure. While both Figures 3-34 and 3-35 illustrate similar information, the main difference between the two plots is that the estimate of the radian frequency ω_r obtained for $\theta_{e,f}$ is based upon the location of the peak of the FFT waveform, while the estimate of the frequency in the $ak + b$ model is based upon the nonlinear least-squares estimate of the electrical angle. One might note that

other techniques for implementing cycle-by-cycle electrical angle prediction could be also implemented, such as estimating the frequency and initial phase for each individual cycle, rather than for the whole waveform, using the FFT. These methods would be qualitatively similar to the method implemented here.

3.4.2 Nonslugging Compressor Characterization

At its core, the liquid slugging FDD method uses a signal-modeling approach for FDD, as described in Chapter 1: the size of the residual between preprocessed waveforms describing fault-free behavior and preprocessed waveforms describing potentially faulty behavior is used to determine the presence of the liquid slugging fault. Since the preprocessed waveforms generated by reciprocating compressors operating in a non-faulty state vary about a mean waveform describing the average behavior of the compressor starts, this variation must be characterized in order to establish limits for the fault detection method. In particular, a significant amount of variation was observed between the preprocessed electrical waveforms for the compressor start transients. This variation comes from three distinct sources: changes in the temperature of the compressor, the initial position of the pistons, and the electrical angle when the voltage is to the stator windings. Each of these sources of variation will be isolated as much as possible, so each may be examined and characterized experimentally and their impact on the liquid slugging FDD method assessed.

Due to the wealth of electrical, mechanical, and thermal interactions taking place inside the compressor, it is often difficult to definitively attribute a particular feature in a set of measurements to a direct cause. While some plausible explanations for interesting features in the data will be offered in this section, the main objective is not the detailed explanation of the wealth of phenomena which can arise during the operation of the compressor, but rather the collection of a set of observations about the compressor's behavior which will guide the construction of and influence the success of electrically-based liquid slugging FDD techniques.

A set of representative d- and q-axis currents are shown in Figures 3-36 and 3-37 in order to emphasize some of the particularly important features of these waveforms. The most notable characteristic of these waveforms is that they both qualitatively look very similar to the i_{ds} and i_{qs} generated by the simulation of the induction motor. In particular,

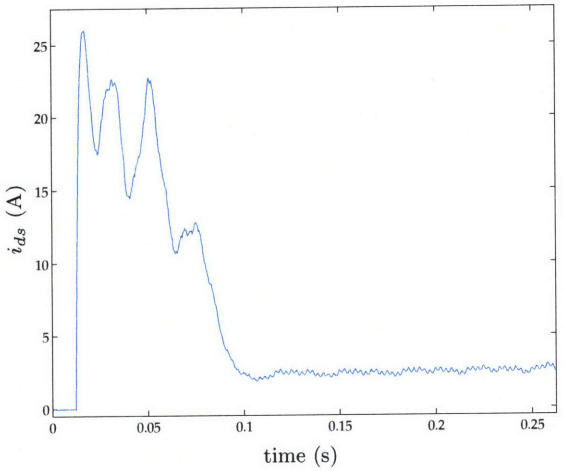


Figure 3-36: i_{ds} for a representative compressor start transient.

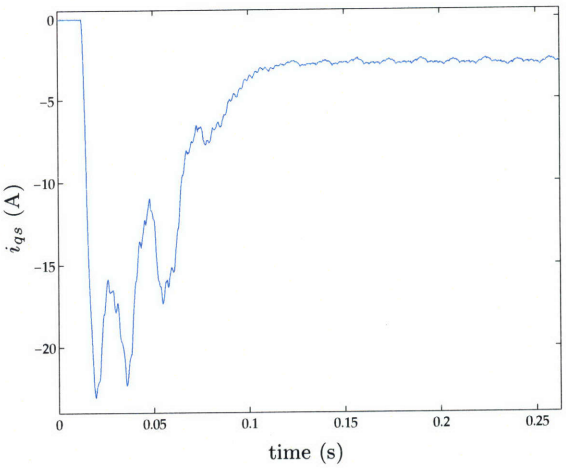


Figure 3-37: i_{qs} for a representative compressor start transient.

it is significant that both the d- and q-axis currents are relatively constant after the start transient, suggesting that the load on the compressor motor can be well-described by a constant inertia and a damping factor proportional to the speed of the rotor, rather than having a time-varying component due to the rotational motion of the compressor internals. Some noise on top of these waveforms are also clearly discernible, but this is largely caused by voltage distortion present on the electric utility (as previously discussed in §2.4.2).

The effects of the changes in the startup waveform were studied by cycling the compressor with a fixed period of 4 minutes on and 4 minutes off over the course of approximately 7 hours, and the full set of electrical and mechanical measurements were recorded over the duration of this series of tests. These time intervals were chosen because the suction and discharge pressures were found to equilibrate by the end of this period. Over the course of these experiments, 75 start transients were collected, providing ample data with which the sources of variation could be studied.

The first, and perhaps most significant, source of variation in the electrical transients is caused by changes in the temperature of the compressor. Many different thermally-based phenomena can potentially cause these changes; for example, temperature changes in the motor will cause corresponding changes in the impedance of the motor windings, as was seen in the fan motor of Chapter 2. The properties of the refrigerant and oil in the system will also change with temperature, as the amount of refrigerant absorbed into the

oil depends on the oil temperature, and the pressure of the refrigerant in the system will also depend on the temperature of the system components. Moreover, the temperature changes can occur over a very long timescale, since some of the components in the system (particularly the compressor body), have a very large thermal mass. The changes in the rotor impedance will occur particularly slowly, as the rotor only directly coupled to the rest of the machine through the air-gap and the two main rotor bearings, so that the heat transfer processes have a significant convective component.

The variation in the preprocessed electrical transient waveform due to temperature changes was isolated by analyzing a set of startup transients which correspond to approximately the same initial piston position. The effect of the initial electrical angle was also minimized during this study by controlling the timing of the compressor start with the set of arbitrary turn-on solid-state relays, so that the initial electrical angle during all of these transients was the same, e.g. $\theta_{ab} = 0$. The variation in the electrical waveforms due to temperature changes can be seen in the plots of i_{ds} in Figures 3-38 and 3-39, and in the plots of i_{qs} in Figures 3-40 and 3-41. Clearly discernible changes in the electrical transients are visible between the first start (labelled start 1) and the 70th start (labelled start 70). Unfortunately, there is a noticeable gap between start 24 and start 55 due to the fact that no start was observed with the same starting piston position in this interval.

A substantial amount of variation is apparent between the first and subsequent starts for both i_{ds} and i_{qs}. One phenomenon which could cause this effect could be the resistive heating of the motor windings, which would have a dramatic effect during the course of the first cycle as the windings are heated above the ambient temperature, but not during successive cycles as the temperature of the windings will not change substantially between cycles. This effect could also be caused by the large difference in the system pressure before the first start and the system pressure before subsequent starts. In general, suction gas-cooled compressors could also be susceptible to a similar effect caused by elevated cylinder pressures during the first start transient due to liquid refrigerant or oil slugging. In these circumstances, refrigerant can migrate into the compressor oil during a long off-cycle; when the compressor is first started, the refrigerant will quickly start to boil out of the oil and potentially cause liquid slugging-like behavior. However, this scenario is unlikely in this compressor, as it is sufficiently small that the motor windings are cooled by

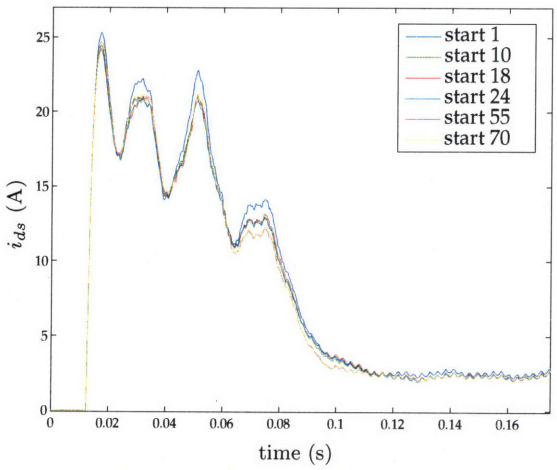


Figure 3-38: i_{ds} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up.

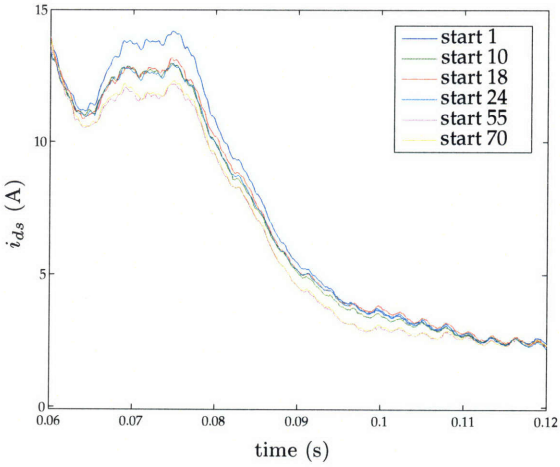


Figure 3-39: Zoomed version of adjacent plot.

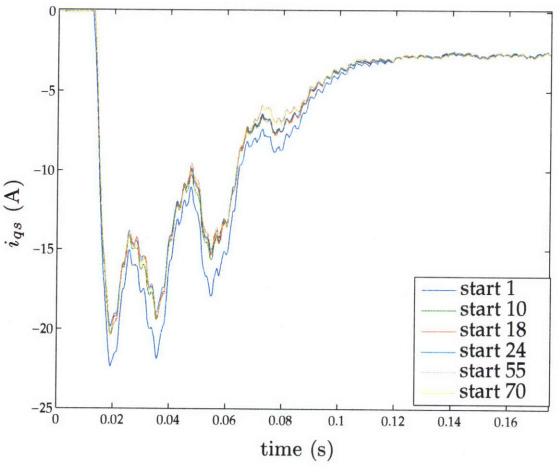


Figure 3-40: i_{qs} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up.

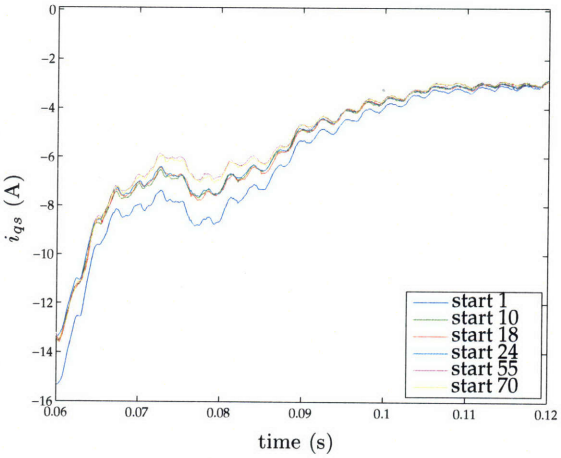


Figure 3-41: Zoomed version of adjacent plot.

convective and radiative heat transfer to the air surrounding the compressor shell, rather than the suction gas.

After the initial start transient, the two currents i_{ds} and i_{qs} gradually change over the series of the tests. These changes could be caused by further changes in the winding impedance as the windings continue to heat until they reach steady state. These changes could also be caused by the gradual heating of the compressor shell, and the gradual heating of the oil, so that less of the refrigerant can migrate to the hot oil during late compressor starts than can migrate to the oil in the early compressor starts. In addition, the variations in the d-axis currents are extremely well correlated to the variations in the q-axis currents, so that little additional information will be obtained from the analysis of the q-axis currents which is not available from the d-axis currents. This research therefore focused primarily on examining the d-axis currents for changes correlated with liquid slugging phenomena.

Another plausible explanation for the difference between the first transient and succeeding transients can be seen in Figures 3-42 and 3-43, in which the rear cylinder pressure P_{rc} and the pressure at the discharge port P_d are illustrated. It is evident in this picture that the cylinder pressure before the first start transient is noticeably lower (approximately 10 psi) than it is during succeeding transients. This effect could be attributed to a variety of causes; for example, all of the refrigerant in the air-conditioning system is near equilibrium with the surrounding lab environment before the compressor turns on for the first time. In comparison, the refrigerant will not be in equilibrium with the laboratory during off-times in the middle of the compressor cycling schedule, since there is insufficient time to reach this equilibrium state. While the system pressure is lower, however, an examination of the peak pressures in Figure 3-42 shows that the peak pressures are the same for the first start and all subsequent starts, since the discharge valves open at a fixed cylinder pressure. This could conceivably have a dramatic effect on i_{ds} and i_{qs}, as the lower system pressure means that the compressor will have to deliver more power in order to accelerate the crankshaft to generate these temporarily higher pressure gradients.

These preliminary observations of the correlation between the preprocessed electrical waveforms and the system pressures suggests that useful compressor diagnostics can be formulated on the basis of electrical measurements. In the example seen here, the changes in the mechanical load on the compressor caused by different system pressure appear to

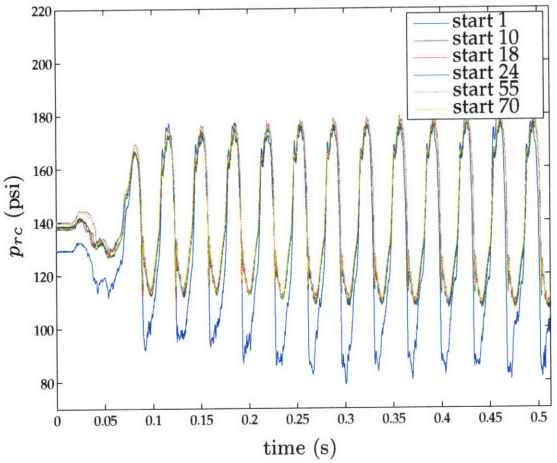


Figure 3-42: P_{rc} for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up.

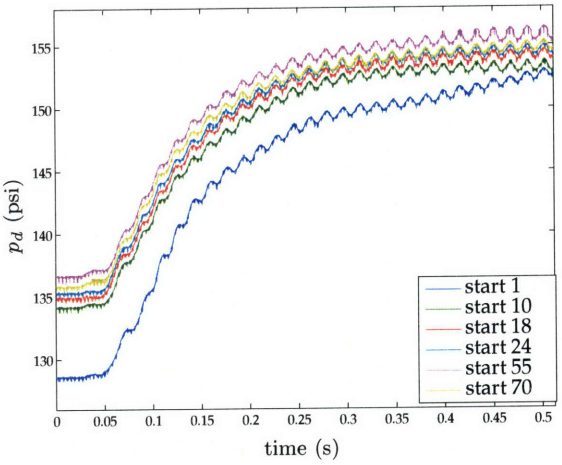


Figure 3-43: P_d for consecutive transient starts with the same starting cylinder position and electrical angle as the compressor windings warm up.

be strongly correlated with the changes in the electrical signature. This dependence on the conditions external to the compressor also increases the amount of variation in the start transients, however, making it necessary for a liquid slugging FDD method to monitor system pressure in order to maximize the sensitivity to small faults. One might note that the changes in the cylinder pressure are still relatively small, however, suggesting that liquid slugs which cause overpressures in the range of even 100-200 psi, which are far lower than the documented overpressures of 1000-2000 psi, will clearly be discernible from normal start transients.

Further examination of the pressure waveforms shows that the cylinder pressure waveforms are very repeatable when the compressor turns on after the first time, and that the baseline pressure before the compressor turns on gradually rises over the sequence of cycles. This observation also correlates with the gradual reduction in the peak values of i_{ds} and i_{qs} over the set of tests, and also with the proposed explanation for the high peak values of i_{ds} for the first transient: as the system pressure rises, the compressor needs to deliver less power to achieve the same peak cylinder pressures. One possible explanation for the gradual rise in the system pressure during off times is that the slow heating of the system components, such as the compressor shell and the condenser fins, has an effect on the pressure in the system during these off-times. Another possible explanation is that less

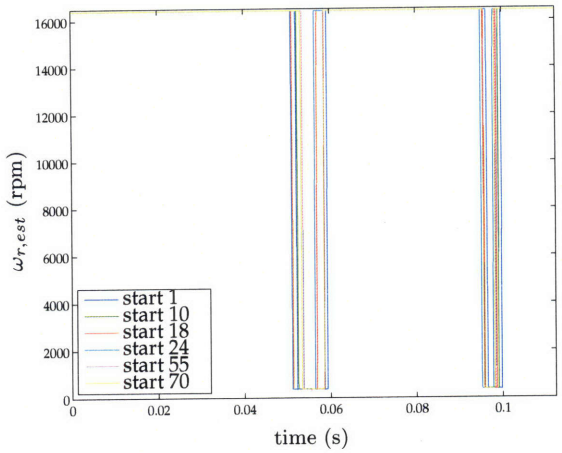


Figure 3-44: Tachometer output during the startup period for the consecutive transient starts with the same starting cylinder position and electrical angle as the stator windings warm up.

refrigerant migrates into the oil as its temperature increases, causing more refrigerant to stay pressurized in the system. Rather than become involved in the spectrum of possible explanations, however, it suffices for the purposes of this research to note the corresponding variations in compressor power during this series of start transients.

As the initial piston position can strongly affect both the cylinder pressure and the electrical transient, the output of one of the position sensors is illustrated in Figure 3-44. This plot clearly shows that the pistons all start in nearly the same position in all of the examined transients. One can also surmise from the small quantity of data shown in this plot that the compressor is accelerating at approximately the same rate in all of these transients, since the times between consecutive edges are nearly the same for all of the waveforms. While it is also theoretically possible to generate speed estimates from these waveforms, the paucity of angle information during much of the waveform makes such estimates extremely sensitive to noise, greatly limiting their usefulness.

The next set of variations to be characterized were those caused by changes in piston position. This characterization data was obtained by using the last twenty transients, numbered 56-75, from the previous dataset. These data were used because the initial electrical angle was constant ($\theta_e = 0$) across all of the transients, and there was minimal temperature variation across all of the data in this set, since the compressor had been cycling for nearly six hours. Additional evidence that there is minimal temperature variation can be seen

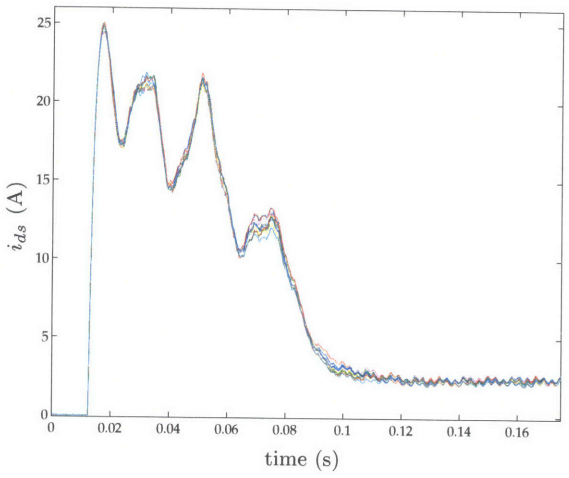


Figure 3-45: i_{ds} for consecutive transient starts with a range of different starting positions when the compressor is warm and the electrical angle is the same.

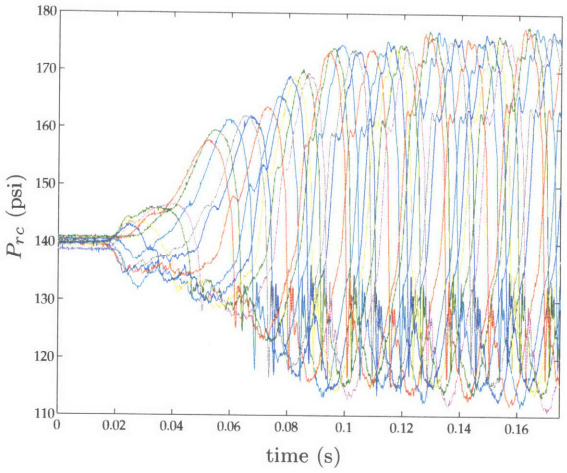


Figure 3-46: P_{rc} for consecutive transient starts with a range of different starting positions when the compressor is warm and the electrical angle is the same.

by considering the minimal amount of variation between transients 55 and 70 in Figure 3-38. The scope of variation can be seen by examining data collected from 11 of these start transients which started in distinct positions; the currents i_{ds} during this start transient are illustrated in Figure 3-45 and the rear cylinder pressure P_{rc} are illustrated in Figure 3-46.

These plots illustrate the fact that the variation in i_{ds} due to different starting positions qualitatively resembles the variation caused by changes in the temperatures. This again suggests that large deviations in the cylinder pressure caused by liquid slugging will be easily detectable, regardless of the initial piston position.

The effects of different initial electrical angles on the waveforms i_{ds} and i_{qs} were also studied in this research. Variations in the electrical transients were introduced by using the solid-state relays in the control system to turn the compressor on at specific points in the electrical line cycle. The thermal effects on the transients were minimized during these experiments by performing these tests at the end of the seven hours of cycling, so that the cycling period and the compressor temperature could be maintained. The effect of the initial piston position was harder to eliminate, as large numbers of transients at each different electrical angle would have to be collected to fully characterize the behavior of the machine, but each of the transients collected at a different electrical angle could be compared to a transient which was started at $\theta_e = 0$ and the same initial piston position. The

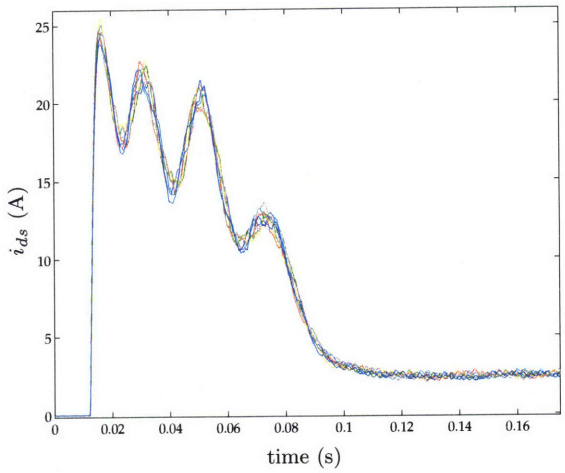


Figure 3-47: i_{ds} when the electrical angle is varied by steps of $\pi/4$ for the compressor when the windings are hot and the initial position of the pistons varies.

electrical angle was varied in steps of $\pi/4$ during these tests, and two separate transients were collected for each step. The range of variation can be seen in Figure 3-47, which illustrates the full set of 8 of these start transients with different starting positions.

Clearly, the variation due to changes in the initial electrical angle is similar to the variation due to the piston position and the thermal effects, the main difference being that changes in the electrical angle affect the entire electrical transient, rather than just the last half of the transient. Nevertheless, the range of variation is not large. The variation in i_{ds} can also be seen by comparing individual start transients in which the electrical angle is varied but the initial piston position and thermal state are the same, as illustrated in Figures 3-48 and 3-49.

These waveforms do look qualitatively similar, providing more support to the conjecture that changes in the mechanical load caused by liquid slugging will be readily identifiable by using this preprocessing method. One possible explanation for the observed variation in i_{ds} is that the changing initial electrical angle excites the magnetic material in the motor differently, slightly modifying the electrical behavior of the machine. As the two waveforms corresponding to $\theta_e = \pi/2$ and $\theta_e = 3\pi/2$ look qualitatively similar, the amplitude of the voltage waveform and the corresponding degree of the magnetic saturation of the iron could have an effect on the stator winding currents. The plots in Figure 3-50, which show two transients with the same initial piston position, same temperature con-

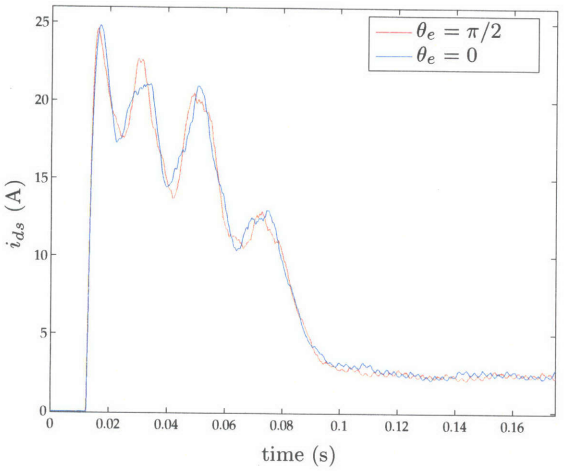


Figure 3-48: i_{ds} for electrical angles of $\theta_e = 0$ and $\theta_e = \pi/2$ when the initial piston position is the same and when the stator windings are warm.

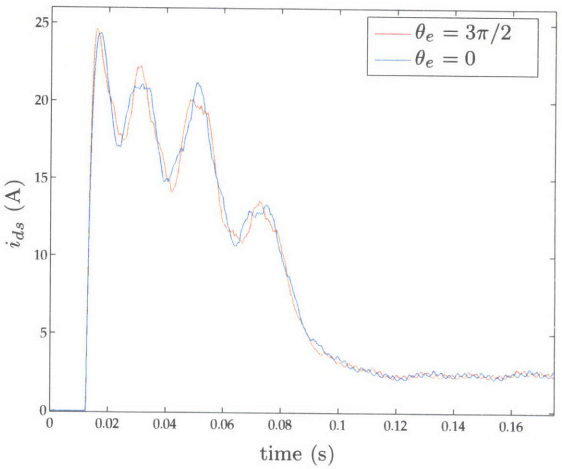


Figure 3-49: i_{ds} for electrical angles of $\theta_e = 0$ and $\theta_e = 3\pi/2$ when the initial piston position is the same and when the stator windings are warm.

ditions, and initial electrical angles differing by π, could also lend support to this theory.

The similarity between these waveforms suggests that the variation due to the electrical angle might be caused by the amplitude of the voltage waveform when it is applied to the stator windings. In this case, the range of voltages from 0 to $\pi/2$ could be sufficient to characterize the shape of i_{ds} for all initial electrical angles.

3.4.3 *Steady-State Liquid Slugging*

The first type of liquid slugging that was experimentally studied was steady-state slugging, due to the relative simplicity of both the manner of inducing this fault in air-conditioning equipment and the means by which it is detected. In general, many of the factors discussed in §3.4.2, such as the changing piston position and the electrical angle, do not affect steady-state slugging diagnostics because these FDD methods are designed to identify changes in the steady-state currents into the stator windings, which do not vary significantly. The identification of transient slugging is comparatively more difficult because the normal transient behavior needs to be characterized before changes from the fault-free transient can be identified.

Steady-state slugging was experimentally studied using a standard protocol. Before

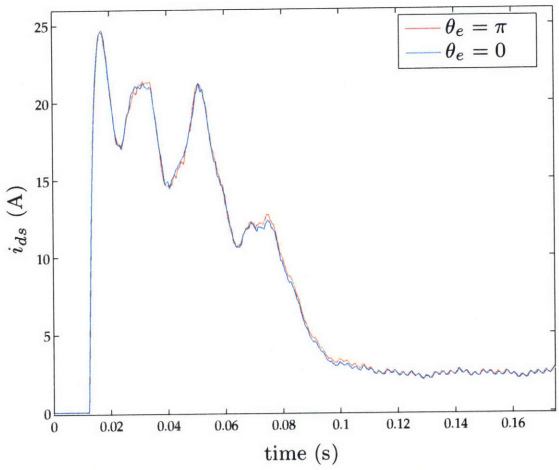


Figure 3-50: Plot illustrating symmetric characteristics of transformation for i_{ds} when the windings are hot and the initial position of the pistons varies.

any slugging experiments were performed, the compressor was run for between 1-2 hours to ensure its operation in thermal steady-state. Cold water, with a temperature between 0 and 3° C was also circulated through the copper tubing around the liquid reservoir so that the refrigerant could condense in the reservoir above the solenoid valve. The refrigerant line connecting the king valve at the bottom of the condenser to the top of the slugging apparatus was also installed and purged of air, so that the slugging apparatus would have a supply of refrigerant. After this sequence of preparations, liquid refrigerant was injected into the compressor head by opening the solenoid valve for prescribed intervals of time, which varied in length according to the desired size of the quantity of liquid injected.

While the preprocessing method discussed in §3.2 was primarily developed for detecting transient liquid slugging, these experiments also investigated the effectiveness of the preprocessing method for detecting steady-state liquid slugging, increasing the applicability of both the preprocessing method and the general slugging diagnostic approach. Before testing the efficacy of this preprocessor on experimental data, an induction motor simulation was run that simulated the expected effect of the liquid slugs by introducing torque pulses through changes in the coefficient β for the damping torque $\beta\omega_r^2$ during the steady-state operation of the motor and examining the resulting effect on the currents i_{ds} and i_{qs}. These simulations had the dual benefit of providing evidence that such changes would be visible on the output of the detection method, as well as providing support for

the assumed mechanism by which the liquid slugs would affect the motor operation; if the observed effects on the motor resembled those of the simulated pulsatile torques, this would strengthen the argument that the actual effect of liquid slugging on the motor is the generation of pulsatile torques on the rotor, as opposed to other unmodeled phenomena. The parameters of the simulated motor and torque pulses were chosen somewhat arbitrarily, and were only used to study the representative effect that torque pulses might have on an induction motor, rather than precisely modeling the exact behavior of the physical system.

The results of such a simulation are illustrated in Figures 3-51 and 3-52. The normal operating load torque on the motor was set to 0.032 N-m at the normal operating speed, while the torque pulses add an additional 1.87 N-m of torque to the load to simulate the effect of slugging on the motor. The size of these torque steps were not chosen because of any particular known correspondence with the experimental system, but rather were only chosen to test whether or not a pulsatile torque would have a manifestation in i_{ds} or i_{qs}, and whether or not this manifestation would resemble the observed changes in the experimental data, which are shown later in this section. The effect of the pulsatile load torques on i_{ds} and i_{qs} is quite discernible in these plots, as the currents respond to the torque steps by changing to increase the torque developed by the machine during the momentary increase in load. In addition, the speed of the rotor drops when the load torque is applied and recovers after the load torque is removed.

Steady-state slugging was studied experimentally by injecting four different quantities of liquid into the compressor head, corresponding to the solenoid valve being opened for 67 ms, 133 ms, 270 ms, and 533 ms. Figures 3-53 through 3-62 illustrate the effects that these liquid injection events had on both i_{ds} and i_{qs}. As noted previously, the precise volume of liquid injected into the cylinder could not be well controlled, due to the limitations of the experimental apparatus; since mechanical measurements of the cylinder pressure are available, Occam's razor suggests that overpressures which are observed in the cylinder during the liquid injection events are most likely caused by the presence of liquid in the cylinder. These steady-state slugging experiments will be discussed in order of the size of the injected quantities of liquid; Figures 3-53 and 3-54 illustrate the effect of opening the solenoid valve for 4 electrical line cycles, or 67 ms.

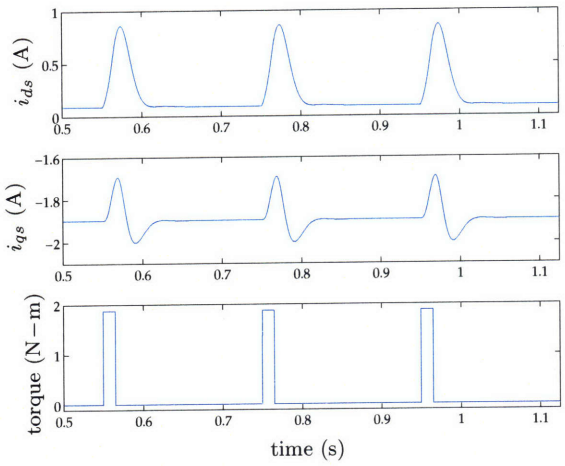


Figure 3-51: Transformed currents with pulsatile torques applied to the motor.

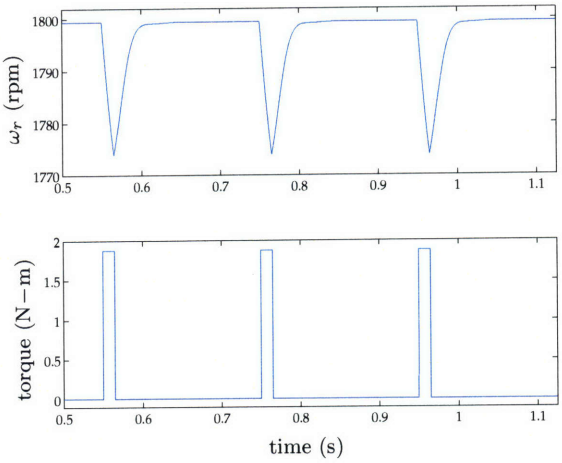


Figure 3-52: Motor speed during torque pulses.

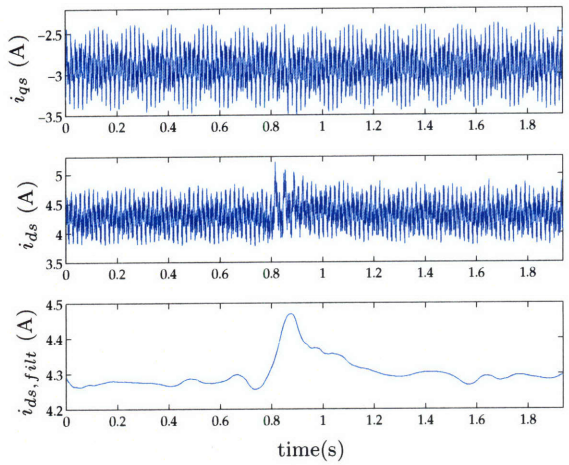


Figure 3-53: Processed electrical measurements for 67 ms steady-state slug.

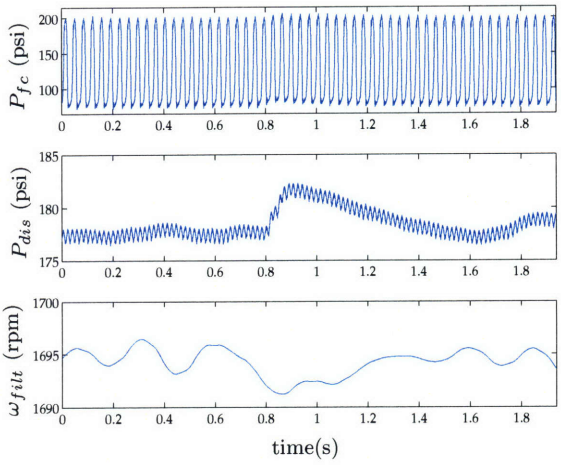


Figure 3-54: Mechanical measurements for 67 ms steady-state slug.

The traces illustrated in Figure 3-53 show that even this small quantity of liquid does indeed have an effect, albeit small, on i_{ds}. Since the noise on this waveform caused by voltage distortion and other unmodeled phenomena would make its use as a robust fault signature somewhat problematic, a wavelet filter bank using a fifth-order Daubechies filter was used to eliminate the structured noise from this signal, generating the waveform seen at the bottom of Figure 3-53 and referred to as $i_{ds,filt}$. The change in the current is much more apparent in this filtered signal, and a simple thresholding or trend checking algorithm could be used to identify the presence of a fault in this waveform.

The ingestion of liquid into the cylinder also has a clear effect on the measured mechanical waveforms illustrated in Figure 3-54. A very small rise in the peak cylinder pressure is apparent around 0.8 sec, where the liquid slug is injected, and a corresponding rise can be seen in the discharge pressure. An estimate of the rotor speed was also generated from the output of the two tachometer channels by counting the number of samples between consecutive edges in the waveform, as is visible at the bottom of this figure. As this estimate also contained a great deal of noise due to artifacts from the processing method, it was also filtered with a similar wavelet filter bank to remove these artifacts. While a slight reduction of the rotor speed during the slugging event is visible in the filtered waveform $\omega_{r,filt}$, the change is not very significant. While these effects are relatively minor for this size of liquid slug, they become more pronounced by injecting larger liquid slugs; the effects of a liquid injection event in which the solenoid is open for 133 ms can be seen in Figures 3-55 and 3-56.

The effects of liquid slugging on both the electrical and the mechanical signals is much more noticeable in these waveforms, as the deviation in $i_{ds,filt}$ is even more significant, and the change in both the front cylinder pressure P_{fc} and the speed $\omega_{r,filt}$ are clearly discernible. The significant change in the discharge pressure P_d is somewhat surprising, as the normal expectation would be that such local changes in the pressure would be attenuated by the large size of the discharge plenum. This pressure rise is not caused by the overpressure in the cylinder, however, but rather by the flashing of the surplus liquid refrigerant as it leaves the cylinder and is entrained with the refrigerant flow exiting through the discharge line. The effect of this flashing was identified by measuring the temperature of the discharge line with a thermocouple and a Campbell 21X datalogger during a series

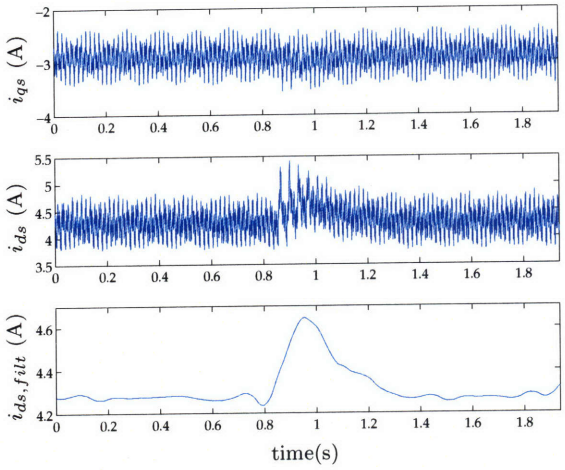


Figure 3-55: Processed electrical measurements for 133 ms steady-state slug.

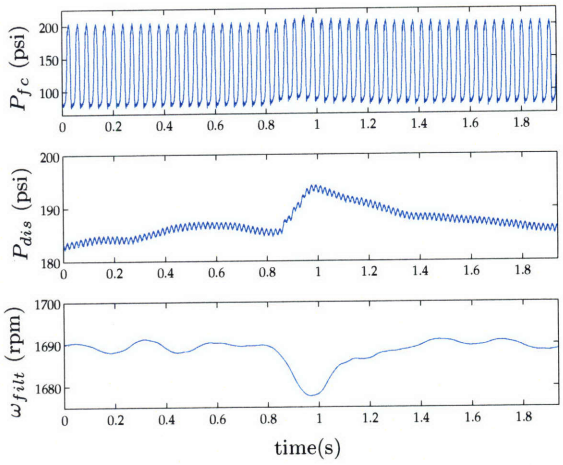


Figure 3-56: Mechanical measurements for 133 ms steady-state slug.

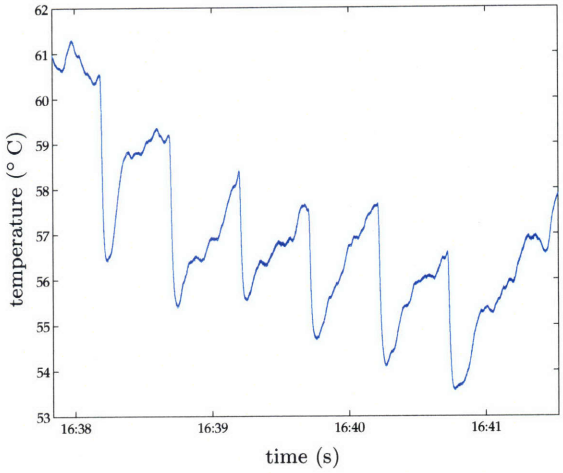


Figure 3-57: Temperature of the discharge line during liquid injection events.

of steady-state liquid slugging events, in which 533 ms liquid slugs were injected into the compressor head 30 seconds apart. The effects on the external temperature of the discharge line is clearly noticeable on the resulting temperature waveform, illustrated in Figure 3-57. The abrupt drop in temperature measured on the outside of the discharge line strongly suggests that liquid refrigerant is flashing as it leaves the compressor, explaining these deviations in the P_d signal. This data serves as additional confirmation of the occurrence of liquid slugging.

While it might appear odd that the front cylinder pressure P_{fc} is being presented for these results even though the liquid slugging apparatus is positioned above the rear cylin-

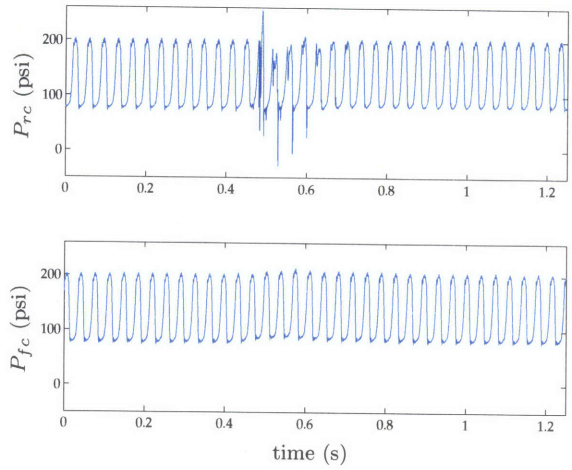


Figure 3-58: Pressure of both cylinders during 133 ms steady-state slug.

der, the justification for examining the front cylinder pressure can be seen by considering both pressure measurements during a 133 ms steady-state slug, as shown in Figure 3-58. While P_{fc} increases somewhat during the slugging event, and gradually returns to the pre-slugging levels, P_{rc} undergoes a number of very large positive and negative pressure spikes during the slugging event. One possible explanation for this fact is that oil-refrigerant mixture present in the liquid slugging apparatus increases the amount of stiction between the valve plate and the reed valve during the slugging event, causing the cylinder pressure to decrease below normal operating levels during the slugging event. Moreover, the output of the pressure sensor during the ingestion of this refrigerant-oil mixture is also highly suspect. The high-pressure stream of refrigerant might also prevent the suction reed valve from closing normally during the liquid injection event, so that the measured cylinder pressure remains much closer to the suction pressure than is normal. As both of these potential interactions are artifacts of the process by which the slugging faults are induced, and do not appear to affect the actual operating state of the compressor, the front cylinder pressure was found to be more useful in demonstrating the representative changes in the cylinder pressure during a steady-state slugging event, since liquid was ingested into this cylinder as well.

Two other fault levels were simulated by injecting different quantities of liquid into the compressor; Figures 3-59 and 3-60 illustrate the effect of injecting 270 ms steady-state slugs into the compressor, while Figures 3-61 and 3-62 illustrate the effect of injecting 533

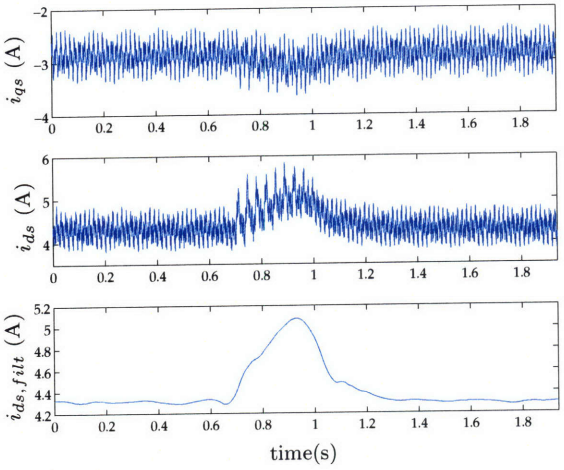


Figure 3-59: Processed electrical measurements for 270 ms steady-state slug.

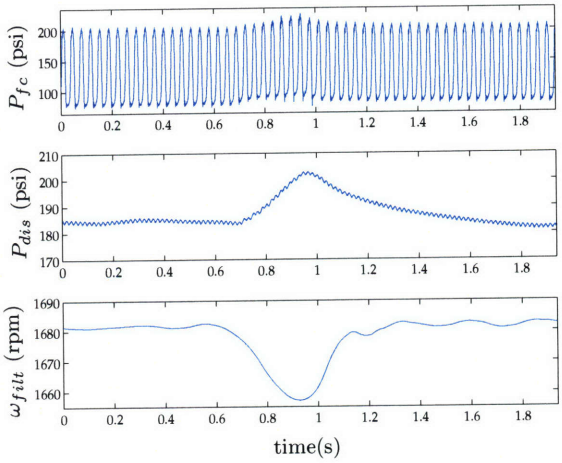


Figure 3-60: Mechanical measurements for 270 ms steady-state slug.

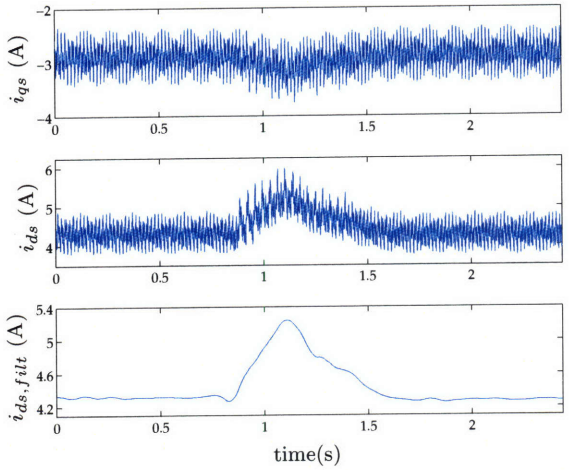


Figure 3-61: Processed electrical measurements for 533 ms steady-state slug.

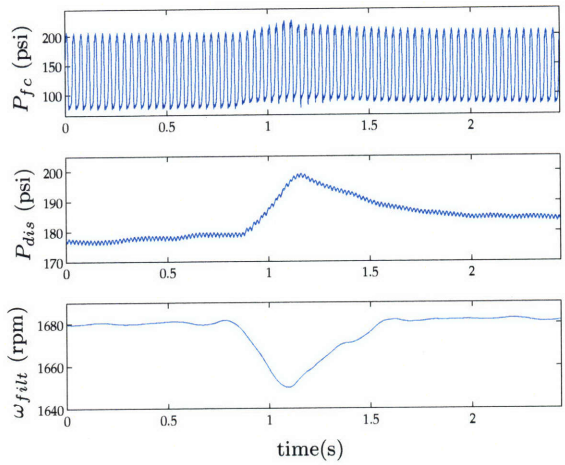


Figure 3-62: Mechanical measurements for 533 ms steady-state slug.

ms steady-state slugs. These plots further illustrate the ability of electrically-based diagnostics to identify changes in the mechanical load on the compressor motor due to liquid slugging. The results shown in Figures 3-59 and 3-61 agree reasonably well with the simulations of the pulsatile torques, in that they show a distinct positive deviation in i_{ds} and a negative deviation in the i_{qs}, although the exact shape of i_{qs} is not replicated from the simulation. This suggests that liquid slugging does indeed have the effect of temporarily causing an increase in the load torque on the compressor motor.

This data supports the use of this preprocessing technique for steady-state FDD meth-

ods because it is able to identify both the presence of the fault as well as the severity of the fault. In examining the set of experimental data, it is clear that the larger quantities of liquid injected into the head produce corresponding increases in the cylinder pressure and discharge pressure, as well as larger reductions in the compressor speed. Moreover, the fact that these steady-state slugging faults manifest changes in the electrical behavior of the system in less than 1 second is also beneficial to the construction of an FDD method; as few other aspects of the system behavior will change on that timescale, these changes in i_{ds} and i_{qs} could isolate the existence of a liquid slugging fault very well.

While this preprocessing method is needed for the detection of transient slugging and has been shown to work well for detecting steady-state slugging, its main disadvantage is that it requires a substantial amount of computational hardware. Another much simpler processing method was therefore investigated for the purposes of detecting steady-state liquid slugging. Since one would expect the electrical power to increase during the ingestion of a liquid slug, and the compressor is approximately balanced during steady-state operation, observations of changes in one stator current can theoretically be used to generate an estimate of the changes in the electrical power during the slugging transient, i.e.

$$\frac{\Delta p_w}{p_{w,0}} \propto \frac{\Delta i_w^2}{i_{w,0}^2}, \tag{3.17}$$

where the electrical power into the stator winding is p_w and the current into the winding is i_w, and the operating point around which the changes occur is denoted by the subscript 0. A simple method of processing the current to look for changes in electrical power was therefore developed, motivated by [101, 112], which consisted of squaring one of the stator currents and passing the resulting signal through a low-pass filter which attenuated the signal's 120 Hz component. A third-order Butterworth filter was used to perform this filtering task. This processing method has the advantage of being very simple and easily implementable in hardware, making it possible to construct a hardware preprocessor that detects liquid slugging when the output of the filter exceeds a set threshold. This processing method was also tested in Matlab and its output is illustrated for the four test sizes of steady-state injection events in Figures 3-63 through 3-66.

These plots illustrate the effectiveness of even very simple electrically-based methods

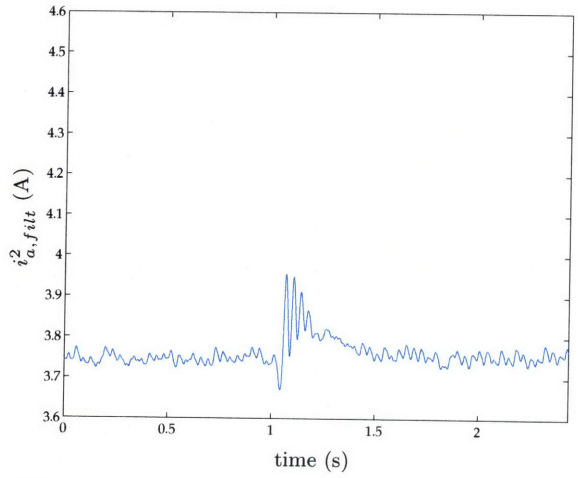


Figure 3-63: Filtered squared current for 67 ms steady-state slug.

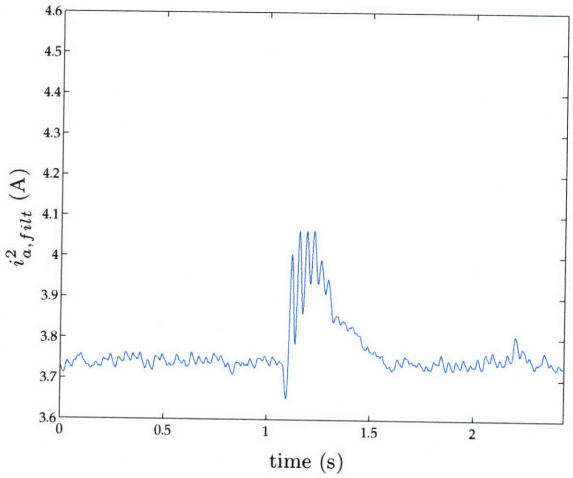


Figure 3-64: Filtered squared current for 133 ms steady-state slug.

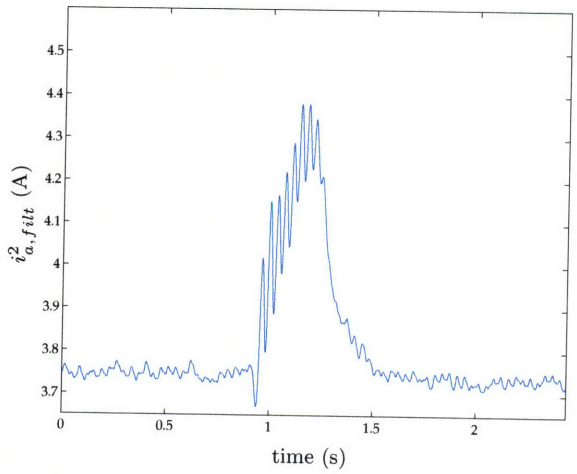


Figure 3-65: Filtered squared current for 270 ms steady-state slug.

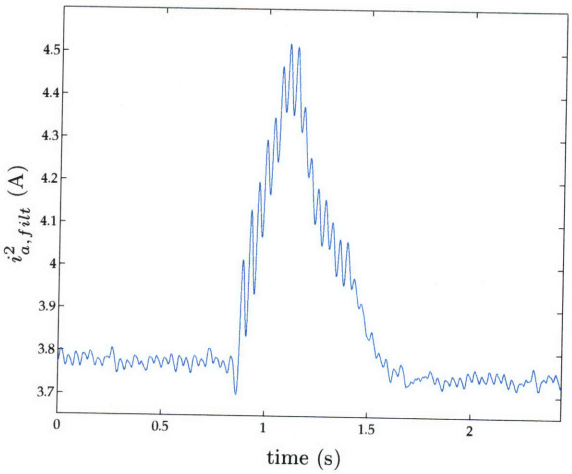


Figure 3-66: Filtered squared current for 533 ms steady-state slug.

for liquid slugging detection. This preprocessing method is able to both identify the presence of liquid slugging, and generate a waveform which is sensitive to the total volume injected as well. Depending on the desired complexity of the FDD system, either of these approaches appear to detect steady-state liquid slugging very well down to very low fault levels. While this method would be effective if a detector for steady-state slugging was desired, the ability of the other preprocessing method to successfully detect steady-state slugging events is useful because the preprocessor is necessary to identify the presence of transient liquid slugging. The sensitivity of both of these fault detectors to low fault levels can also be noted via the fact that hundreds of such slugs have been ingested by the compressor in the experimental apparatus at this point, and no sign of damage of impending failure is evident.

3.4.4 *Transient Liquid Slugging*

Transient liquid slugging phenomena were studied via the same experimental methodology used to investigate steady-state slugging. Due to the many sources of variation which affect i_{ds} and i_{qs}, as seen in §3.4.2, these sources of variation will also be studied separately in order to ascertain their influence on the electrical transient during a slugging event. A number of induction motor simulations were also run in order to investigate the potential changes in i_{ds} and i_{qs} during a transient slugging event. The load torque $\beta\omega_r^2$ during the motor's startup transient was changed between these simulations to simulate the effect of liquid in the cylinder. This model for the liquid slugging was used because of its intuitive appeal; one would expect that more torque would be required for the motor to repeatedly clear the cylinder of liquid ingested from the head and suction line. As in §3.4.3, the parameters for the simulated motor and the changes in torque were only chosen to study the comparative changes on i_{ds} and i_{qs}. The results of these simulations can be seen in Figures 3-67 and 3-68.

While there is initially no difference between the slugging and nonslugging behavior for either i_{ds} and i_{qs}, there is a large difference between the two conditions after approximately 0.17 sec in both waveforms. This effect makes some physical sense, as the load torque $\beta\omega_r^2$ is minimal while ω_r is small, so that most of the energy is accelerating the inertia of the rotor. As the rotor accelerates, the torque $\beta\omega_r$ becomes much larger, and has a

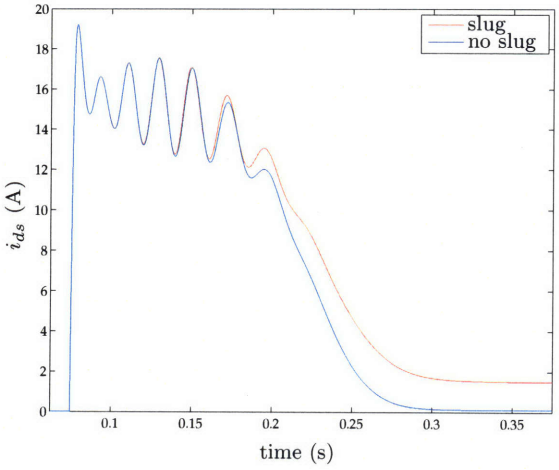


Figure 3-67: Simulated i_{ds} for higher load torque in comparison to lower load torque.

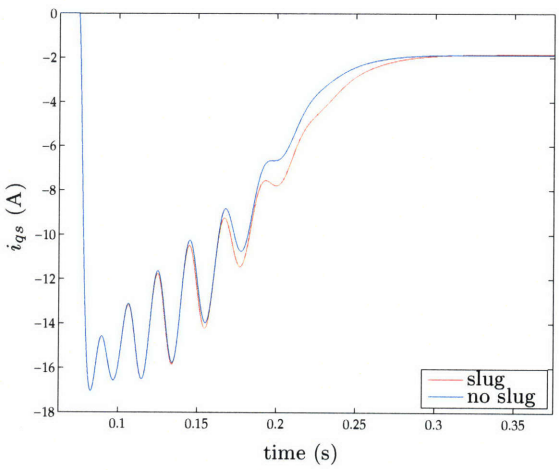


Figure 3-68: Simulated i_{qs} for higher load torque in comparison to lower load torque.

substantial effect on i_{ds} and i_{qs}.

In studying the effects of the initial piston position, the initial electrical angle, and the compressor temperature on the electrical start transient, it must be noted that the thermal changes do not have the same experimental significance for the transient slugging diagnostics as much as the other two phenomena do. In general, the operation of the compressor for any significant length of time (i.e. longer than a minute or two) will cause the temperature of the compressor to be higher than ambient temperature. These higher compressor temperatures are not conducive to liquid slugging, as liquid refrigerant in the compressor head or cylinder will either evaporate or migrate elsewhere in the refrigerant loop. This research will therefore focus primarily on evaluating the performance of the slugging diagnostic with variations caused by the piston position and electrical angle.

The first set of experimental data illustrates the starting electrical signature i_{ds} and the corresponding rear cylinder pressure P_{rc} for two slugging transients and two nonslugging transients. Each of these waveforms were collected by turning on the motor after it had remained off overnight, ensuring that the motor temperature is nearly the same in all cases, and all waveforms were collected when the motor was started with the same initial piston position and the same electrical angle $\theta_e = 0$. Figure 3-69 shows very clear differences between the two sets of traces corresponding to slugging and nonslugging starts, and Figure 3-70 indicates that the differences in these waveforms can be attributed to elevated

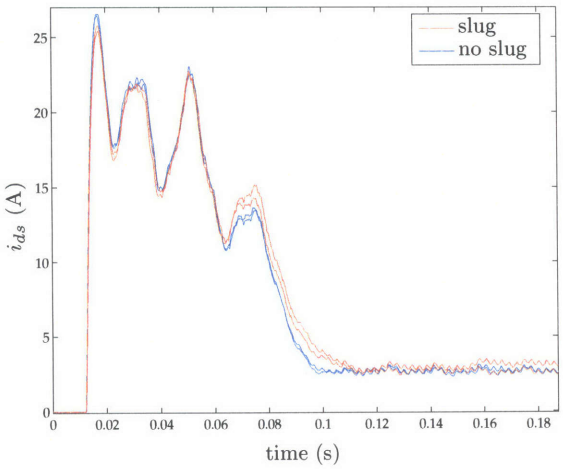


Figure 3-69: i_{ds} during transient slugging for 2 slugging tests in comparison to 2 cold nonslugging tests.

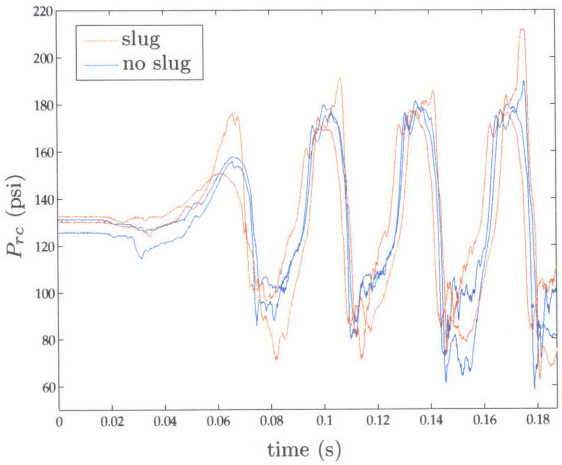


Figure 3-70: P_{rc} during transient slugging for 2 slugging tests in comparison to 2 cold nonslugging tests.

cylinder pressures.

These results demonstrate the effectiveness of electrically-based methods for the identification of transient liquid slugging. Moreover, it is notable that there is considerable qualitative agreement between the shapes of the simulated i_{ds} and the experimentally observed i_{ds} in that the effect of the liquid slugging on i_{ds} is manifested predominantly at the end of the transient. This fact further suggests that the variation caused by different starting angles will not have a large effect on the detectability of the fault, as the end of the transients with different initial electrical angles were observed in §3.4.2 to be very similar. In constructing a fault detector for transient liquid slugging, it is clear that the interval between 0.6 and 0.12 sec should be the primary section of time analyzed to distinguish between faulty and non-faulty behavior.

The data illustrated in the slugging data also makes it clear that the liquid slugging transient are not very repeatable, despite the fact that the solenoid valve was opened for the same interval of time in both of the experiments. This variation can be partly ascribed to the particular construction of the slugging apparatus; since the liquid is being injected into the head instead of directly into the cylinder, the liquid refrigerant could collect in any number of locations, including either of the cylinders, the compressor head, or the discharge valve and suction line. While the refrigerant would ideally have been injected directly into the cylinder, the experimental implementation of such an apparatus would

have increased the cylinder's clearance volume by many orders of magnitude, dictating that the present approach be adopted.

In the process of experimental investigation it became clear that the compressor behavior during a cold start had significant variations that were manifested in both the pressure waveforms and in i_{ds} and i_{qs}. One possible source for this variation is due to the compressor oil which was carried out of the condenser and collected in the slugging apparatus. This oil could seep through the slugging apparatus while the compressor was off and gradually accumulate in the compressor head and in the cylinder[4], so that the following time that the compressor was turned on would generate higher-than-expected cylinder pressures due to this oil slugging. While the detection of high cylinder pressures due to oil slugging could also potentially be useful, this phenomenon made it difficult to compare the liquid slugging starts with starts in which the cylinder pressures were representative of nonslugging behavior. The oil slugging was mitigated by injecting liquid slugs from a fresh tank of refrigerant, rather than using refrigerant from the bottom of the condenser. This process did disturb the mass balance in the system, but the quantities of refrigerant injected were relatively small, and the system was evacuated and recharged on a regular basis to minimize the effects on the system.

Though this is a somewhat plausible explanation for the observed variation in the nonslugging transients which were collected when the compressor was cold, the possibility of the existence of other phenomena which also had the same effect suggested the development of other methods of evaluating the effectiveness of the transient slugging FDD method be used. To ensure that the slugging data could be compared with true nonslugging data, the nonslugging data was not collected when the compressor was cold, but rather when the compressor was "warm," meaning that it was run for four minutes, and then was switched off for four minutes, before the nonslugging start transient was collected. Data collected during this warm transient were assured to be free of any slugging phenomena, since the temperature of the compressor was higher than the ambient temperature. Measurements of the compressor shell temperature confirmed that the temperature of the motor was only slightly higher than it was before the test, suggesting that the

[4]This oil was observe to slowly accumulate on the outside of fittings and on the compressor head, suggesting this mode of operation.

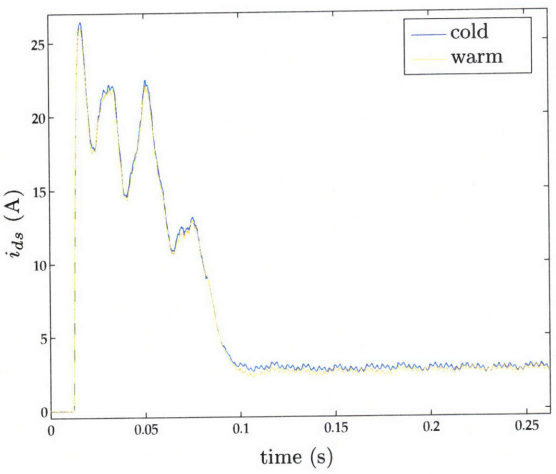


Figure 3-71: D-axis current for both cold and warm nonslugging tests.

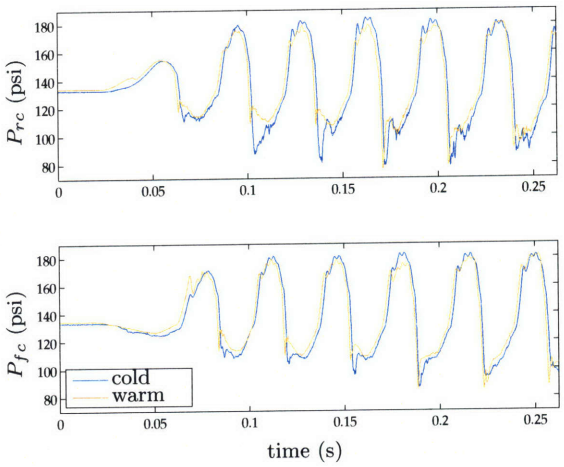


Figure 3-72: Cylinder pressures P_{fc} and P_{rc} for both cold and warm nonslugging tests.

thermal changes in the compressor transients would not dominate the effect of the liquid slugging. Figures 3-71 and 3-72, which illustrate two start transients with the same initial piston position, one of which is a cold nonslugging transient, without the spurious high peak pressures, and the other of which is a warm start transient. The high degree of overlap in i_{ds} in these waveforms suggests that the thermal effects will not dominate the effects from liquid slugging.

Qualitatively similar results to those shown in Figures 3-69 and 3-70 can be seen in comparing liquid slugging transients to the warm nonslugging transients. As usual, the sources of variation between these sets of transients were minimized by identifying transients with the same initial piston position and setting the initial electrical angle $\theta_e = 0$. The current i_{ds} is illustrated in Figure 3-73, with the tail of the transient magnified to illustrate the effects of the higher torque in Figure 3-74, while i_{qs} is shown in Figure 3-75 with the same magnification illustrated in Figure 3-76. The agreement between the plots and the simulated compressor behavior can also be noted.

Measurements of both the front and the rear cylinder pressures P_{rc} and P_{fc} during the set of electrical transients shown in Figures 3-73 through 3-76 show higher peak cylinder pressures during the slugging transients than during the nonslugging transients, as is expected. These cylinder pressures are illustrated in Figure 3-77. It must be noted that the difference between the system pressure before the compressor turns on and the peak cylin-

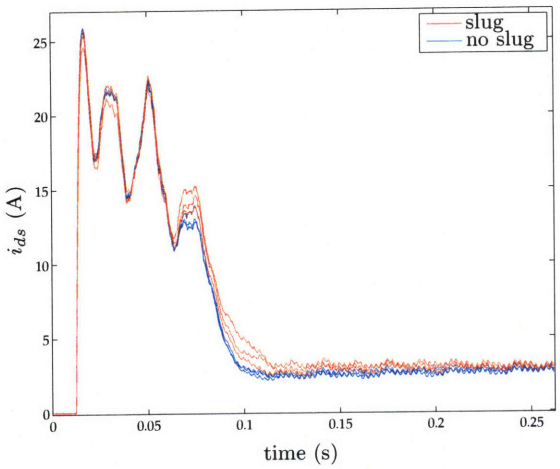


Figure 3-73: D-axis current during transient slugging with the compressor starting in the same position, compared to slightly warm nonslugging starts.

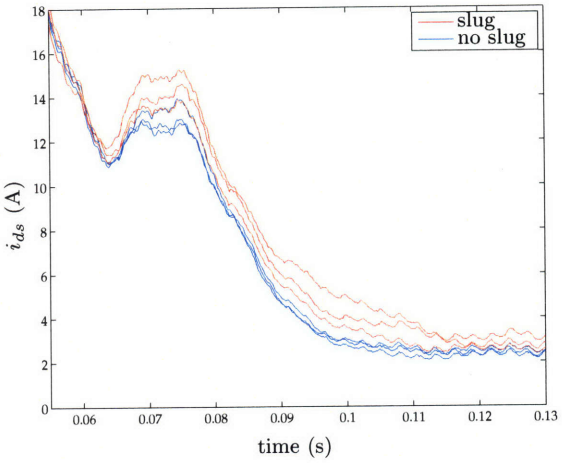


Figure 3-74: Zoomed picture of the same data.

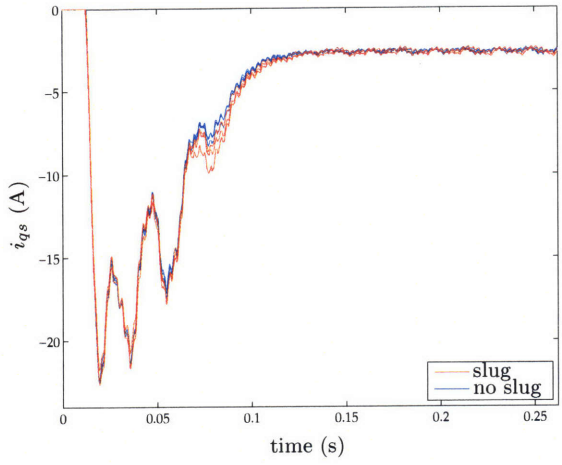


Figure 3-75: Q-axis current during transient slugging with the compressor starting in the same position, compared to slightly warm nonslugging starts.

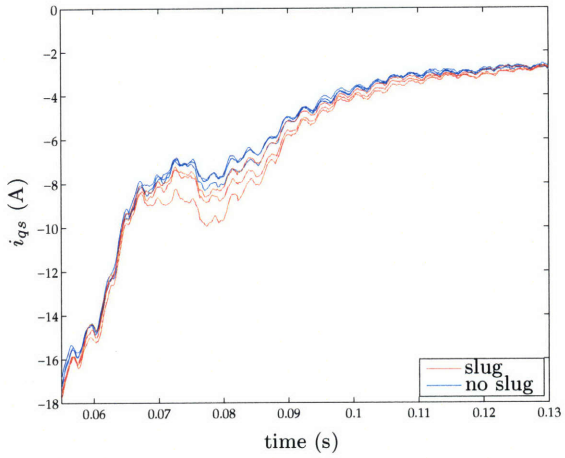


Figure 3-76: Zoomed picture of the same data.

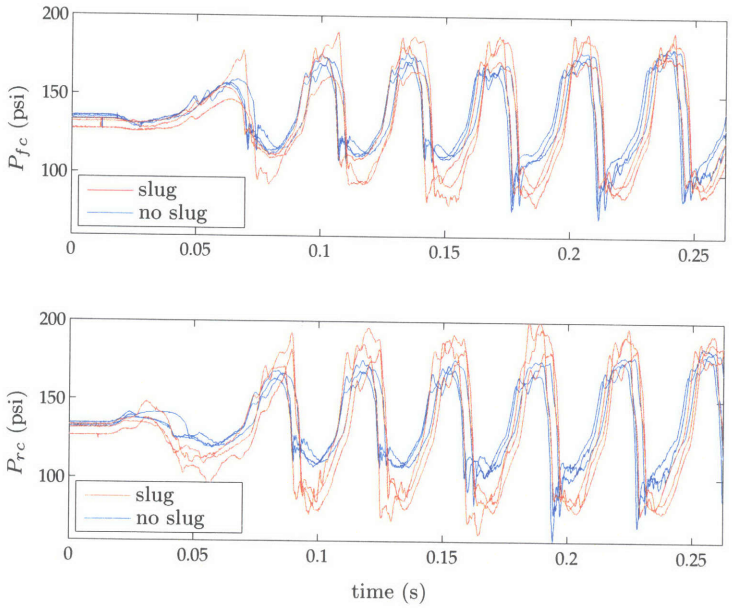


Figure 3-77: Cylinder pressures during transient liquid slugging events.

der pressures during its operation in the few cycles dominates the mechanical load seen by the compressor motor. For example, the slugging pressure transient in Figure 3-77 with the lowest initial system pressure corresponds to slugging transient i_{ds} in Figure 3-73 with the highest peak value between 0.06 and 0.12 sec.

In comparing the measured peak pressures with the expected peak pressures, it is notable that these pressures are much lower than would be expected from the literature. Given the measurements of 1000-2000 psi reported in [131, 133, 137] and the experimental damage attributed to liquid slugging seen in Figures 3-6 and 3-7, the peak cylinder pressures would be expected to be far higher than those measured, which are only 30-50 psi greater than normal operating pressures. While this may in part be caused by small quantities of liquid ingested into the cylinder, it is likely that the surprisingly low peak cylinder pressures during slugging events are caused by the geometry of the compressor itself. Since displacement of the cylinder is small (approximately 23 cc), only relatively small quantities of liquid can be ingested by the compressor installed in the experimental air-conditioning system. Furthermore, as the area of the discharge valve is approximately 20% of the cylinder bore, relatively little additional torque will be required for the compressor motor to push all of the liquid out of the discharge port. An indication of the

relative difference between this compressor and other compressors which have exhibited problems with liquid slugging can be seen in the the specifications given in [131], for which overpressures on the order of 1500 psi are reported, which suggest that the displacement of the compressor under examination is approximately 13,000 cc. Clearly, this difference in the equipment could explain some of these observed effects.

Though the experimental compressor is relatively insusceptible to liquid slugging faults, it is in many regards an excellent platform to test the effectiveness of the detection method. If slugging can be detected in this compressor, it should be much easier to detect on other larger compressors that experience much higher peak pressures during liquid slugging events. The sensitivity of the electrical waveform to even these small changes in the peak pressure suggests that this diagnostic approach will be very successful at detecting slugging in larger equipment well below the threshold of damage, as many liquid slugs of this size have been injected into this compressor with no apparent consequences to the health of the machine.

In evaluating the performance of this transient liquid slugging FDD method, it was also necessary to examine the effect of changes in the electrical angle. The initial electrical angle of the voltage when the compressor turned on was changed to $\pi/2$, π, and $3\pi/2$ during three slugging tests by using the control system and the solid-state relays, and the resulting transient waveforms for i_{ds} were compared to transients with the same initial position and with $\theta_e = 0$. These results are illustrated in Figures 3-78, 3-79, and 3-80. The effects of liquid slugging are evident in the i_{ds} waveforms as well as in the pressure waveforms, suggesting that the slugging diagnostic method will function regardless of the variations in the initial electrical angle. It is also notable that the characteristics of the variation in the electrical angle seen in §3.4.2 are repeated here; the difference between the slugging and the nonslugging transients with the same initial position but with starting angles that differ by π is the same as the difference between the slugging and nonslugging transients with no difference in their initial electrical angle (i.e. $\theta_e = 0$).

The effect of changes in the piston position were also studied for their effect on the transient slugging diagnostic. The range of the previous plots implies that the success of the diagnostic does not depend on the specific angle of the piston, as each set of plots was observed at a different initial piston position. This variation was studied by collecting

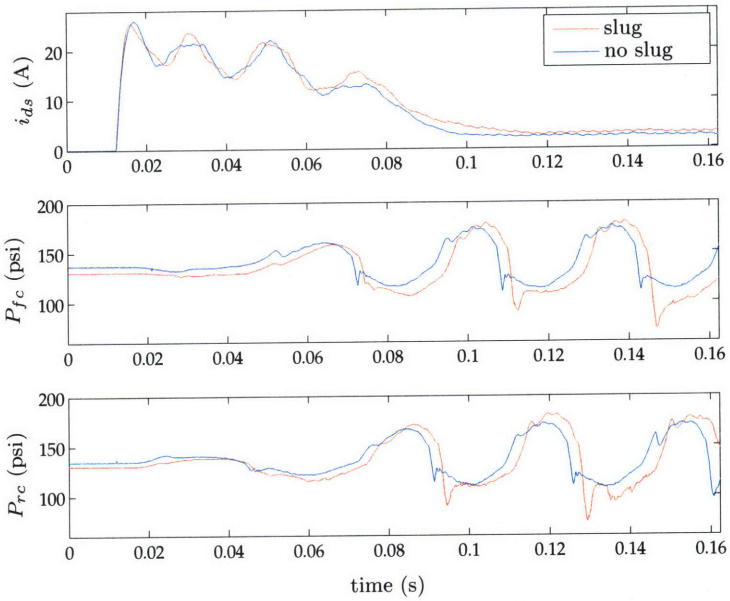


Figure 3-78: Measured variables during transient slugging start with initial electrical angle $\theta = \pi/2$.

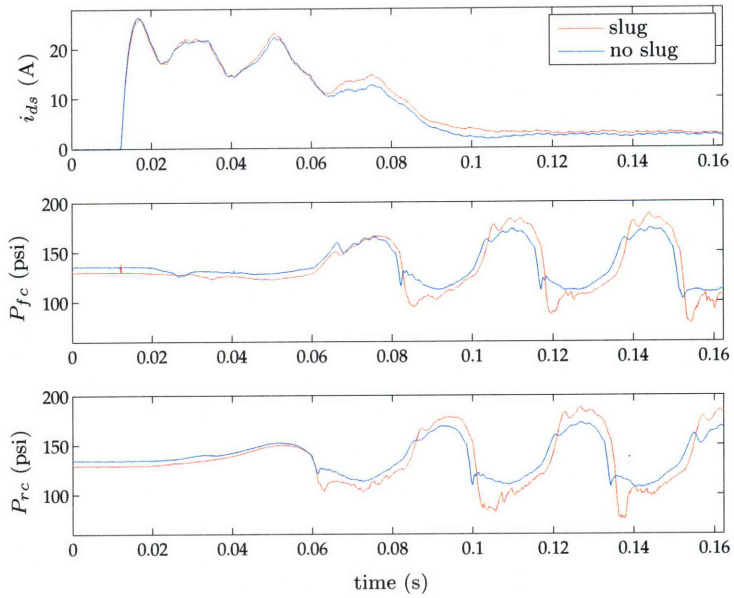


Figure 3-79: Measured variables during transient slugging start with initial electrical angle $\theta = \pi$.

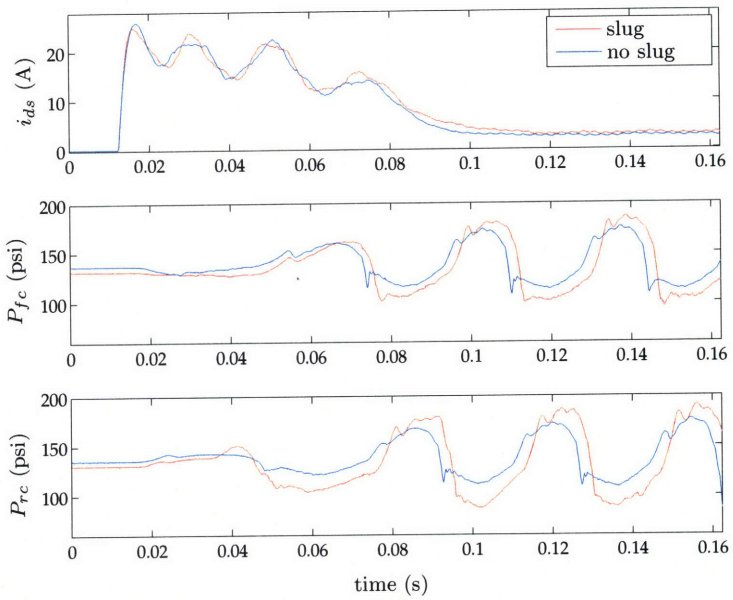


Figure 3-80: Measured variables during transient slugging start with initial electrical angle $\theta = 3\pi/2$.

a set of slugging transient starts and a set of nonslugging transient starts during which the initial electrical angle and the temperature of the motor were the same, but the initial piston position was varied. Sets of both the warm nonslugging starts and the cold nonslugging starts were collected to compare each to the set of slugging starts, although the set of cold nonslugging starts may not accurately describe the nonslugging behavior of the compressor. Figures 3-81 and 3-82 illustrate these sets of data in comparing the cold nonslugging data to the slugging data, while Figures 3-83 and 3-84 show the comparison between warm nonslugging data and the slugging data. Pressure waveforms are not illustrated for these datasets, as the high degree to which the waveforms overlap makes a plot of these waveforms basically meaningless.

These plots effectively illustrate the crux of the challenges posed in attempting to identify liquid slugging in compressors which require relatively little additional torque to eject the liquid from the cylinder. A clear difference is visible between the mean of i_{ds} for the warm nonslugging data and i_{ds} for the slugging data, but the distributions of each of these waveforms clearly overlap. This demonstrates that the variation in the electrical transients due to the piston position are significant, as these variations are of nearly the same scale as the changes due to the liquid slugging. Comparison between the cold nonslugging data

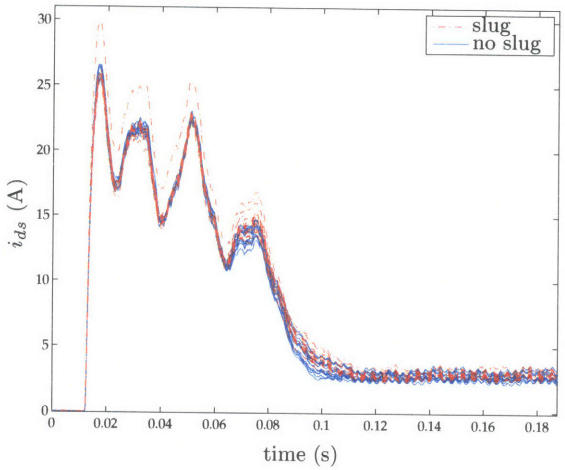


Figure 3-81: D-axis current during transient slugging for all slugging tests in comparison to the cold nonslugging tests.

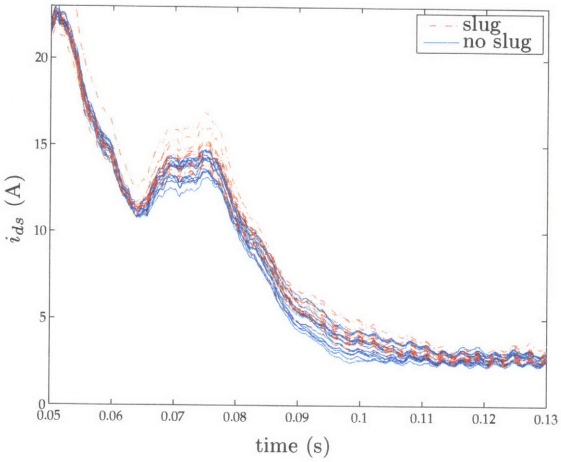


Figure 3-82: Zoomed picture of the same data.

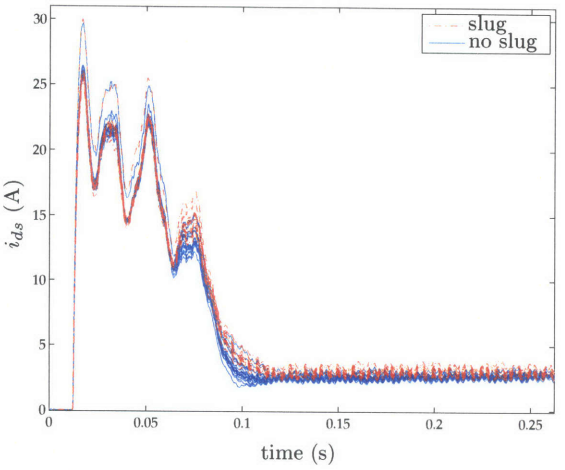


Figure 3-83: D-axis current during transient slugging for all slugging tests in comparison to the warm nonslugging tests.

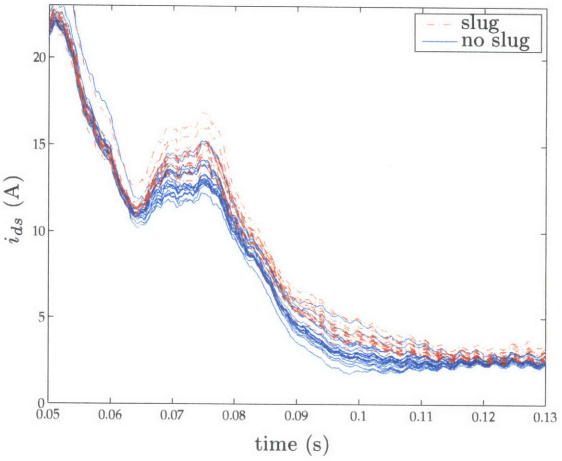


Figure 3-84: Zoomed picture of the same data.

and the slugging data shows an even greater degree of overlap, due in part to the potential for some level of slugging-like behavior in the ostensibly "nonslugging" transients. Even in this plot, however, a perceptible difference is discernible between the means of the two waveforms.

As is apparent from the observed data, the experimentally observed liquid slugging phenomena are very near the lower thresholds for successful fault detection. With validation requiring additional experimental study, results from this research suggest that the fault detection methods developed in this research will be able to detect repeated liquid slugging events. Such an FDD method could be initialized by first collecting a series of observations of healthy compressor starts and calculating the mean of this series of waveforms. After this initial training period, the average of a number of the most recent starts (perhaps 15-20) could be calculated every time the compressor started; if liquid slugging events occurred during a series of compressor starts, the average would begin to deviate from the data collected during the training period and liquid slugging could be declared. Unfortunately, this diagnostic method would not be effective for detecting individual slugging starts, as it will be impossible to discern whether or not the variation in a given electrical transient is caused by variation due to different piston position or variation due to liquid slugging without any additional information.

Such concerns are largely secondary, however, as the experimental data indicates that liquid slugging does not present a significant danger of damage for compressors like that which was used in the experimental apparatus. Practically speaking, it would not be effective to implement a method for detecting liquid slugging faults on this compressor. For larger compressors with correspondingly higher peak pressures during slugging events and which are more susceptible to damage, this research indicates that FDD methods which are based upon the analysis of the compressor's electrical inputs would be able to reliably detect both transient and steady-state liquid slugging faults before they caused significant damage.

3.5 Summary

Liquid slugging is a relatively common source of faults in reciprocating compressors which is difficult to detect with mechanical sensors; research presented in this chapter suggests that electrically-based fault detection methods are effective at detecting the fault with high sensitivity. A preprocessing method based upon the Park transformation was used to identify the presence of liquid slugging from measurements made at the compressor's electrical terminals, and the effectiveness of the proposed fault detection method was tested and validated on an experimental air-conditioning apparatus incorporating a set of electrical and mechanical sensors which was built expressly for that purpose.

Chapter 4

Refrigerant Charge Diagnostics

To put it glibly, the air-conditioning fault which has probably received the largest amount of public attention in the last 25 years concerns leaks in the refrigerant circuit. This and other related faults have significant consequences both for the air-conditioner and for the users of the air-conditioning system. This chapter will begin by discussing the prevalence and the impact of these faults, and then will survey previous work which has been done in developing FDD methods to identify and mitigate their effects. The FDD approach and methodology used in this research will then be described, and this will be followed by a demonstration of the effectiveness of these methods as experimentally investigated in a residence in North Carolina.

4.1 Motivation

Most air-conditioning systems are designed to use a specified mass, or charge, of refrigerant in the refrigerant circuit; when the amount of refrigerant present in the circuit falls below this value, the system is generally referred to as being undercharged. As with the other faults discussed in Chapters 2 and 3, refrigerant undercharge can be caused by a number of different phenomena, and can also be the cause of or contribute to a number of abnormal behaviors of the air-conditioner. This section will explore some of the possible reasons that an air-conditioning system might be undercharged, as well as the effects of this state. A number of fault surveys will also be presented which underscore the widespread nature of of this fault, and lend additional perspective to the practical utility of an FDD method

which could effectively detect it. Some methods for detecting this fault which have previously been developed also will be presented in order to suggest the variety of different approaches which could be used to detect it, as well identify some of the strengths and weaknesses inherent in any of the possible approaches.

4.1.1 Background

In general, the operation of most residential air-conditioning systems is based upon the use of the latent heat of vaporization of refrigerants flowing through heat exchangers to transfer energy efficiently from the indoor air flowing through the evaporator to the outdoor air flowing through the condenser. Since the most efficient operation of these systems occurs when most of the energy transfer occurs through evaporation or condensation, it is essential that the mass of refrigerant used to transfer this energy be calibrated appropriately for the expected cooling load so that the refrigerant will not be heated far above the heat of vaporization at the operating pressure in the evaporator, or cooled far below the heat of condensation at the operating pressure in the condenser. Naturally, the cooling load and ambient temperatures of the heat exchangers will vary somewhat, but it is clear that the designer of an air-conditioning system must specify the mass of the system in order to achieve expected performance bounds. Too little refrigerant will result in a reduction of the cooling capacity of the equipment, since the superheated refrigerant in the evaporator cannot absorb as much thermal energy as evaporating refrigerant, and too much refrigerant could result in system pressures which are higher than designed and other undesirable phenomena, such as liquid slugging. Eliminating either of these conditions is therefore desirable, but this research will focus on the detection of the state of undercharge.

Perhaps the most well-known reason for identifying refrigerant undercharge in an air-conditioning system is that leaks in the refrigerant circuit, which are perhaps the most common reason for the presence of the undercharged state, can have an effect on the environment. Notwithstanding the current political debate regarding global warming, it is clear that some of the common refrigerants incorporating chlorofluorocarbons (CFCs) can contribute to the depletion of the ozone layer. Fault detection systems which could accurately detect and undercharge condition could mitigate such effects by reducing the quantities of refrigerant lost through leaks. Moreover, these refrigerants, such as R-12 and

R-22, can be fairly expensive: a thirty pound tank of R-22 (perhaps the refrigerant which has been used most frequently in residential air-conditioners) costs approximately $130 at current prices. By minimizing the amount of refrigerant lost to the environment via such leaks, the cost of servicing air-conditioning units could be reduced substantially.

Aside from concerns directly connected to the loss of refrigerant, undercharge faults can affect the operation of the air-conditioner in a number of other different ways. While the most significant effect of an undercharged system is a reduction in the unit's cooling capacity, this does not necessarily always have the result of decreasing the ability of the air-conditioner to cool the space. While small discrepancies in the system charge can be mitigated through modulation of an active expansion valve, such as a thermostatic expansion valve (TXV) or an electronic expansion valve (EXV), reduced cooling capacity is typically mitigated via the thermostat or control system used to regulate the cycling of the air-conditioner; if the cooling capacity of the air-conditioner is decreased, the air-conditioner will run longer to cool the space down. Correspondingly, undercharge faults can cause the power consumption of the air-conditioner to increase. The higher refrigerant temperatures at the outlet of the evaporator during undercharge faults can also affect the health of the compressor, since high temperatures can cause thermally-related faults, including reduced lubrication, higher motor temperatures, and reduced equipment lifetime.

While leaks in the refrigerant circuit are perhaps the most common cause of undercharge faults, another common cause is poor equipment service. To a large extent, the difficulty of determining the proper charge for an air-conditioning system in the absence of documentation makes this service fault understandable, as a service technician may not be able to determine the proper level of charge for the system by using only a gauge set. Nevertheless, undercharge faults can have a significant effect on system performance regardless of their source, be it improper commissioning, poor service, or leaks.

A number of different surveys have been published which study the incidence of incorrect charge faults, due to the effect of refrigerant undercharge on the aggregate population of air-conditioners in cities, as well as the effect of refrigerants on the atmosphere. One study conducted from 2001-2004 of 503 air-conditioners at 181 buildings in the Pacific Northwest found that 46% of the units had an incorrect refrigerant charge (not within 5% of the rated charge) [31]. Another set of studies conducted between 1998-2001 ex-

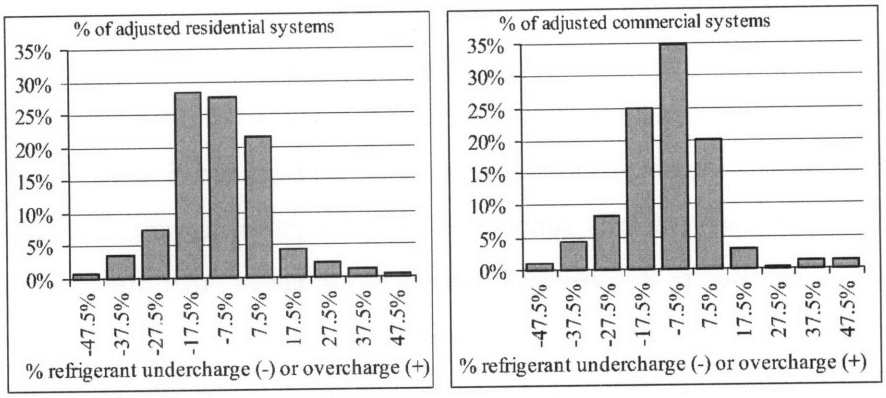


Figure 4-1: Results of incorrect refrigerant charge survey, from [42].

amined approximately 13,000 commercial and residential air-conditioners located in California. One notable statistic from these studies [100] regarding 4,168 of the commercial air-conditioners with fixed orifice expansion (FOX) devices and TXVs is that 72% of the units have the wrong refrigerant charge. Another survey from California during the same period [42] examined a cross-section of 8873 residential units and 4385 commercial units, and found that 56% of the commercial units had an incorrect refrigerant charge, 60% of the commercial systems had an incorrect refrigerant charge. The deviation from the expected charge level for the range of these units was also presented in this paper, and is in Figure 4-1. These plots make it clear that poor service and refrigerant leaks are both pertinent causes of incorrect refrigerant charge, as the presence of overcharge can only be attributed to poor service, while the larger number of units with undercharge than overcharge could be ascribed to the presence of both leaks and poor service in a system.

One of the important consequences of this relatively high rate of faults in air-conditioners is that the energy consumption of the unit can be markedly increased as it tries to cool spaces down with equipment that has a reduced cooling capacity. One estimate of the potential increase in energy costs due to improper charging of residential air-conditioning units is that such faults increase energy usage in homes by 20% and that 17.6 Terawatt-hours of energy are wasted nationally via such faults [77].

Due to the range of causes and effects of refrigerant undercharge faults, an automated FDD system which could detect the presence of this fault would clearly be beneficial for a variety of reasons. Such a system would not only benefit the equipment owner via

more efficient and effective operation of the air-conditioning equipment, but they would also assist service technicians by making it possible to more effectively repair the air-conditioning unit, as well as society at large by decreasing energy usage of this equipment on a widespread basis. As such benefits are widely acknowledged, a number of previous research efforts have been directed towards accurately detecting this fault; these efforts will be reviewed in the next section.

4.1.2 Previous Work

It is not altogether surprising that there is a wealth of published literature on the effect of undercharge on air-conditioning units and methods of detecting this fault, given the impact that the refrigerant charge has on the performance of the air-conditioner. In one particularly salient paper [57], the authors study the effect of undercharge on the performance of a 3-ton system with a fixed-orifice expansion valve as the level of undercharge varies from 50% to 100% of the normal system charge. The results of their study concluded that there were no appreciable changes in the unit's performance when the charge is within 10% of its specified level, but that the performance decreased rapidly when the system charge falls below this point. They also study the financial impact of a charge level which is 15% below the specified charge, and conclude that the service expense is recouped within 3-4 months of the repair. Similar studies were also performed in [48], which studied the effect of undercharge and overcharge on systems with capillary tube expansion devices and reached comparable conclusions.

Another direction of study, pursued in [58], experimentally investigated the effect of incorrect refrigerant charge on system parameters for the purpose of identifying a set of measurements which are very sensitive to the charge level. The behavior of a water-to-water chiller was experimentally studied via a set of 10 temperatures and 2 pressures, and the authors reach the conclusion that the most sensitive measurements to changes in the refrigerant charge are the degree of superheat at the evaporator outlet, the degree of subcooling at the condenser outlet, and the discharge pressure.

The related subject of charge inventory, or the fraction of refrigerant charge present in the different system components, is studied in [64]. An accurate determination of charge inventory is important for methods which use temperature measurements, as the locations

for the sensors are dependent upon the quantities of liquid in the system components. The authors estimate the quantity of refrigerant in each of the components using measurements of the refrigerant and air-side conditions, and then estimate the total charge present in the system by by summing up the contributions from the individual components. The work was validated experimentally on 3 different unitary air-conditioners with TXVs.

A number of different approaches to refrigerant undercharge detection have been developed and tested, with a similar number of different variations in the architecture of the sensor network and the FDD approach. Many of these refrigerant undercharge diagnostics are included as part of a more comprehensive FDD system, such as those described in Chapter 1, and the variety of techniques used to implement FDD methods in that chapter can be applied to the detection of refrigerant undercharge as well. In particular, the methods used in [93, 94, 115] were applied to the development of a FDD system specifically designed for identifying refrigerant charge in [77]. The approach used in this method relied upon a subset of four of the temperature measurements used in [94], and validated the performance of the method on four distinct systems, including a heat pump and a residential split air-conditioning system.

Other researchers have developed systems exclusively designed to detect undercharge using similar methods. One such FDD system, discussed in [147] analyzed a set of 16 temperature, pressure, and mass flow measurements of the steady-state behavior of a liquid-to-liquid chiller, and identified the presence of undercharge using artificial neural networks that were trained on a set of validated normal performance data. This system could detect both undercharge and overcharge at a fault level of 33%. In comparison, a model-based approach for identifying undercharge is used in [107]; this approach analyzes the innovations generated by a Kalman filter that is tracking the performance of a multi-regressive system model. The FDD method was validated using a liquid-to-liquid chiller controlled by a TXV, with a wide variety of temperature, pressure, mass flow, and power measurements, and the authors found that the residual for the suction pressure yielded the most reliable fault signatures.

While both of the above approaches use large numbers of sensors, a limited sensor approach is taken in [99] by evaluating the sensitivity of the evaporating temperature and the subcooling temperature to refrigerant undercharge. The authors experimentally inves-

tigated the sensitivity of these measurements on a 2-ton heat pump with both a FOX and a TXV, and found that the evaporating temperature was a reliable indicator for the FOX but not the TXV, while the liquid subcooling temperature was found to be a good indicator in both cases.

A related use for a refrigerant undercharge FDD method is investigated in [11], in which a system is developed that detects leaks in supermarket refrigeration systems. The approach taken in this system is somewhat different, in that it compares predictions of the liquid volume fraction in a liquid receiver with the predicted value, with differences being indicative of a fault. A set of eight measurements of temperatures, pressures, and speeds are used to train a neural network which describes the system, and a Bayesian approach is used to detect the probability of the loss of refrigerant, given measured values of a number of other observed variables. This FDD method was validated with data collected from an experimental system, and they identified a leak equivalent to 1% of the full system charge.

While the low cost of temperature measurements makes the development of FDD methods that incorporate these measurements attractive, other sensing approaches can also be used to detect refrigerant leaks. Some of the considerations and potential approaches for identifying refrigerant leakage are discussed in [135]; potential approaches used to detect faults include infrared photometry, the use of semiconductor cells, ultrasound detection, and the use of ionization and platinum diodes.

One particularly notable aspect of many of these methods is that they require the use of a large number of thermal and mechanical sensors. As discussed in Chapter 1, these sensors can be both expensive and susceptible to bias and failure, potentially compromising the reliability of the FDD method. In addition, the susceptibility of compressors to electrical faults, such as shorted windings, has motivated compressor and air-conditioner manufacturers to develop diagnostic systems which identify faults in the compressor motor. In response to both of these concerns, the objective of this research was the development of a method or set of methods for identifying refrigerant undercharge based upon measurements of the compressor's power. Such an FDD method would further extend the capabilities of the electrically-based FDD systems which have already been developed, and the increased reliability of the electrical measurements could improve the performance of the FDD method over the lifetime of the air-conditioning unit. Rather than rely principally

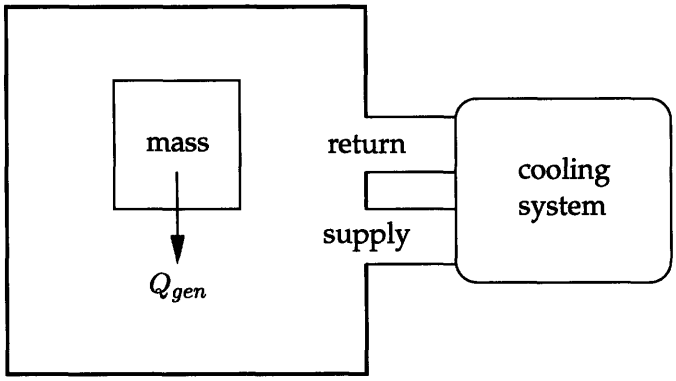


Figure 4-2: Schematic diagram of a residence, adapted from [103].

on the transient behavior of the air-conditioner, however, this diagnostic approach looks at another characteristic of the compressor: its cycling behavior.

4.2 Cycling Detection Methodology

In order to provide background for the approach used to detect refrigerant undercharge that was developed in this research, it is useful to review another method that identifies undercharge via changes in the cycling behavior, as described in [102, 103]. This method can be developed by using a very simple model of a residence, illustrated in Figure 4-2.

Modeling the residence as having an adiabatic boundary for the moment, this system can be described with a very simple heat balance; when the system is in steady state so that the amount of thermal energy in the space Q_S remains constant, the thermal energy generated Q_{gen} by the mass must be equivalent to the amount of thermal energy Q_c removed by the cooling system, i.e.

$$Q_S = Q_{gen} - Q_c = 0. \tag{4.1}$$

Moreover, the net cooling or heating rate in the space can be related to the temperature of the space by

$$Q_S = Mc\frac{dT_S}{dt}, \tag{4.2}$$

where M is the mass present in the space, c is the specific heat of that mass, and T_S is the temperature of the space referred to a fixed reference temperature.

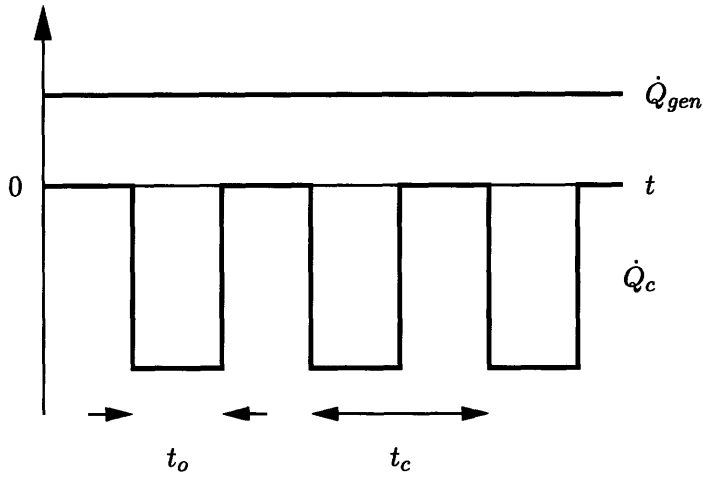


Figure 4-3: Illustration of heat flow in the residence, adapted from [103].

As the cooling capacity $\dot{Q}_c$[1] of any effective cooling system is greater than the rate at which thermal energy is transferred into the space $\dot{Q}_{gen}$, the cooling system will not have to run continuously to keep the temperature in the space within a defined temperature range, and the system will instead cycle. This behavior is illustrated in Figure 4-3.

The quasi-steady-state behavior which is maintained as the cooling system cycles is governed by the fact that the amount of energy which is removed during the time that the cooling system is operating must be equal and opposite in sign to the total amount of thermal energy input to the space. This can be written as

$$\dot{Q}_c t_o = \dot{Q}_{gen} t_c \tag{4.3}$$

where the length of time that the cooling system runs is t_o, and the length of one complete cycling period is t_c. Such an equivalence can be observed in Figure 4-3 by noting that the areas under each of the curves for $\dot{Q}_{gen}$ and $\dot{Q}_c$ for the respective times are equal.

The cycling behavior of the cooling system is typically controlled by a thermostat, which turns on the cooling system when the temperature reaches a high setpoint T_H and then turns off the cooling system when the temperature reaches a low setpoint T_L. By using this information, it is possible to express the heat balance for the space while the

[1] $\dot{Q}$ indicates thermal energy transferred per unit time.

cooling system is running as

$$Mc\frac{(T_H - T_L)}{t_o} = \dot{Q}_c - \dot{Q}_{gen} \tag{4.4}$$

This expression can be combined with Equation (4.3) to develop a relationship between the run fraction t_o/t_c and the run time t_o,

$$\left(1 - \frac{t_o}{t_c}\right)\dot{Q}_c = \frac{Mc(T_H - T_L)}{t_o} \tag{4.5}$$

$$\frac{t_o}{t_c} = 1 - \frac{Mc(T_H - T_L)}{\dot{Q}_c t_o} \tag{4.6}$$

This may be further simplified by recognizing that many of the terms on the right-hand side of this equation can be combined to represent a figurative minimum run time $t_{o,min}$, which can be expressed as

$$t_{o,min} = \frac{Mc(T_H - T_L)}{\dot{Q}_c} \tag{4.7}$$

This minimum run time represents the limit of the minimum amount of time required for the cooling system to bring the temperature of the space from T_H to T_L as the run fraction tends to zero.

While this is an admittedly simple model for describing the thermal behavior of a residence, its appeal lies in its simplicity. Under the approximate assumptions that the thermal mass Mc of a residence does not change over time and that the thermostat control setpoints T_H and T_L also do not change, observed changes in $t_{o,min}$ will reflect changes in $\dot{Q}_c$, and can be used as an indicator of refrigerant undercharge faults. Run-times and cycle times can be measured on field-installed systems, and changes in $t_{o,min}$ can be identified from the model (4.7) via regression analysis. A change due to undercharge, for example, will correspond to an increase in $t_{o,min}$, while the effect of sealing ductwork will correspond to a decrease in $t_{o,min}$.

Though results illustrating the efficacy of this detection method are presented in [103], the model's assumption that the house has an adiabatic boundary raises considerable questions regarding the sensitivity of the method to changes in the cooling load that are caused by changes in the external temperature or humidity. Specifically, if the $\dot{Q}_{gen}$ term is mod-

ified so that, rather than represent a constant term, it represents the thermal energy input due to the external temperature, i.e.

$$\dot{Q}_{gen} = Mc\frac{d(T_o - T_i)}{dt}, \tag{4.8}$$

where T_i is the building's internal temperature and T_o is temperature external to the building. This relation can be used with (4.3) to formulate a related expression for the run-time of the cooling system, i.e.

$$t_o = \frac{Mc(T_o - T_i)}{\dot{Q}_c}. \tag{4.9}$$

This equation makes intuitive sense; the cooling system will run for a longer period of time if either the difference between the indoor and outdoor temperature increases, or if the cooling capacity of the air-conditioner $\dot{Q}_c$ decreases. The dependence of t_o on both $\Delta T = T_o - T_i$ and Q_c suggests that an improved metric for refrigerant charge which is related to the changes in the capacity of the unit will incorporate measurements of both the run-time of the air-conditioner and ΔT.

In order to test the diagnostic method suggested by [103] and evaluate alternative methods of identifying undercharge faults by using observations of the air-conditioner's cycling behavior, a set of exploratory experiments were performed on a residence located in North Carolina. A photograph of this house, owned by Rob and Karyn Cox and graciously made available for these experiments, is shown in Figure 4-4. The house has approximately 2000 square feet of internal floor space, and is cooled by a central air-conditioning system which is divided into two zones: one air handler serves the ground floor, while the other air handler serves the second story. Both of these air handlers are installed in the attic, and flexible ductwork was used to distribute the cooled air from these air handlers throughout their respective zones.

Both of the zones were serviced by the same type of air-conditioning equipment. A Copeland single-phase capacitor-run scroll compressor (model no. CR22KF-PFV-130) was used in each refrigerant loop, with R-22 used as the refrigerant. The information on the nameplate of the condensing unit stated that the units were shipped with 62 oz of refrigerant; according to [160], this represents the proper refrigerant charge for 30 feet of liquid line. After determining that 42 feet of tubing were required to span the distance between

Figure 4-4: Test facility for undercharge fault detection experiments.

the compressor and the air handler in the attic, an additional 6.96 oz of refrigerant were added to the nominal system charge [160, p. 785], so that the total system charge was 68.96 oz of refrigerant.

The compressor power was monitored via a standard NILM installation, as described in [32,123]. Due to the fact that both compressors were connected to the same electrical distribution circuit, the current to both compressors were monitored with one current transducer, while the two air handlers were monitored separately with independent current transducers. The behavior of each compressor could be deduced from the signals generated by the two air handlers by examining the relative timing of the fan activation. The utility voltage supplying all of these loads was monitored by a LEM LV-25P voltage transducer, and the current into the compressors was monitored with a LEM LA-100T current transducer. The outputs of these sensors were interfaced with an Advantech PCI-1710 12-bit data acquisition card that was installed in a 800 MHz PC running Debian Linux. Each of these signals was preprocessed by the standard NILM preprocessor described in [127] so that spectral envelope estimates corresponding to the real power consumption of each load (the aggregate signal for the two compressors, and the signal for each individual fan) were written to files which were broken into hour-long sections.

Additional thermal instrumentation was also installed in order to monitor the local variations in temperature, humidity, and sunlight, which could affect the cooling load on the air-conditioning system. This instrumentation was purchased from Onset Computer Corporation, and is listed in Appendix C. In the course of the data analysis, it became clear that a number of useful diagnostic methods (described in §4.3) could be formulated solely by using measurements of the outdoor air temperature (obtained by using a high-accuracy U12 Hobo temperature logger, manufactured by Onset Corp., which was located under the eaves of a tool shed) and measurements of the indoor air temperature (obtained by using another U12 Hobo temperature sensor which was positioned on top of the upstairs thermostat. The small number of sensors which were needed for these diagnostic methods was encouraging, as small numbers of required sensors tend to increase the reliability of the FDD method.

Due to the unmodeled and potentially complex interactions between the two zones in the house, only the upstairs zone was operated during all of the experiments. The main reason for operating the air-conditioning system in this way was that it would make it easier to model and simulate the observed thermal dynamics, rather than solely relying on empirical data to explain the observed phenomena. While this proved to be a useful strategy, the cooling capacity of the single air-conditioner was insufficient to compensate for the thermal load on hot days; since the appreciation of the building occupants for the pursuit of science did not extend so far as to welcome heat stroke and other related afflictions, there were a number of days in which both air-conditioning units were run and which were not included in the data analyzed.

Two different sets of data were acquired in these experiments, corresponding to a fully-charged state and a 20% undercharged state. Initially, a 30% undercharge fault (corresponding to a system charge of 48.3 oz of refrigerant) was induced in the system due to the fact that this level of fault was originally found in the system, but the extremely long run-times which resulted when attempting to cool the house with only one air-conditioning unit a readjustment of the fault level. A total of 14 days of observations in the interval of August 9 - September 5 were obtained with the air-conditioning system at a regular charge level, and 15 days of observations between September 5 and 24 were obtained with the system at a 20% undercharge level. Only one air-conditioner operated during most of

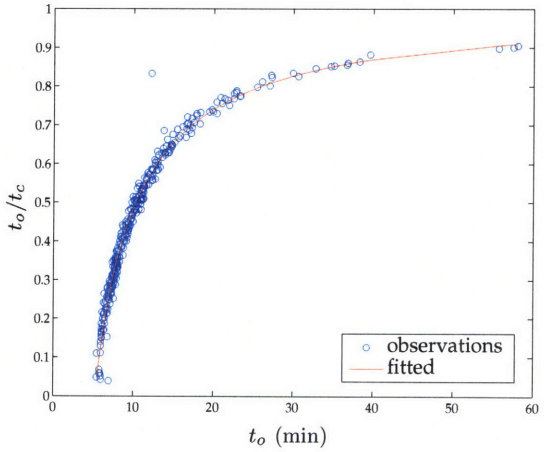


Figure 4-5: Full charge experiment with temperatureless charge diagnostic method.

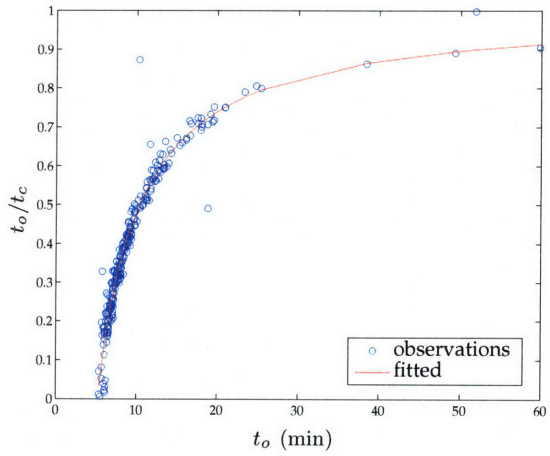


Figure 4-6: Undercharge experiment with temperatureless charge diagnostic method.

the days in these intervals which were not used for data analysis, but the data was not useable for these days because of a malfunction with the data acquisition software.

After the data corresponding to the two states of refrigerant charge were acquired, they were fused together so that the correspondences between the two datasets could be analyzed. This was largely executed via a series of Perl scripts written expressly for this purpose. The biggest challenge in fusing the datasets was the time synchronization between the two; the bulk of the time difference was identified by simply comparing the time on the laptop which was used to launch the temperature loggers with the time on the computer which acquired the power data. The time offset was further adjusted in order to bring the cycling behavior apparent in the power data in synchrony with the transients visible in the temperature data.

With the completion of the data acquisition process, the undercharge detection method described in [103] was tested on the two sets of data. The results of completing a least squares fit on the observed data can be seen in Figures 4-5 and 4-6. It is apparent from this data that there is a considerable spread in the observed run-times for the system in both states, and that the resulting least squares fits are very similar. This similarity is particularly evident in the comparison of the fitted waveforms, illustrated in Figure 4-7, and in the numerical estimates of the values of $t_{o,min}$, which is 5.1440 minutes for the regularly charged system and 5.1509 minutes for the undercharged system. While it is indeed encouraging that the minimum run-time increases in the presence of an undercharge fault,

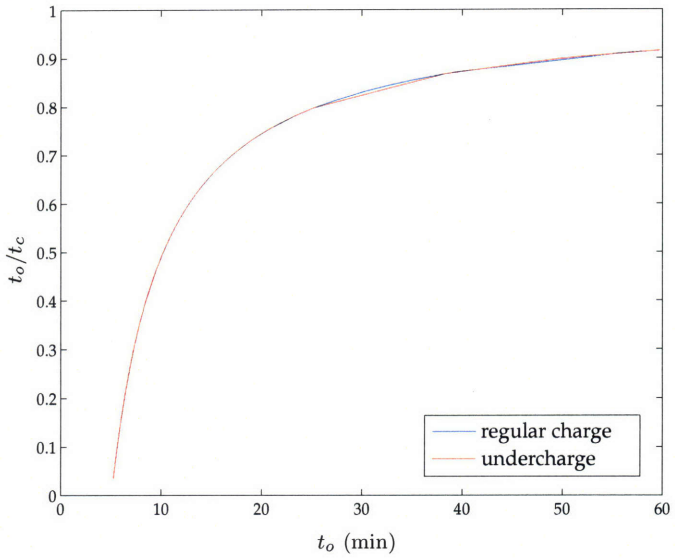


Figure 4-7: Least squares fits for both regular charge and undercharge data for temperatureless charge diagnostic method.

the fact that the detected change is on the order of 0.005 minutes for a 20% change in the system charge suggests that this method might only be effective for identifying gross variations in the system charge, e.g. greater than 20%.

There are a few potential explanations for the poor performance of this method. One particularly likely reason that this FDD method performed poorly on this data is that it appears that substantially more data was used to generate the plots in this research than was used in [103]. This difference in the quantity of data could have the result of making the potential variation in the cycle times large enough that unmodeled effects, such as the dependence of Q_c on the relative humidity or on the outdoor temperature, could significantly bias the estimates of $t_{o,min}$.

Due to the apparent shortcomings of this approach, this research developed two related methods of identifying refrigerant undercharge based upon measurements of the indoor temperature, the outdoor temperature, and the cycling behavior as determined from observations of the motor current. These methods will be described in the following sections.

4.3 Cycling Diagnostics Using Temperature Measurements

In consideration of the apparent shortcomings of the methods described in [103] and the relationship between the run-time t_o and the temperature difference $\Delta T = T_{out} - T_{in}$ suggested by (4.9), a series of approaches for identifying undercharge faults were investigated that analyzed changes in the $(t_o, \Delta T)$ behavior in both fully charged and undercharged states. As these analyses are all based upon observations made from the fused datasets, it is helpful to illustrate the sets of temperature and electrical power data for two representative days in order to point out some of the particularly relevant features. To this end, the behavior of the fully charged system over one day (August 13, 2007) is shown in Figure 4-8, and the behavior of the undercharged system over one day (September 24, 2007), is shown in 4-9. Three sets of data are illustrated in these figures: the outdoor temperature, the indoor temperature, and the electrical power input to the compressor. The compressor's cycling behavior can be calculated from this plot by analyzing the times that it turns on and off. The average steady-state power consumption of the compressor over each cycle of operation is also illustrated in this plot.

One of the first notable observations which arises from an examination of these two days is that ΔT, as it is defined, has both positive and negative values. During the early hours of the morning, the indoor temperature is higher than the outdoor temperature, while the outdoor temperature is higher than the indoor temperature during the middle of the afternoon. Accordingly, the changes in t_o will be analyzed for both positive and negative values of ΔT. Furthermore, it is interesting to note in both of these figures that the indoor temperature does not oscillate between two fixed values, but that these thresholds appear to gradually increase as the outdoor temperature increases. This is most likely an effect caused by the slow thermal behavior of the indoor temperature sensor, rather than a behavior of the system itself [4, 108]. The changes in the steady-state compressor power with temperature are also quite noticeable; this effect is largely due to the fact that higher outdoor temperatures which result in higher pressures in the condenser can cause the mechanical load on the compressor to increase. This increase in the load on the compressor causes the compressor's power to increase commensurately.

One additional pertinent observation that can be made by examining the cycling be-

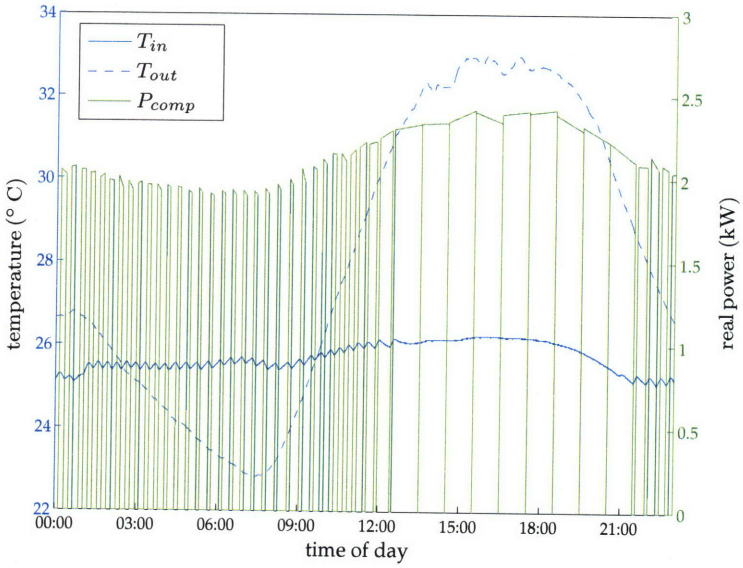


Figure 4-8: Measured variables on August 13, 2007, when the system was fully charged.

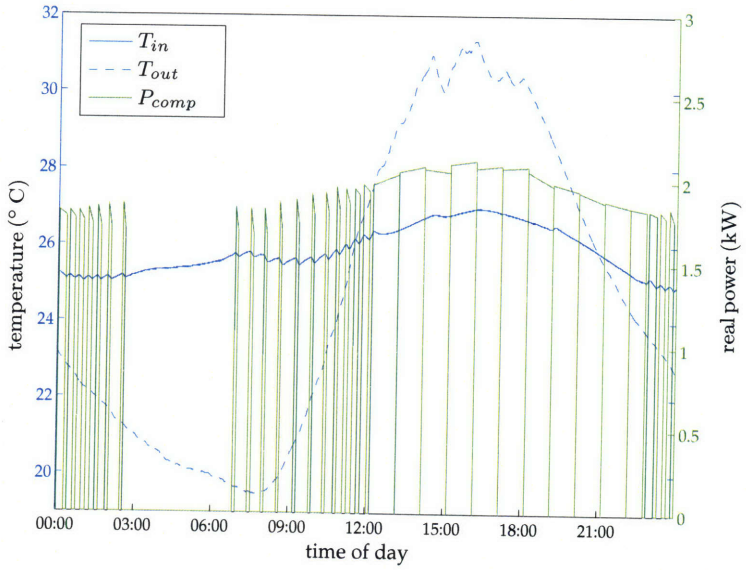


Figure 4-9: Measured variables on September 24, 2007, when the system was undercharged.

havior of the compressor in Figure 4-8 between approximately 1:00 pm and 10:00 pm is that the compressor appears to be running for an hour continuously, and only shutting off for extremely brief periods of time. This apparent behavior is actually a processing artifact that was intentionally introduced into the data to improve the performance of the FDD method. Because the compressor ran continuously during this time interval, it was necessary to develop a method of processing the data which could incorporate the changes in the temperature, rather than calculate ΔT by using the average T_o and T_i over the full range of this interval. Compressor run-times which extended for longer than an hour were therefore broken into hour-long pieces, so that the compressor apparently turned off and on within the same second in the processed data. This made it possible to calculate the average temperature over the interval of an hour, which was much more representative of the system's behavior.

In comparing the data representing the fully charged system and the undercharged system, significant differences between the cycling behavior on each day are apparent even in the unprocessed data. For example, though September 24 is approximately 2° C cooler than August 24, the compressor runs for a longer interval of time in the middle of the day (from 12:00 pm to 11:00 pm), than the fully charged system does (which runs from 1:00 pm to 10:00 pm). While these days are intended only to illustrate the representative behavior of the system, this strongly suggests that refrigerant undercharge detection methods which use observations of the cycling behavior will be able to successfully distinguish the behavior of a fully charged system from the behavior of an undercharged system.

The first FDD method which used changes in the cycling behavior as a fault detection metric directly applied the relation suggested by (4.9): for a given ΔT, the run-time t_o will increase if the cooling capacity $\dot{Q}_c$ decreases. In developing a diagnostic approach incorporating this observation, the temperature measurements were separated into bins 0.05° C wide, and the two different sets of data corresponding to fully charged and undercharged behavior were compared on the basis of these run-times. This was accomplished by aggregating all of the fused cycling and temperature datasets for each state of the air-conditioner and generating histograms for each temperature bin which characterized the behavior of the air-conditioner in each of the two states (undercharged and fully charged). The characteristics of these distributions in run-time with changes in temperature can be seen in the

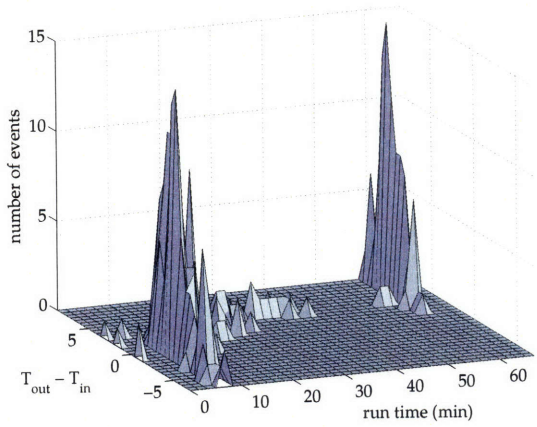


Figure 4-10: Normal charge experiment cycling histograms.

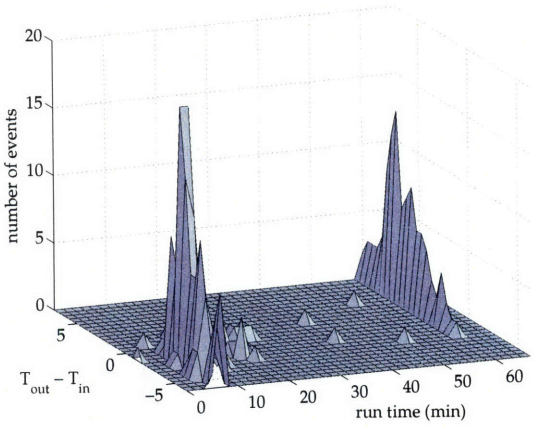


Figure 4-11: Undercharged experiment cycling histograms.

diagrams illustrated in Figures 4-10 and 4-11.

The behavior of the air-conditioner exhibited in these figures corresponds closely with expectations, as the undercharged system exhibits longer run-times than the fully charged system for many values of ΔT. This can be most easily seen by separately comparing the run-times of the system for each given ΔT in the undercharged and regularly charged states. For the regularly charged system, Figure 4-10 illustrates the fact that the minimum t_o is approximately 8 minutes for negative values of ΔT. As the temperature difference ΔT increases to approximately 2° C, the run-time for the air-conditioner gradually increases, as is expected. Further increases in ΔT beyond 4° C result in run-times equal to or greater than an hour, since the cooling capacity of the air-conditioner $\dot{Q}_{ac}$ is insufficient to remove the thermal energy $\dot{Q}_{gen}$.

In comparison, the behavior of the air-conditioner during the undercharge state, illustrated in Figure 4-11, is visibly different. The cycling period of the air-conditioner in this state is also approximately 8 minutes for large negative values of ΔT, but the air-conditioner has to cycle more frequently for a comparatively lower ΔT than was required for the fully charged system in order have the same cooling effect. It is also apparent that the undercharged air-conditioning system must run for an hour or longer when the temperature difference ΔT is greater than 2° C. This temperature is somewhat lower than that observed from the system in the regularly charged state, suggesting that the system is undercharged because of the observed higher run-time for many values of ΔT.

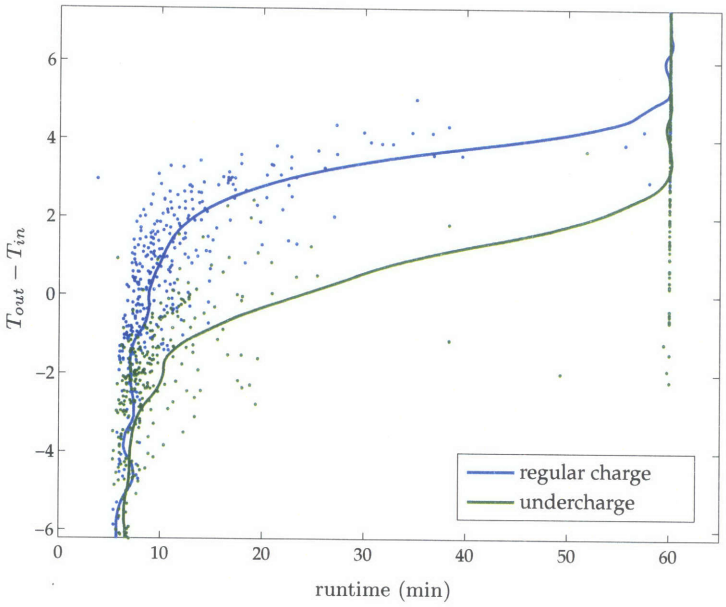


Figure 4-12: Filtered output of both charge experiments for the histogram cycling detection method.

The variation in the average cycling time due to the changes in the ΔT can be treated as a signal which expresses $t_o = f(\Delta T)$. The behavior of this signal is single valued, since the set of cycling times in each temperature bin can only have one average value, though other sources of variation in the system, such as dependence of the thermal load on incident sunlight and temperature, will cause noise in this system so that t_o is not a smooth signal. Nevertheless, the underlying behavior of the system can be described by a filtered version of this signal. Figure 4-12 illustrates the two sets of signals; the individual points represent the observations in the $t_o - \Delta T$ space, while the solid lines represent the filtered versions of the means of the run-time distributions. A wavelet filter bank with fourth-order Daubechies filters was used to filter these signals, as it was desired to remove the structured high-frequency features of the noise present in the means without substantially attenuating the edges present in the underlying signal.

It is clear from Figure 4-12 that the "characteristic curves" which are illustrated describe the underlying behavior of the system. This set of observations of the system in the $t_o - \Delta T$ plane is equivalent to looking down from above on the 3-dimensional surfaces shown in Figures 4-10 and 4-11. The comparison of these underlying signals which are observed from the filter again makes intuitive sense; the run-times of the air-conditioning system in

both the undercharged and regularly charged state are about the same when $Delta - T$ is less than -2° C, since both systems have sufficient cooling capacity for the cooling load in these conditions. As the ΔT increases, however, the reduced cooling capacity of the undercharged system causes t_o to increase for lower values of ΔT than are required by the regularly charged system. This effect also causes the undercharged system to run for an hour or longer when ΔT is greater than 2° C, while the capacity of the regularly charged system is large enough that it only has to run for approximately 30 minutes to offset the thermal inputs during this period of time.

These characteristic curves describing the system in either an undercharged or fully charged state can be used to classify new sets of incoming data by comparing new sets of data and classifying the new data according to the characteristic curve to which it is closest. In order to test the ability of these characteristic curves to accurately describe such new data, a validation procedure was carried out on two days of data selected from each charge state. For this validation process for the FDD method, a characteristic curve was generated for each dataset, representing the system in its two states of charge, without including the two days of validation data. This validation data was then tested to determine the closest characteristic curve. The results of completing this process are illustrated in Figures 4-13 and 4-14; it is apparent in Figure 4-13 that the set of data observed during the fully charged state are much closer to the characteristic curve corresponding to a state of regular charge, while the dataset observed during the undercharged state are generally closer to the characteristic curve for the undercharged behavior. While the dispersion is relatively large for the undercharged dataset, the profusion of data points for which the system is running for at least an hour at relatively small values of ΔT provides strong evidence that the system is undercharged.

This classification process can also be implemented by using an automatic classification routine, as might be done in a commercial product. One approach tested in this research used a support vector machine (SVM) method [153], implemented in Matlab, to classify each point in the validation data as lying closer to one of the two characteristic curves. The results of this classification process found that 69.4% of the points in the validation dataset drawn from the fully charged data were classified as closer to the fully charged characteristic curve, rather than the undercharged characteristic curve, while 86.8% of the

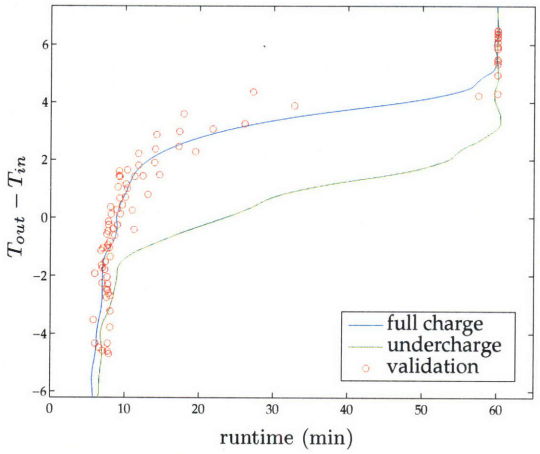


Figure 4-13: Validation of technique; validation data was drawn from full charge experiment.

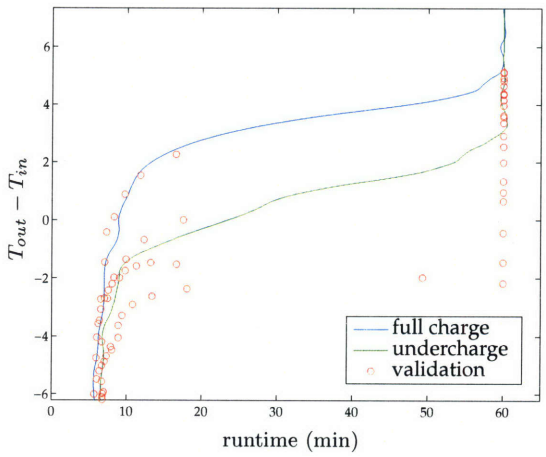


Figure 4-14: Validation of technique; validation data was drawn from undercharged experiment.

points in the validation data drawn from the undercharged dataset were found to be closer to the undercharged characteristic curve. These results strongly suggest that automated numerical classification techniques could be implemented with such characteristic curves to identify the presence of undercharge faults.

This classification method could be implemented in an FDD method in which the HVAC technician characterized the performance of the system to undercharge faults in the course of commissioning the system by running the system at low levels of refrigerant charge before the building is fully occupied. As the amount of time required for such a diagnostic might be prohibitive, an alternative implementation of the FDD method which could be implemented in the absence of knowledge about the characteristic curves which describe the undercharged system would calculate the initial characteristic curve during a baseline commissioning process, and subsequent sets of data could be compared to this initial curve to evaluate how close the two sets of data are. Since an average distance metric could be generated from each of these datasets, a threshold could be established which alerted the system user to the presence of a fault when the average distance between the most recent set of observations and the baseline characteristic curve was sufficiently large.

Not surprisingly, changes in the cycling behavior also affect the transient behavior of the compressor motor during its startup. These changes can be clearly seen by examining the two transients shown in Figure 4-15, which are representative of the motor behavior in

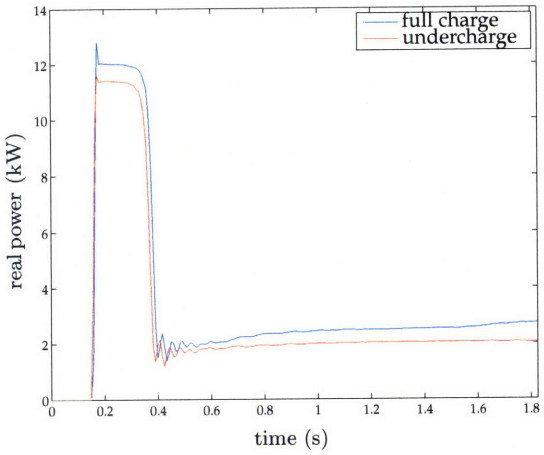


Figure 4-15: Compressor startup transients during normal cycling behavior for the air-conditioning system in both states of charge.

both the fully charged and undercharged conditions. One particularly noticeable changes which can occur is that the average compressor power during the initial acceleration of the rotor is higher for regular charge than it is during undercharge.

These changes in the startup transient are principally caused by the fact that the initial power during the startup transient is related directly to the impedance of the windings, so that changes in the winding impedance will result in changes in this initial power. As discussed in Chapter 2, the temperature of the windings has a substantial effect on their impedance. Due to the fact that the run-times for the compressor are typically longer when the air-conditioning system is undercharged, the increased resistive heating of the motor windings can cause the stator and rotor winding impedances to increase, resulting in a lower power consumption during the initial startup transient. The correlation of this behavior to the different run-times can be seen by comparing two start transients, representing the different states of charge, which were collected when the compressor was off for a number of hours, as shown in Figure 4-16. It is clear in this plot that initial power of both of these transients are nearly identical, suggesting that the changes in the transient are caused not by the refrigerant charge directly, but rather by the changes in the operational behavior that are associated with changes in refrigerant charge.

While these diagnostic features are clearly affected either directly or indirectly by the mass of refrigerant in the system, they will also be affected by ΔT, since that will have an effect on the pressures in the system. Other phenomena, such as humidity or incident solar

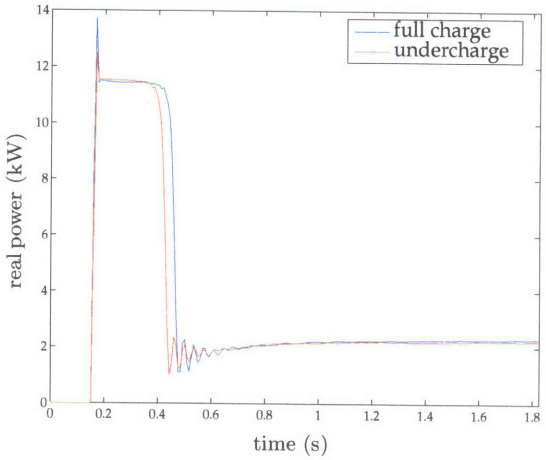


Figure 4-16: Compressor startup transients after the unit has been off for a number of hours, in both states of charge.

radiation, will also have an effect, though that effect was not large enough to identify in these experiments. In order to develop a second fault diagnostic metric which would not require the observation of the compressor over long periods of time, these observations of the initial electrical power as a function of ΔT were incorporated into a fault detection method. The behavior of this fault symptom is shown in Figure 4-17, which illustrates the set of observations of the peak power during the startup transient as a function of ΔT.

It is clear from Figure 4-17 that, while the peak power is dependent upon ΔT, it is also largely dependent upon the refrigerant charge. As expected, the transient power is lower for the undercharged system than it is for the fully charged system. The center of mass of each of the two clusters is clearly different, even though there are a few outlying datapoints. The center of each of these datasets was therefore calculated and is provided in Table 4.1. Such observations of transient characteristics could be used for a fault detection metric by collecting a set of baseline data representing the fault-free behavior of the system, and then calculating the distance of the new center of mass from datasets collected over recent time periods to the original center of mass; if the length of the vector between the two datasets is greater than a threshold, a fault could be declared.

The final fault diagnostic approach that was developed used a different aspect of the cycling behavior to identify the presence of a fault. Rather than only measuring a series of datapoints representing the pairs $(t_o, \Delta T)$, the evolution of the system in time was incorporated into the diagnostic method in order to use state information about the air-

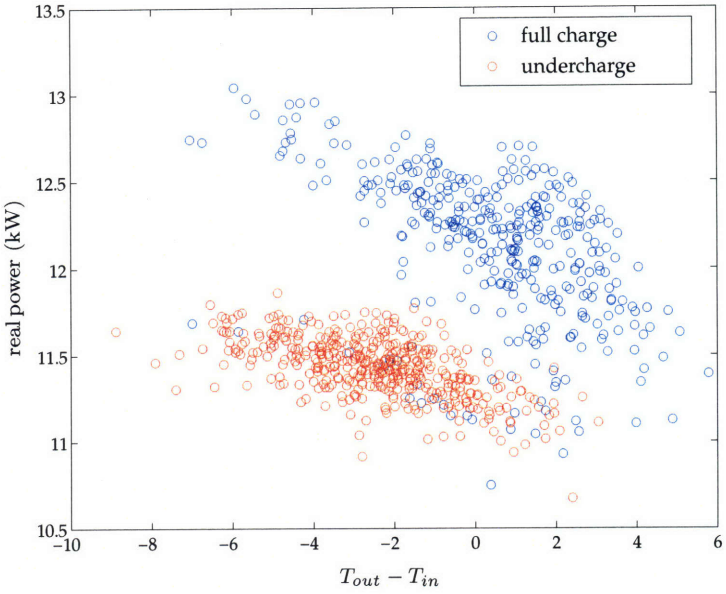


Figure 4-17: Scatter plot showing initial compressor power versus the ΔT across the house.

state	average ΔT (° C)	average P_{tran} (kW)
regular charge	0.459	12.120
undercharge	-2.462	11.426

Table 4.1: Average ΔT and peak transient power during the startup transient for both states of refrigerant charge.

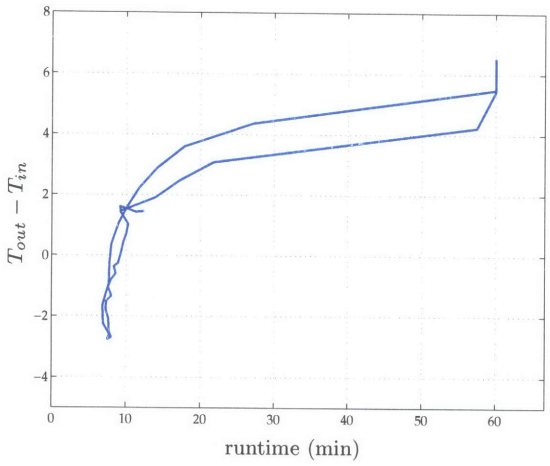


Figure 4-18: Plot of runtime against ΔT over the course of Aug. 13.

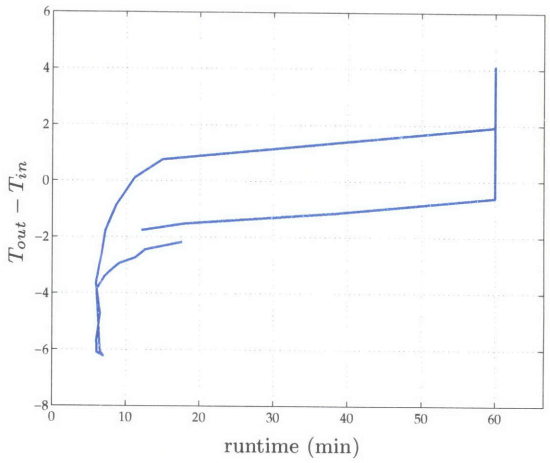


Figure 4-19: Plot of runtime against ΔT over the course of Sep. 24.

conditioning system's thermal behavior. Such an approach uses the fact that the present datapoint $(t_o, \Delta T)$ is not only a function of the current operating conditions, but is also dependent upon the previous cycling behavior of the system. In general, this current cycling behavior will depend on how well the system has previously been able to cool down the residence. This method equates to looking at trajectories in the $\Delta T - t_o$ plane observed over the course of 24 hours. One method of using and representing such information is illustrated in Figures 4-18 and 4-19.

Though these plots can initially be difficult to interpret, reference to the time-series data for the same days, illustrated in Figures 4-8 and 4-9, can help to better understand the exhibited behavior. Referring to Figures 4-8 and 4-18, the initial cycle of the regularly charged system on August 13, 2007 corresponds to $(t_o, \Delta T)$ = (12 min, 1.5° C), since it is dark and the house is cooling down. From 3:00-9:00 am, T_o is less than T_i, so that the cooling load is relatively small and t_o remains between 8 and 10 minutes. As T_o increases, however, the run-time of the air-conditioner has to increase in order to compensate for the increased cooling load, so the trajectory in $(t_o, \Delta T)$ space accordingly follows the upper arm of the plot shown in Figure 4-18. By 1:00 pm, the system has to running continuously because the cooling load is greater than the cooling capacity of the air-conditioner, and the output of the plot stays fixed at 60 minutes as the temperature rises until approximately 4 pm and begins to fall thereafter. As T_o decreases, however, the cooling load on the air-conditioner is reduced, making it possible for the system to begin cycling again by approximately 10

pm, as the cooling capacity is able to remove the amount of heat radiated into the space during the run-time t_o. This transition from non-cycling behavior to cycling behavior can be identified as movement along the lower arm of the trajectory in Figure 4-18, until the system reaches approximately the same point in $(t_o, \Delta T)$ space that began the day.

The trajectory of the undercharged system shown in Figure 4-19 differs from that of the fully charged day in a few key respects. Noting that T_o for September 24 is approximately 2° C lower than August 13, the initial operating point in $(t_o, \Delta T)$ space is (17 min, -2° C), which represents a much longer run time for a lower ΔT than was observed in the fully charged system. As T_o decreases, t_o remains at approximately 8 minutes until 10:00 am, when ΔT is approximately 0° C. As T_o continues increasing, t_o also increases rapidly as the reduced cooling capacity is unable to effectively cool the residence, and the air-conditioner runs continuously from 12 pm to 11 pm. Of particular significance is the fact that ΔT is negative by the time that t_o is less than 60 minutes; since the unit's cooling capacity is reduced, it has to continue to run until the temperature at the thermostat is low enough that it can cycle again. This cycling behavior resumes when the operating point of the system is located at approximately (60 min, -1° C), after which the system is able to cycle with t_o less than an hour.

In comparing the behavior of the air-conditioning system in both of these scenarios, it is clear that the reduction in cooling capacity has a substantial effect on the air-conditioner. More specifically, it takes longer for the undercharged system to remove the thermal energy input to the house over the portion of the day during the peak temperatures than it does for the fully charged system. The air-conditioning system therefore has to run longer towards the end of the day because of this effect. These figures underscore the significance of this effect; while the cooling load on the undercharged day is less than that of the regularly charged day due to lower peak temperatures, the amount of time required for the air-conditioner to cool the residence is still longer for the observed undercharge day than it is for the fully charged day. Moreover, as T_o is generally correlated with the amount of incident solar radiation, the longer run-times at the end of the day will cause the air-conditioner to run during more negative ΔT. This is responsible for the effect seen in Figure 4-19, in which t_o of the system is larger for lower values of ΔT at the end of the day, as compared to the response of the same system when it is fully charged.

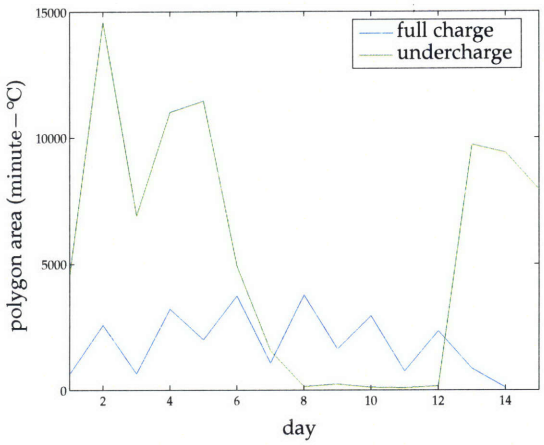


Figure 4-20: Plot of polygon areas for the two different states of refrigerant charge as a function of time.

The manifestation of the differences in these trajectories, as is apparent in Figures 4-18 and 4-19, is that the area enclosed by the trajectory of the undercharged system is larger than the comparable area for the fully charged system. This dependence of the trajectory area upon the amount of system charge can be used as a fault signature; by computing the trajectory area for each day of operation, the state of the system can be evaluated on the basis of the trend in these computed areas, since changes in the system charge will result in an increase in the areas. This diagnostic was computed for each observed day of both the fully charged and undercharged datasets, and the results of these computations are illustrated in Figure 4-20.

While it is clear that the average trajectory area for the undercharged system is higher than that for the fully charged system, it is notable that the computed area for the undercharged days numbered 7 through 12 is far below the areas of the other undercharged days. The reason for this dramatic change is that low temperatures on those days reduced the cooling load, so that the air-conditioning system only cycled for brief periods of time. This illustrates the fact that one of the requirements for this diagnostic is that the system must run for an hour or more to trace out the full trajectory of its response, making it essential that an FDD method based upon this diagnostic test each day to ensure that the full range of run-times has been exercised, preventing deceptively low trajectory areas. Each day was evaluated on this basis, and the trajectory areas were computed for those days which exhibited the full range of run-times. The results of these calculations are presented

state	total # of days	# with full excitation	area [°C-min]
regular charge	14	11	2269
undercharge	15	10	8221

Table 4.2: Areas of the trajectories for undercharged and fully charged datasets.

in Table 4.2, where it is clear that the average trajectory areas for the undercharged system is much larger than the area for the fully charged system. This could be used in a FDD method by computing the trajectory area for each day of operation which exercised the system over the full range of run-time, and faults could be identified on the basis of changes in this area.

It is important to note that this diagnostic method is strongly dependent upon the preprocessing method; if the upper threshold for the run-times was set to 30 minutes, the trajectory areas would change dramatically. Since the underlying system behavior used for this FDD method is that the reduced cooling capacity due to undercharge will cause the system to run longer for lower cooling loads, this diagnostic approach will be effective over a range of thresholds. Nevertheless, the fault detection capabilities may be dependent upon the proper selection of this upper threshold, so that further investigation of the dependence of this metric on these upper thresholds is important.

4.4 Method Validation

On the basis of the experimental data collected from the residential air-conditioning system, the approaches used to identify undercharge faults on air-conditioning equipment via two temperature measurements and one power measurement appear to be very effective. In order to better understand the characteristics of the system which give rise to the diagnostic features which were used for the characteristic curve and trajectory-based FDD methods, a thermal model of a building was developed to test and validate the methods' performance. A model of a very simple building was formulated for this purpose; this model had a total wall area of 200 m^2, a floor area of 100 m^2, and was built on a 0.1 m thick concrete slab. Furthermore, the windows were assumed to occupy 20% of the total wall area, so that the total window area was 50 m^2, and the conductance of the window

Parameter	Value
C_{wall}	2×10^7 J/K
C_{air}	2×10^7 J/K
R_{wall}	0.01 K/W
R_{air}	0.003 K/W
R_{window}	0.006 K/W
Q_l	1580 W
fully charged Q_{ac}	3596 W
undercharged Q_{ac}	3040 W

Table 4.3: Parameters for the building simulation.

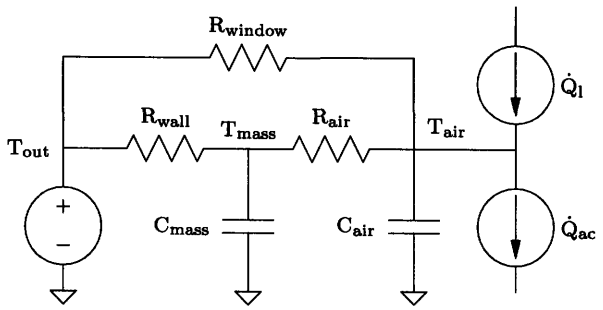


Figure 4-21: Circuit description of the thermal behavior for the thermal model.

was assumed to be 3 W/m^{2}K, while the conductance of the wall was assumed to be 0.5 W/m^{2}K. Finally, the convection-radiation heat transfer coefficient between the air and a surface was assumed to be 8 W/m^{2}K.

The simplest thermal model of this simulated building which exhibited behavior resembling the experimental data is shown below in Figure 4-21. This model is described by two capacitances C_m and C_a, which characterize the thermal mass of the building and of the air, respectively, and three resistances R_w, R_g, and R_a, which describe the thermal resistances of the wall, the window glass, and the air. This model also has three inputs: the outdoor temperature T_{out}, the heat input from occupants, equipment, and solar radiation through windows Q_l, and the cooling effect of the air-conditioner Q_{ac}. The two temperature nodes in this model T_{mass} and T_{air} are analogous to the temperature at the thermostat (T_{mass}) and the temperature at the output of the air-conditioner (T_{air}). The parameters for these circuit values were calculated from the characteristics of the thermal model, and are listed in Table 4.3.

A simulation of the building's temperature response was developed using this model.

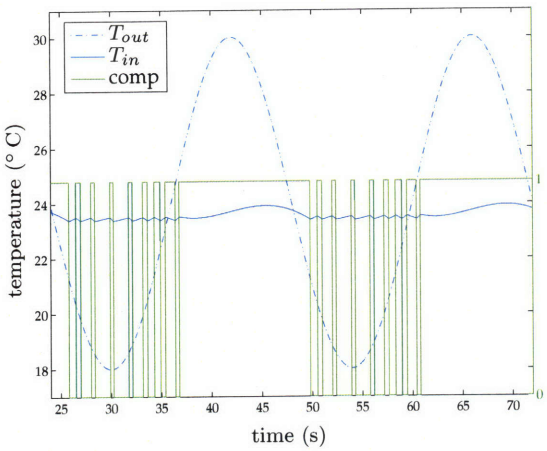


Figure 4-22: Simulated fully charged temperature and cycling response over two days.

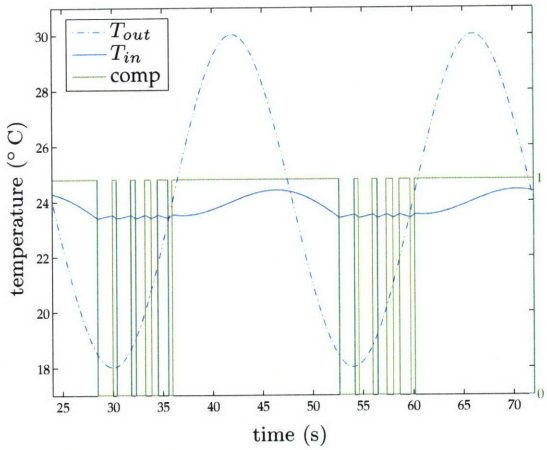


Figure 4-23: Simulated undercharged temperatures and cycling response over two days.

This simulation used a sinusoidally varying T_{out} which ranged from 18° C to 30° C, and a constant heat load Q_l of 1580 W. The temperature on the space was controlled by a thermostat with very simple deadband control, in which the cooling system Q_{ac} was turned on when the temperature T_1 went above 23.5° C, and then turned off when the it went below 23.4° C. The value of Q_{ac} was changed from a value of 3596 W when the system was fully charged to a value of 3040 W when the system was undercharged, to simulate the reduction in the cooling capacity caused by an undercharge fault. This dynamic model of the system was then simulated by using a forward-Euler fixed-step method to integrate the solution of the differential equations describing the system over a three day time interval; standard variable-step methods were not used due to the fact that the algorithms which determine step length would interact poorly with the deadband control in the simulation function. The output of these simulations for two days of both the fully charged simulation and the undercharged simulation can be seen in Figures 4-23 and 4-22.

Comparing the cycling behavior in Figures 4-23 and 4-22 to the experimentally observed cycling behavior shown in Figures 4-8 and 4-9, it is apparent that the two thermal responses are very similar. Since the detailed internal behavior of the air-conditioning system governing its power consumption was not simulated, only the times that the air-conditioner turns on and off are represented in Figures 4-23 and 4-22.

This set of simulated temperatures and cycling behavior was preprocessed to gener-

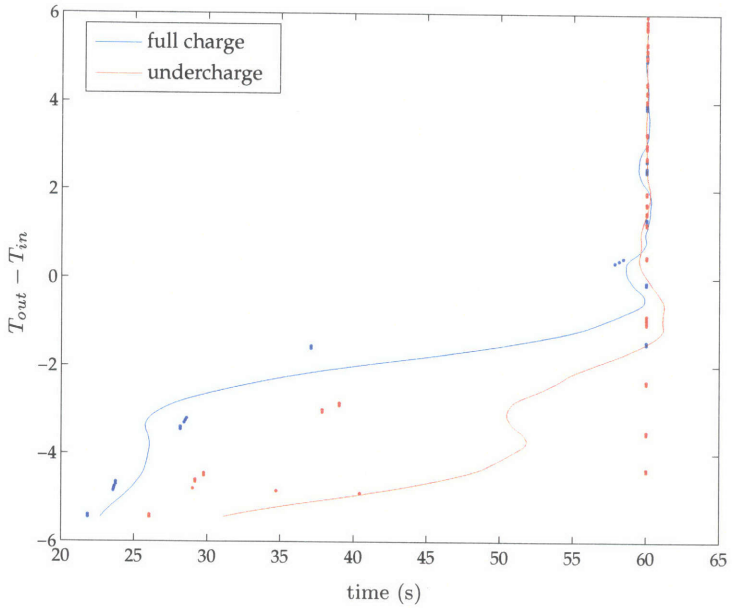


Figure 4-24: Characteristic curves for both undercharged and fully charged sets of simulated data.

ate the same set of fault signatures which were developed experimentally. The first set of fault signatures were obtained via a process identical to that which was described in §4.3, in which the full set of observed (t_o, ΔT) pairs were grouped into bins of $\Delta T = 0.05°$ C, and then the resulting waveform was filtered to obtain a characteristic curve. These characteristic curves were generated for the undercharged and fully charged datasets, and are illustrated in Figure 4-24.

It is clear from Figure 4-24 that these characteristic curves are qualitatively very similar to those derived from experimental data, suggesting that the behavior of the experimental system responsible for the shape and changes in the characteristic curves with refrigerant charge is well described by the simple model discussed in this section. While this set of curves is not as smooth as those obtained from the experimental data, this is due to the fact that the thermal behavior is nearly identical from one day to the next because the data is simulated, resulting in artifacts in the characteristic curves due to the particular sets of observations. This fact can be seen by comparing the datapoints signifying the observations, shown as dots in Figure 4-24, which are in nearly identical locations. These observations stand in contrast to the observations illustrated in Figure 4-12, which vary substantially.

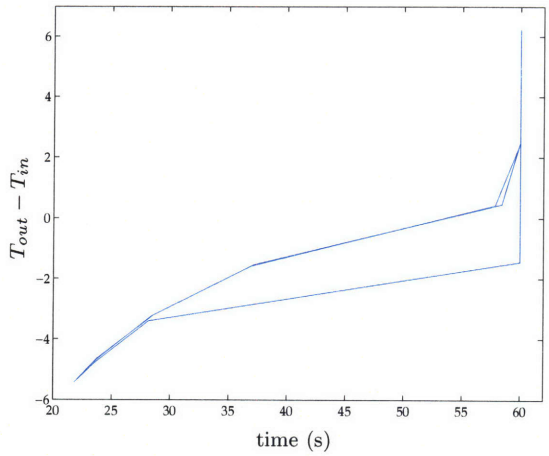


Figure 4-25: Simulated regularly charged day.

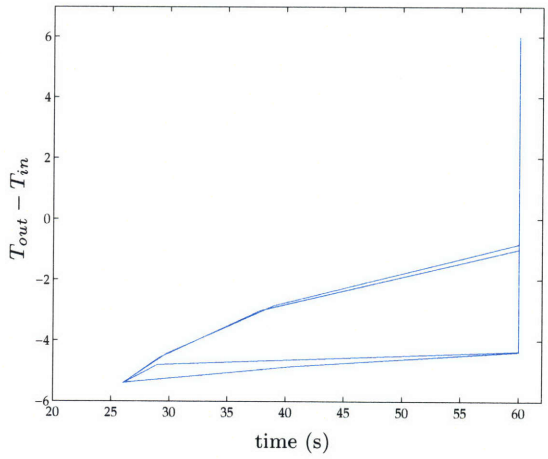


Figure 4-26: Simulated undercharged day.

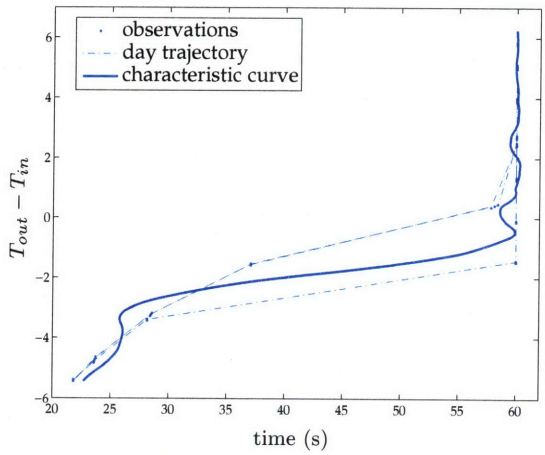


Figure 4-27: Simulated fully charged day with both diagnostics.

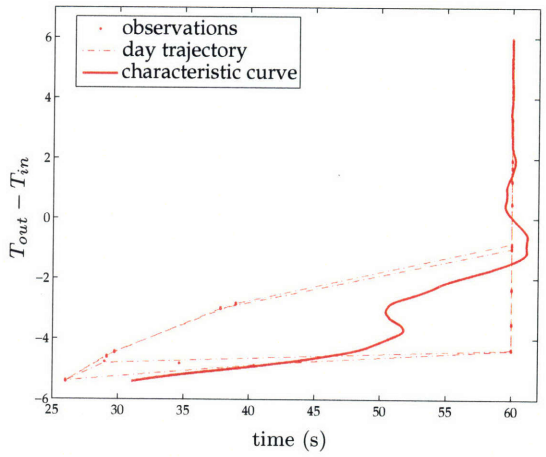


Figure 4-28: Simulated undercharged day with both diagnostics.

The results of applying the third diagnostic approach, in which the trajectory in $t_o - ^\circ$ C space is analyzed and the area enclosed over a day of operation is computed, are illustrated in Figures 4-26 and 4-25. These plots also illustrate the efficacy of applying this FDD method to the problem of undercharge detection, and show that a relatively simple model of the house exhibits similar hysteresis-like behavior with the corresponding changes in the cooling capacity of the air-conditioning unit.

The repeatability of the simulation output also makes it possible to better visualize the relationship between the two different cycling diagnostic approaches. The output of both of the diagnostic approaches can be seen in Figures 4-27 and 4-28 for the fully charged

behavior and the undercharged behavior, respectively; it is clear from both of these figures that the cycling histograms, which compute the average behavior of the air-conditioning system over many days, effectively also compute the average between the top and the bottom sides of the system's trajectory in $(t_o, \Delta T)$ space over the course of many days.

4.5 Summary

Over the course of this chapter, the analysis of the air-conditioner's cycling behavior was shown to be an effective means of identifying refrigerant undercharge in field-installed residential equipment. In comparison to many of the fault detection approaches discussed in §4.1.2, these diagnostic methods used a relatively limited set of measurements: the electrical power into the compressor, the outdoor sensible temperature, and the indoor sensible temperature. Some or all of these diagnostic approaches could be integrated into a FDD system bundled with pre-existing compressor fault detection systems, adding significant value for minimal extra cost.

FDD methods which identify refrigerant undercharge faults via changes in the cycling behavior have two inherent limitations. The first of these limitations is that the baseline performance of the system must be characterized before changes in the system performance due to faults can be detected. Such an approach limits the ability of the FDD method to detect commissioning faults, and only allows it to detect operational faults. Due to the variation in the construction of these systems, however, the development of a theoretical metric for a priori performance would be very difficult to determine, as the installation contractor may encounter unforeseen obstacles in installing the system that even the system designer did not foresee. Such considerations, as well as the large number of FDD systems which also require a baseline system characterization, suggest that this limitation is widely accepted to be necessary in a pragmatic sense.

The other related limitation of this method is that, since cycling-based diagnostic approaches require the system performance to change measurably as the cooling capacity decreases, air-conditioning systems which are oversized will only exhibit changes in their cycling behavior when the cooling capacity has been dramatically reduced. In systems where the cooling capacity is only slightly larger than the cooling load, changes in the

cooling capacity will result in measurably different run-times. In comparison, the average run-time of oversized air-conditioning systems will not change dramatically if the cooling capacity of the unit, even in an undercharged state, is still more than adequate for the cooling load. Though the cycling diagnostic approach will not detect small losses in refrigerant charge, this FDD method will be able to identify undercharge faults as soon as they are sufficiently large to reduce the cooling capacity to the point that they could affect occupant comfort. As such detection thresholds are of prime importance to any building manager, this aspect of the fault detection system should not be problematic.

The different attributes of the diagnostic techniques suit each technique to a slightly different hardware implementation, which could be useful in adapting these detection methods to a variety of general FDD systems. For example, the diagnostic technique based upon the characteristic curves does not require a high-speed data acquisition system, but it does require a significant number of observations (perhaps 7-10 days) and the precise measurement of on- and off-times. In comparison, the diagnostic technique based upon the transient behavior requires a great deal of resolution of the signal, but only for brief periods of time. Finally, the benefit of the trajectory-based diagnostic technique is that, while it makes the same number of measurements as the characteristic-curve technique, it only requires one day of data, assuming that that day has a sufficiently large range of excitation. Ideally, all of these methods could be integrated into one FDD system, which used the sensitivity of each of the individual techniques to enhance the sensitivity of the overall FDD system.

While the experimental data that was obtained provides ample evidence of its effectiveness in some residential situations, a variety of other questions regarding the performance of the general cycling approach to identifying refrigerant undercharge faults remain. For example, further investigation of the general method's dependence on the incident solar radiation and the humidity will be essential to the practical application of this approach to general residences. It is likely that large variations in humidity will particularly affect the trajectory-based diagnostic, as the air-conditioning system will have to run for longer in order to remove heat from both the latent load and the sensible load, so that it will be difficult for an FDD system to distinguish between a scenario in which a fully charged system is cooling a residence with high humidity and a scenario in which an undercharged sys-

tem is cooling a residence with low humidity. In such a situation, the other two diagnostic approaches will probably be more robust if they collect a sufficient amount of data to span the range of variation in the humidity.

Furthermore, the ability of these methods to identify undercharge on other types of buildings and other types of air-conditioning systems also remains an important topic of further research. Multi-zone air-conditioning systems, in particular, represent an interesting and important direction, as unequal cooling loads in different zones might cause the cycling behavior of one system to dominate while the other system does not have to run very often. Similarly, changes in the load caused by variations in the number of occupants, furniture, and so forth could have significant effects on the performance of these diagnostic techniques which need to be addressed. While these changes may not occur with great frequency in a residential setting, further study of these problems is essential to producing a robust fault detection method.

Most importantly, additional data evaluating the performance of these methods over a much longer span of time is essential. Though it is very encouraging that these methods worked over a period as long as two weeks, monitoring the performance of this method over the course of an entire cooling season in a number of different residences will be important to understanding the limitations of this method in a variety of field-installed equipment.

Chapter 5

Discussion & Future Work

Over the course of this research, a variety of methods were studied to detect mechanical faults in vapor-compression air-conditioning equipment by using electrical measurements. These methods were found to effectively identify the presence of faults. Three particular groups of faults were studied in depth: airflow faults, liquid slugging faults, and refrigerant undercharge faults. Fault detection methods for each of these faults were investigated by reviewing previous work to understand the challenges and potential approaches that could be used, and the FDD methods developed in this research were experimentally tested and validated.

The principal benefit associated with the use of these electrically-based FDD techniques is that electrical instrumentation is often easier to install, more reliable, and less expensive than comparable mechanical instrumentation. As air-conditioning equipment is susceptible to electrical faults, such as shorted motor windings or broken rotor bars, many FDD systems already include electrical instrumentation similar to that used in this research, so that the techniques developed represent a substantial value-added function for the FDD system, rather than new and additional hardware. Moreover, electrically-based FDD techniques can have inherent advantages over mechanical sensors in some circumstances. For example, the mechanical instrumentation required to detect liquid slugging faults is expensive and very delicate, while comparable electrical instrumentation is relatively inexpensive and easy to install, and can detect the faulty condition well below thresholds which cause damage. It is also apparent from the results of this research that the electrical signals contain a significant amount of information about the mechanical state of the sys-

tem; though many electrical processing methods discard this information, its effective use can provide an abundance of additional information about the state of the system.

While this research has been focused on the development of electrically-based FDD techniques, these techniques are not intended as a replacement for traditional fault detection techniques using thermal and mechanical sensors. As discussed in Chapter 1, these sensors directly measure the variables of interest, allowing for higher sensitivity and more precise fault diagnostics than are possible with electrically-based methods. By combining electrical, mechanical, and thermal sensors in an FDD method designed to detect mechanical faults, the reliability and the sensitivity of the fault detection method to a wide range of faults could be substantially expanded beyond presently demonstrated capabilities of published FDD systems. Such a formulation for FDD systems could improve the performance of both the system being monitored as well as the FDD system itself.

The results of this research suggest a wide array of potential avenues for future work. One particularly interesting direction would be the study of a FDD system which incorporated electrical, mechanical, and thermal sensors. This would entail the investigation of a number of different concerns, such as the precise location and type of sensors to install, as well as the experimental verification of the FDD method's ability to correctly identify multiple simultaneous faults. Such a method would also present an excellent opportunity for studying sensor faults and redundancy techniques, to ensure that sensor faults could be identified as robustly as the system faults. The range of potential application for such a system could be investigated in detail as well, as such an FDD system could be installed either on new equipment or as a retrofit; because each of these applications places different constraints on the FDD system, understanding and accommodating these constraints would be quite useful.

Another important direction for future research involves testing these FDD techniques on a variety of field-installed equipment. Though the results illustrated in this research were obtained on commercially available air-conditioners, much of the equipment was still built and constructed in a laboratory at MIT. A thorough test of these FDD methods in air-conditioners that are subjected to the wide range of operating conditions found in a variety of building installations would better demonstrate the ability of the FDD methods to detect the range of faults to which air-conditioners are susceptible. Practical issues re-

garding the number of faults which might occur and the effect of ancillary phenomena on the performance of the FDD techniques could also be studied in this environment. Field studies would also provide an excellent opportunity to study the interaction of the fault detection system with the system user and maintainer, answering questions of whether the homeowner or the service technician should be informed of the fault, how to store fault information, and other related questions.

Though the above considerations pertain most directly to the development of comprehensive FDD methods, they also apply to the particular fault detection techniques developed in this thesis. To more fully examine these issues, each particular FDD technique will be discussed in the following sections in order to comment upon their measured performance, explore some potential applications for these methods, and suggest directions for future work.

5.1 *Airflow Diagnostics*

Faults related to the volumetric airflow through the evaporator coils were the first class of faults investigated in this research, as presented in Chapter 2. The particular fault conditions studied in detail were air filter blockage and leaks in the ventilation system. These faults were identified by estimating the speed and the torque developed by the motor, and using these estimates along with a fan curve, which relates the volumetric airflow through the fan to the power and speed applied to the fan, to generate estimates of the airflow. These airflow estimates were obtained dynamically via observations of motor currents and voltages, rather than using any mechanical measurements. After experimentally testing and validating this method, it was found to identify the volumetric airflow through the air-handler reasonably well. The changes in the airflow were then used to determine the presence of faults.

One of the first considerations that should be addressed in the future development of this FDD method involves the further refinement of the airflow predictions. While the proposed airflow estimation method did not use any estimates of the motor's mechanical losses due to an interest in minimizing the amount of *a priori* information required for its operation, the analysis of the mechanical losses in the system for the purposes of devel-

oping methods of accounting for the dominant loss mechanisms in the airflow estimation method would be very useful. For example, the windage and bearing losses might be relatively easy to estimate on the basis of some simple geometrical information about the motor. By folding such estimates of the mechanical losses into the overall airflow prediction algorithm, the quality of the airflow estimates might be markedly improved.

Developing additional understanding of the other potential sources of variation in the system would also be very helpful for improving the quality of the airflow estimates. Since this system variation was found to have a significant effect on the fan performance over the course of the day in §2.9, it would be useful to develop a theory which explains and compensates for these sources of variation in the system in order to improve the implementation of this airflow estimation method for commercial applications. More detailed and exact validation tests could also improve the performance of the method, as the results from the portable hot-wire anemometer could be validated by more precise airflow measurements using more elaborate flow conditioning systems. Such tests would also allow the minimum detectable blockage or leakage to be determined, among other similar information.

One of the main traits of this FDD method is that the fan curve must be known in order to generate the airflow estimates. Though the airflow estimates made with the fan curve agreed with the measured airflow in these experiments, situations such as damage to the blower wheel or installation-specific airflow conditions (such as pre-rotation) could sufficiently affect the fan's performance that the flow predictions will not agree with the measured flow. These effects should be studied in order to better understand the susceptibility of the estimates to bias due to such conditions in a field setting. In addition, additional work in modeling the characteristics of the fan curves could yield numerical estimates of fan curves that describe the operation of the fan over a wider range of operation than the curves used in this research, which were generated with a polynomial curve fit. In a practical setting, the fans installed in air-handlers also might not be tightly specified by the manufacturer, preventing the specific set of coefficients describing the fan from being used by the FDD system. Additional knowledge about the commercial aspects of the manufacturing and distribution system for air-handlers could suggest strategies for ensuring that such a FDD system is loaded with correct set of fan curves.

Another potentially fruitful and important direction that could be pursued by future research involves the detailed study of the manner by which faults occur in ventilation systems. While the fault models used in developing and evaluating the performance of this airflow estimation method were fairly simple, the manner by which faults occur in field-installed equipment might be more complicated. For example, leaks in the ventilation system might be caused by either poor quality of installation or degradation in duct sealing compounds, such as mastic. In addition, blockage may occur due to poorly set dampers, accumulation of debris on filters or the evaporator, or other phenomena. Because the operation of the FDD method is principally based upon detecting changes in the change in pressure across the fan, as given by Equation (2.9), the method may not be able to distinguish between a fault-free system and a system with simultaneous leakage and blockage, increasing the importance of detailed studies of faults and the assessment of ancillary mechanical sensors that might enhance fault detection. Field studies of the performance of this airflow estimation method with simulated fault conditions that more closely match practically occurring faults might suggest potential ways of distinguishing between these classes of faults, or simply characterize the performance of the method in these scenarios.

Additional motor modeling and characterization may also be necessary for the implementation of this method in many commercially-available air-handling units. Since single-phase motor motors are commonly used in residential systems, one important direction for this research entails the development of parameter estimation routines for these types of motors. The parameter identification routines that were developed for the three-phase motor in the experimental air-handling unit could be adapted for these single-phase motors, allowing torque speed curves to be estimated for these units. Furthermore, additional modifications may be necessary to the parameter identification method to reflect the cycling behavior often found in air-handling units; while this method obtained estimates of the motor during continuous operation, the temperature of the motor when the air-handling unit is operated at a duty ratio of 50% or less will result in electrical parameters which differ from both the hot and the cold parameters. Parameter estimation methods which interpolate between the hot and cold parameters or use of more sophisticated models which characterize the behavior of the motor over a wide range of operation

could thus be very useful in describing the changes in the torque-speed characteristic of the motor over a wide range of operation.

Finally, this airflow estimation method could be integrated into a building control and energy management system. For example, the inclusion of this method into a ventilation control system, which modulates dampers depending on the outdoor and indoor air temperatures and humidity [68], would reduce the amount of mechanical instrumentation needed to implement such a system. Furthermore, the airflow estimates produced by this FDD method could be used to provide building occupants with useful information about the energy consumption of the building as well as information about the performance of the ventilation system, which could be extremely useful information in applications such as demand-side energy management.

5.2 Liquid Slugging Diagnostics

Steady-state and transient liquid slugging phenomena in reciprocating compressors were also studied in this research. Variations in the electrical power consumed by the motor caused by the higher cylinder pressures during slugging events were used to identify the occurrence of these events, and a preprocessor was developed which allowed these observed power variations to be distinguished from other sources of variation inherent in the system. These detection methods were experimentally tested on a semi-hermetic reciprocating compressor which was outfitted with additional mechanical instrumentation to directly measure the mechanical variables of interest, and the electrically-based fault detection method was shown to effectively identify the occurrence of liquid slugging phenomena well below the threshold of damage.

Beyond the particular goal of developing an FDD method that can identify liquid slugging events, one prime objective of this research was the study of methods to identify changes in the mechanical load of an electromechanical system using observations of the electrical variables. This research explored a variety of concerns involved in developing robust diagnostics for these systems, and proposed a few alternative solutions to those problems. Though the design of these diagnostics can be more challenging than for the analogous mechanical diagnostics, this research demonstrated of the potential benefits of

such an approach, and some strategies for accomplishing these objectives.

While the ability of this diagnostic method to successfully identify the slugging events was evident in the results presented in §3.4.4, it was also clear from these results that the experimental compressor under study was not very susceptible to damage from liquid slugging. Other experiments involving the injection of even larger quantities of liquid than those illustrated in Chapter 3 yielded no detectable change in the peak pressures, suggesting that the peak cylinder pressures achievable in this compressor were reached in the course of these experiments. One clear extension of this work is therefore the development of an analogous set of experiments on a reciprocating compressor which is more susceptible to damage from liquid slugging, and which will exhibit peak cylinder pressures akin to those discussed in [133,134]. Such a set of experiments will verify the ability of the FDD method to identify slugging phenomena which can cause damage to the compressor, as well as characterize the fault signatures which could be used to automate the FDD method developed in this research.

Additional research into the incidence of liquid slugging in the field is also essential to the further study of this phenomenon. Due to the difficulty of directly sensing liquid slugging, the fault surveys discussed in Chapter 1 deduce the presence of liquid slugging from the evidence of a variety of other mechanical faults. By using the fault detection methods proposed in this research, fault surveys will be able to directly study the frequency and the size of liquid slugs as they occur in field-installed equipment without any mechanical modifications to the compressor itself. This type of data will clarify and direct future research, as it will be possible to understand if and how liquid slugging affects field-installed equipment, and it will also help to set fault detection thresholds accordingly. The simultaneous measurement of additional data, such as outdoor temperature, crankcase temperature, suction pressure, and cycling intervals, will also make it possible to better understand the reasons for the occurrence of liquid slugging and prevent the phenomenon from happening in the future. Such data could also be useful to compressor manufacturers, as more detailed knowledge about the maximum cylinder pressures experienced in the field could result in lighter compressor designs which are better suited for the demands of the application, rather than for extreme conditions which never happen.

Though this method was ostensibly designed to identify liquid slugging events, its

principle of operation identifies unusually high transient cylinder pressures. These events are of general interest, regardless of the cause, because compressor damage is typically caused by the high pressures rather than the simple presence of the liquid. This diagnostic method could therefore potentially detect a variety of different fault conditions which cause overpressures in the compression chamber, such as oil slugging. Moreover, the dramatic effect that these high pressures had on the mechanical load of the compressor motor suggests that this fault detection method could also effectively identify overpressures in other types of compressors, such as scroll, screw, or rolling piston compressors, as unusually high peak pressures in any compression chamber will cause the load on any compressor to change in response. This suggests that the experimental verification and study of the sensitivity of this diagnostic approach across different types of compression equipment will also be a fruitful direction of research.

Field studies of the effects of slugging and overpressure phenomena in other types of compressors would further direct potential applications of this fault detection method. Higher compression pressures may not result in severe damage to system components in other types of compressors, but the sensitivity of the method to these fault signatures could present interesting and novel diagnostic and control applications. For example, one popular area of current research studies liquid injection in scroll compressors to reduce compressor work per unit suction mass [2,16]. Because the fault detection techniques proposed in this research are sensitive to changes in the pressure of the compression chamber, these techniques may be able to be coupled with temperature measurements which provide evidence of lower compressor temperatures to better control this liquid injection strategy.

Another potentially productive direction for future research involves the merger of mechanical and electrical fault detection strategies for compressors. A great deal of research into detecting the variety of faults affecting motors, but these methods have not been integrated with fault detection strategies proposed in Chapter 3. Integration of detection methods for both of these faults into a common FDD system would allow the effect of the electrical faults on the liquid slugging fault detection method, and vice versa, to be studied. In addition, research into such a system would also make it possible to study the ability of such an FDD system to identify both mechanical and electrical faults simultane-

ously. Such a system could be extremely useful when integrated into a larger FDD system which identified the full range of faults in air-conditioning equipment.

One of the potential applications of an electrically-based liquid slugging FDD method is the incorporation of the slugging detection algorithm into a control strategy for electronic expansion valves (EXVs). Due to the danger of liquid slugging, air-conditioning systems are usually controlled so that the refrigerant exiting the evaporator is 3-4°C above the saturation temperature at the suction pressure, thus ensuring that only refrigerant vapor is entering the compressor. Instead of controlling the mass flow through the expansion valve on the basis of the temperature of the suction vapor, the liquid slugging diagnostic could control the EXV so that it was closed just enough to prevent liquid from being injected into the compressor. The efficiency gains realized by such a control strategy would be relatively small, but the net savings over the life of the air-conditioner could be substantial.

5.3 *Refrigerant Charge Diagnostics*

The third and final set of faults studied in this research involved refrigerant undercharge faults, or faults in which the mass of refrigerant in the air-conditioning system was below that needed for the system to operate most efficiently. Changes in the cycling behavior of the compressor, caused by the reduction in the cooling capacity of the equipment, were used to identify the presence of this fault. These FDD methods were developed on the basis of experimental measurements taken on a air-conditioning system installed in a residence, and building simulations were also constructed to validate the FDD method and characterize the thermal behavior of the house which gives rise to the observed phenomena.

One of the main benefits of this FDD method is that it is not tied to the internal thermodynamic behavior of the air-conditioning system, but instead detects the fault based only upon the interactions of the air-conditioning system with its associated cooling load. The FDD method is therefore not dependent upon any special attributes of the system, allowing faults to be detected on a wide variety of equipment with little additional information (e.g. compressor maps or detailed system models.) The small amount of additional

information required by this FDD method also suggests that this technique for detecting refrigerant undercharge might be particularly suited as a retrofit for previously installed air-conditioning equipment, as only changes in the system behavior and three easily obtained measurements are needed to identify the presence of a fault.

Of the variety of directions for future work on this FDD method, perhaps the one which will most quickly improve the performance of this method is the study of its implementation in a variety of different residential buildings over a long period of time. Such a study is necessary for a few reasons, the first of which is that a relatively limited sample of data was used to develop the fault detection approach, while the method's seasonal performance will be much more indicative of its overall effectiveness than the four weeks of data used in this research. As there is a considerable amount of variation in construction of (not to mention occupant behavior in) air-conditioned buildings due to differences in thermal mass, solar heat gains, building envelope, and other related quantities, tests of this FDD method in a range of building types would explore the limits of this method's effectiveness and provide additional insights which could be used to improve its performance. Such a field study would also help to clarify other conditions to which this method may be sensitive, such as changes in the humidity or the thermal capacitance of a space.

Additional studies of this FDD method's sensitivity to the fault level will also help to refine and improve its performance. Though it is not expected that cycling diagnostics will identify changes in the charge level that are less than 10% of the system charge [57], this theoretical limitation should be field-tested because of its dependence on the system size; if many installed air-conditioning systems are undersized, the cycling detection method may be able to detect smaller changes in the refrigerant charge, while oversized systems might not exhibit significant changes in their cycling behavior until the fault level is higher than 10%.

The performance of this method and its sensitivity to undercharge faults in a variety of buildings might also be improved by integrating other temperature measurements into the FDD method. While this research demonstrated the benefits of incorporating such measurements, and examined the effects of including observations of the outdoor and indoor temperatures, the additional information gained from a single measurement of the subcooling or evaporating temperatures may greatly enhance the performance of the FDD

method. Field studies in which a range of thermal measurements are observed and integrated into an analysis of the cycling behavior may well result in a system which has a higher sensitivity to faults and works well in a wide variety of buildings.

Appendix A

Fan Estimation Code

This appendix contains the variety of code needed to execute and process the liquid slugging data described in Chapter 2. The code is broken up into five sections.

§**A.1:** The C program used to take the output of the optical tachometer and generate speed estimates in rpm.

§**A.2:** The next set of Matlab code identifies the speed estimates from representative data. `speed_est.m` generates the speed estimates by loading data and passing it through the function `spect_gen.m`, which filters the data and generates the spectrum in the region of interest.

§**A.3:** This section of code prepares raw motor data for parameter estimation routines. The main code used in this section is `volt_est.m`, and the other helper functions are listed after it. All of these helper functions are used to implement a Gauss-Newton solver which identifies the parameters of the first two line cycles of the utility voltage.

§**A.4:** This section contains the actual C programs which estimate the parameters of the induction machine. `motorest1.c` performs the current-only minimization, and `motorest2.c` performs the minimization against the current and a set of torque-speed curves obtained from a dynamometer. These two pieces of code are largely the same, but both were included for completeness. The filter coefficients used in both of these pieces of code are calculated in the Matlab file `filter_gen.m`.

§**A.5:** Representative airflow estimation code is given in this section; `flow_est.m` takes a collection of motor parameters and a speed, calculates the mechanical power, and then locates the point on the fan curve which corresponds to the airflow, while `speed_flow_proc.m` executes this process on representative data.

A.1 Tachometer Data Processing

A.1.1 tachproc2.c

```
#include <math.h>                                                    0
#include <stdlib.h>
#include <string.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <values.h>
#include <math.h>
#include <err.h>

#include <gsl/gsl_vector.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>

// this is from fan-diag/code/tachproc_pci/gsl/tachproc2.c
// to make: gcc -g -Wall -lgsl -lgslcblas -lm -o tachproc2 tachproc2.c
// Use: ./tachproc infile record_length > outfile

// process the output of the tachometer to give a reading in rpm.

int main (int argc, char *argv[]) {

  // setup.

  double fs = 100e3/7.0;
  double Ts = 1.0/fs;

  unsigned long int k = 0;

  double fIa, fIb, fIc, fVab, fVbc, fVca, fTach;

  // read in datafile

  FILE *INFILE = fopen(argv[1], "r"); // The input file
  long int N = atol(argv[2]);

  gsl_vector *Tach = gsl_vector_calloc(N);

  if (INFILE == NULL)
    err(1, "can't open file");

  for (k = 0; k < N; k++) {
    fscanf(INFILE, "%lf %lf %lf %lf %lf %lf %lf", &fVbc, &fIa, &fVca, &fIb, &fVab
        , &fIc, &fTach);
    gsl_vector_set(Tach, k, fTach);
  }

  fclose(INFILE);

  double last_value = 0; // whether the tach is high or low: last point
  char state = 0;       // the present state of the input
  short thresh = 300;  // threshold for detecting a transition.
  int num_edge = 0;
  long int last_sample = 0;
  double tach_in;

  for (k = 0; k < Tach->size; k++) {
```

```
  tach_in = gsl_vector_get(Tach, k);

  // If this is the first datapoint, we can just go to the next iteration,
  // since we can only see an edge on the second point and after.

  if (k == 1) {
    last_value = tach_in;
    if (tach_in > 2600) {
      state = 1;
    } else if (tach_in < 2400) {
      state = 0;
      k++;
      continue;
    }
  }

  if ( (state == 0) && ( (tach_in - last_value) > thresh ) ) {

    if (num_edge == 128) {
      printf("%.5e %.5e\n", (double) Ts*k, (double) (60.0/(Ts*(k-last_sample))))
          ;
      last_sample = k;
      num_edge = 0;
      last_value = tach_in;
      continue;
    } else if (num_edge < 128) {
      num_edge++;
    }

    state = 1;    // change the state, since it's transitioned to 1.

  } else if ( (state == 1) && ( (tach_in - last_value) < -thresh) ) {

    if (num_edge == 128) {
      printf("%.5e %.5e\n", (double) Ts*k, (double) (60.0/(Ts*(k-last_sample))))
          ;
      last_sample = k;
      num_edge = 0;
      last_value = tach_in;
      continue;
    } else if (num_edge < 128) {
      num_edge++;
    }

    state = 0;

  }

  last_value = tach_in;

 }

 return(0);

}
```

A.2 Speed Harmonic Processing

A.2.1 speed_est.m

```
% estimate the speed from the slot harmonics.

% original code from fan-diag/2008-03-05/speed_est.m

clear
clear all

Nfs = 2^16;
start = 3e5;
nR = 48; % number of rotor slots.

% unblocked data: average speed for test02 was 987.94 rpm.
s_unb = (1200-983.44)/1200;    % speed from tachometer
% s_unb = (1200-983.44)/1200;   % speed from harmonics
fshp0_unb = 60*((nR+0)*(1-s_unb)/3 + 1) % 850 Hz: no
fshm0_unb = 60*((nR+0)*(1-s_unb)/3 - 1) % 730 Hz: no
fshp1_unb = 60*((nR+1)*(1-s_unb)/3 + 1) % 866 Hz: yes
fshm1_unb = 60*((nR+1)*(1-s_unb)/3 - 1) % 747 Hz: no
fshp3_unb = 60*((nR+1)*(1-s_unb)/3 + 3) % 987 Hz: yes
fshm3_unb = 60*((nR+1)*(1-s_unb)/3 - 3) % 627 Hz: yes

((fsh/60) - 1)*(3/(nR+1))*1200

oA = (1:1:5000)';
oB = ones(size(oA));
unb = load('~/static/fan-diag/2008-02-12/test03');
iu = detrend(unb(:,2));
[f_iu, fx_iu, f_wall] = spect_gen(iu, start, Nfs);
ind = (2.75e3+1:4.85e3)';
plot(fshp0_unb*oB, oA, 'r-.', fshm0_unb*oB, oA, 'r-.', fshp1_unb*oB, oA, 'r-.',
    fshm1_unb*oB, oA, 'r-.', fshp3_unb*oB, oA, 'r-.', fshm3_unb*oB, oA, 'r-.',
    fx_iu(ind), f_iu(ind));
axis([fx_iu(ind(1)) fx_iu(ind(end)) 0 5e3])
```

A.2.2 spect_gen.m

```
function [f_out, fx_out, fwall] = spect_gen(data, start, Nfs)

% find the relevant peaks in the spectrum of the motor current.

   f_samp = 100e3/7; %# sampling frequency

   N_cyc = 200; %# use 200 cycles.
   f_wall = 60; %# wall frequency.
   N_samp = round((N_cyc/f_wall)*f_samp); %# the data length of N_cyc.

   ind = start+(1:N_samp)';
   hannmeister = hanning(N_samp);
   raw = data(ind);
```

```
   %# plot the spectra to check.
   f_raw = abs(fft(hannmeister.*raw, Nfs));
   fx_raw = (0:Nfs/2-1)'*(f_samp/Nfs);
   % plot(fx_raw, f_raw(1:Nfs/2))

   ind_f = (1:500)';
   % plot(fx_raw(ind_f), f_raw(ind_f))                                    20
   max_ind = find(f_raw(ind_f)==max(f_raw(ind_f)));
   fwall = (max_ind/(Nfs/2))*(f_samp/2);

   %# notch filter the 60Hz component out.
   w1a = 30*2/f_samp;
   w1b = 100*2/f_samp;
   [b1, a1] = ellip(3, 1, 80, [w1a w1b], 'stop');
   ub2 = filter(b1, a1, raw);

   %# take a look at the spectra                                          30
   f_ub2 = abs(fft(hannmeister.*ub2, Nfs));
   fx_ub2 = (0:Nfs/2-1)'*(f_samp/Nfs);
   % plot(fx_ub2, f_ub2(1:Nfs/2))

   %# bandpass the frequency spikes of interest
   w2a = 600*2/f_samp;
   w2b = 1600*2/f_samp;
   [b2, a2] = ellip(6, 0.5, 80, [w2a w2b]);
   ub3 = filter(b2, a2, ub2);
                                                                        40
   %# take a look at the spectra
   f_out = abs(fft(hannmeister.*ub3, Nfs));
   fx_out = (0:Nfs/2-1)'*(f_samp/Nfs);
   % plot(fx_out, f_out(1:Nfs/2))

end
```

A.3 Estimation Preprocessing

A.3.1 volt_est.m

```
function Aout = volt_est(file, Nstart)                                  0

   % now find the dq0 transformation of the voltage, based upon the
   % first cycle of phase A. Also spit out the scaled currents.

   i_sc = 2.01942344520708e-02; % the scale factor for the current data.

   % channels for the pci1710: [v_bc i_a v_ca i_b v_ab i_c]
   % fs = 100e3/7

   % hot data                                                             10

   ind1 = Nstart+(1:75000)';
   preind = (1:(ind1(1)-1000))';

   Vbc = file(ind1, 1) - mean(file(preind, 1));
```

```
Vca = file(ind1, 3) - mean(file(preind, 3));
Vab = file(ind1, 5) - mean(file(preind, 5));

Ia = i_sc*(file(ind1, 2) - mean(file(preind, 2)));
Ia = [0; Ia(2:length(Ia))];                                          20
Ib = i_sc*(file(ind1, 4) - mean(file(preind, 4)));
Ib = [0; Ib(2:length(Ib))];
Ic = i_sc*(file(ind1, 6) - mean(file(preind, 6)));
Ic = [0; Ic(2:length(Ic))];

% need to shift the wall voltages so that they are neutral referenced.

deltatowye = [1 -1 0; 0 1 -1; -1 0 1; 1 1 1];
Vdelta = [Vab Vbc Vca];
Vwye = zeros(size(Vdelta));                                          30

for k = (1:length(Vdelta))
   Vwye(k,:) = (deltatowye \ [Vdelta(k,:) 0]')';
end

% now on to the estimation.

Fs = 100e3/7;
Ts = 1/Fs;
F0 = 60.0;                                                           40
N = Fs/F0;

% estimate the characteristics of the first two cycles

yobs = Vwye(1:ceil(2*N), 1);
ind = (0:length(yobs)-1)';
t = Ts*ind;

K = 1;
Kvec = []; iter = [];                                                50
mu = [1000; 2*pi*60; 0.7];

while (K < length(yobs))

   yhat = sinfunc(mu, zeros(size(t)), t, length(yobs));
   interval = findk(yobs(K:length(yobs)), yhat(K:length(yobs)), 0.2);
   K = min([length(yobs) K+interval]);

   Kvec = [Kvec; K];
   mu = GNL('sinfunc', mu, yobs, t, K);                              60

end

% now use that first cycle to find the dq0 transform of the wall.

Vlab = (120*sqrt(2)/mu(1))*Vwye;
Vdq0 = zeros(size(Vlab));
th_vec = zeros(size(t));

for (k = 1:size(Vlab,1))                                             70
```

```
      th = 2*pi*60*Ts*(k-1);
      th_vec(k) = th;

      T = (2/3)*[cos(th) cos(th - 2*pi/3) cos(th + 2*pi/3);
              -sin(th) -sin(th - 2*pi/3) -sin(th + 2*pi/3);
              0.5 0.5 0.5];

      Vdq0(k,:) = (T*Vlab(k,:)')';

   end

   Aout = [Ia Ib Ic Vdq0];

end
```

A.3.2 GNL.m

```
function [theta] = GNL(f, theta0, yobs, t, K)

 % implement modified Gauss-Newton search with telescoping dataset
 % size adjustment.

 theta = theta0;

 %# limiting tolerances
 xtol = 1e-4;
 ftol = 1e-4;
 gtol = eps;
 maxiter = 100 * length(theta);

 %# initializing stuff
 fvec = feval(f, theta, yobs, t, K);
 oldnorm = norm(fvec);
 info = 0;
 iter = 0;

 %# Begin loop

 while (info == 0)

   [fjac, fvec] = fdjac2(f, theta, yobs, t, K);

   fjac = [fjac; .1*eye(length(theta))];
   fvec = [fvec; .1*(theta(:)-theta0(:))];
   iter = iter + 1;

   [u,s,v] = svd(fjac, 0);
   utf = u'*fvec;

   %# eliminate the almost zero singular values
   s = diag(diag(s) .* (1 - (diag(s) < sqrt(eps))));
   dx = diag(s) == 0;
   delta = -v*diag((1-dx)./(diag(s)+dx))*utf;
   jcnorms = sqrt(diag(v*s*s*v')); # column norms.
```

```
  theta = theta + delta;

  %# compute norm of the scaled gradient                                    40
  gnorm = 0;
  if oldnorm != 0
    dx = delta / oldnorm;
    gnorm = max(abs(dx.*(jcnorms != 0)./(jcnorms+(jcnorms==0))));
  end

  %# is gradient norm less than gtol?
  if (gnorm < gtol)
     info = 1;
     sprintf('gradient norm = %.3e, less than gtol = %.3e', gnorm, gtol);  50
     break
  end

  %# Too many iterations?
  if (iter >= maxiter)
     info = 1;
     sprintf('number of function evaluations exceeds %d', maxiter);
     break
  end
                                                                           60
  %# is step in parameter space less than xtol*norm(theta)?
  if (norm(delta) <= xtol*norm(theta))
     info = 1;
     sprintf('maximum relative step less than %.3e', xtol);
     break
  end

 end

endfunction                                                                70
```

A.3.3 fdjac2.m

```
function [fjac, fvec] = fdjac2(f, theta, yobs, t, K)                        0

 %# forward difference approximation
 small = sqrt(eps);
 fvec = feval(f, theta, yobs, t, K);
 fjac = zeros(size(fvec,1), size(theta,1));
 maxstep = 0.1;

 for i = 1:size(fjac,2)
   smu = theta(i);
   h = sign(theta(i))*min([max(abs([small small*smu])) maxstep]);           10
   theta(i) = theta(i) + h;
   fjac(:,i) = (feval(f, theta, yobs, t, K) - fvec) / h;
   theta(i) = smu;
 end

endfunction
```

A.3.4 findk.m

```
function [K,mu] = findk(yobs, yhat, ltol)

 N = length(yhat);

 if (N < 4)
  K=N;
  return
 end

 for K = 4:N
  t = linspace(0,1,K)';
  A = [ones(K,1) t t.*t];

  err = yobs(1:K)-yhat(1:K);
  mu = A \ err;

  if(abs(mu(2)) > sqrt(eps) && abs(mu(2))*ltol <= abs(mu(3)))
   K = min([K N]);
   return
  end

 end

endfunction
```

A.3.5 sinfunc.m

```
function err = sinfunc(mu, yobs, t, N)
   yhat = mu(1)*sin(mu(2)*t(1:N) + mu(3));
   err = yhat-yobs(1:N);
end
```

A.4 Motor Parameter Estimation

A.4.1 motorest1.c

```
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <values.h>
#include <math.h>
#include <err.h>

#include <gsl/gsl_vector.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_multifit_nlin.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_errno.h>
```

```
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_spline.h>

// taken from motor-diag/2008-03-13/motorest1.c
// to make: gcc -g -Wall -lgsl -lgslcblas -lm -o motorest1 motorest1.c
// usage: $ ./motorest1 test03.asc params.dat

// This code is intended to be a general purpose piece of estimation code
// that incorporates the envelope pre-estimation code into the whole, so
// that only one program needs to be run. The final parameters for the
// system are dumped into the output file specified by the user
// (params.dat, above).

// This is basically a concatenation of 2008-03-11/motorest-ienv.c and
// 2008-03-11/motorest-i.c. It does not minimize against a set of
// torque-speed observations.

// The filter coefficients used in the envelope calculation are calculated
// in ~/bucket/crlaugh/nlest/2008-02-19/preest.m; they are the coefficients
// of a 3rd order butterworth filter, and zi are the initial states of the
// filter so that the forward-backward filtering works right. See the
// reference from filtfilt.m for more details.

// The initial stator resistance has to be set as a preprocessor directive
// for the first estimation pass.

int motor_f (const gsl_vector *params, void *data, gsl_vector *f);
int motor_df (const gsl_vector *params, void *data, gsl_matrix *fjac);
int motor_fdf (const gsl_vector *params, void *data, gsl_vector *f, gsl_matrix *
    J);
int motor (double t, const double y[], double dydt[], void *mu_struct);
void fdjac(void (*fcn)(gsl_vector *, void *, gsl_vector *), gsl_vector *theta,
    void *data, gsl_vector *fvec, gsl_matrix *fjac);
void motorsim (gsl_vector *theta, void *data, gsl_vector *y_out);
void param_env (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc);
void param_curr (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
    ;
void postproc_env (gsl_vector *y_out, gsl_vector *y_in, void *data);
void postproc_curr (gsl_vector *y_out, gsl_vector *y_in, void *data);
void estimate (gsl_vector *theta_out, void *data);
void envelope_gen (gsl_vector *out, gsl_vector *x);
void envelope_filter (gsl_vector *data_out, gsl_vector *data_in);
void envelope_wrapper(gsl_vector *data_out, gsl_vector *data_in, gsl_vector *t);

#define R_STATOR 6.1

/*******************************************
Define datatypes.
*******************************************/

struct data {
  void *lossfcn;       // loss function
  void *param_gen;     // take the vector of estimated parameters to the
      parameters for simulation
  void *postproc;
```

```
 void *motor_f;
 void *motor_df;
 void *motor_fdf;
 gsl_vector *y0;      // initial state of the machine to simulate
 gsl_vector *yobs;    // vector of observations
 gsl_vector *t;       // time vector
 gsl_interp_accel *Vd_acc; // interpolation accelerator for Vd
 gsl_interp_accel *Vq_acc; // interpolation accelerator for Vd
 gsl_spline *Vd_spline; // spline structure for Vd
 gsl_spline *Vq_spline; // spline structure for Vq
 gsl_vector *theta_sc; // scale factor for the parameters
 gsl_vector *theta;  // the glorious parameters
 double regp;
};


struct motor_helper {
 gsl_vector *mu_sim; // parameters which have been scaled
 gsl_interp_accel *Vd_acc; // interpolation accelerator for Vd
 gsl_interp_accel *Vq_acc; // interpolation accelerator for Vd
 gsl_spline *Vd_spline; // spline structure for Vd
 gsl_spline *Vq_spline; // spline structure for Vq
};


/*****************************************
 Define functions.
*****************************************/


int motor_f (const gsl_vector *params, void *data, gsl_vector *f) {

 // calculate motor outputs for levenberg-marquardt.

 unsigned long int i;

 void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
     data *) data)->lossfcn;
 gsl_vector *yobs = ((struct data *) data)->yobs;
 long int N = yobs->size;

 gsl_vector *yout = gsl_vector_calloc(N);
 gsl_vector *theta_1 = gsl_vector_calloc(params->size);

 gsl_vector *theta0 = ((struct data *) data)->theta;
 double regp = ((struct data *) data)->regp;

 for (i = 0; i < params->size; i++)
  gsl_vector_set(theta_1, i, gsl_vector_get(params, i));

 // regularize the parameters.
 gsl_vector_sub(theta_1, theta0);
 gsl_vector_div(theta_1, theta0);
 gsl_vector_scale(theta_1, regp);
```

```
  // calculate the motor response to the input, given the parameters
  lossfcn(params, data, yout);

  // populate the top part of the output vector
  for (i = 0; i < N; i++)
    gsl_vector_set(f, i, gsl_vector_get(yout, i));

  // populate the bottom part of the output vector, for regularization
  for (i = 0; i < params->size; i++)                                    130
    gsl_vector_set(f, N+i, gsl_vector_get(theta_1, i));

  // cleanup.

  gsl_vector_free(yout);
  gsl_vector_free(theta_1);

  return GSL_SUCCESS;

}                                                                       140


int motor_df (const gsl_vector *params, void *data, gsl_matrix *fjac) {

  // derivatives calculator for 1-m.

  unsigned long int i;

  void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
      data *) data)->lossfcn;
  gsl_vector *yobs = ((struct data *) data)->yobs;                      150
  long int N = yobs->size;

  gsl_vector *fvec = gsl_vector_calloc(N);
  gsl_matrix *jac = gsl_matrix_calloc(N, params->size);

  gsl_vector *jac_vec = gsl_vector_calloc(params->size);
  gsl_matrix *eye_params = gsl_matrix_calloc(params->size, params->size);
  gsl_matrix_set_identity(eye_params);

  // calculate the finite-difference jacobian.                          160
  fdjac(lossfcn, params, data, fvec, jac);

  // set up the top part of the jacobian for nlsq
  for (i = 0; i < N; i++) {
    gsl_matrix_get_row(jac_vec, jac, i);
    gsl_matrix_set_row(fjac, i, jac_vec);
  }

  // set up the bottom part for regularization
  for (i = 0; i < params->size; i++) {                                  170
    gsl_matrix_get_row(jac_vec, eye_params, i);
    gsl_matrix_set_row(fjac, N+i, jac_vec);
  }

  // cleanup
  gsl_vector_free(fvec);
```

```
 gsl_matrix_free(jac);
 gsl_vector_free(jac_vec);
 gsl_matrix_free(eye_params);

 return GSL_SUCCESS;

}


int motor_fdf (const gsl_vector *params, void *data, gsl_vector *f, gsl_matrix *
    J) {

 // helper routine.

 motor_f(params, data, f);
 motor_df(params, data, J);

 return GSL_SUCCESS;

}


int motor (double t, const double y[], double dydt[], void *mu_struct) {

 // motor simulation program.

 double Lm, Lls, Llr, Las, Lar, D;
 double idr, iqr, iqs, ids;
 double T;

 double P = 6.0;

 double we = 2.0*M_PI*60.0;
 double vds, vqs, vdr, vqr;

 // interpolation algorithms to allow the variable-step ode solver to function.
 gsl_interp_accel *Vd_acc = ( (struct motor_helper *) mu_struct)->Vd_acc;
 gsl_interp_accel *Vq_acc = ( (struct motor_helper *) mu_struct)->Vq_acc;
 gsl_spline *Vd_spline = ( (struct motor_helper *) mu_struct)->Vd_spline;
 gsl_spline *Vq_spline = ( (struct motor_helper *) mu_struct)->Vq_spline;

 vds = gsl_spline_eval(Vd_spline, t, Vd_acc);
 vqs = gsl_spline_eval(Vq_spline, t, Vq_acc);
 vdr = 0.0;
 vqr = 0.0;

 gsl_vector *mu = ( (struct motor_helper *) mu_struct)->mu_sim;

 // set parameters.
 double rs = gsl_vector_get( (gsl_vector *) mu, 0);
 double rr = gsl_vector_get( (gsl_vector *) mu, 1);
 double Xm = gsl_vector_get( (gsl_vector *) mu, 2);
 double Xls = gsl_vector_get( (gsl_vector *) mu, 3);
 double Xlr = gsl_vector_get( (gsl_vector *) mu, 4);
 double B = gsl_vector_get( (gsl_vector *) mu, 5);
 double K = gsl_vector_get( (gsl_vector *) mu, 6);
```

```
 double lamqs = y[0];
 double lamds = y[1];
 double lamqr = y[2];
 double lamdr = y[3];
 double wr = y[4];

 Lm = Xm/we;
 Lls = Xls/we;                                                    240
 Llr = Xlr/we;
 Las = Lls + Lm;
 Lar = Llr + Lm;

 D = Lm*Lm - Las*Lar;

 // simulation equations from Krause, &c.
 idr = (Lm*lamds - Las*lamdr)/D;
 iqr = (Lm*lamqs - Las*lamqr)/D;
 iqs = (Lm*lamqr - Lar*lamqs)/D;                                  250
 ids = (Lm*lamdr - Lar*lamds)/D;

 dydt[0] = (vqs - we*lamds - rs*iqs);
 dydt[1] = (vds + we*lamqs - rs*ids);
 dydt[2] = (vqr - (we - (P/2.0)*wr)*lamdr - rr*iqr);
 dydt[3] = (vdr + (we - (P/2.0)*wr)*lamqr - rr*idr);

 T = (3.0/2.0)*(P/2.0)*(lamqr*idr - lamdr*iqr);

 dydt[4] = K*(T - B*wr*wr);                                       260
 dydt[5] = we;

 return GSL_SUCCESS;

}


void param_env (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
    {

 // properly pre-condition the parameters for the estimation; this little   270
     doohickey turns those
 // pre-conditioned parameters into something useful for the motor simulation.

 gsl_vector_set(theta_out, 0, R_STATOR); // Rs
 gsl_vector_set(theta_out, 1, gsl_vector_get(theta_sc, 0)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 0)) + 1.0)); // Rr
 gsl_vector_set(theta_out, 2, gsl_vector_get(theta_sc, 1)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 1)) + 1.0)); // Xm
 gsl_vector_set(theta_out, 3, gsl_vector_get(theta_sc, 2)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 2)) + 1.0)); // Xls
 gsl_vector_set(theta_out, 4, gsl_vector_get(theta_sc, 3)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 3)) + 1.0)); // Xlr
 gsl_vector_set(theta_out, 5, gsl_vector_get(theta_sc, 4)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 4)) + 1.0)); // B
 gsl_vector_set(theta_out, 6, gsl_vector_get(theta_sc, 5)*10.0/(exp(-0.8*
     gsl_vector_get(theta, 5)) + 1.0)); // K
```

```
}

void param_curr (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
     {

  // pre-conditioning again, different than last time, since we don't need to
      fix Rs.

  gsl_vector_set(theta_out, 0, gsl_vector_get(theta_sc, 0)*gsl_vector_get(theta,
       0)); // Rs
  gsl_vector_set(theta_out, 1, gsl_vector_get(theta_sc, 1)*gsl_vector_get(theta,
       1)); // Rr
  gsl_vector_set(theta_out, 2, gsl_vector_get(theta_sc, 2)*gsl_vector_get(theta,
       2)); // Xm
  gsl_vector_set(theta_out, 3, gsl_vector_get(theta_sc, 3)*gsl_vector_get(theta,
       3)); // Xls
  gsl_vector_set(theta_out, 4, gsl_vector_get(theta_sc, 4)*gsl_vector_get(theta,
       4)); // Xlr
  gsl_vector_set(theta_out, 5, gsl_vector_get(theta_sc, 5)*gsl_vector_get(theta,
       5)); // B
  gsl_vector_set(theta_out, 6, gsl_vector_get(theta_sc, 6)*gsl_vector_get(theta,
       6)); // K

}

void postproc_env (gsl_vector *y_out, gsl_vector *y_in, void *data) {

  // little wrapper to take the motor output, generate the envelopes,
  // and calculate the residuals.

  gsl_vector *yobs = ( (struct data *) data)->yobs;
  gsl_vector *t_vec = ( (struct data *) data)->t;

  envelope_wrapper(y_out, y_in, t_vec); // generate the envelope
  gsl_vector_sub(y_out, yobs);

}

void postproc_curr (gsl_vector *y_out, gsl_vector *y_in, void *data) {

  // little wrapper part II to pass the currents through and calculate
  // the residuals.

  gsl_vector *yobs = ( (struct data *) data)->yobs;
  gsl_vector_sub(y_in, yobs);
  gsl_vector_memcpy(y_out, y_in);

}

void motorsim (gsl_vector *theta, void *data, gsl_vector *y_out) {

  // driver which runs the ode simulation routine on the motor differential
      equations,
  // and generates a set of simulated output data.
```

```
int i, k;

double we = 2*M_PI*60.0;

double lamqs, lamds, lamqr, lamdr;
double Lm, L1s, L1r, Las, Lar, D;
double iqs, ids, th;

double t, ti;
double h = 1e-6;

// variable setup.
gsl_vector *yobs = ( (struct data *) data)->yobs;
gsl_vector *y0 = ( (struct data *) data)->y0;
gsl_vector *t_vec = ( (struct data *) data)->t;
gsl_vector *theta_sc = ( (struct data *) data)->theta_sc;

long int N = yobs->size;
int states = y0->size;
double y[states];

// interpolation accelerator setup.
gsl_interp_accel *Vd_acc = ( (struct data *) data)->Vd_acc;
gsl_interp_accel *Vq_acc = ( (struct data *) data)->Vq_acc;
gsl_spline *Vd_spline = ( (struct data *) data)->Vd_spline;
gsl_spline *Vq_spline = ( (struct data *) data)->Vq_spline;

void (*param_gen) (gsl_vector *, gsl_vector *, gsl_vector *) = ((struct data
    *) data)->param_gen;
gsl_vector *mu_sim = gsl_vector_calloc(7); // # of parameters for the
    simulation (not necessarily for estimation)
param_gen(mu_sim, theta, theta_sc); // scale parameters

struct motor_helper Msim = {mu_sim, Vd_acc, Vq_acc, Vd_spline, Vq_spline};

double Xm = gsl_vector_get(mu_sim, 2);
double Xls = gsl_vector_get(mu_sim, 3);
double Xlr = gsl_vector_get(mu_sim, 4);

Lm = Xm/we;
Lls = Xls/we;
Llr = Xlr/we;
Las = Lls + Lm;
Lar = Llr + Lm;
D = Lm*Lm - Las*Lar;

gsl_matrix *Data = gsl_matrix_calloc(N, states);

// run ode solver.
const gsl_odeiv_step_type *T = gsl_odeiv_step_rkf45;

gsl_odeiv_step *s = gsl_odeiv_step_alloc(T, states);
gsl_odeiv_control *c = gsl_odeiv_control_y_new(1e-2, 1.0);
gsl_odeiv_evolve *e = gsl_odeiv_evolve_alloc(states);

gsl_odeiv_system sys = {motor, NULL, states, (void *) &Msim};
```

```
// move initial state from vector to array.
for (i = 0; i < states; i++)
  y[i] = gsl_vector_get(y0, i);

t = gsl_vector_get(t_vec, 0);

// populate time vector.
for (i = 0; i < states; i++)
  gsl_matrix_set(Data, 0, i, y[i]);

// ode solver from time step to time step.
for (i = 1; i < N; i++) {

  ti = gsl_vector_get(t_vec, i);

  while (t < ti)
    gsl_odeiv_evolve_apply(e, c, s, &sys, &t, ti, &h, y);

  for (k = 0; k < states; k++)
    gsl_matrix_set(Data, i, k, y[k]);

}

// generate lab frame currents to compare to observations.
gsl_vector *Ias = gsl_vector_calloc(N);
gsl_vector *Ibs = gsl_vector_calloc(N);
gsl_vector *Ics = gsl_vector_calloc(N);

for (i = 0; i < N; i++) {

  lamqs = gsl_matrix_get(Data, i, 0);
  lamds = gsl_matrix_get(Data, i, 1);
  lamqr = gsl_matrix_get(Data, i, 2);
  lamdr = gsl_matrix_get(Data, i, 3);
  th = gsl_matrix_get(Data, i, 5);

  iqs = (Lm*lamqr - Lar*lamqs)/D;
  ids = (Lm*lamdr - Lar*lamds)/D;

  gsl_vector_set(Ias, i, (cos(th)*ids - sin(th)*iqs));
  gsl_vector_set(Ibs, i, (cos(th - 2*M_PI/3)*ids - sin(th - 2*M_PI/3)*iqs));
  gsl_vector_set(Ics, i, (cos(th + 2*M_PI/3)*ids - sin(th + 2*M_PI/3)*iqs));

  gsl_vector_set(y_out, i, gsl_vector_get(Ias, i));

}

// generate envelope residuals or current residuals, depending on the step.
gsl_vector *y_out_proc = gsl_vector_calloc(y_out->size); // allocate the
    envelope
void (*postproc) (gsl_vector *, gsl_vector *, void *) = ((struct data *) data)
    ->postproc;
postproc(y_out_proc, y_out, data);   // form the residual
gsl_vector_memcpy(y_out, y_out_proc); // put the result back in y_out
```

```
  // cleanup.
  gsl_odeiv_evolve_free(e);
  gsl_odeiv_control_free(c);
  gsl_odeiv_step_free(s);

  gsl_vector_free(y_out_proc);                                          440
  gsl_vector_free(mu_sim);
  gsl_matrix_free(Data);

  gsl_vector_free(Ias);
  gsl_vector_free(Ibs);
  gsl_vector_free(Ics);

}

void envelope_filter (gsl_vector *data_out, gsl_vector *data_in) {      450

  // this function should just run the given dataset through the filter.

  long int i;
  double b[4], a[4];

  b[0] = 3.565800529847785e-08; // all of these coefficients are for this
     particular configuration
  b[1] = 1.069740158954335e-07; // (sampling rate, &c.)
  b[2] = 1.069740158954335e-07;
  b[3] = 3.565800529847785e-08;                                         460

  a[0] = 1.0;
  a[1] = -2.986805334783035e+00;
  a[2] = 2.973697575591248e+00;
  a[3] = -9.868919555441706e-01;

  gsl_vector *zi_init = gsl_vector_calloc(3);
  gsl_vector_set(zi_init, 0, 9.999999648979718e-01);
  gsl_vector_set(zi_init, 1, -1.986805478519675e+00);
  gsl_vector_set(zi_init, 2, 9.868919917508654e-01);                    470

  gsl_vector *zi = gsl_vector_calloc(zi_init->size);

  for (i = 0; i < zi->size; i++)
    gsl_vector_set(zi, i, gsl_vector_get(data_in, 0)*gsl_vector_get(zi_init, i));

  gsl_vector_set(data_out, 0, b[0]*gsl_vector_get(data_in, 0) + gsl_vector_get(
      zi, 0));
  gsl_vector_set(zi, 0, b[1]*gsl_vector_get(data_in, 0) + gsl_vector_get(zi, 1)
      - a[1]*gsl_vector_get(data_out, 0));
  gsl_vector_set(zi, 1, b[2]*gsl_vector_get(data_in, 0) + gsl_vector_get(zi, 2)
      - a[2]*gsl_vector_get(data_out, 0));
  gsl_vector_set(zi, 2, b[3]*gsl_vector_get(data_in, 0) - a[3]*gsl_vector_get(     480
      data_out, 0));

  for (i = 1; i < data_in->size; i++) {
    gsl_vector_set(data_out, i, b[0]*gsl_vector_get(data_in, i) + gsl_vector_get(
        zi, 0));
```

```
    gsl_vector_set(zi, 0, b[1]*gsl_vector_get(data_in, i) + gsl_vector_get(zi, 1)
        - a[1]*gsl_vector_get(data_out, i));
    gsl_vector_set(zi, 1, b[2]*gsl_vector_get(data_in, i) + gsl_vector_get(zi, 2)
        - a[2]*gsl_vector_get(data_out, i));
    gsl_vector_set(zi, 2, b[3]*gsl_vector_get(data_in, i) - a[3]*gsl_vector_get(
        data_out, i));
  }

  // cleanup.
  gsl_vector_free(zi);                                                    490
  gsl_vector_free(zi_init);

}


void envelope_gen (gsl_vector *out, gsl_vector *x) {

  long int i;
  int nfact = 9;
                                                                          500
  // setup.
  gsl_vector *y = gsl_vector_calloc(x->size+(2*nfact));
  gsl_vector *y_temp = gsl_vector_calloc(x->size+(2*nfact));

  for (i = 0; i < nfact; i++)
    gsl_vector_set(y, i, 2.0*gsl_vector_get(x, 0) - gsl_vector_get(x, nfact-i));

  for (i = 0; i < x->size; i++)
    gsl_vector_set(y, i+nfact, gsl_vector_get(x, i));
                                                                          510
  for (i = 0; i < nfact; i++)
    gsl_vector_set(y, nfact+(x->size)+i, 2.0*gsl_vector_get(x, (x->size)-1) -
        gsl_vector_get(x, (x->size)-1-i));

  // forward filtering.
  envelope_filter(y_temp, y);
  gsl_vector_memcpy(y, y_temp); // so that y always has the data we care about.
  gsl_vector_reverse(y);

  // backward filtering.
  envelope_filter(y_temp, y);                                             520
  gsl_vector_memcpy(y, y_temp); // so that y always has the data we care about.
  gsl_vector_reverse(y);

  // setting the output.
  for (i = 0; i < out->size; i++)
    gsl_vector_set(out, i, gsl_vector_get(y, nfact+i));

  // clenaup.
  gsl_vector_free(y);
  gsl_vector_free(y_temp);                                                530

}
```

```
void envelope_wrapper(gsl_vector *data_out, gsl_vector *data_in, gsl_vector *t)
    {

  // generate the envelopes for the real and the reactive power.

  // setup.
  long int i;                                                                    540
  double ti;
  double we = 2.0*M_PI*60.0;

  gsl_vector *P = gsl_vector_calloc(data_in->size);
  gsl_vector *Q = gsl_vector_calloc(data_in->size);
  gsl_vector *Penv = gsl_vector_calloc(data_in->size);
  gsl_vector *Qenv = gsl_vector_calloc(data_in->size);

  for (i = 0; i < data_in->size; i++) {
   ti = gsl_vector_get(t, i);                                                    550
   gsl_vector_set(P, i, cos(we*ti)*gsl_vector_get(data_in, i));
   gsl_vector_set(Q, i, sin(we*ti)*gsl_vector_get(data_in, i));
  }

  envelope_gen(Penv, P); // generate envelope for real power.
  envelope_gen(Qenv, Q); // generate envelope for reactive power.

  // calculate the full envelope.
  for (i = 0; i < P->size; i++)
   gsl_vector_set(data_out, i, 2.0*sqrt(pow(gsl_vector_get(Penv, i), 2.0) + pow(  560
       gsl_vector_get(Qenv, i), 2.0)));

  // cleanup.
  gsl_vector_free(P);
  gsl_vector_free(Q);
  gsl_vector_free(Penv);
  gsl_vector_free(Qenv);

}

                                                                                 570
void fdjac(void (*fcn)(gsl_vector *, void *, gsl_vector *), gsl_vector *theta,
    void *data, gsl_vector *fvec, gsl_matrix *fjac) {

  // calculate the finite difference jacobian.

  long int i;                        // initialize variables
  double theta_sign;
  double h;
  double small = sqrt1(DBL_EPSILON); // the smallest step to take
  double smu;                        // to save the theta in
  double maxstep = 0.1;              // the biggest step to take                  580

  gsl_vector *fvec_temp = gsl_vector_calloc(fvec->size);

  gsl_vector *step_vec_a = gsl_vector_calloc(2); // helping vectors for
      determining step size
  gsl_vector *step_vec_b = gsl_vector_calloc(2);
```

```
  gsl_vector_set(step_vec_a, 0, small); // setting some parameters of help
      vectors
  gsl_vector_set(step_vec_b, 1, maxstep);

  gsl_vector_set_zero(fvec);          // initialize the residual                 590
  gsl_matrix_set_zero(fjac);          // initialize the jacobian

  fcn(theta, data, fvec);         // record the evaluation at this theta

  for (i = 0; i < theta->size; i++) {

    smu = gsl_vector_get(theta, i);     // save the theta before perturbing it

    theta_sign = GSL_SIGN(gsl_vector_get(theta, i));
    gsl_vector_set(step_vec_a, 1, fabs(small*gsl_vector_get(theta, i)));          600
    gsl_vector_set(step_vec_b, 0, gsl_vector_max(step_vec_a));
    h = theta_sign*gsl_vector_min(step_vec_b); // calculate the size of the test
        step

    gsl_vector_set(theta, i, gsl_vector_get(theta, i) + h); // temporarily
        perturb theta

    fcn(theta, data, fvec_temp);   // calculate output with temp theta

    gsl_vector_sub(fvec_temp, fvec);    // return the result in fvec_temp
    gsl_vector_scale(fvec_temp, (1.0/h)); // return the result in fvec_temp again
    gsl_matrix_set_col(fjac, i, fvec_temp); // save result in col of jacobian      610

    gsl_vector_set(theta, i, smu);      // restore value of theta
  }

  gsl_vector_free(fvec_temp);           // clean up
  gsl_vector_free(step_vec_a);
  gsl_vector_free(step_vec_b);

}
                                                                                620
void estimate (gsl_vector *theta_out, void *data) {

  // run the estimation routine with the canned l-m package.

  // setup.
  gsl_vector *yobs = ((struct data *) data)->yobs;
  gsl_vector *theta0 = ((struct data *) data)->theta;

  unsigned long int i, iter = 0;
  long int N = yobs->size;                                                      630
  int nparams = theta0->size;
  long int status;
  long int maxiter = 1000*nparams;

  // initialize solver.
  const gsl_multifit_fdfsolver_type *T;
  gsl_multifit_fdfsolver *s;
  gsl_multifit_function_fdf f;
```

```
 T = gsl_multifit_fdfsolver_lmsder;                                          640

 f.f = ((struct data *) data)->motor_f;
 f.df = ((struct data *) data)->motor_df;
 f.fdf = ((struct data *) data)->motor_fdf;
 f.p = nparams;
 f.params = data;
 f.n = N+nparams;

 s = gsl_multifit_fdfsolver_alloc (T, N+nparams, nparams);
 gsl_multifit_fdfsolver_set(s, &f, theta0);                                  650
 iter = 0;

 // run solver.
 do {

   iter++;
   status = gsl_multifit_fdfsolver_iterate (s);

   if (status)
     break;                                                                  660

   status = gsl_multifit_test_delta(s->dx, s->x, 1e-4, 1e-4);

 } while (status == GSL_CONTINUE && iter < maxiter);

 printf("number of iterations = %ld\n", iter);

 for (i = 0; i < nparams; i++)
   gsl_vector_set(theta_out, i, gsl_vector_get(s->x, i));
                                                                             670
 // cleanup.
 gsl_multifit_fdfsolver_free(s);

}


/*****************************************
 Do the estimation.
*****************************************/
                                                                             680
int main (int argc, char *argv[]) {

 unsigned long int i;
 double regp = 0.01;

 double fIa, fIb, fIc, fVd, fVq, fV0;
 double fs = 100e3/7;   // sample frequency
 double Ts = 1/fs;      // sample period
 unsigned long int N = 75000; // the length of the data vector.
 int states = 6;        // the number of state variables for the simulation.   690

 gsl_vector *Ia = gsl_vector_calloc(N); // the size of the datafile
 gsl_vector *Ib = gsl_vector_calloc(N); // the size of the datafile
 gsl_vector *Ic = gsl_vector_calloc(N); // the size of the datafile
```

```
double Vd[N], Vq[N], t_spline[N];

/* load the motor current and voltage data. */
FILE *MOTOR_IN = fopen(argv[1], "r"); // The input file
                                                                        700
if (MOTOR_IN == NULL)
  err(1, "can't open file");

for (i = 0; i < N; i++) {
  fscanf(MOTOR_IN, "%lf %lf %lf %lf %lf %lf", &fIa, &fIb, &fIc, &fVd, &fVq, &
      fV0);

  gsl_vector_set(Ia, i, fIa);
  gsl_vector_set(Ib, i, fIb);
  gsl_vector_set(Ic, i, fIc);
  Vd[i] = fVd;                                                          710
  Vq[i] = fVq;
}

fclose(MOTOR_IN);

/* set up the matrices and initialize them. */
gsl_vector *y0 = gsl_vector_calloc(states);
gsl_vector *t = gsl_vector_calloc(N);

for (i = 0; i < t->size; i++) {          // Populate the time vector and array  720
    .
  gsl_vector_set(t, i, i*Ts);
  t_spline[i] = i*Ts;
}

gsl_vector *Ia_filt = gsl_vector_calloc(Ia->size); // the envelope vector.
envelope_wrapper(Ia_filt, Ia, t);       // generate the envelope.


/* Do the pre-estimation step. */
                                                                        730
int params_env = 6;          // the number of parameters to pre-estimate.

// the initial parameter scales (set by the user).
gsl_vector *theta_sc_pre = gsl_vector_calloc(params_env);
gsl_vector_set(theta_sc_pre, 0, 1.0); // Rr
gsl_vector_set(theta_sc_pre, 1, 10.0); // Xm
gsl_vector_set(theta_sc_pre, 2, 1.0); // Xls
gsl_vector_set(theta_sc_pre, 3, 1.0); // Xlr
gsl_vector_set(theta_sc_pre, 4, 1e-4); // B
gsl_vector_set(theta_sc_pre, 5, 10.0); // K                              740

// initial guess (initialize to 1)
gsl_vector *theta0_pre = gsl_vector_calloc(params_env);
gsl_vector_set_all(theta0_pre, 1.0);

/* set up the spline interpolation thingy. */
gsl_interp_accel *Vd_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Vq_acc = gsl_interp_accel_alloc();
gsl_spline *Vd_spline = gsl_spline_alloc(gsl_interp_cspline, N);
```

```
gsl_spline *Vq_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline_init(Vd_spline, t_spline, Vd, N);
gsl_spline_init(Vq_spline, t_spline, Vq, N);

// set up structure.
struct data Dpre = {&motorsim, &param_env, &postproc_env, &motor_f, &motor_df,
     &motor_fdf, y0, Ia_filt, t, Vd_acc, Vq_acc, Vd_spline, Vq_spline,
    theta_sc_pre, theta0_pre, regp};

// do the first estimation step.
gsl_vector *theta_out1 = gsl_vector_calloc(params_env);
gsl_vector *theta_pre = gsl_vector_calloc(params_env+1);
printf("pre-estimation: ");
estimate(theta_out1, &Dpre);
param_env(theta_pre, theta_out1, theta_sc_pre);


/* Do the final estimation step. */

int params_curr = 7;   // the number of parameters to estimate.

// the final parameter scales (set by the program).
gsl_vector *theta_sc = gsl_vector_calloc(params_curr);
gsl_vector_set(theta_sc, 0, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 0)
    )))); // Rs
gsl_vector_set(theta_sc, 1, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 1)
    )))); // Rr
gsl_vector_set(theta_sc, 2, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 2)
    )))); // Xm
gsl_vector_set(theta_sc, 3, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 3)
    )))); // Xls
gsl_vector_set(theta_sc, 4, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 4)
    )))); // Xlr
gsl_vector_set(theta_sc, 5, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 5)
    )))); // B
gsl_vector_set(theta_sc, 6, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 6)
    )))); // K

// initial guess
gsl_vector *theta0 = gsl_vector_calloc(params_curr);
gsl_vector_set(theta0, 0, gsl_vector_get(theta_pre, 0)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 0))))); // Rs
gsl_vector_set(theta0, 1, gsl_vector_get(theta_pre, 1)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 1))))); // Rr
gsl_vector_set(theta0, 2, gsl_vector_get(theta_pre, 2)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 2))))); // Xm
gsl_vector_set(theta0, 3, gsl_vector_get(theta_pre, 3)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 3))))); // Xls
gsl_vector_set(theta0, 4, gsl_vector_get(theta_pre, 4)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 4))))); // Xlr
gsl_vector_set(theta0, 5, gsl_vector_get(theta_pre, 5)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 5))))); // B
gsl_vector_set(theta0, 6, gsl_vector_get(theta_pre, 6)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 6))))); // K

// set up structure.
```

```
  struct data Dcurr = {&motorsim, &param_curr, &postproc_curr, &motor_f, &
      motor_df, &motor_fdf, y0, Ia, t, Vd_acc, Vq_acc, Vd_spline, Vq_spline,
      theta_sc, theta0, regp};

  // do the second estimation step.
  gsl_vector *theta_out2 = gsl_vector_calloc(params_curr);
  gsl_vector *theta_final = gsl_vector_calloc(params_curr);
  printf("estimation: ");
  estimate(theta_out2, &Dcurr);
  param_curr(theta_final, theta_out2, theta_sc);

  /* report the results. */

  FILE *MOTOR_OUT = fopen(argv[2], "w");
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 0)); // Rs
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 1)); // Rr
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 2)); // Xm
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 3)); // Xls
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 4)); // Xlr
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 5)); // B
  fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 6)); // K
  fprintf(MOTOR_OUT, "\n");
  fclose(MOTOR_OUT);


  /* Free the allocated space. */

  gsl_vector_free(Ia);
  gsl_vector_free(Ib);
  gsl_vector_free(Ic);

  gsl_vector_free(theta_sc_pre);
  gsl_vector_free(theta0_pre);
  gsl_vector_free(theta0);
  gsl_vector_free(theta_sc);

  gsl_vector_free(Ia_filt);
  gsl_vector_free(theta_pre);
  gsl_vector_free(theta_final);
  gsl_vector_free(y0);
  gsl_vector_free(t);

  gsl_spline_free(Vd_spline);
  gsl_spline_free(Vq_spline);
  gsl_interp_accel_free(Vd_acc);
  gsl_interp_accel_free(Vq_acc);

  return(0);

}
```

A.4.2 motorest2.c

```
#include <math.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <values.h>
#include <math.h>
#include <err.h>

#include <gsl/gsl_vector.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_multifit_nlin.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <gsl/gsl_spline.h>

// taken from 2008-03-11/motorest2.c
// to make: gcc -g -Wall -lgsl -lgslcblas -lm -o motorest2 motorest2.c
// usage: $ ./motorest2 test03.asc params.dat

// only small modifications from motorest1.c were needed.

// This code is intended to be a general purpose piece of estimation code
// which incorporates the envelope pre-estimation code into the whole, so
// that only one program needs to be run. The final parameters for the
// system are dumped into the output file specified by the user (params.dat,
// above).

// This is basically a concatenation of 2008-03-11/motorest-ienv.c and
// 2008-03-11/motorest-tsi.c. This program minimizes against both the
// phase a current and a set of torque-speed observations.

// The filter coefficients used in the envelope calculation are calculated
// in ~/bucket/crlaugh/nlest/2008-02-19/preest.m; they are the coefficients
// of a 3rd order butterworth filter, and zi are the initial states of the
// filter so that the forward-backward filtering works right. See the
// reference from filtfilt.m for more details.

int motor_f_pre (const gsl_vector *params, void *data, gsl_vector *f);
int motor_df_pre (const gsl_vector *params, void *data, gsl_matrix *fjac);
int motor_fdf_pre (const gsl_vector *params, void *data, gsl_vector *f,
    gsl_matrix *J);
int motor_f_tsi (const gsl_vector *params, void *data, gsl_vector *f);
int motor_df_tsi (const gsl_vector *params, void *data, gsl_matrix *fjac);
int motor_fdf_tsi (const gsl_vector *params, void *data, gsl_vector *f,
    gsl_matrix *J);
int motor (double t, const double y[], double dydt[], void *mu_struct);
void fdjac(void (*fcn)(gsl_vector *, void *, gsl_vector *), gsl_vector *theta,
    void *data, gsl_vector *fvec, gsl_matrix *fjac);
void motorsim (gsl_vector *theta, void *data, gsl_vector *y_out);
void krausedude (gsl_vector *theta, void *data, gsl_vector *torque_diff);
void param_env (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc);
void param_curr (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
    ;
void postproc_env (gsl_vector *y_out, gsl_vector *y_in, void *data);
```

```
void postproc_curr (gsl_vector *y_out, gsl_vector *y_in, void *data);
void estimate (gsl_vector *theta_out, void *data);
void envelope_gen (gsl_vector *out, gsl_vector *x);
void envelope_filter (gsl_vector *data_out, gsl_vector *data_in);
void envelope_wrapper(gsl_vector *data_out, gsl_vector *data_in, gsl_vector *t);

#define R_STATOR 6.1                                                    60

/*****************************************
Define datatypes.
*****************************************/

struct data {
  void *lossfcn;       // loss function
  void *krausefcn;     // torque-speed data function
  void *param_gen;     // take the vector of estimated parameters to the
     parameters for simulation
  void *postproc;                                                       70
  void *motor_f;
  void *motor_df;
  void *motor_fdf;
  gsl_vector *y0;      // initial state of the machine to simulate
  gsl_vector *yobs;    // vector of observations
  gsl_vector *t;       // time vector
  long int TS_N;       // number of points of torque-speed data
  gsl_vector *speed_data; // measured speed data
  gsl_vector *torque_data; // measured torque data
  gsl_interp_accel *Vd_acc; // interpolation accelerator for Vd          80
  gsl_interp_accel *Vq_acc; // interpolation accelerator for Vd
  gsl_spline *Vd_spline; // spline structure for Vd
  gsl_spline *Vq_spline; // spline structure for Vq
  gsl_vector *theta_sc; // scale factor for the parameters
  gsl_vector *theta;   // the glorious parameters
  double regp;
};


struct motor_helper {                                                   90
  gsl_vector *mu_sim; // parameters which have been scaled
  gsl_interp_accel *Vd_acc; // interpolation accelerator for Vd
  gsl_interp_accel *Vq_acc; // interpolation accelerator for Vd
  gsl_spline *Vd_spline; // spline structure for Vd
  gsl_spline *Vq_spline; // spline structure for Vq
};



/*****************************************                             100
 Define functions.
*****************************************/

int motor_f_pre (const gsl_vector *params, void *data, gsl_vector *f) {

  // initial envelope estimation; does not minimize against t-s curve.

  unsigned long int i;
```

```
 void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
     data *) data)->lossfcn;
 gsl_vector *yobs = ((struct data *) data)->yobs;
 long int N = yobs->size;

 gsl_vector *yout = gsl_vector_calloc(N);
 gsl_vector *theta_1 = gsl_vector_calloc(params->size);

 gsl_vector *theta0 = ((struct data *) data)->theta;
 double regp = ((struct data *) data)->regp;

 for (i = 0; i < params->size; i++)
   gsl_vector_set(theta_1, i, gsl_vector_get(params, i));

 gsl_vector_sub(theta_1, theta0);
 gsl_vector_div(theta_1, theta0);
 gsl_vector_scale(theta_1, regp);

 lossfcn(params, data, yout);

 for (i = 0; i < N; i++)
   gsl_vector_set(f, i, gsl_vector_get(yout, i));

 for (i = 0; i < params->size; i++)
   gsl_vector_set(f, N+i, gsl_vector_get(theta_1, i));

 gsl_vector_free(yout);
 gsl_vector_free(theta_1);

 return GSL_SUCCESS;

}

int motor_df_pre (const gsl_vector *params, void *data, gsl_matrix *fjac) {

 // initial envelope estimation; does not minimize against t-s curve.

 unsigned long int i;

 void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
     data *) data)->lossfcn;
 gsl_vector *yobs = ((struct data *) data)->yobs;
 long int N = yobs->size;

 gsl_vector *fvec = gsl_vector_calloc(N);
 gsl_matrix *jac = gsl_matrix_calloc(N, params->size);

 gsl_vector *jac_vec = gsl_vector_calloc(params->size);
 gsl_matrix *eye_params = gsl_matrix_calloc(params->size, params->size);
 gsl_matrix_set_identity(eye_params);

 fdjac(lossfcn, params, data, fvec, jac);

 for (i = 0; i < N; i++) {
   gsl_matrix_get_row(jac_vec, jac, i);
```

```
    gsl_matrix_set_row(fjac, i, jac_vec);
  }

  for (i = 0; i < params->size; i++) {
    gsl_matrix_get_row(jac_vec, eye_params, i);
    gsl_matrix_set_row(fjac, N+i, jac_vec);
  }

  gsl_vector_free(fvec);
  gsl_matrix_free(jac);
  gsl_vector_free(jac_vec);
  gsl_matrix_free(eye_params);

  return GSL_SUCCESS;

}

int motor_fdf_pre (const gsl_vector *params, void *data, gsl_vector *f,
    gsl_matrix *J) {

  motor_f_pre(params, data, f);
  motor_df_pre(params, data, J);

  return GSL_SUCCESS;

}

int motor_f_tsi (const gsl_vector *params, void *data, gsl_vector *f) {

  // second estimation step which minimized against t-s observations.

  unsigned long int i;

  void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
      data *) data)->lossfcn;
  void (*krausefcn) (gsl_vector *theta, void *data, gsl_vector *torque_diff) =
      ((struct data *) data)->krausefcn;
  gsl_vector *yobs = ((struct data *) data)->yobs;
  long int TS_N = ((struct data *) data)->TS_N;
  gsl_vector *theta0 = ((struct data *) data)->theta;
  double regp = ((struct data *) data)->regp;

  long int N = yobs->size;

  gsl_vector *yout = gsl_vector_calloc(N);
  gsl_vector *tdata_out = gsl_vector_calloc(TS_N);
  gsl_vector *theta_1 = gsl_vector_calloc(params->size);

  for (i = 0; i < params->size; i++)
    gsl_vector_set(theta_1, i, gsl_vector_get(params, i));

  gsl_vector_sub(theta_1, theta0);
  gsl_vector_div(theta_1, theta0);
  gsl_vector_scale(theta_1, regp);

  lossfcn(params, data, yout);
```

```
  krausefcn(params, data, tdata_out);

  for (i = 0; i < N; i++)
    gsl_vector_set(f, i, gsl_vector_get(yout, i));
                                                                          220
  // include t-s observations in residual.
  for (i = 0; i < TS_N; i++)
    gsl_vector_set(f, N+i, gsl_vector_get(tdata_out, i));

  for (i = 0; i < params->size; i++)
    gsl_vector_set(f, TS_N+N+i, gsl_vector_get(theta_1, i));

  gsl_vector_free(yout);
  gsl_vector_free(theta_1);
                                                                          230
  return GSL_SUCCESS;

}


int motor_df_tsi (const gsl_vector *params, void *data, gsl_matrix *fjac) {

  unsigned long int i;

  void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct    240
      data *) data)->lossfcn;
  void (*krausefcn) (gsl_vector *theta, void *data, gsl_vector *torque_diff) =
      ((struct data *) data)->krausefcn;
  gsl_vector *yobs = ((struct data *) data)->yobs;
  long int TS_N = ((struct data *) data)->TS_N;

  long int N = yobs->size;

  gsl_vector *fvec1 = gsl_vector_calloc(N);
  gsl_matrix *jac1 = gsl_matrix_calloc(N, params->size);
  gsl_vector *fvec2 = gsl_vector_calloc(TS_N);
  gsl_matrix *jac2 = gsl_matrix_calloc(TS_N, params->size);                250

  gsl_vector *jac_vec = gsl_vector_calloc(params->size);
  gsl_matrix *eye_params = gsl_matrix_calloc(params->size, params->size);
  gsl_matrix_set_identity(eye_params);

  fdjac(lossfcn, params, data, fvec1, jac1);
  fdjac(krausefcn, params, data, fvec2, jac2);

  for (i = 0; i < N; i++) {
    gsl_matrix_get_row(jac_vec, jac1, i);                                  260
    gsl_matrix_set_row(fjac, i, jac_vec);
  }

  // include t-s observations in jacobian.
  for (i = 0; i < TS_N; i++) {
    gsl_matrix_get_row(jac_vec, jac2, i);
    gsl_matrix_set_row(fjac, N+i, jac_vec);
  }
```

```
  for (i = 0; i < params->size; i++) {                                    270
    gsl_matrix_get_row(jac_vec, eye_params, i);
    gsl_matrix_set_row(fjac, N+TS_N+i, jac_vec);
  }

  gsl_vector_free(fvec1);
  gsl_matrix_free(jac1);
  gsl_vector_free(fvec2);
  gsl_matrix_free(jac2);
  gsl_vector_free(jac_vec);
  gsl_matrix_free(eye_params);                                            280

  return GSL_SUCCESS;

}


int motor_fdf_tsi (const gsl_vector *params, void *data, gsl_vector *f,
    gsl_matrix *J) {

  motor_f_tsi(params, data, f);
  motor_df_tsi(params, data, J);                                          290

  return GSL_SUCCESS;

}

int motor (double t, const double y[], double dydt[], void *mu_struct) {

  double Lm, L1s, L1r, Las, Lar, D;
  double idr, iqr, iqs, ids;
  double T;                                                               300

  double P = 6.0;

  double we = 2.0*M_PI*60.0;
  double vds, vqs, vdr, vqr;

  gsl_interp_accel *Vd_acc = ( (struct motor_helper *) mu_struct)->Vd_acc;
  gsl_interp_accel *Vq_acc = ( (struct motor_helper *) mu_struct)->Vq_acc;
  gsl_spline *Vd_spline = ( (struct motor_helper *) mu_struct)->Vd_spline;
  gsl_spline *Vq_spline = ( (struct motor_helper *) mu_struct)->Vq_spline;  310

  vds = gsl_spline_eval(Vd_spline, t, Vd_acc);
  vqs = gsl_spline_eval(Vq_spline, t, Vq_acc);
  vdr = 0.0;
  vqr = 0.0;

  gsl_vector *mu = ( (struct motor_helper *) mu_struct)->mu_sim;

  double rs = gsl_vector_get( (gsl_vector *) mu, 0);
  double rr = gsl_vector_get( (gsl_vector *) mu, 1);                      320
  double Xm = gsl_vector_get( (gsl_vector *) mu, 2);
  double Xls = gsl_vector_get( (gsl_vector *) mu, 3);
  double Xlr = gsl_vector_get( (gsl_vector *) mu, 4);
  double B = gsl_vector_get( (gsl_vector *) mu, 5);
```

```
  double K = gsl_vector_get( (gsl_vector *) mu, 6);

  double lamqs = y[0];
  double lamds = y[1];
  double lamqr = y[2];
  double lamdr = y[3];                                                    330
  double wr = y[4];

  Lm = Xm/we;
  Lls = Xls/we;
  Llr = Xlr/we;
  Las = Lls + Lm;
  Lar = Llr + Lm;

  D = Lm*Lm - Las*Lar;
                                                                          340
  idr = (Lm*lamds - Las*lamdr)/D;
  iqr = (Lm*lamqs - Las*lamqr)/D;
  iqs = (Lm*lamqr - Lar*lamqs)/D;
  ids = (Lm*lamdr - Lar*lamds)/D;

  dydt[0] = (vqs - we*lamds - rs*iqs);
  dydt[1] = (vds + we*lamqs - rs*ids);
  dydt[2] = (vqr - (we - (P/2.0)*wr)*lamdr - rr*iqr);
  dydt[3] = (vdr + (we - (P/2.0)*wr)*lamqr - rr*idr);
                                                                          350
  T = (3.0/2.0)*(P/2.0)*(lamqr*idr - lamdr*iqr);

  dydt[4] = K*(T - B*wr*wr);
  dydt[5] = we;

  return GSL_SUCCESS;

}

void krausedude (gsl_vector *theta, void *data, gsl_vector *torque_diff) {  360

  // calculate the torque-speed points; analogous to krause.m.

  unsigned long int i;

  gsl_vector *theta_sc = ( (struct data *) data)->theta_sc;

  gsl_vector *speed_data = ( (struct data *) data)->speed_data;
  gsl_vector *torque_data = ( (struct data *) data)->torque_data;
                                                                          370
  double nph = 3.0;
  double P = 6.0;
  double we = 2.0*M_PI*60.0;
  double ws = we/(P/2.0);
  double V1 = 120.0;
  double s, alpha, beta, I2sq, Telec;

  double Rs = gsl_vector_get(theta_sc, 0)*gsl_vector_get(theta, 0);
  double Rr = gsl_vector_get(theta_sc, 1)*gsl_vector_get(theta, 1);
  double Xm = gsl_vector_get(theta_sc, 2)*gsl_vector_get(theta, 2);       380
```

```
  double Xls = gsl_vector_get(theta_sc, 3)*gsl_vector_get(theta, 3);
  double Xlr = gsl_vector_get(theta_sc, 4)*gsl_vector_get(theta, 4);

  double R1eq = (Xm*Xm*Rs)/(Rs*Rs + (Xm + Xls)*(Xm + Xls));
  double X1eq = (Xm*Rs*Rs + Xm*Xls*(Xm + Xls))/(Rs*Rs + (Xm + Xls)*(Xm + Xls));

  double s_sync = 3600.0/(P/2.0);

  for (i = 0; i < speed_data->size; i++) {
                                                                          390
    s = (s_sync - gsl_vector_get(speed_data, i))/s_sync;

    alpha = Rs*(R1eq + (Rr/s)) - (Xls + Xm)*(X1eq + Xlr);
    beta = Rs*(X1eq + Xlr) + (Xls + Xm)*(R1eq + (Rr/s));

    I2sq = (V1*V1*Xm*Xm)/(alpha*alpha + beta*beta);
    Telec = 141.6*(1/ws)*nph*I2sq*(Rr/s); // in oz-in

    gsl_vector_set(torque_diff, i, Telec); // put the result in here.
  }                                                                       400

  gsl_vector_sub(torque_diff, torque_data); // subtract from it the measured
      torque

}


void param_env (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
    {

  // properly pre-condition the parameters for the estimation; this little
      doohickey turns those
  // pre-conditioned parameters into something useful for the motor simulation.   410

  gsl_vector_set(theta_out, 0, R_STATOR); // Rs
  gsl_vector_set(theta_out, 1, gsl_vector_get(theta_sc, 0)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 0)) + 1.0)); // Rr
  gsl_vector_set(theta_out, 2, gsl_vector_get(theta_sc, 1)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 1)) + 1.0)); // Xm
  gsl_vector_set(theta_out, 3, gsl_vector_get(theta_sc, 2)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 2)) + 1.0)); // Xls
  gsl_vector_set(theta_out, 4, gsl_vector_get(theta_sc, 3)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 3)) + 1.0)); // Xlr
  gsl_vector_set(theta_out, 5, gsl_vector_get(theta_sc, 4)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 4)) + 1.0)); // B
  gsl_vector_set(theta_out, 6, gsl_vector_get(theta_sc, 5)*10.0/(exp(-0.8*
      gsl_vector_get(theta, 5)) + 1.0)); // K

}                                                                         420

void param_curr (gsl_vector *theta_out, gsl_vector *theta, gsl_vector *theta_sc)
    {

  // pre-conditioning again, different than last time.
```

```
  gsl_vector_set(theta_out, 0, gsl_vector_get(theta_sc, 0)*gsl_vector_get(theta,
      0)); // Rs
  gsl_vector_set(theta_out, 1, gsl_vector_get(theta_sc, 1)*gsl_vector_get(theta,
      1)); // Rr
  gsl_vector_set(theta_out, 2, gsl_vector_get(theta_sc, 2)*gsl_vector_get(theta,
      2)); // Xm
  gsl_vector_set(theta_out, 3, gsl_vector_get(theta_sc, 3)*gsl_vector_get(theta,
      3)); // Xls
  gsl_vector_set(theta_out, 4, gsl_vector_get(theta_sc, 4)*gsl_vector_get(theta,
      4)); // Xlr
  gsl_vector_set(theta_out, 5, gsl_vector_get(theta_sc, 5)*gsl_vector_get(theta,
      5)); // B
  gsl_vector_set(theta_out, 6, gsl_vector_get(theta_sc, 6)*gsl_vector_get(theta,
      6)); // K

}

void postproc_env (gsl_vector *y_out, gsl_vector *y_in, void *data) {

  gsl_vector *yobs = ( (struct data *) data)->yobs;
  gsl_vector *t_vec = ( (struct data *) data)->t;

  envelope_wrapper(y_out, y_in, t_vec); // generate the envelope
  gsl_vector_sub(y_out, yobs);

}

void postproc_curr (gsl_vector *y_out, gsl_vector *y_in, void *data) {

  gsl_vector *yobs = ( (struct data *) data)->yobs;
  gsl_vector_sub(y_in, yobs);
  gsl_vector_memcpy(y_out, y_in);

}

void motorsim (gsl_vector *theta, void *data, gsl_vector *y_out) {

  int i, k;

  double we = 2*M_PI*60.0;

  double lamqs, lamds, lamqr, lamdr;
  double Lm, Lls, Llr, Las, Lar, D;
  double iqs, ids, th;

  double t, ti;
  double h = 1e-6;

  gsl_vector *yobs = ( (struct data *) data)->yobs;
  gsl_vector *y0 = ( (struct data *) data)->y0;
  gsl_vector *t_vec = ( (struct data *) data)->t;
  gsl_vector *theta_sc = ( (struct data *) data)->theta_sc;

  long int N = yobs->size;
  int states = y0->size;
  double y[states];
```

```
gsl_interp_accel *Vd_acc = ( (struct data *) data)->Vd_acc;
gsl_interp_accel *Vq_acc = ( (struct data *) data)->Vq_acc;
gsl_spline *Vd_spline = ( (struct data *) data)->Vd_spline;
gsl_spline *Vq_spline = ( (struct data *) data)->Vq_spline;
                                                                        480
void (*param_gen) (gsl_vector *, gsl_vector *, gsl_vector *) = ((struct data
    *) data)->param_gen;
gsl_vector *mu_sim = gsl_vector_calloc(7); // # of parameters for the
    simulation (not necessarily for estimation)
param_gen(mu_sim, theta, theta_sc);

struct motor_helper Msim = {mu_sim, Vd_acc, Vq_acc, Vd_spline, Vq_spline};

double Xm = gsl_vector_get(mu_sim, 2);
double Xls = gsl_vector_get(mu_sim, 3);
double Xlr = gsl_vector_get(mu_sim, 4);
                                                                        490
Lm = Xm/we;
Lls = Xls/we;
Llr = Xlr/we;
Las = Lls + Lm;
Lar = Llr + Lm;
D = Lm*Lm - Las*Lar;

gsl_matrix *Data = gsl_matrix_calloc(N, states);

const gsl_odeiv_step_type *T = gsl_odeiv_step_rkf45;                    500

gsl_odeiv_step *s = gsl_odeiv_step_alloc(T, states);
gsl_odeiv_control *c = gsl_odeiv_control_y_new(1e-2, 1.0);
gsl_odeiv_evolve *e = gsl_odeiv_evolve_alloc(states);

gsl_odeiv_system sys = {motor, NULL, states, (void *) &Msim};

for (i = 0; i < states; i++)
  y[i] = gsl_vector_get(y0, i);
                                                                        510
t = gsl_vector_get(t_vec, 0);

for (i = 0; i < states; i++)
  gsl_matrix_set(Data, 0, i, y[i]);

for (i = 1; i < N; i++) {

  ti = gsl_vector_get(t_vec, i);

  while (t < ti)                                                        520
    gsl_odeiv_evolve_apply(e, c, s, &sys, &t, ti, &h, y);

  for (k = 0; k < states; k++)
    gsl_matrix_set(Data, i, k, y[k]);

}

gsl_vector *Ias = gsl_vector_calloc(N);
```

```
  gsl_vector *Ibs = gsl_vector_calloc(N);
  gsl_vector *Ics = gsl_vector_calloc(N);

  for (i = 0; i < N; i++) {

    lamqs = gsl_matrix_get(Data, i, 0);
    lamds = gsl_matrix_get(Data, i, 1);
    lamqr = gsl_matrix_get(Data, i, 2);
    lamdr = gsl_matrix_get(Data, i, 3);
    th = gsl_matrix_get(Data, i, 5);

    iqs = (Lm*lamqr - Lar*lamqs)/D;
    ids = (Lm*lamdr - Lar*lamds)/D;

    gsl_vector_set(Ias, i, (cos(th)*ids - sin(th)*iqs));
    gsl_vector_set(Ibs, i, (cos(th - 2*M_PI/3)*ids - sin(th - 2*M_PI/3)*iqs));
    gsl_vector_set(Ics, i, (cos(th + 2*M_PI/3)*ids - sin(th + 2*M_PI/3)*iqs));

    gsl_vector_set(y_out, i, gsl_vector_get(Ias, i));

  }

  gsl_vector *y_out_proc = gsl_vector_calloc(y_out->size); // allocate the
      envelope

  void (*postproc) (gsl_vector *, gsl_vector *, void *) = ((struct data *) data)
      ->postproc;
  postproc(y_out_proc, y_out, data);   // form the residual
  gsl_vector_memcpy(y_out, y_out_proc); // put the result back in y_out

  gsl_odeiv_evolve_free(e);
  gsl_odeiv_control_free(c);
  gsl_odeiv_step_free(s);

  gsl_vector_free(y_out_proc);
  gsl_vector_free(mu_sim);
  gsl_matrix_free(Data);

  gsl_vector_free(Ias);
  gsl_vector_free(Ibs);
  gsl_vector_free(Ics);

}

void envelope_filter (gsl_vector *data_out, gsl_vector *data_in) {

  // this function should just run the given dataset through the filter.

  long int i;
  double b[4], a[4];

  b[0] = 3.565800529847785e-08; // all of these coefficients are for this
      particular configuration
  b[1] = 1.069740158954335e-07;
  b[2] = 1.069740158954335e-07;
  b[3] = 3.565800529847785e-08;
```

```
  a[0] = 1.0;
  a[1] = -2.986805334783035e+00;
  a[2] = 2.973697575591248e+00;
  a[3] = -9.868919555441706e-01;

  gsl_vector *zi_init = gsl_vector_calloc(3);
  gsl_vector_set(zi_init, 0, 9.999999648979718e-01);
  gsl_vector_set(zi_init, 1, -1.986805478519675e+00);                    590
  gsl_vector_set(zi_init, 2, 9.868919917508654e-01);

  gsl_vector *zi = gsl_vector_calloc(zi_init->size);

  for (i = 0; i < zi->size; i++)
   gsl_vector_set(zi, i, gsl_vector_get(data_in, 0)*gsl_vector_get(zi_init, i));

  gsl_vector_set(data_out, 0, b[0]*gsl_vector_get(data_in, 0) + gsl_vector_get(
      zi, 0));
  gsl_vector_set(zi, 0, b[1]*gsl_vector_get(data_in, 0) + gsl_vector_get(zi, 1)
      - a[1]*gsl_vector_get(data_out, 0));
  gsl_vector_set(zi, 1, b[2]*gsl_vector_get(data_in, 0) + gsl_vector_get(zi, 2)   600
      - a[2]*gsl_vector_get(data_out, 0));
  gsl_vector_set(zi, 2, b[3]*gsl_vector_get(data_in, 0) - a[3]*gsl_vector_get(
      data_out, 0));

  for (i = 1; i < data_in->size; i++) {
   gsl_vector_set(data_out, i, b[0]*gsl_vector_get(data_in, i) + gsl_vector_get(
       zi, 0));
   gsl_vector_set(zi, 0, b[1]*gsl_vector_get(data_in, i) + gsl_vector_get(zi, 1)
       - a[1]*gsl_vector_get(data_out, i));
   gsl_vector_set(zi, 1, b[2]*gsl_vector_get(data_in, i) + gsl_vector_get(zi, 2)
       - a[2]*gsl_vector_get(data_out, i));
   gsl_vector_set(zi, 2, b[3]*gsl_vector_get(data_in, i) - a[3]*gsl_vector_get(
       data_out, i));
  }

  gsl_vector_free(zi);                                                   610
  gsl_vector_free(zi_init);

}


void envelope_gen (gsl_vector *out, gsl_vector *x) {

  long int i;
  int nfact = 9;
                                                                         620
  gsl_vector *y = gsl_vector_calloc(x->size+(2*nfact));
  gsl_vector *y_temp = gsl_vector_calloc(x->size+(2*nfact));

  for (i = 0; i < nfact; i++)
   gsl_vector_set(y, i, 2.0*gsl_vector_get(x, 0) - gsl_vector_get(x, nfact-i));

  for (i = 0; i < x->size; i++)
   gsl_vector_set(y, i+nfact, gsl_vector_get(x, i));
```

```
 for (i = 0; i < nfact; i++)                                                    630
  gsl_vector_set(y, nfact+(x->size)+i, 2.0*gsl_vector_get(x, (x->size)-1) -
      gsl_vector_get(x, (x->size)-1-i));

 envelope_filter(y_temp, y);
 gsl_vector_memcpy(y, y_temp); // so that y always has the data we care about.
 gsl_vector_reverse(y);

 envelope_filter(y_temp, y);
 gsl_vector_memcpy(y, y_temp); // so that y always has the data we care about.
 gsl_vector_reverse(y);
                                                                                640
 for (i = 0; i < out->size; i++)
  gsl_vector_set(out, i, gsl_vector_get(y, nfact+i));

 gsl_vector_free(y);
 gsl_vector_free(y_temp);

}


void envelope_wrapper(gsl_vector *data_out, gsl_vector *data_in, gsl_vector *t)  650
    {

 long int i;
 double ti;
 double we = 2.0*M_PI*60.0;

 gsl_vector *P = gsl_vector_calloc(data_in->size);
 gsl_vector *Q = gsl_vector_calloc(data_in->size);
 gsl_vector *Penv = gsl_vector_calloc(data_in->size);
 gsl_vector *Qenv = gsl_vector_calloc(data_in->size);
                                                                                660
 for (i = 0; i < data_in->size; i++) {
  ti = gsl_vector_get(t, i);
  gsl_vector_set(P, i, cos(we*ti)*gsl_vector_get(data_in, i));
  gsl_vector_set(Q, i, sin(we*ti)*gsl_vector_get(data_in, i));
 }

 envelope_gen(Penv, P);
 envelope_gen(Qenv, Q);

 for (i = 0; i < P->size; i++)                                                  670
  gsl_vector_set(data_out, i, 2.0*sqrt(pow(gsl_vector_get(Penv, i), 2.0) + pow(
      gsl_vector_get(Qenv, i), 2.0)));

 gsl_vector_free(P);
 gsl_vector_free(Q);
 gsl_vector_free(Penv);
 gsl_vector_free(Qenv);

}
                                                                                680
void fdjac(void (*fcn)(gsl_vector *, void *, gsl_vector *), gsl_vector *theta,
    void *data, gsl_vector *fvec, gsl_matrix *fjac) {
```

```
  long int i;                          // initialize variables
  double theta_sign;
  double h;
  double small = sqrtl(DBL_EPSILON); // the smallest step to take
  double smu;                    // to save the theta in
  double maxstep = 0.1;          // the biggest step to take

  gsl_vector *fvec_temp = gsl_vector_calloc(fvec->size);                    690

  gsl_vector *step_vec_a = gsl_vector_calloc(2); // helping vectors for
     determining step size
  gsl_vector *step_vec_b = gsl_vector_calloc(2);

  gsl_vector_set(step_vec_a, 0, small); // setting some parameters of help
     vectors
  gsl_vector_set(step_vec_b, 1, maxstep);

  gsl_vector_set_zero(fvec);           // initialize the residual
  gsl_matrix_set_zero(fjac);           // initialize the jacobian
                                                                            700
  fcn(theta, data, fvec);          // record the evaluation at this theta

  for (i = 0; i < theta->size; i++) {

    smu = gsl_vector_get(theta, i);     // save the theta before perturbing it

    theta_sign = GSL_SIGN(gsl_vector_get(theta, i));
    gsl_vector_set(step_vec_a, 1, fabs(small*gsl_vector_get(theta, i)));
    gsl_vector_set(step_vec_b, 0, gsl_vector_max(step_vec_a));
    h = theta_sign*gsl_vector_min(step_vec_b); // calculate the size of the test   710
       step

    gsl_vector_set(theta, i, gsl_vector_get(theta, i) + h); // temporarily
       perturb theta

    fcn(theta, data, fvec_temp);   // calculate output with temp theta

    gsl_vector_sub(fvec_temp, fvec);    // return the result in fvec_temp
    gsl_vector_scale(fvec_temp, (1.0/h)); // return the result in fvec_temp again
    gsl_matrix_set_col(fjac, i, fvec_temp); // save result in col of jacobian

    gsl_vector_set(theta, i, smu);      // restore value of theta                720
  }

  gsl_vector_free(fvec_temp);          // clean up
  gsl_vector_free(step_vec_a);
  gsl_vector_free(step_vec_b);

}

void estimate (gsl_vector *theta_out, void *data) {
                                                                            730
  gsl_vector *yobs = ((struct data *) data)->yobs;
  gsl_vector *theta0 = ((struct data *) data)->theta;
  long int TS_N = ((struct data *) data)->TS_N;
```

```
 unsigned long int i, iter = 0;
 long int N = yobs->size;
 int nparams = theta0->size;
 long int status;
 long int maxiter = 1000*nparams;

 const gsl_multifit_fdfsolver_type *T;
 gsl_multifit_fdfsolver *s;
 gsl_multifit_function_fdf f;

 T = gsl_multifit_fdfsolver_lmsder;

 f.f = ((struct data *) data)->motor_f;
 f.df = ((struct data *) data)->motor_df;
 f.fdf = ((struct data *) data)->motor_fdf;
 f.p = nparams;
 f.params = data;
 f.n = N+TS_N+nparams;

 s = gsl_multifit_fdfsolver_alloc (T, N+TS_N+nparams, nparams);
 gsl_multifit_fdfsolver_set(s, &f, theta0);
 iter = 0;

 do {

   iter++;
   status = gsl_multifit_fdfsolver_iterate (s);

   if (status)
     break;

   status = gsl_multifit_test_delta(s->dx, s->x, 1e-4, 1e-4);

 } while (status == GSL_CONTINUE && iter < maxiter);

 printf("number of iterations = %ld\n", iter);

 for (i = 0; i < nparams; i++)
   gsl_vector_set(theta_out, i, gsl_vector_get(s->x, i));

 gsl_multifit_fdfsolver_free(s);

}


/*****************************************
 Do the estimation.
*****************************************/

int main (int argc, char *argv[]) {

 unsigned long int i;
 double regp = 0.01;

 double fIa, fIb, fIc, fVd, fVq, fV0;
```

```
double fTd, fSd;
double fs = 100e3/7;   // sample frequency
double Ts = 1/fs;      // sample period
unsigned long int N = 75000; // the length of the data vector.
int states = 6;        // the number of state variables for the simulation.

unsigned long int TS_N = 15; // the number of torque-speed data entries: 15
    for hot
// unsigned long int TS_N = 7; // the number of torque-speed data entries: 7
    for cold

gsl_vector *Ia = gsl_vector_calloc(N); // the size of the datafile
gsl_vector *Ib = gsl_vector_calloc(N); // the size of the datafile
gsl_vector *Ic = gsl_vector_calloc(N); // the size of the datafile

gsl_vector *speed_data = gsl_vector_calloc(TS_N);
gsl_vector *torque_data = gsl_vector_calloc(TS_N);

double Vd[N], Vq[N], t_spline[N];


/* load the motor current and voltage data. */
FILE *MOTOR_IN = fopen(argv[1], "r"); // The input file

if (MOTOR_IN == NULL)
 err(1, "can't open file");

for (i = 0; i < N; i++) {
 fscanf(MOTOR_IN, "%lf %lf %lf %lf %lf %lf", &fIa, &fIb, &fIc, &fVd, &fVq, &
     fV0);

 gsl_vector_set(Ia, i, fIa);
 gsl_vector_set(Ib, i, fIb);
 gsl_vector_set(Ic, i, fIc);
 Vd[i] = fVd;
 Vq[i] = fVq;
}

fclose(MOTOR_IN);


/* load the torque-speed data. */
FILE *TSDATA_IN = fopen("hot_torque_speed_data.asc", "r");
// FILE *TSDATA_IN = fopen("cold_torque_speed_data.asc", "r");

if (TSDATA_IN == NULL)
 err(1, "can't open file");

for (i = 0; i < TS_N; i++) { //
 fscanf(TSDATA_IN, "%lf %lf", &fSd, &fTd);

 gsl_vector_set(speed_data, i, fSd);
 gsl_vector_set(torque_data, i, fTd);
}

fclose(TSDATA_IN);
```

```
/* set up the matrices and initialize them. */
gsl_vector *y0 = gsl_vector_calloc(states);
gsl_vector *t = gsl_vector_calloc(N);

for (i = 0; i < t->size; i++) {          // Populate the time vector and array
    .
 gsl_vector_set(t, i, i*Ts);                                                    850
 t_spline[i] = i*Ts;
}

gsl_vector *Ia_filt = gsl_vector_calloc(Ia->size); // the envelope vector.
envelope_wrapper(Ia_filt, Ia, t);       // generate the envelope.


/* Do the pre-estimation step. */

int params_env = 6;          // the number of parameters to pre-estimate.        860

gsl_vector *theta_sc_pre = gsl_vector_calloc(params_env); // The parameter
    scales
gsl_vector_set(theta_sc_pre, 0, 1.0); // Rr
gsl_vector_set(theta_sc_pre, 1, 10.0); // Xm
gsl_vector_set(theta_sc_pre, 2, 1.0); // Xls
gsl_vector_set(theta_sc_pre, 3, 1.0); // Xlr
gsl_vector_set(theta_sc_pre, 4, 1e-4); // B
gsl_vector_set(theta_sc_pre, 5, 10.0); // K

// initial guess                                                                870
gsl_vector *theta0_pre = gsl_vector_calloc(params_env);
gsl_vector_set_all(theta0_pre, 1.0);

/* set up the spline interpolation thingy. */
gsl_interp_accel *Vd_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Vq_acc = gsl_interp_accel_alloc();
gsl_spline *Vd_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline *Vq_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline_init(Vd_spline, t_spline, Vd, N);
gsl_spline_init(Vq_spline, t_spline, Vq, N);                                    880

struct data Dpre = {&motorsim, &krausedude, &param_env, &postproc_env, &
    motor_f_pre, &motor_df_pre, &motor_fdf_pre, y0, Ia_filt, t, 0, speed_data,
    torque_data, Vd_acc, Vq_acc, Vd_spline, Vq_spline, theta_sc_pre, theta0_pre
    , regp};

gsl_vector *theta_out1 = gsl_vector_calloc(params_env);
gsl_vector *theta_pre = gsl_vector_calloc(params_env+1);
printf("pre-estimation: ");
estimate(theta_out1, &Dpre);
param_env(theta_pre, theta_out1, theta_sc_pre);

                                                                                890
/* Do the final estimation step. */

int params_curr = 7; // the number of parameters to estimate.
```

```
gsl_vector *theta_sc = gsl_vector_calloc(params_curr); // The parameter scales
gsl_vector_set(theta_sc, 0, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 0)
    )))); // Rs
gsl_vector_set(theta_sc, 1, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 1)
    )))); // Rr
gsl_vector_set(theta_sc, 2, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 2)
    )))); // Xm
gsl_vector_set(theta_sc, 3, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 3)
    )))); // Xls
gsl_vector_set(theta_sc, 4, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 4)
    )))); // Xlr
gsl_vector_set(theta_sc, 5, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 5)
    )))); // B
gsl_vector_set(theta_sc, 6, pow(10.0, floor(log10(gsl_vector_get(theta_pre, 6)
    )))); // K

// initial guess
gsl_vector *theta0 = gsl_vector_calloc(params_curr);
gsl_vector_set(theta0, 0, gsl_vector_get(theta_pre, 0)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 0))))); // Rs
gsl_vector_set(theta0, 1, gsl_vector_get(theta_pre, 1)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 1))))); // Rr
gsl_vector_set(theta0, 2, gsl_vector_get(theta_pre, 2)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 2))))); // Xm
gsl_vector_set(theta0, 3, gsl_vector_get(theta_pre, 3)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 3))))); // Xls
gsl_vector_set(theta0, 4, gsl_vector_get(theta_pre, 4)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 4))))); // Xlr
gsl_vector_set(theta0, 5, gsl_vector_get(theta_pre, 5)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 5))))); // B
gsl_vector_set(theta0, 6, gsl_vector_get(theta_pre, 6)/pow(10.0, floor(log10(
    gsl_vector_get(theta_pre, 6))))); // K

struct data Dcurr = {&motorsim, &krausedude, &param_curr, &postproc_curr, &
    motor_f_tsi, &motor_df_tsi, &motor_fdf_tsi, y0, Ia, t, TS_N, speed_data,
    torque_data, Vd_acc, Vq_acc, Vd_spline, Vq_spline, theta_sc, theta0, regp};

gsl_vector *theta_out2 = gsl_vector_calloc(params_curr);
gsl_vector *theta_final = gsl_vector_calloc(params_curr);
printf("estimation: ");
estimate(theta_out2, &Dcurr);
param_curr(theta_final, theta_out2, theta_sc);


/* Report the results. */

FILE *MOTOR_OUT = fopen(argv[2], "w");
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 0)); // Rs
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 1)); // Rr
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 2)); // Xm
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 3)); // Xls
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 4)); // Xlr
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 5)); // B
fprintf(MOTOR_OUT, "%.5e ", gsl_vector_get(theta_final, 6)); // K
fprintf(MOTOR_OUT, "\n");
```

```
  fclose(MOTOR_OUT);


  /* Free the allocated space. */

  gsl_vector_free(Ia);
  gsl_vector_free(Ib);                                                    940
  gsl_vector_free(Ic);

  gsl_vector_free(theta_sc_pre);
  gsl_vector_free(theta0_pre);
  gsl_vector_free(theta0);
  gsl_vector_free(theta_sc);

  gsl_vector_free(Ia_filt);
  gsl_vector_free(theta_pre);
  gsl_vector_free(theta_final);                                           950
  gsl_vector_free(y0);
  gsl_vector_free(t);

  gsl_spline_free(Vd_spline);
  gsl_spline_free(Vq_spline);
  gsl_interp_accel_free(Vd_acc);
  gsl_interp_accel_free(Vq_acc);

  return(0);
                                                                          960
}
```

A.4.3 filter_gen.m

```
% generate the coefficients used in the filtering method for the envelope      0
% in the motor parameter estimation.

Fs = 100e3/7;
fc = 0.25*60/(Fs/2); % cutoff frequency
[b,a] = butter(3, fc);

xp = Ip;

nfilt = 4;
nfact = 3*(nfilt-1); % length of edge transients                              10

% set up filter's initial conditions to remove dc offset problems at the
% beginning and end of the sequence

% use sparse matrix to solve system of linear equations for initial conditions
% zi are the steady-state states of the filter b(z)/a(z) in the state-space
% implementation of the 'filter' command.

A = [eye(nfilt-1) - [-a(2:nfilt).' [eye(nfilt-2); zeros(1,nfilt-2)]]];
bhat = [b(2:nfilt).' - a(2:nfilt).'*b(1)];                                    20

zi = A \ bhat;
```

```
% and save the coefficients from zi in the estimation code.
```

A.5 Airflow Prediction Generation

A.5.1 speed_flow_proc.m

```
% calculate how good the flow is, given the parameters and so forth of the
% system

% code taken from fan-diag/2008-02-21/speed_flow_proc.m

clear
clear all

% get the speed information.

speedfile = load('~/static/fan-diag/2008-02-21/testdata_rpm');
t1 = speedfile(:,1);
s1 = speedfile(:,2);

t1o = zeros(size(t1));

for (k = 1:length(t1o))
   t1o(k) = datenum(2008, 02, 21, 20, 43, t1(k));
end

plot(t1o, s1)
datetick('x', 15);

% unblocked speed
mean(s1(1:5.5e4)) % the mean of this signal is 999.18 rpm.

% estimate the airflow n' stuff.

airflow_proc % this loads all of the airflow data.
mean_mat = zeros(size(midu01));
std_mat = zeros(size(midu01));

for k = (1:6);
   for l = (1:4);
      mean_mat(k,l) = mean([midu01(k,l) midu02(k,l)]);
      std_mat(k,l) = std([midu01(k,l) midu02(k,l)]);
   end
end

mean(mean(mean_mat)) % the mean unblocked airflow is 8.13 m/s from these 5
    traverses
mean(mean(mean_mat-2*std_mat)) % the lower bound for unblocked airflow is 7.90 m
    /s
mean(mean(mean_mat+2*std_mat)) % the upper bound for unblocked airflow is 8.36 m
    /s

% the volumetric flow through the 16 inch duct is
% (x m/s)*(3.2808 ft/m)*(60 sec/min)*(8.125 in)^2*pi*(1 ft^2/144 in^2)
```

```
%                                    flow: [(lower bound) (mean) (upper bound)]
[7.90 8.13 8.36]*3.2808*60*(8.125)^2*pi/144 % unblocked flow: [2239.71 2304.92
    2370.13]

%# conversion factors.
%# power = torque * 2*pi * rpm                                                  50
%# 141.6 oz-in = 1 N-m
%# 192 oz-in = 1 ft-lb
%# 1 hp = (550 ft-lb)/(sec)
%# 1 W = (1 N-m)/sec
%# 0.73756 N-m = 1 ft-lb
%# horsepower = (x oz-in)[(1 ft-lb)/(192 oz-in)][1/550][2*pi/60](y rpm)
%# for the blocked case: (300)*(1/192)*(1/550)*(2*pi/60)*(1164) = 0.3463 hp
%# for the unblocked case: (980)*(1/192)*(1/550)*(2*pi/60)*(975) = 0.9475 hp

                                                                                60

%# the DD11-10 fan parameters: brake horsepower coefficients
FP = load('fan_curve_params');

cfm = (1:1:3330)';
CFM = [ones(length(cfm),1) cfm cfm.^2 cfm.^3 cfm.^4 cfm.^5];
FPcurve = CFM*FP';

FPcurve_u = FPcurve.*(999/1000)^3;
plot(cfm, FPcurve_u, cfm(2250:2380), FPcurve_u(2250:2380), 'ro');
axis([1800 2800 0.6 1.2]); xlabel('airflow (cfm)');                             70

% calculate the measured horsepower from the torque speed curve.

motorparams % load the measured set of motor parameters.

% unblocked data: speed = 999 rpm.
[pout0206A, flow0206A] = flow_est(muh0206A, 999) % [1.0150 2519]
[pout0206B, flow0206B] = flow_est(muh0206B, 999) % [1.0230 2532];

[pout0207A, flow0207A] = flow_est(muh0207A, 999) % [1.0041 2500];               80
[pout0207B, flow0207B] = flow_est(muh0207B, 999) % [1.0014 2496];

[pout0208A, flow0208A] = flow_est(muh0208A, 999) % [1.0041 2500];
[pout0208B, flow0208B] = flow_est(muh0208B, 999) % [1.0106 2511];

[pout0212A, flow0212A] = flow_est(muh0212A, 999) % [0.99237 2480];
[pout0212B, flow0212B] = flow_est(muh0212B, 999) % [0.97679 2453];
```

A.5.2 flow_est.m

```
function [pout, flow] = flow_est(params, speed)                                 0

% estimate the flow given motor parameters and the motor speed.
% the speed is in rpm, the params are as they come out of the estimation code.

   % motor parameters.
   Rs = params(1);
   Rr = params(2);
```

```
Xm = params(3);
Xls = params(4);
Xlr = params(5);
B = params(6);
K = params(7);

% simulate the t-s curve
fe = 60;
V = 120;
P = 6;
mu = [Rs; Rr; Xm; Xls; Xlr];
p = [V; P; fe; mu];
Tsim = krause(p, speed);

%# horsepower = (x N-m)[(141.6 oz-in)/(1 N-m)][(1 ft-lb)/(192 oz-in)
    ][1/550][2*pi/60](y rpm)
pout = (Tsim)*(141.6)*(1/192)*(1/550)*(2*pi/60)*(speed);

%# the DD11-10 fan parameters: brake horsepower coefficients
FP = load('fan_curve_params');

cfm = (1:1:2700)';
CFM = [ones(length(cfm),1) cfm cfm.^2 cfm.^3 cfm.^4 cfm.^5];
FPcurve = CFM*FP';

FPcurve1 = FPcurve.*(speed/1000)^3;

indx = find(abs(FPcurve1-pout)==min(abs(FPcurve1-pout)));
flow = cfm(indx);
```

Appendix B

Liquid Slugging Information

This appendix contains the variety of code needed to execute and process the liquid slugging data described in Chapter 3, as well as drawings of the new compressor head and electrical schematics. There are six sections in this appendix.

§**B.1:** A representative solenoid control program is given which was used to collect the data in this chapter.

§**B.2:** This section contains a demonstration of the operation of the full preprocessor in Matlab or Octave. The main piece of code which performs the preprocessing is in `dq0preproc.m`, while the remainder of the programs are helper functions. In particular, `GNL3.m`, `findk3.m`, and `fdjac23.m` are analogues of the equivalent programs given in Appendix A, but these operate on three phases of observations, rather than on just one. This version of the preprocessor is presented for ease of understanding.

§**B.3:** The same function of the preprocessor in the previous section, but much better performance can be achieved by implementing this version, coded in C.

§**B.4:** Two useful helper functions are provided; `process.pl` takes a datafile with raw currents and voltages and identifies the location of transients and chops the datafile up into pieces containing those transients, while `indexsift.pl` takes a file containing the indices of events and pulls the data at those indices out of a datafile.

§**B.5:** The set of compressor drawings for the new head.

§**B.6:** Schematics for the instrumentation board and the solenoid control board are given in this section.

B.1 Solenoid Control Program

B.1.1 slug_delay.c

```
// cycling program for compressor and solenoid. interleave slug and nonslug        0
   starts.

#include <pic.h>
#include <pic168xa.h>
```

```
// Appropriate config words. Low-voltage programming is disabled
__CONFIG(HS & WDTDIS & PWRTEN & BOREN & LVPDIS);

void putch(unsigned char byte) {
  while (!TXIF); /* Txif is set when TXREG is empty */                    10
  TXREG = byte; /* Move the the ascii char to the tx register */
}

unsigned char getbyte() {
  while (RCIF == 0); // Bit is set when data in RCR has been xferred to RCREG
  return RCREG;
}

int start_count = 0;
int run_count = 0;                                                        20
int sec_count = 0;
int max_times = 0;
int slug_number_one = 250; // the number of cycles for the solenoid.
int slug_number_two = 100; // the number of cycles for the solenoid.
int fill_number = 4;
int on_time = 14400;
int off_time = 14400;
int marker_time = 1200;

void main(void) {                                                         30

  unsigned char byte='0';

  /* Interrupt Initialization */

  TRISA0 = 0;
  TRISA1 = 0;
  TRISA3 = 0;
  TRISA5 = 0;
  TRISB1 = 0;                                                             40
  TRISB2 = 0;
  TRISB0 = 1; // probably need to change this line, given that the interrupt is
      only one line

  RA0 = 0; // Compressor relays
  RA1 = 0;
  RB2 = 0;

  // RB1 = 0; // Condenser fan
  // RA3 = 0; // Evaporator fan
  RA5 = 0; // Solenoid relay                                              50

  INTF = 0; // Clear interrupt
  PIR1 = 0; // Clearing peripheral interrupts. I'm not sure if I should,
  PIE1 = 0; // but it seems reasonable given that I'm not using them.
  INTEDG = 1; // Interrupt on rising edge
  INTCON = 0x00;
  // INTCON = 0x90; // Global enable and INTE enabled
  // PORTB = 0x00;
```

```
 while(1) {

   byte=getbyte();

   if (byte == 'g') {

     RA0 = 0; // compressor relays
     RA1 = 0;
     RB2 = 0;

     RA5 = 0; // solenoid relay

     INTCON = 0x90;

   }

   if (byte == 's') {

     INTCON = 0x00;

     RA0 = 0; // compressor relays
     RA1 = 0;
     RB2 = 0;

     RA5 = 0; // solenoid relay
   }

 }
}

#define FOSC 20000000UL
#define CYCLES_PER_USEC ((FOSC / 4) / 1000000UL)
#define __delay_cycles(x) do { \
      unsigned char _i; \
      _i = ((x) - 1) / 3; \
      while(--_i != 0) continue; \
      if((((x) - 1) % 3) >= 1) asm("nop"); \
      if((((x) - 1) % 3) >= 2) asm("nop"); \
      } while(0)
static void usleep(unsigned short microseconds)
{
 microseconds -= 11;
 while(microseconds >= 16) {
   __delay_cycles(16UL * CYCLES_PER_USEC - 13);
   microseconds -= 16;
 }
}


static void interrupt isr (void) {

 if (INTF) {

   //   usleep(4166);
```

```
if (max_times == 100) {

  INTCON = 0x00;

}                                                                    120

if (start_count == (off_time-marker_time)) {

  putch('M');

}

if (start_count == off_time) {

  if (run_count <= slug_number_one) {                                130

    RA5 = 1;

    RA0 = 0;
    RA1 = 0;
    RB2 = 0;

    run_count++;
  }
  else if (run_count > slug_number_one && run_count <= slug_number_two) {   140

    RA5 = 1;

    RA0 = 1;
    RA1 = 1;
    RB2 = 1;

    run_count++;
  }
  else if (run_count > slug_number_two && run_count <= 1800) {       150

    RA5 = 0;

    RA0 = 1;
    RA1 = 1;
    RB2 = 1;

    run_count++;
  }
  else if (run_count > 1800 && run_count <= (1800 + fill_number) ) {  160

    RA0 = 1;
    RA1 = 1;
    RB2 = 1;

    RA5 = 1;
    run_count++;
  }
  else if (run_count > (1800 + fill_number) && run_count <= on_time) {
                                                                     170
```

```
  RA5 = 0;

  RA0 = 1;
  RA1 = 1;
  RB2 = 1;

  run_count++;
 }
 else if (run_count == (on_time + 1)) {
                                                                          180
  if (sec_count < (off_time-marker_time)) {

   RA5 = 0;

   RA0 = 0;
   RA1 = 0;
   RB2 = 0;

   sec_count++;
  }                                                                       190
  else if (sec_count == (off_time-marker_time)) {

   putch('M');

   RA5 = 0;

   RA0 = 0;
   RA1 = 0;
   RB2 = 0;
                                                                          200
   sec_count++;
  }
  else if (sec_count > (off_time-marker_time) && sec_count <= off_time) {

   RA5 = 0;

   RA0 = 0;
   RA1 = 0;
   RB2 = 0;
                                                                          210
   sec_count++;
  }
  else if (sec_count > off_time && sec_count < (off_time + on_time) ) {

   RA5 = 0;

   RA0 = 1;
   RA1 = 1;
   RB2 = 1;
                                                                          220
   sec_count++;
  }
  else {

   RA0 = 0;
   RA1 = 0;
```

```
        RB2 = 0;

        RA5 = 0;
        run_count = 0;
        sec_count = 0;
        start_count = 0;
        max_times++;
       }
     }
   }
   else {
    start_count++;
   }

   INTF = 0;
  }
}
```

B.2 Matlab DQ0 Preprocessor

B.2.1 dq0preproc.m

```
% matlab version of preprocessor, to test out all strategies.
% continuation of work from 2008-04-21/3.

clear;
clear all;

% load observed datafile from compressor.
load ~/static/rtu-diag/2008-04-10/test01_cmp.001

% do the interpolation in a more straightforward way.

ind = (1:length(test01_cmp))';

fs = 192e3/24;
Ts = 1/fs;
F0 = 60.0;
N = fs/F0;

ind1 = (ind-1);
ind2 = (ind-1)+(1/6);
ind3 = (ind-1)+(1/3);
ind4 = (ind-1)+(1/2);
ind5 = (ind-1)+(2/3);
ind6 = (ind-1)+(5/6);

Vab_interp = test01_cmp(:,1);
Ian_interp = interp1(Ts*ind2, test01_cmp(:,2), Ts*ind1, 'spline');
Vbc_interp = interp1(Ts*ind3, test01_cmp(:,3), Ts*ind1, 'spline');
Ibn_interp = interp1(Ts*ind4, test01_cmp(:,4), Ts*ind1, 'spline');
Vca_interp = interp1(Ts*ind5, test01_cmp(:,5), Ts*ind1, 'spline');
Icn_interp = interp1(Ts*ind6, test01_cmp(:,6), Ts*ind1, 'spline');
```

```
Iabc = [Ian_interp Ibn_interp Icn_interp](2:size(ind,1),:);

deltatowye = [1 -1 0; 0 1 -1; -1 0 1; 1 1 1];
Vdelta = [Vab_interp Vbc_interp Vca_interp](2:size(ind,1),:);
Vwye = zeros(size(Vdelta));

for k = (1:length(Vdelta))
   Vwye(k,:) = (deltatowye \ [Vdelta(k,:) 0]')';
end                                                                    40


% estimate the characteristics of the first two complete cycles

start = find(Vwye(1:ceil(N),1)==max(Vwye(1:ceil(N),1)))+4;

ind = start+(1:(length(Vwye)-start))';
yobs = Vwye(ind,:);
ind_obs = (1:length(ind))';
tobs = Ts*ind_obs;                                                     50

yobs_short = Vwye(ind(1:ceil(N)), :);
ind_short = (1:length(yobs_short))';
t_short = Ts*(ind_short-1);


% first step of estimation: estimate all parameters.

K = 1;
Kvec = []; iter = [];                                                  60
mu = [3000; 2*pi*60; 0.1];

while (K < length(yobs_short))

   yhat = cosfunc3a(mu, zeros(size(yobs_short)), t_short, size(yobs_short,1));
   y = [yhat(1:(size(yhat,1)/3)) yhat((size(yhat,1)/3+1):(2*size(yhat,1)/3))
       yhat((2*size(yhat,1)/3+1):size(yhat,1))];

   interval = findk3(yobs_short(K:length(yobs_short),:), y(K:length(y),:), 0.2);
   K = min([size(yobs_short,1) K+interval]);
                                                                       70
   Kvec = [Kvec; K];
   mu = GNL('cosfunc3a', mu, yobs_short, t_short, K);

end

Vmag = mu(1);
omega_e = mu(2);
phase_1 = mu(3);

                                                                       80
% second step: estimate the variation in the phase.

phase_est = zeros(size(yobs));
phi_prev = 0;

for i = 1:ceil(N):size(yobs,1)
```

```
   ymin = yobs((i:i+ceil(N)-1),:);
   tmin = tobs(i:i+ceil(N)-1);

   mu = 0.1;
   K = 1; Kvec = []; iter = [];

   while (K < length(ymin))

      yhat = cosfunc3b(mu, zeros(size(ymin)), tmin, Vmag, omega_e, length(ymin),
           phi_prev);
      y = [yhat(1:(size(yhat,1)/3)) yhat((size(yhat,1)/3+1):(2*size(yhat,1)/3))
          yhat((2*size(yhat,1)/3+1):size(yhat,1))];

      interval = findk3(ymin(K:length(ymin),:), y(K:length(y),:), 0.2);
      K = min([length(ymin) K+interval]);

      Kvec = [Kvec; K];
      mu = GNL3('cosfunc3b', mu, ymin, tmin, Vmag, omega_e, K, phi_prev);

   end

   phi_prev = mu + phi_prev;

   if (phi_prev > 2*pi)
      phi_prev = phi_prev - 2*pi;
   elseif (phi_prev < -2*pi)
      phi_prev = phi_prev + 2*pi;
   end

   phase_est(i) = phi_prev;

   if (rem(i-1, ceil(N)*500) == 0)
      warning('length = %d', i);
   end

end

indx = find(phase_est);
uphase = unwrap(phase_est(indx));

A = [tobs(indx) ones(size(indx))];
b = uphase;
mu_fit = A\b;

% time variation that can't be fit with static coefficients.
plot(tobs(indx), uphase-A*mu_fit);

% interpolate the phase to compute full fitted sine wave.
indx_interp = (indx(1):indx(length(indx)))';
t_interp = tobs(indx_interp);
phase_interp = interp1(tobs(indx), uphase, t_interp, 'spline');

% test these voltage estimates.

yhat = Vmag*cos(omega_e*t_interp + mod(phase_interp, 2*pi));
```

```
yres = yobs(indx_interp,1)-yhat;
% plot(t_interp, yres);

yhat = Vmag*cos(omega_e*t_interp + mod(phase_interp, 2*pi) - 2*pi/3);
yres = yobs(indx_interp,2)-yhat;
% plot(t_interp, yres);

yhat = Vmag*cos(omega_e*t_interp + mod(phase_interp, 2*pi) + 2*pi/3);
yres = yobs(indx_interp,3)-yhat;
% plot(t_interp, yres);                                                    150


% do the dq0 transformation.

Vdq0 = zeros(length(t_interp),3);
Idq0 = zeros(length(t_interp),3);

for k = 1:length(t_interp)

   th = omega_e*t_interp(k) + phase_interp(k);                             160

   T = (2/3)*[ cos(th) cos(th-2*pi/3) cos(th+2*pi/3);
            -sin(th) -sin(th-2*pi/3) -sin(th+2*pi/3);
             0.5    0.5          0.5];

   Idq0(k,:) = (T*Iabc(start+k,:)')';
   Vdq0(k,:) = (T*Vwye(start+k,:)')';

end
```

B.2.2 GNL3.m

```
function [theta] = GNL3(f, theta0, yobs, t, Vmag, we, K, phi_prev)       0

 theta = theta0;

 %# limiting tolerances
 xtol = 1e-4;
 ftol = 1e-4;
 gtol = eps;
 maxiter = 100 * length(theta);

 %# initializing stuff                                                    10
 fvec = feval(f, theta, yobs, t, Vmag, we, K, phi_prev);
 oldnorm = norm(fvec);
 info = 0;
 iter = 0;

 %# Begin loop

 while (info == 0)

   [fjac, fvec] = fdjac23(f, theta, yobs, t, Vmag, we, K, phi_prev);     20

   fjac = [fjac; .1*eye(length(theta))];
```

```
   fvec = [fvec; .1*(theta(:)-theta0(:))];
   iter = iter + 1;

   [u,s,v] = svd(fjac, 0);
   utf = u'*fvec;

   %# eliminate the almost zero singular values
   s = diag(diag(s) .* (1 - (diag(s) < sqrt(eps))));                    30
   dx = diag(s) == 0;
   delta = -v*diag((1-dx)./(diag(s)+dx))*utf;
   jcnorms = sqrt(diag(v*s*s*v')); % column norms.

   theta = theta + delta;

   %# compute norm of the scaled gradient
   gnorm = 0;
   if oldnorm != 0
     dx = delta / oldnorm;                                              40
     gnorm = max(abs(dx.*(jcnorms != 0)./(jcnorms+(jcnorms==0))));
   end

   %# is gradient norm less than gtol?
   if (gnorm < gtol)
      info = 1;
      sprintf('gradient norm = %.3e, less than gtol = %.3e', gnorm, gtol);
      break
   end
                                                                        50
   %# Too many iterations?
   if (iter >= maxiter)
      info = 1;
      sprintf('number of function evaluations exceeds %d', maxiter);
      break
   end

   %# is step in parameter space less than xtol*norm(theta)?
   if (norm(delta) <= xtol*norm(theta))
      info = 1;                                                         60
      sprintf('maximum relative step less than %.3e', xtol);
      break
   end

 end

endfunction
```

B.2.3 findk3.m

```
function len = findk3(yobs, yhat, ltol)                                 0

 N = size(yhat,1);

 if (N < 4)
   len = N;
   return
```

```
  end

  kvec = zeros(1,size(yhat,2));
                                                                          10
  for i = 1:size(yhat,2)

     for K = 4:N
        t = linspace(0,1,K)';
        A = [ones(K,1) t t.*t];

        err = yobs(1:K,i)-yhat(1:K,i);
        mu = A \ err;

        if (abs(mu(2)) > sqrt(eps) && abs(mu(2))*ltol <= abs(mu(3)))      20
           K = min([K N]);
           break
        end
     end

     kvec(i) = K;

  end

  len = min(kvec);                                                        30

endfunction
```

B.2.4 fdjac23.m

```
function [fjac, fvec] = fdjac23(f, theta, yobs, t, Vmag, we, K, phi_prev)   0

  %# forward difference approximation
  small = sqrt(eps);
  fvec = feval(f, theta, yobs, t, Vmag, we, K, phi_prev);
  fjac = zeros(size(fvec,1), size(theta,1));
  maxstep = 0.1;

  for i = 1:size(fjac,2)
    smu = theta(i);
    h = sign(theta(i))*min([max(abs([small small*smu])) maxstep]);        10
    theta(i) = theta(i) + h;
    fjac(:,i) = (feval(f, theta, yobs, t, Vmag, we, K, phi_prev) - fvec) / h;
    theta(i) = smu;
  end

endfunction
```

B.2.5 cosfunc3a.m

```
function err = cosfunc3a(mu, yobs, t, N)                                  0

   yhat = zeros(N,size(yobs,2));

   yhat(:,1) = mu(1)*cos(mu(2)*t(1:N) + mu(3));
```

```
   yhat(:,2) = mu(1)*cos(mu(2)*t(1:N) - 2*pi/3 + mu(3));
   yhat(:,3) = mu(1)*cos(mu(2)*t(1:N) + 2*pi/3 + mu(3));

   err = yhat-yobs(1:N,:);
   err = err(:);

endfunction
```

B.2.6 cosfunc3b.m

```
function err = cosfunc3b(mu, yobs, t, Vmag, omega_e, N, phi_prev)

   phi = phi_prev + mu;
   yhat = zeros(N,size(yobs,2));

   yhat(:,1) = Vmag*cos(omega_e*t(1:N) + phi);
   yhat(:,2) = Vmag*cos(omega_e*t(1:N) - 2*pi/3 + phi);
   yhat(:,3) = Vmag*cos(omega_e*t(1:N) + 2*pi/3 + phi);

   err = yhat-yobs(1:N,:);
   err = err(:);

endfunction
```

B.3 C DQ0 Preprocessor

B.3.1 dq0prep.c

```
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <values.h>
#include <math.h>
#include <err.h>

#include <gsl/gsl_vector.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_multifit_nlin.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_spline.h>

// DQ0 version of the preprocessor.

// taken from rtu-diag/preprocessor/v2008-04-30/
// use: ./dq0prep datafile outfile
// to make: gcc -g -Wall -lgsl -lgslcblas -lm -o dq0prep dq0prep.c

/* Define prototypes. */
```

```
void cosfunc (gsl_vector *theta, void *est, gsl_matrix *yhat);
void lossfcn1 (gsl_vector *mu, void *est, gsl_vector *err);
void lossfcn2 (gsl_vector *mu, void *est, gsl_vector *err);
void mu_sim (gsl_vector *mu, void *est, gsl_matrix *yout);
long int findk3 (gsl_matrix *yhat, gsl_matrix *yobs, void *est, unsigned long
    int Kin);
void fdjac(gsl_vector *theta, void *data, gsl_vector *fvec, gsl_matrix *fjac);
void data_interp (double **read_array, gsl_matrix *Iabc, gsl_matrix *Vdelta,
    gsl_vector *t);
double ** data_read (char *filename, unsigned long int *N);
gsl_matrix * make_deltatowye (void);
void wyesolve (gsl_matrix *Vdelta, gsl_matrix *Vwye);
void makepark (double th, gsl_matrix *T);
int nlest_f (const gsl_vector *mu, void *est, gsl_vector *fvec);
int nlest_df (const gsl_vector *mu, void *est, gsl_matrix *fjac);
int nlest_fdf (const gsl_vector *mu, void *est, gsl_vector *f, gsl_matrix *J);
gsl_vector * nlt_estimate (gsl_vector *mu0, void *est);
gsl_vector * phase_interp (gsl_matrix *phase_corr_matrix, gsl_vector *tobs,
    unsigned long int Nobs);
gsl_matrix * make_submatrix (gsl_matrix *in, long int start_row, long int
    start_col, long int length_row, long int length_col);
double get_initial_phase (gsl_vector *x);

/* Define structures. */

struct est {
  void *lossfcn;
  void *cosfunc;
  gsl_matrix *yobs;
  gsl_vector *t;
  gsl_vector *mu;
  double mag;
  double omega_e;
  double phi_prev;
  double ltol;
  unsigned long int Nsim;
  double regp;
};

/* Define functions. */

void cosfunc (gsl_vector *theta, void *est, gsl_matrix *yhat) {

  long int i;
  double ti;

  gsl_vector *t = ((struct est *) est)->t;
  long int N = ((struct est *) est)->Nsim;

  double mag = gsl_vector_get(theta, 0);
  double we = gsl_vector_get(theta, 1);
  double phi0 = gsl_vector_get(theta, 2);
  double phi_prev = gsl_vector_get(theta, 3);
  double phi = phi0 + phi_prev;
```

```
  for (i = 0; i < N; i++) {
    ti = gsl_vector_get(t, i);
    gsl_matrix_set(yhat, i, 0, mag*cos(we*ti + phi));
    gsl_matrix_set(yhat, i, 1, mag*cos(we*ti - 2*M_PI/3 + phi));
    gsl_matrix_set(yhat, i, 2, mag*cos(we*ti + 2*M_PI/3 + phi));
  }

}


void lossfcn1 (gsl_vector *mu, void *est, gsl_vector *err) {

  unsigned long int i;

  void (*cosfunc) (gsl_vector *mu, void *est, gsl_matrix *yhat) = ((struct est
      *) est)->cosfunc;
  long int N = ((struct est *) est)->Nsim;
  gsl_matrix *yobs = ((struct est *) est)->yobs;

  gsl_vector *theta = gsl_vector_calloc(4);
  gsl_vector_set(theta, 0, gsl_vector_get(mu, 0));
  gsl_vector_set(theta, 1, gsl_vector_get(mu, 1));
  gsl_vector_set(theta, 2, gsl_vector_get(mu, 2));
  gsl_vector_set(theta, 3, 0.0);

  gsl_matrix *yhat = gsl_matrix_calloc(N, yobs->size2);
  gsl_matrix_view yobs_temp = gsl_matrix_submatrix(yobs, 0, 0, N, yobs->size2);
  cosfunc(theta, est, yhat);
  gsl_matrix_sub(yhat, &yobs_temp.matrix);

  for (i = 0; i < N; i++) {
    gsl_vector_set(err, i, gsl_matrix_get(yhat, i, 0));
    gsl_vector_set(err, N+i, gsl_matrix_get(yhat, i, 1));
    gsl_vector_set(err, 2*N+i, gsl_matrix_get(yhat, i, 2));
  }

  gsl_matrix_free(yhat);
  gsl_vector_free(theta);

}

void lossfcn2 (gsl_vector *mu, void *est, gsl_vector *err) {

  unsigned long int i;

  void (*cosfunc) (gsl_vector *theta, void *est, gsl_matrix *yhat) = ((struct
      est *) est)->cosfunc;
  gsl_matrix *yobs = ((struct est *) est)->yobs;
  long int N = ((struct est *) est)->Nsim;

  gsl_vector *theta = gsl_vector_calloc(4);
  gsl_vector_set(theta, 0, ((struct est *) est)->mag);
  gsl_vector_set(theta, 1, ((struct est *) est)->omega_e);
  gsl_vector_set(theta, 2, gsl_vector_get(mu, 0));
  gsl_vector_set(theta, 3, ((struct est *) est)->phi_prev);
```

```
  gsl_matrix *yhat = gsl_matrix_calloc(N, yobs->size2);
  gsl_matrix_view yobs_temp = gsl_matrix_submatrix(yobs, 0, 0, N, yobs->size2);
  cosfunc(theta, est, yhat);
  gsl_matrix_sub(yhat, &yobs_temp.matrix);

  for (i = 0; i < N; i++) {
    gsl_vector_set(err, i, gsl_matrix_get(yhat, i, 0));
    gsl_vector_set(err, N+i, gsl_matrix_get(yhat, i, 1));
    gsl_vector_set(err, 2*N+i, gsl_matrix_get(yhat, i, 2));
  }                                                                    140

  gsl_matrix_free(yhat);

}


void mu_sim (gsl_vector *mu, void *est, gsl_matrix *yout) {

  unsigned long int i;
                                                                       150
  void (*lossfcn) (gsl_vector *theta, void *est, gsl_vector *y_out) = ((struct
      est *) est)->lossfcn;
  gsl_vector *sim_out = gsl_vector_calloc(yout->size1*yout->size2);

  lossfcn(mu, est, sim_out);

  for (i = 0; i < yout->size1; i++) {
    gsl_matrix_set(yout, i, 0, gsl_vector_get(sim_out, i));
    gsl_matrix_set(yout, i, 1, gsl_vector_get(sim_out, (yout->size1)+i));
    gsl_matrix_set(yout, i, 2, gsl_vector_get(sim_out, 2*(yout->size1)+i));
  }                                                                    160

  gsl_vector_free(sim_out);

}


long int findk3 (gsl_matrix *yhat, gsl_matrix *yobs, void *est, unsigned long
    int Kin) {

  long int N = yobs->size1;
  long int len = N-Kin;                                                170
  double ltol = ((struct est *) est)->ltol;

  long int i, K, col;     // initialize vectors
  int tau_len;
  double fn1, fn2, t_K;

  gsl_vector *Ksave = gsl_vector_calloc(yobs->size2);

  if (len < 4) {        // need at least PARAMS samples for lmsder to work.
    K = len;                                                           180
    return (GSL_MIN(N, K+Kin));
  }

  for (col = 0; col < yobs->size2; col++) {
```

```
for (K = 4; K < len; K++) {

 gsl_vector *err = gsl_vector_calloc(K); // initialize the matrices and
     vectors
 gsl_vector *res = gsl_vector_calloc(K); // the residual from the QR decomp
 gsl_matrix *A = gsl_matrix_calloc(K, 3); // the linearity fitting matrix

 tau_len = GSL_MIN(A->size1, A->size2); // this is needed for the QR decomp
 gsl_vector *tau = gsl_vector_calloc(tau_len);
 gsl_vector *mu = gsl_vector_calloc(3); // the vector of fit parameters

 // filling up the linearity matrix
 for (i = 0; i < K; i++) {
  t_K = ((double) i)/((double) (K-1)); // the terms to fill the matrix

  gsl_matrix_set(A, i, 0, 1.0);
  gsl_matrix_set(A, i, 1, t_K);
  gsl_matrix_set(A, i, 2, t_K*t_K);
 }

 // snag the appropriate components from the simulated and observed data
     vectors

 gsl_vector_view yobs_K = gsl_matrix_column(yobs, col);
 gsl_vector_view yobs_Ksub = gsl_vector_subvector(&yobs_K.vector, Kin, K);

 gsl_vector_view yhat_K = gsl_matrix_column(yhat, col);
 gsl_vector_view yhat_Ksub = gsl_vector_subvector(&yhat_K.vector, Kin, K);

 gsl_vector_memcpy(err, &yhat_Ksub.vector);
 gsl_vector_sub(err, &yobs_Ksub.vector); // compute the error between the
     sim and obs

 gsl_linalg_QR_decomp(A, tau);     // generate the decomposition for the
     lssolve next
 gsl_linalg_QR_lssolve(A, tau, err, mu, res); // do the linearity fit

 fn1 = fabsl(gsl_vector_get(mu, 1)); // helper terms to evaluate quality of
     fit
 fn2 = fabsl(gsl_vector_get(mu, 2));

 if ( (fn1 > sqrt(DBL_EPSILON)) && (fn1*ltol <= fn2)) { // fit quality test

   K = GSL_MIN(K, len);

   gsl_vector_free(err); // cleanup
   gsl_vector_free(res);
   gsl_matrix_free(A);
   gsl_vector_free(tau);
   gsl_vector_free(mu);

   break;
 }

 gsl_vector_free(err); // cleanup
```

```
    gsl_vector_free(res);
    gsl_matrix_free(A);
    gsl_vector_free(tau);
    gsl_vector_free(mu);
                                                                        240
   }

   gsl_vector_set(Ksave, col, K);

  }

  return (GSL_MIN(N, gsl_vector_min(Ksave)+Kin));

  gsl_vector_free(Ksave);
                                                                        250
}


void fdjac(gsl_vector *theta, void *data, gsl_vector *fvec, gsl_matrix *fjac) {

  long int i;                        // initialize variables
  double theta_sign;
  double h;
  double small = sqrtl(DBL_EPSILON); // the smallest step to take
  double smu;                        // to save the theta in              260
  double maxstep = 0.1;              // the biggest step to take

  void (*lossfcn) (gsl_vector *theta, void *data, gsl_vector *y_out) = ((struct
      est *) data)->lossfcn;

  gsl_vector *fvec_temp = gsl_vector_calloc(fvec->size);

  gsl_vector *step_vec_a = gsl_vector_calloc(2); // helping vectors for
      determining step size
  gsl_vector *step_vec_b = gsl_vector_calloc(2);

  gsl_vector_set(step_vec_a, 0, small); // setting some parameters of help   270
      vectors
  gsl_vector_set(step_vec_b, 1, maxstep);

  gsl_vector_set_zero(fvec);          // initialize the residual
  gsl_matrix_set_zero(fjac);          // initialize the jacobian

  lossfcn(theta, data, fvec);        // record the evaluation at this theta

  for (i = 0; i < theta->size; i++) {

   smu = gsl_vector_get(theta, i);    // save the theta before perturbing it  280

   theta_sign = GSL_SIGN(gsl_vector_get(theta, i));
   gsl_vector_set(step_vec_a, 1, fabs(small*gsl_vector_get(theta, i)));
   gsl_vector_set(step_vec_b, 0, gsl_vector_max(step_vec_a));
   h = theta_sign*gsl_vector_min(step_vec_b); // calculate the size of the test
       step
```

```
    gsl_vector_set(theta, i, gsl_vector_get(theta, i) + h); // temporarily
        perturb theta

    lossfcn(theta, data, fvec_temp);  // calculate output with temp theta
                                                                          290
    gsl_vector_sub(fvec_temp, fvec);    // return the result in fvec_temp
    gsl_vector_scale(fvec_temp, (1.0/h)); // return the result in fvec_temp again
    gsl_matrix_set_col(fjac, i, fvec_temp); // save result in col of jacobian

    gsl_vector_set(theta, i, smu);      // restore value of theta
  }

  gsl_vector_free(fvec_temp);           // clean up
  gsl_vector_free(step_vec_a);
  gsl_vector_free(step_vec_b);                                            300

}


void data_interp (double **read_array, gsl_matrix *Iabc, gsl_matrix *Vdelta,
    gsl_vector *t) {

  unsigned long int i;
  unsigned long int N = t->size;
  double Ts = (gsl_vector_get(t, 1) - gsl_vector_get(t, 0));
                                                                          310
  double *Vbc = malloc(N*sizeof(double));
  double *Vca = malloc(N*sizeof(double));
  double *Ia = malloc(N*sizeof(double));
  double *Ib = malloc(N*sizeof(double));
  double *Ic = malloc(N*sizeof(double));

  double *t_vbc = malloc(N*sizeof(double));
  double *t_vca = malloc(N*sizeof(double));
  double *t_ia = malloc(N*sizeof(double));
  double *t_ib = malloc(N*sizeof(double));                                320
  double *t_ic = malloc(N*sizeof(double));

  for (i = 0; i < N; i++) {
    gsl_matrix_set(Vdelta, i, 0, read_array[i][0]); // we are going to
        synchronize to this channel.

    Ia[i] = read_array[i][1];
    Vbc[i] = read_array[i][2];
    Ib[i] = read_array[i][3];
    Vca[i] = read_array[i][4];
    Ic[i] = read_array[i][5];                                             330

    t_ia[i] = Ts*(((double) i) + (1.0/6.0));
    t_vbc[i] = Ts*(((double) i) + (1.0/3.0));
    t_ib[i] = Ts*(((double) i) + (1.0/2.0));
    t_vca[i] = Ts*(((double) i) + (2.0/3.0));
    t_ic[i] = Ts*(((double) i) + (5.0/6.0));
  }

  /* Set up interpolation structures. */
```

```
gsl_interp_accel *Vbc_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Vca_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Ia_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Ib_acc = gsl_interp_accel_alloc();
gsl_interp_accel *Ic_acc = gsl_interp_accel_alloc();

gsl_spline *Vbc_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline *Vca_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline *Ia_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline *Ib_spline = gsl_spline_alloc(gsl_interp_cspline, N);
gsl_spline *Ic_spline = gsl_spline_alloc(gsl_interp_cspline, N);

gsl_spline_init(Vbc_spline, t_vbc, Vbc, N);
gsl_spline_init(Vca_spline, t_vca, Vca, N);
gsl_spline_init(Ia_spline, t_ia, Ia, N);
gsl_spline_init(Ib_spline, t_ib, Ib, N);
gsl_spline_init(Ic_spline, t_ic, Ic, N);

/* Do the interpolation. */

for (i = 0; i < N; i++) {
  gsl_matrix_set(Iabc, i, 0, gsl_spline_eval(Ia_spline, gsl_vector_get(t, i),
      Ia_acc));
  gsl_matrix_set(Iabc, i, 1, gsl_spline_eval(Ib_spline, gsl_vector_get(t, i),
      Ib_acc));
  gsl_matrix_set(Iabc, i, 2, gsl_spline_eval(Ic_spline, gsl_vector_get(t, i),
      Ic_acc));

  gsl_matrix_set(Vdelta, i, 1, gsl_spline_eval(Vbc_spline, gsl_vector_get(t, i)
      , Vbc_acc));
  gsl_matrix_set(Vdelta, i, 2, gsl_spline_eval(Vca_spline, gsl_vector_get(t, i)
      , Vca_acc));
}

gsl_spline_free(Vbc_spline);
gsl_spline_free(Vca_spline);
gsl_spline_free(Ia_spline);
gsl_spline_free(Ib_spline);
gsl_spline_free(Ic_spline);

gsl_interp_accel_free(Vbc_acc);
gsl_interp_accel_free(Vca_acc);
gsl_interp_accel_free(Ia_acc);
gsl_interp_accel_free(Ib_acc);
gsl_interp_accel_free(Ic_acc);

free(Vbc);
free(Vca);
free(Ia);
free(Ib);
free(Ic);

free(t_vbc);
free(t_vca);
free(t_ia);
```

```
  free(t_ib);
  free(t_ic);

}

gsl_matrix * make_deltatowye (void) {

  gsl_matrix *A = gsl_matrix_calloc(4,3);
                                                                    400
  gsl_matrix_set(A, 0, 0, 1);
  gsl_matrix_set(A, 0, 1, -1);

  gsl_matrix_set(A, 1, 1, 1);
  gsl_matrix_set(A, 1, 2, -1);

  gsl_matrix_set(A, 2, 0, -1);
  gsl_matrix_set(A, 2, 2, 1);

  gsl_matrix_set(A, 3, 0, 1);                                       410
  gsl_matrix_set(A, 3, 1, 1);
  gsl_matrix_set(A, 3, 2, 1);

  return (A);

}


double ** data_read (char *filename, unsigned long int *N) {
                                                                    420
  unsigned long int i, k;

  int block_length = 100000;
  int vars = 6;

  FILE *MOTOR_IN = fopen(filename, "r"); // The input file

  if (MOTOR_IN == NULL)
    err(1, "can't open file");
                                                                    430
  double **read_array = malloc(block_length*sizeof(double *));
  if (read_array == NULL)
    printf("out of memory, line %d\n", __LINE__);

  for (i = 0; i < block_length; i++) {
    read_array[i] = malloc(vars*sizeof(double));
    if (read_array[i] == NULL)
      printf("out of memory, line %d\n", __LINE__);
  }
                                                                    440
  i = 0; // initialize the line counter.

  while (!feof(MOTOR_IN)) {

    if ((i > 0) && (i % block_length) == 0) {
```

```
    read_array = realloc(read_array, (i+block_length)*sizeof(double *));

    for (k = i; k < (i+block_length); k++) {
     read_array[k] = malloc(vars*sizeof(double));
     if (read_array[i] == NULL)
      printf("out of memory; line: %d\n", __LINE__);
    }

   }

   fscanf(MOTOR_IN, "%lf %lf %lf %lf %lf %lf",
        &read_array[i][0],
        &read_array[i][1],
        &read_array[i][2],
        &read_array[i][3],
        &read_array[i][4],
        &read_array[i][5]);

   i++;

  }

  *N = (i-1);
  return (read_array);

}


void wyesolve (gsl_matrix *Vdelta, gsl_matrix *Vwye) {

  unsigned long int i, k;
  unsigned long int N = Vdelta->size1;

  gsl_matrix *delta_to_wye = make_deltatowye();
  gsl_vector *tau = gsl_vector_calloc(delta_to_wye->size2);
  gsl_vector *res = gsl_vector_calloc(delta_to_wye->size1); // the residual from
      the QR decomp
  gsl_vector *b = gsl_vector_calloc(delta_to_wye->size1);
  gsl_linalg_QR_decomp(delta_to_wye, tau);

  for (i = 0; i < N; i++) {

   for (k = 0; k < 3; k++)
    gsl_vector_set(b, k, gsl_matrix_get(Vdelta, i, k));

   gsl_vector_view Vwye_k = gsl_matrix_row(Vwye, i);
   gsl_linalg_QR_lssolve(delta_to_wye, tau, b, &Vwye_k.vector, res);

  }

  gsl_vector_free(tau);
  gsl_vector_free(b);
  gsl_matrix_free(delta_to_wye);
  gsl_vector_free(res);

}
```

```
void makepark (double th, gsl_matrix *T) {

  gsl_matrix_set(T, 0, 0, cos(th));
  gsl_matrix_set(T, 0, 1, cos(th - 2*M_PI/3));
  gsl_matrix_set(T, 0, 2, cos(th + 2*M_PI/3));

  gsl_matrix_set(T, 1, 0, -sin(th));                                    510
  gsl_matrix_set(T, 1, 1, -sin(th - 2*M_PI/3));
  gsl_matrix_set(T, 1, 2, -sin(th + 2*M_PI/3));

  gsl_matrix_set(T, 2, 0, 0.5);
  gsl_matrix_set(T, 2, 1, 0.5);
  gsl_matrix_set(T, 2, 2, 0.5);

  gsl_matrix_scale(T, (2.0/3.0));

}                                                                       520


int nlest_f (const gsl_vector *mu, void *est, gsl_vector *fvec) {

  unsigned long int i;

  void (*lossfcn) (gsl_vector *theta, void *est, gsl_vector *y_out) = ((struct
      est *) est)->lossfcn;
  gsl_matrix *yobs = ((struct est *) est)->yobs;
  long int N = ((struct est *) est)->Nsim;
  double regp = ((struct est *) est)->regp;                             530

  gsl_vector *yout = gsl_vector_calloc(N*yobs->size2);
  gsl_vector *theta_1 = gsl_vector_calloc(mu->size);

  gsl_vector *theta0 = ((struct est *) est)->mu;
  for (i = 0; i < mu->size; i++)
    gsl_vector_set(theta_1, i, gsl_vector_get(mu, i));

  gsl_vector_sub(theta_1, theta0);
  gsl_vector_div(theta_1, theta0);                                      540
  gsl_vector_scale(theta_1, regp);

  lossfcn(mu, est, yout);

  for (i = 0; i < N*yobs->size2; i++)
    gsl_vector_set(fvec, i, gsl_vector_get(yout, i));

  for (i = 0; i < mu->size; i++)
    gsl_vector_set(fvec, N*yobs->size2+i, gsl_vector_get(theta_1, i));
                                                                        550
  gsl_vector_free(yout);
  gsl_vector_free(theta_1);

  return GSL_SUCCESS;

}
```

```
int nlest_df (const gsl_vector *mu, void *est, gsl_matrix *fjac) {
 unsigned long int i;

 gsl_matrix *yobs = ((struct est *) est)->yobs;
 long int N = ((struct est *) est)->Nsim;
 double regp = ((struct est *) est)->regp;

 gsl_vector *fvec = gsl_vector_calloc(N*yobs->size2);
 gsl_matrix *jac = gsl_matrix_calloc(N*yobs->size2, mu->size);
 gsl_vector *jac_vec = gsl_vector_calloc(mu->size);
 gsl_matrix *regp_matrix = gsl_matrix_calloc(mu->size, mu->size);

 for (i = 0; i < mu->size; i++)
  gsl_matrix_set(regp_matrix, i, i, regp);

 fdjac(mu, est, fvec, jac);

 for (i = 0; i < N*yobs->size2; i++) {
  gsl_matrix_get_row(jac_vec, jac, i);
  gsl_matrix_set_row(fjac, i, jac_vec);
 }

 for (i = 0; i < mu->size; i++) {
  gsl_matrix_get_row(jac_vec, regp_matrix, i);
  gsl_matrix_set_row(fjac, N*yobs->size2+i, jac_vec);
 }

 gsl_vector_free(fvec);
 gsl_matrix_free(jac);
 gsl_vector_free(jac_vec);
 gsl_matrix_free(regp_matrix);

 return GSL_SUCCESS;

}


int nlest_fdf (const gsl_vector *mu, void *est, gsl_vector *f, gsl_matrix *J) {

 nlest_f(mu, est, f);
 nlest_df(mu, est, J);

 return GSL_SUCCESS;

}


gsl_vector * nlt_estimate (gsl_vector *mu0, void *est) {

 unsigned long int i, K = 0;

 unsigned long int iter = 0; // parameters for nlsq solver
 long int status;
```

```
unsigned long int maxiter;

void (*lossfcn) (gsl_vector *theta, void *est, gsl_vector *y_out) = ((struct
    est *) est)->lossfcn;
void (*cosfunc) (gsl_vector *theta, void *est, gsl_matrix *yhat) = ((struct
    est *) est)->cosfunc;
gsl_matrix *yobs = ((struct est *) est)->yobs;
gsl_vector *t = ((struct est *) est)->t;
gsl_vector *mu_pre = ((struct est *) est)->mu;
double mag = ((struct est *) est)->mag;                                    620
double omega_e = ((struct est *) est)->omega_e;
double phi_prev = ((struct est *) est)->phi_prev;
double ltol = ((struct est *) est)->ltol;
unsigned long int N = ((struct est *) est)->Nsim;
double regp = ((struct est *) est)->regp;

gsl_matrix *z = gsl_matrix_calloc(yobs->size1, yobs->size2);
gsl_matrix *yhat = gsl_matrix_calloc(N, yobs->size2);

gsl_vector *mu0_hat = gsl_vector_calloc(mu_pre->size);                     630
gsl_vector_memcpy(mu0_hat, mu_pre);
gsl_vector *t_hat = gsl_vector_calloc(t->size);
gsl_vector_memcpy(t_hat, t);

struct est Sim = {lossfcn, cosfunc, z, t_hat, mu0_hat, mag, omega_e, phi_prev,
     ltol, N, regp};
struct est Est = {lossfcn, cosfunc, yobs, t_hat, mu0_hat, mag, omega_e,
    phi_prev, ltol, N, regp};

const gsl_multifit_fdfsolver_type *T;
T = gsl_multifit_fdfsolver_lmsder;
gsl_multifit_fdfsolver *s;                                                 640
gsl_multifit_function_fdf f;
maxiter = 600*mu0->size;

f.f = &nlest_f;
f.df = &nlest_df;
f.fdf = &nlest_fdf;
f.p = mu0->size;
f.params = &Est;

while (K < (N-1)) {                                                        650

  mu_sim(mu0, &Sim, yhat);

  K = findk3(yhat, yobs, &Est, K);

  gsl_matrix *y_K = gsl_matrix_calloc(K, yobs->size2);
  y_K = make_submatrix(yobs, 0, 0, K, yobs->size2);
  gsl_vector *t_K = gsl_vector_calloc(K);
  gsl_vector_view t_view = gsl_vector_subvector(t, 0, K);
  gsl_vector_memcpy(t_K, &t_view.vector);                                  660

  f.n = 3*K+mu0->size;
  Est.Nsim = K;
  Est.yobs = y_K;
```

```
  Est.t = t_K;
  Est.mu = mu0;

  s = gsl_multifit_fdfsolver_alloc (T, 3*K+mu0->size, mu0->size);
  gsl_multifit_fdfsolver_set(s, &f, mu0);
  iter = 0;                                                          670

  do {

    iter++;
    status = gsl_multifit_fdfsolver_iterate (s);

    if (status)
     break;

    status = gsl_multifit_test_delta(s->dx, s->x, 1e-4, 1e-4);       680

  } while (status == GSL_CONTINUE && iter < maxiter);

  for (i = 0; i < mu0->size; i++)
    gsl_vector_set(mu0, i, gsl_vector_get(s->x, i));

  gsl_matrix_free(y_K);
  gsl_vector_free(t_K);
  gsl_multifit_fdfsolver_free(s);
                                                                     690
 }

 gsl_matrix_free(z);
 gsl_matrix_free(yhat);
 gsl_vector_free(mu0_hat);
 gsl_vector_free(t_hat);

 // generate output.

 gsl_vector *mu_out = gsl_vector_calloc(mu0->size);                  700

 for (i = 0; i < mu0->size; i++)
   gsl_vector_set(mu_out, i, gsl_vector_get(mu0, i));

 return(mu_out);

}


gsl_vector * phase_interp (gsl_matrix *phase_corr_matrix, gsl_vector *tobs,   710
    unsigned long int Nobs) {

 unsigned long int i;

 double phi_out;
 double phi_last = gsl_matrix_get(phase_corr_matrix, 0, 1);
 double corr_factor = 0;
 double threshold = M_PI;

 // unwrap the phase.
```

```
                                                                          720
gsl_vector *unwrap_phase = gsl_vector_calloc(phase_corr_matrix->size1);
gsl_vector_set(unwrap_phase, 0, phi_last);

for (i = 1; i < phase_corr_matrix->size1; i++) {

 phi_out = gsl_matrix_get(phase_corr_matrix, i, 1) + corr_factor;

 if (phi_out > phi_last + threshold) {
  corr_factor -= 2*M_PI;
  phi_out -= 2*M_PI;                                                      730
 } else if (phi_out < phi_last - threshold) {
  corr_factor += 2*M_PI;
  phi_out += 2*M_PI;
 }

 gsl_vector_set(unwrap_phase, i, phi_out);
 phi_last = phi_out;

}
                                                                          740
// interpolate the phase.

double t_uw_interp[phase_corr_matrix->size1], p_uw_interp[phase_corr_matrix->
    size1];

for (i = 0; i < phase_corr_matrix->size1; i++) {
 t_uw_interp[i] = gsl_matrix_get(phase_corr_matrix, i, 0);
 p_uw_interp[i] = gsl_vector_get(unwrap_phase, i);
}

gsl_interp_accel *phase_acc = gsl_interp_accel_alloc();                   750
gsl_spline *phase_spline = gsl_spline_alloc(gsl_interp_cspline,
    phase_corr_matrix->size1);
gsl_spline_init(phase_spline, t_uw_interp, p_uw_interp, phase_corr_matrix->
    size1);

gsl_vector *phase_obs = gsl_vector_calloc(Nobs);

for (i = 0; i < Nobs; i++)
 gsl_vector_set(phase_obs, i, gsl_spline_eval(phase_spline, gsl_vector_get(
     tobs, i), phase_acc));

gsl_spline_free(phase_spline);
gsl_interp_accel_free(phase_acc);                                         760


// rewrap the phase for the dq0 transformation.

gsl_vector *wrap_phase = gsl_vector_calloc(phase_obs->size);
corr_factor = 0;

for (i = 0; i < phase_obs->size; i++) {

 phi_out = gsl_vector_get(phase_obs, i) + corr_factor;                    770
```

```
    if (phi_out > 2*M_PI) {
      corr_factor -= 2*M_PI;
      phi_out -= 2*M_PI;
    } else if (phi_out < -2*M_PI) {
      corr_factor += 2*M_PI;
      phi_out += 2*M_PI;
    }

    gsl_vector_set(wrap_phase, i, phi_out);                                    780

  }

  gsl_vector_free(unwrap_phase);
  gsl_vector_free(phase_obs);

  return(wrap_phase);

}                                                                              790

gsl_matrix * make_submatrix (gsl_matrix *in, long int start_row, long int
    start_col, long int length_row, long int length_col) {

  gsl_matrix *out = gsl_matrix_calloc(length_row, length_col);
  gsl_matrix_view mat_view = gsl_matrix_submatrix(in, start_row, start_col,
      length_row, length_col);
  gsl_matrix_memcpy(out, &mat_view.matrix);

  return(out);

}                                                                              800

double get_initial_phase (gsl_vector *x) {

  double phi;

  double x0 = gsl_vector_get(x, 0);
  double x1 = gsl_vector_get(x, 1);
  double m1 = (x0+x1)/2.0;
                                                                               810
  int test1 = (x1 > x0);
  int test2 = ((x1 - x0)/m1 > 0.98);
  int test3 = (GSL_SIGN(x0) == -GSL_SIGN(x1));

  if ((test2) && (!test3)) {
    if (x1 > 0) {
      phi = 7.0*M_PI/4.0;
      return(phi);
    } else if (x1 < 0) {
      phi = M_PI;                                                              820
      return(phi);
    }
  } else if (test3) {
    if (x0 > 0) {
      phi = M_PI/2.0;
```

```
    return(phi);
   } else if (x0 < 0) {
    phi = 3.0*M_PI/2.0;
    return(phi);
   }
  } else {
   if (test1) {
    if (x1 > 0) {
     phi = 7.0*M_PI/4.0;
     return(phi);
    } else if (x1 < 0) {
     phi = 5.0*M_PI/4.0;
     return(phi);
    }
   } else if (!test1) {
    if (x1 > 0) {
     phi = M_PI/4.0;
     return(phi);
    } else if (x1 < 0) {
     phi = 3.0*M_PI/4.0;
     return(phi);
    }
   }
  }

  return(0);

}


int main (int argc, char *argv[]) {

  unsigned long int i, N, Nshort;

  double fs = 192e3/24.0; // sample frequency
  double Ts = 1.0/fs;     // sample period
  double F0 = 60.0;       // utility frequency
  double Nc = fs/F0;      // number of samples per cycle
  double ltol = 0.2;      // fitting error for squared term in findk3
  double regp = 0.1;      // regularization parameter

  // read in variables from file.
  double **read_array = data_read(argv[1], &N);

  // set up process variables.
  gsl_matrix *Iabc = gsl_matrix_calloc(N, 3);
  gsl_matrix *Vdelta = gsl_matrix_calloc(N, 3);
  gsl_matrix *Vwye = gsl_matrix_calloc(N, 3);
  gsl_vector *t = gsl_vector_calloc(N);

  for (i = 0; i < N; i++)    // populate the time vector
   gsl_vector_set(t, i, Ts*i);

  // generate the interpolated variables.
  data_interp(read_array, Iabc, Vdelta, t);
  free(read_array);
```

```
// solve for the wye-referenced voltages.
wyesolve(Vdelta, Vwye);
gsl_matrix_free(Vdelta);

// find the parameters of the first line cycle.
gsl_vector_view Van = gsl_matrix_column(Vwye, 0);
gsl_vector_view Van_1 = gsl_vector_subvector(&Van.vector, 0, ceil(Nc));
double phi0 = get_initial_phase(&Van_1.vector);                          890

if (phi0 == 0)
 errx(1, "error in initial phase estimate, line %d\n", __LINE__);

gsl_matrix *yobs_short = gsl_matrix_calloc(ceil(Nc), Vwye->size2);
yobs_short = make_submatrix(Vwye, 0, 0, ceil(Nc), Vwye->size2);
gsl_vector *t_short = gsl_vector_calloc(yobs_short->size1);
Nshort = yobs_short->size1;

for (i = 0; i < yobs_short->size1; i++)                                  900
 gsl_vector_set(t_short, i, Ts*i);

gsl_vector *mu0_1 = gsl_vector_calloc(3);
gsl_vector_set(mu0_1, 0, 3000);
gsl_vector_set(mu0_1, 1, 2*M_PI*F0);
gsl_vector_set(mu0_1, 2, phi0);

gsl_vector *mu1 = gsl_vector_calloc(mu0_1->size);
struct est Est1 = {&lossfcn1, &cosfunc, yobs_short, t_short, mu0_1, 0, 0, 0,
    ltol, Nshort, regp};
                                                                         910
mu1 = nlt_estimate(mu0_1, &Est1);

if ((GSL_SIGN(gsl_vector_get(mu1, 0)) < 0) || (GSL_SIGN(gsl_vector_get(mu1, 1)
    ) < 0 ))
 errx(1, "Initial parameter estimate incorrect: |V| = %lf, omega_e = %lf\n",
     gsl_vector_get(mu1, 0),
     gsl_vector_get(mu1, 1));

// second step: estimate the variation in the phase over the whole dataset.
double Vmag = gsl_vector_get(mu1, 0);
double omega_e = gsl_vector_get(mu1, 1);                                 920
double phi_prev = 0.0;

gsl_vector *mu0_2 = gsl_vector_calloc(1);
gsl_vector_set(mu0_2, 0, 0.1);

gsl_vector *mu2 = gsl_vector_calloc(mu0_2->size);
gsl_matrix *yobs_iter = gsl_matrix_calloc(ceil(Nc), Vwye->size2);
gsl_vector *t_iter = gsl_vector_calloc(ceil(Nc));
gsl_matrix *phase_corr_matrix = gsl_matrix_calloc(floor(N/ceil(Nc)), 2);
long int phase_ctr = 0;                                                  930

for (i = 0; i < N; i += ceil(Nc)) {

 if (i+ceil(Nc) > N) {
  break;
```

```
  }

  gsl_matrix_view yobs_iter_view = gsl_matrix_submatrix(Vwye, i, 0, ceil(Nc),
      Vwye->size2);
  gsl_matrix_memcpy(yobs_iter, &yobs_iter_view.matrix);
  gsl_vector_view t_iter_view = gsl_vector_subvector(t, i, ceil(Nc));
  gsl_vector_memcpy(t_iter, &t_iter_view.vector);

  struct est Est2 = {&lossfcn2, &cosfunc, yobs_iter, t_iter, mu0_2, Vmag,
      omega_e, phi_prev, ltol, ceil(Nc), regp};

  mu2 = nlt_estimate(mu0_2, &Est2);

  phi_prev += gsl_vector_get(mu2, 0);

  if (phi_prev > 2*M_PI) {
    phi_prev = phi_prev - 2*M_PI;
  } else if (phi_prev < -2*M_PI) {
    phi_prev = phi_prev + 2*M_PI;
  }

  gsl_matrix_set(phase_corr_matrix, phase_ctr, 0, Ts*i);
  gsl_matrix_set(phase_corr_matrix, phase_ctr, 1, phi_prev);

  gsl_vector_set(mu0_2, 0, gsl_vector_get(mu2, 0));

  ++phase_ctr;

}

// unwrap the phase so that we can interpolate.
gsl_vector *phase_obs = gsl_vector_calloc(N);
phase_obs = phase_interp(phase_corr_matrix, t, N);

// compute the dq0 transformation of the currents.
double Vsc = sqrt(2.0)*120.0/Vmag;
double Isc = 1.282020756512273e-02;
double th;

gsl_matrix *Vdq0 = gsl_matrix_calloc(Vwye->size1, Vwye->size2);
gsl_matrix *Idq0 = gsl_matrix_calloc(Iabc->size1, Iabc->size2);
gsl_matrix *Tdq = gsl_matrix_calloc(3, 3);
gsl_vector *work = gsl_vector_calloc(3);

FILE *OUTFILE = fopen(argv[2], "w"); // The input file

if (OUTFILE == NULL)
  err(1, "can't open file");

for (i = 0; i < Vdq0->size1; i++) {

  th = omega_e*gsl_vector_get(t, i) + gsl_vector_get(phase_obs, i);
  makepark(th, Tdq);

  gsl_vector_view I_lab_row = gsl_matrix_row(Iabc, i);
  gsl_vector_view I_dq0_row = gsl_matrix_row(Idq0, i);
```

```
  gsl_vector_view V_lab_row = gsl_matrix_row(Vwye, i);
  gsl_vector_view V_dq0_row = gsl_matrix_row(Vdq0, i);

  gsl_blas_dgemv(CblasNoTrans, Isc, Tdq, &I_lab_row.vector, 0.0, work);
  gsl_vector_memcpy(&I_dq0_row.vector, work);
  gsl_vector_set_zero(work);

  gsl_blas_dgemv(CblasNoTrans, Vsc, Tdq, &V_lab_row.vector, 0.0, work);
  gsl_vector_memcpy(&V_dq0_row.vector, work);
  gsl_vector_set_zero(work);

  fprintf(OUTFILE, "%lf %lf %lf %lf %lf %lf\n",
        gsl_matrix_get(Vdq0, i, 0),
        gsl_matrix_get(Vdq0, i, 1),
        gsl_matrix_get(Vdq0, i, 2),
        gsl_matrix_get(Idq0, i, 0),
        gsl_matrix_get(Idq0, i, 1),
        gsl_matrix_get(Idq0, i, 2));

 }

 fclose(OUTFILE);

 // cleanup.
 gsl_vector_free(t);
 gsl_matrix_free(Iabc);
 gsl_matrix_free(Vwye);

 gsl_vector_free(mu0_1);
 gsl_vector_free(mu1);
 gsl_matrix_free(yobs_short);
 gsl_vector_free(t_short);

 gsl_vector_free(mu0_2);
 gsl_vector_free(mu2);
 gsl_matrix_free(yobs_iter);
 gsl_vector_free(t_iter);

 gsl_matrix_free(phase_corr_matrix);
 gsl_vector_free(phase_obs);

 gsl_matrix_free(Vdq0);
 gsl_matrix_free(Idq0);
 gsl_matrix_free(Tdq);
 gsl_vector_free(work);

 // finish.
 return(0);

}
```

B.4 Perl Helper Code

B.4.1 process.pl

```
#!/usr/bin/perl -w

# usage: ./process.pl filename*

open(INDEXFILE, ">>indexfile");

for ($argnum = 0; $argnum < @ARGV; $argnum++) {

  $ARGV[$argnum] =~ /([\w\d_]*).(\d\d\d)$/;

  $basename = $1;
  $filecountout = $2;
  open(FILE, "$ARGV[$argnum]");

  $print_len = 9.99e5;
  $thresh = 1e3;
  $lockout_len = 1e5;

  # initialize variables

  $last_xa = 0.0;
  $last_xb = 0.0;
  $last_xc = 0.0;

  $k = 0;
  $k_lockout = 0;
  $k_print = 0;
  $print_flag = 0;
  $print_ctr = 0;

  # execute the main maneuvers.

  while (<FILE>) {

    if ($k < $k_lockout) {
      $k++;
      next;
    }

    @line = split;          # divide the file into its constituent entries.

    $raw_ia = $line[1];     # get the currents out of the file.
    $raw_ib = $line[3];
    $raw_ic = $line[5];

    $raw_vab = $line[0];     # get the voltages out of the file.
    $raw_vbc = $line[2];
    $raw_vca = $line[4];

    if ($k < 10000) {       # prepare to compute the averages
      $mean_ia_vec[$k] = $raw_ia;
      $mean_ib_vec[$k] = $raw_ib;
```

```
  $mean_ic_vec[$k] = $raw_ic;

  $mean_vab_vec[$k] = $raw_vab;
  $mean_vbc_vec[$k] = $raw_vbc;
  $mean_vca_vec[$k] = $raw_vca;

  $k++;
  next;
}

if ($k == 10000){           # compute the averages.
  $mean_ia = average(@mean_ia_vec);
  $mean_ib = average(@mean_ib_vec);
  $mean_ic = average(@mean_ic_vec);

  $mean_vab = average(@mean_vab_vec);
  $mean_vbc = average(@mean_vbc_vec);
  $mean_vca = average(@mean_vca_vec);
}

$ia = sprintf("%5d", $raw_ia - $mean_ia); # subtract off the averages.
$ib = sprintf("%5d", $raw_ib - $mean_ib);
$ic = sprintf("%5d", $raw_ic - $mean_ic);
$vab = sprintf("%5d", $raw_vab - $mean_vab);
$vbc = sprintf("%5d", $raw_vbc - $mean_vbc);
$vca = sprintf("%5d", $raw_vca - $mean_vca);

if (@ia_save < 1000) { # save the last 1000 values.

  push(@ia_save, $ia); # if less than 1000 values are currently saved.
  push(@ib_save, $ib);
  push(@ic_save, $ic);
  push(@vab_save, $vab);
  push(@vbc_save, $vbc);
  push(@vca_save, $vca);

} else {

  push(@ia_save, $ia); # keep the depth of saved values at 1000.
  push(@ib_save, $ib);
  push(@ic_save, $ic);
  push(@vab_save, $vab);
  push(@vbc_save, $vbc);
  push(@vca_save, $vca);

  shift(@ia_save);
  shift(@ib_save);
  shift(@ic_save);
  shift(@vab_save);
  shift(@vbc_save);
  shift(@vca_save);
}

$xa = $ia**2;     # generate the prep waveform.
$xb = $ib**2;
$xc = $ic**2;
```

```
$dxa = $xa - $last_xa; # generate the diffs of the waveform.
$dxb = $xb - $last_xb;                                                    110
$dxc = $xc - $last_xc;

if ($print_flag == 1) {

  if ($print_ctr == $print_len) { # stop printing at the right length
    print INDEXFILE "$k_print\n";
    $print_flag = 0;
    $print_ctr = 0;
    close(DATA_OUT);
                                                                          120
    `./dq0prep tempfile tempfile_proc`;
    rename("tempfile_proc", "$basename"."_proc".".$filecountout");
    unlink("tempfile");

    $k_lockout = $lockout_len + $k;
    $k++;
    next;
  }

  # if it is a spurious transient (e.g. just a power spike), this will     130
  # reject it because the levels go back down to zero.

  if (!((($xa > $thresh) && ($dxa > $thresh)) || (($xb > $thresh) && ($dxb >
      $thresh)) || (($xc > $thresh) && ($dxc > $thresh)))) {
    $print_flag = 0;
    $print_ctr = 0;
    close(DATA_OUT);
    unlink("tempfile");
    $k_lockout = $lockout_len + $k;
    $k++;
    next;                                                                 140
  }

  # print the data to the file.

  print DATA_OUT "$vab $ia $vbc $ib $vca $ic\n";
  $print_ctr++;
  $k++;
  next;

}                                                                         150

# detect the start transient.

if ((($xa > $thresh) && ($dxa > $thresh)) || (($xb > $thresh) && ($dxb >
    $thresh)) || (($xc > $thresh) && ($dxc > $thresh))) {

  if ($k == $k_lockout) {    # in case the compressor's still on.
    $k_lockout = $k + $lockout_len;
    $k++;
    next;
  } else {                   # edge detected.                             160
    $k_print = $k;
```

```
      $print_flag = 1;
      open(DATA_OUT, ">tempfile");

      # print the 1000 saved values (including the present one)
      # before the transient.

      for ($i = 0; $i < @ia_save; $i++) {
        print DATA_OUT "$vab_save[$i] $ia_save[$i] $vbc_save[$i] $ib_save[$i]
            $vca_save[$i] $ic_save[$i]\n";
      }                                                                    170

      $k++;
      next;
     }

   }

   # set up for the next datapoint.

   $last_xa = $xa;                                                         180
   $last_xb = $xb;
   $last_xc = $xc;

   $k++;

  }

  close(FILE);

}                                                                          190

close(INDEXFILE);
rename("indexfile", "$basename.index");


# helper functions.

sub average {

  my(@vector);                                                             200
  my($sum);

  $sum = 0;

  @vector = @_;
  $len = @vector;

  foreach $_ (@vector) {
    $sum += $_;
  }                                                                        210

  return $sum/$len;

}
```

B.4.2 indexsift.pl

```
#!/usr/bin/perl -w

# usage: ./indexsift.pl datafile.index 1000000 datafiles*

# 1000000 lines are typically used for the compressor.

$index_file = $ARGV[0]; # the file of indices.
$leng = $ARGV[1]; # the length of the desired events.
$offset = 999; # the offset from the index, so that
             # there is zero padding at the beginning.

open(INDICES, $index_file);
@index = <INDICES>;
$index_length = $#index;
$num_args = @ARGV - 2;

for ($i = 0; $i < @index; $i++) {
   $index[$i] -= $offset;
}

if ($index_length+1 != $num_args) {
  print "the length of the indexfile does not equal the number of files to
      process.\n";
  exit (0);
}

$ind_num = 0;
$start = $index[$ind_num];
$end = $start+$leng;

for ($i = 2; $i < @ARGV; $i++) {

  open(DATA_IN, $ARGV[$i]);

  $ARGV[$i] =~ /([\w\d_]*).(\d\d\d)$/;
  $basename = $1;
  $filecountout = $2;

  open(DATA_IN, "$ARGV[$i]");
  open(DATA_OUT, ">$basename"."_proc".".$filecountout");

  $k = 0;

  while(<DATA_IN>) {

    if ($k < $start) {
      $k++;
      next;
    }

    $line = $_;

    if ( ($k >= $start) || eof(DATA_IN) ) {
      if ($k < $end) {
```

```
      print DATA_OUT $line;
    }
  }

  if ( ($k == $end) && ($ind_num < $index_length)) {
    $start = $index[++$ind_num];
    $end = $start+$leng;
    close(DATA_IN);
    close(DATA_OUT);
    last;
  }

  $k++;

 }

}

close(INDICES);
```

B.5 Compressor Head Drawings

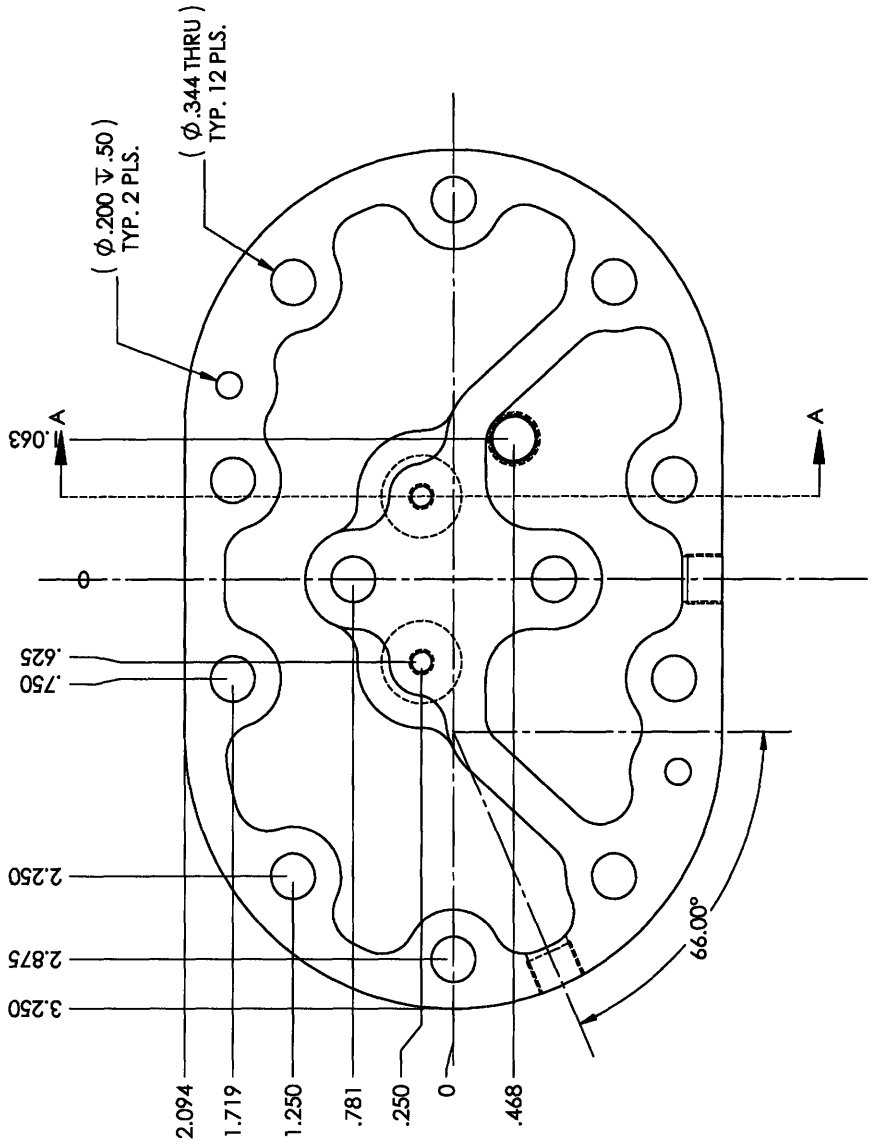


Figure B-1: Bottom view of compressor head.

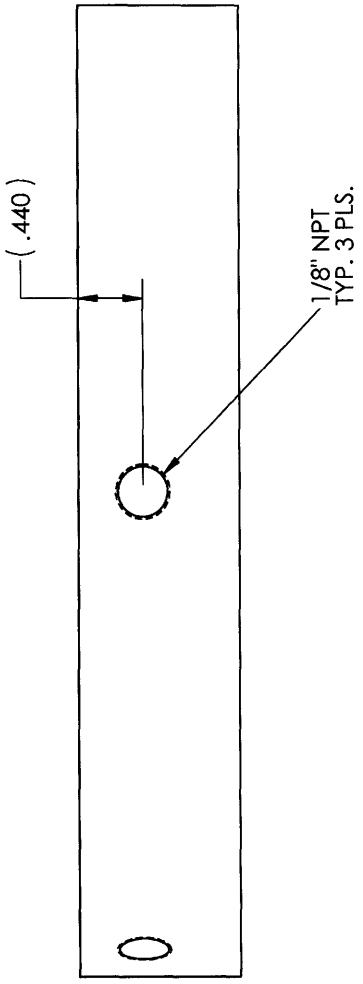


Figure B-2: Left side view of compressor head.

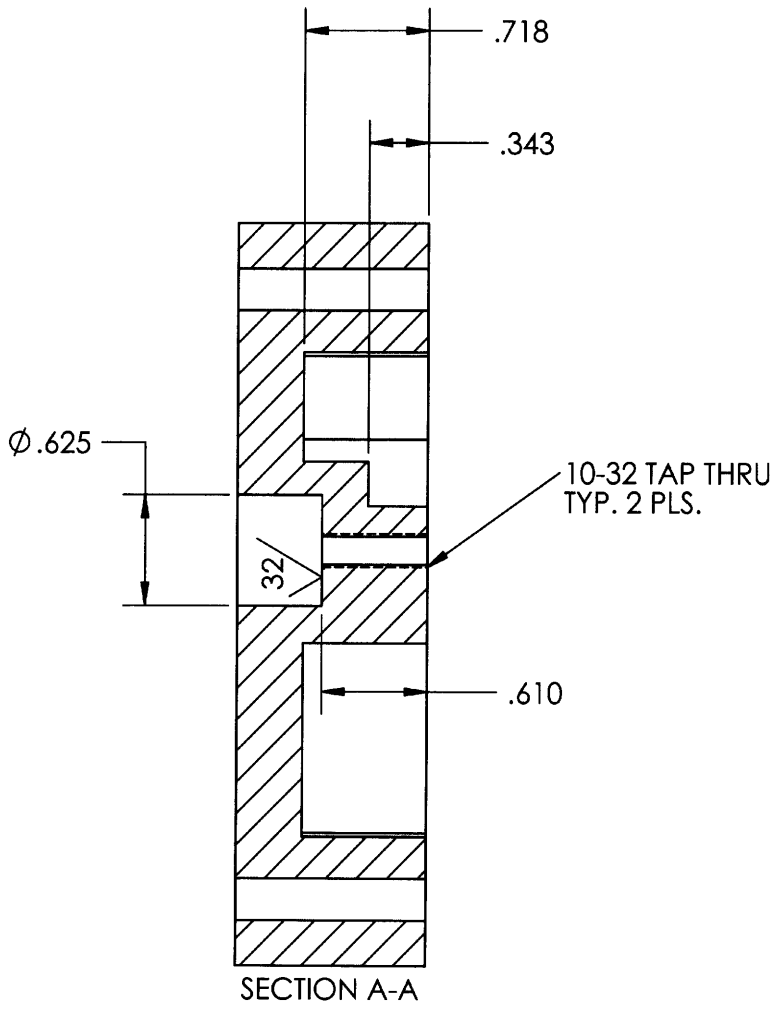


Figure B-3: Cross-section of compressor head.

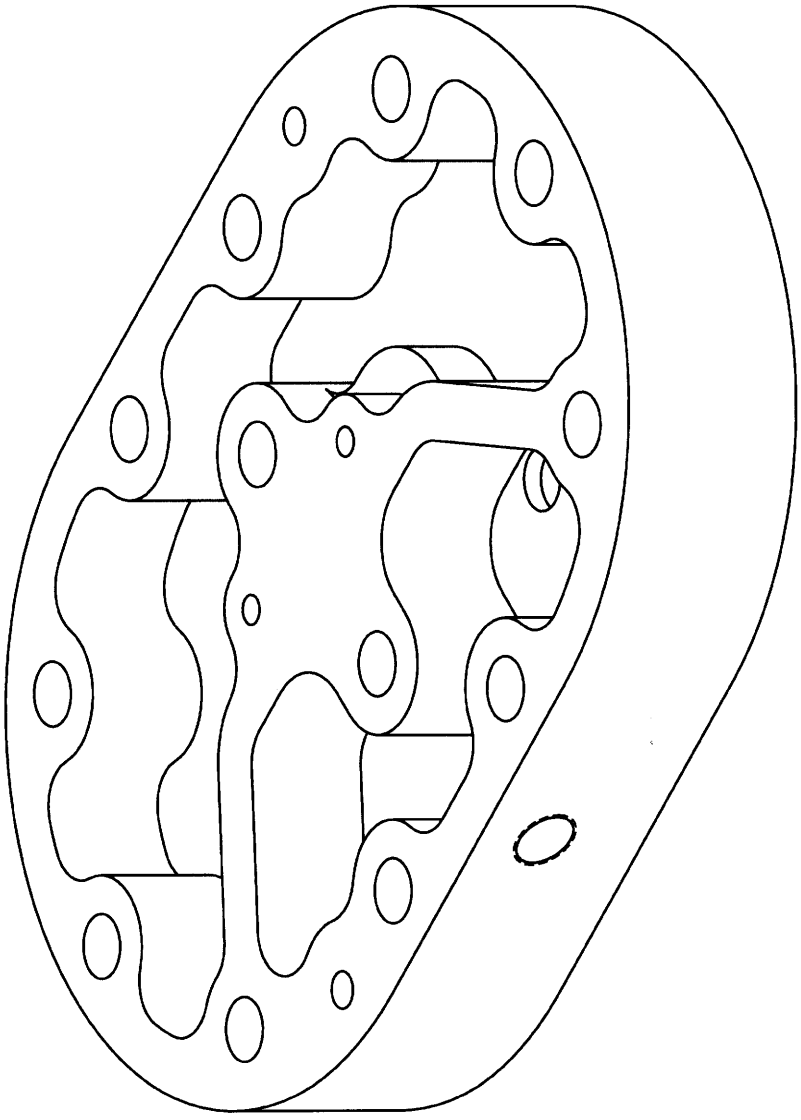


Figure B-4: 3-D drawing of compressor head.

B.6 Electrical Schematics

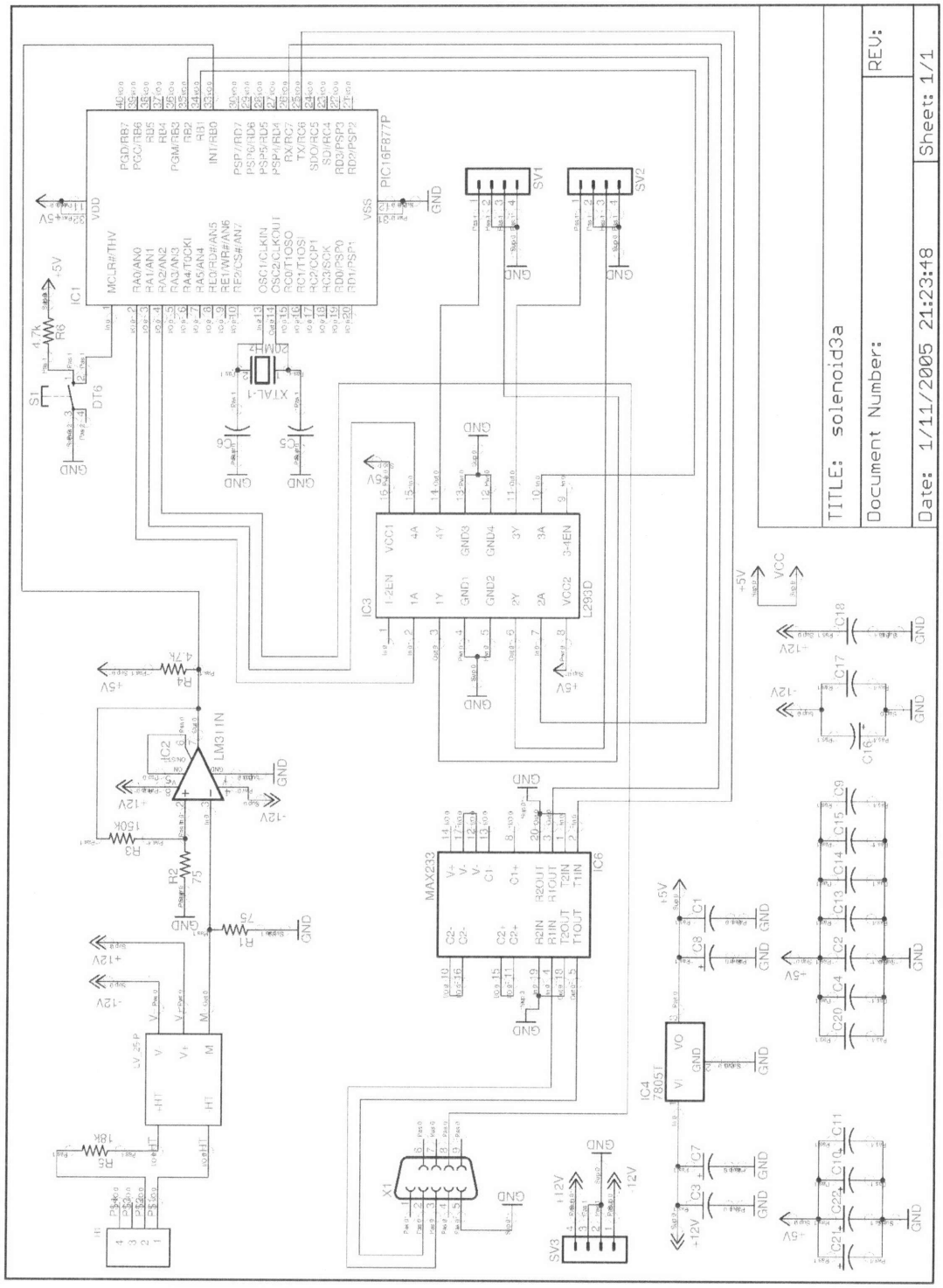


Figure B-5: Schematic of the control board for the solid-state relays.

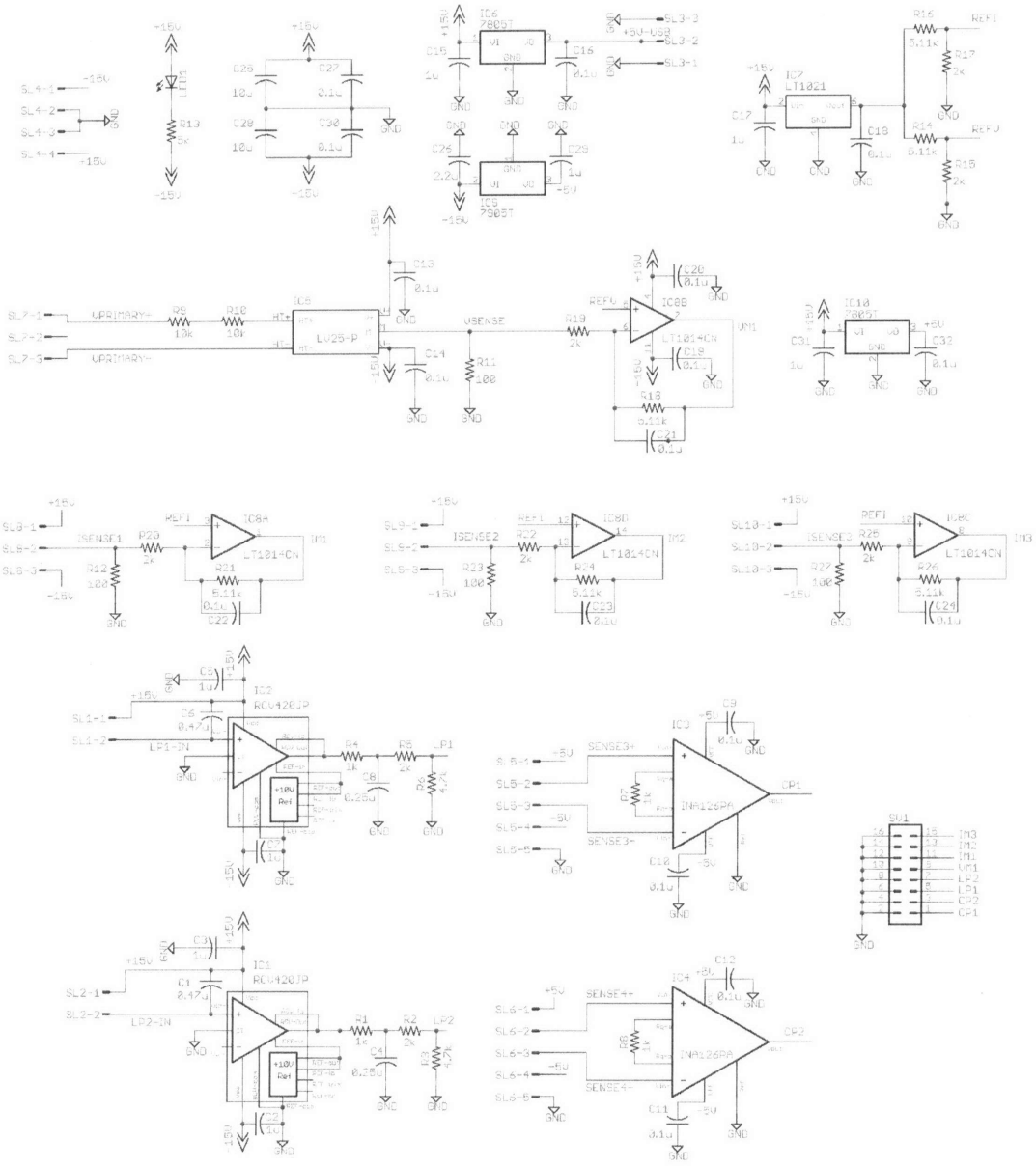


Figure B-6: Schematic of instrumentation board for cylinder pressure sensors and suction and discharge pressure sensors.

Appendix C

Refrigerant Charge Information

This appendix contains a selection of code and information relevant to the undercharge diagnostic methods discussed in Chapter 4. This information is broken into four distinct sections, three of which contain Matlab code.

§**C.1:** In this section, the information on the thermal instrumentation installed at the test residence is summarized for future reference.

§**C.2:** The data fusing code is provided in this section. The code in `eventid1.m` does event detection on a single channel of real power data as output from prep, and it records the times that the compressor turns on and off, as well as the averaged power values over that interval. `syncdata.m` synchronizes the data geenrated by `eventid1.m` and the temperature data, and `postprocess.m` takes in the assortment of datafiles and generates a file with all of the relevant information for the operation of the diagnostic methods.

§**C.3:** The actual fault detection method is listed in this code, which is a relatively short file that operates on the two sets of undercharge and regular charge data.

§**C.4:** The method is validated in three pieces of code, provided here. The building simulation is run in `house_real.m`, while the forward euler solver is given in `rfeuler.m` and the wrapper which calls the forward Euler solver is given in `rhouse.m`.

C.1 Thermal Instrumentation

Table on following page.

Function	Serial Number	Instrumentation	Logging Interval	Location
Discharge Temp	1064384	external temp	1m 15s	inside condenser unit (tiewrapped to pipe, insulated)
Suction Temp	1064381	external temp	1m 15s	inside condenser unit (tiewrapped to pipe, insulated)
Cond Temp	1064382	external temp	1m 15s	tiewrapped to middle of coil, insulated
Evap Temp	1064383	external temp	1m 15s	tiewrapped to bend in coils on edge of evap
Evap R/H (return)	1064387	internal temp & humidity	2m 30s	sitting on top of evap coil housing (aluminum)
Liquid Line temp	1064386	external temp	1m 15s	tiewrapped to liquid line near evaporator in attic
Outdoor Temp	678795	internal temp	10s	under eaves of shed
Upstairs Thermostat Temp	678794	internal temp	10s	on top thermostat in bedroom
Evap Air Temp	678796	internal temp	1m 15s	in supply air duct on one side
Downstairs Thermostat Temp	678792	internal temp	10s	on top of thermostat in living room
Sunlight, &c	1185219	sunlight, humidity, temperature	1m	on steel pole in yard

C.2 Data Synchronization Scripts

C.2.1 eventid1.m

```
function [out_matrix, outcode] = eventid1(y, incode)

% this is just a functionization of ident_events1.

% codevec(1): (normal operation) (binary)
% codevec(2): (on at the end of the last start transient) (binary)
% codevec(3): (start index) (value)
% codevec(4:end): (the rest of the values depend on the outcode given)

% from Rob's nilmconst.m file (event detector).

%   y = load('~/static/charge-diag/state4/p0813.1');
%   incode = soutcode;

   FILTER_LENGTH = 32;
   FILTER_CUTOFF = .1;
   GETEVENTS_THRESHOLD = 1200; % Threshold used in event detection
   GETEVENTS_SKIP = 500; % Number of points skipped after the first event
       detection
   GETEVENTS_MINAMPLITUDE = 1e3;
   GETEVENTS_SSDELAY = 1e4;

   Ts = 1/120;
   outcode = [];
   out_matrix = [];
   post_matrix = [];

   b = fir1(FILTER_LENGTH, FILTER_CUTOFF); % Create event detection filter
   thresh = GETEVENTS_THRESHOLD;

   % filter the input.

   fy = filter(b,1,y);
   dy = y - fy;


   % find the places where the levels change abruptly (aka something turns on.)

   ify = [];
   ify = find(abs(diff(abs(dy)>thresh)) == 1);


   % if the unit is on during the whole datafile:

   if ((isempty(ify) && (mean(fy) > GETEVENTS_MINAMPLITUDE)) && incode(2))

      if ((length(fy) - incode(3)) > 120*60*60)

         transient_proc = [incode(4:length(incode))';
                           y];

         for (pt = 1:floor((length(y)-incode(3))/(120*60*60)))
```

```
        chunk = transient_proc(120*60*60*(pt-1)+(1:(120*60*60)));

        A = [ones(size(chunk)) (1:length(chunk))'];
        mu = A \ chunk;
        chunk_approx = A*mu;

        chunk_start = incode(3) + 120*60*60*(pt-1);

        output_line = [Ts*chunk_start;                                    60
                       Ts*length(chunk);;
                       chunk_approx(1);
                       chunk_approx(length(chunk_approx))]';

        out_matrix = [out_matrix;
                      output_line];
      end

      chunk_start = incode(3) + 120*60*60*pt;
      chunk = transient_proc(120*60*60*pt:length(transient_proc));        70

      outcode(1) = 0;
      outcode(2) = 1;
      outcode(3) = - (length(y) - chunk_start);
      outcode = [outcode chunk'];
      return;

   end

end                                                                       80


% use the ssdelay to identify only discrete events.

for (k = 1:length(ify))

   if ( (ify(k) < 100) && (mean(y((ify(k)-floor(ify(k)/2)):ify(k))) >
       GETEVENTS_MINAMPLITUDE) )
      continue
   else
      indt = ify(k);                                                      90
      first_index = k+1;
      break;
   end

end

for k = first_index:length(ify)

   test1 = ((ify(k) - indt(length(indt))) < GETEVENTS_SKIP);
   test2 = (abs((y(ify(k)+ceil(GETEVENTS_SKIP/2)) - y(ify(k)-ceil(    100
       GETEVENTS_SKIP/2)))) < GETEVENTS_THRESHOLD);

   if (test1 || test2)
      continue
   else
```

```
      indt = [indt; ify(k)];
   end
end

ind_out = indt;

% if an event starts before the beginning of the datafile, but concludes
    during
% the datafile.

if (mean(y(floor(indt(1)/2):indt(1))) > GETEVENTS_MINAMPLITUDE)

   if (incode(2))
      index_end = ind_out(1);
      transient_end = y(1:index_end);

      transient_proc = [incode(4:length(incode))';
                  transient_end];

      A = [ones(size(transient_proc)) (1:length(transient_proc))'];
      mu = A \ transient_proc;
      transient_approx = A*mu;

      output_line = [Ts*incode(3);
                  Ts*(index_end - incode(3));
                  transient_approx(1);
                  transient_approx(length(transient_approx))]';

      out_matrix = output_line;

      ind_out = ind_out(2:length(ind_out));

   else
      error('Spurious event: error 1');
   end

end


% if an event starts, but doesn't conclude before the end of the datafile.

if (mean(y(indt(length(indt))+(1:floor((length(y)-indt(length(indt)))/2)))) >
     thresh)

   transient = y(ind_out(length(ind_out)):length(y));

   if (length(transient) > 120*60*60)

      for (pt = 1:floor(length(transient)/(120*60*60)))

         chunk = transient(120*60*60*(pt-1)+(1:(120*60*60)));
         ndx1 = (1:length(chunk))';
         ndx1a = flipud(ndx1);
         chunk_rev = chunk(ndx1a);
         chunk_out = zeros(size(chunk));
```

```
            chunk_out(1) = chunk_rev(1);
                                                                          160
            for pt2 = 2:length(chunk_rev);

               db = abs(chunk_rev(pt2) - chunk_rev(pt2-1));

               if ((db > 500) && (length(chunk_rev)-pt2 < 100))
                  break;
               else
                  chunk_out(pt2) = chunk_rev(pt2);
               end
                                                                          170
            end

            chunk_proc = flipud(chunk_out(find(chunk_out)));

            A = [ones(size(chunk_proc)) (1:length(chunk_proc))'];
            mu = A \ chunk_proc;
            chunk_approx = A*mu;

            chunk_start = ind_out(length(ind_out)) + 120*60*60*(pt-1);
                                                                          180
            output_line = [Ts*chunk_start;
                           Ts*length(chunk);
                           chunk_approx(1);
                           chunk_approx(length(chunk_approx))]';

            post_matrix = [post_matrix;
                           output_line];
         end

         chunk = transient(120*60*60*pt+1:length(transient));             190
         chunk_start = ind_out(length(ind_out)) + 120*60*60*pt;

         outcode(1) = 0;
         outcode(2) = 1;
         outcode(3) = - (length(y) - chunk_start);
         outcode = [outcode chunk'];

      else

         transient_start = ind_out(length(ind_out));                      200

         outcode(1) = 0;
         outcode(2) = 1;
         outcode(3) = - (length(y) - transient_start);
         outcode = [outcode transient'];

      end

      ind_out = ind_out(1:(length(ind_out)-1));
                                                                          210
   end


   % now ind_out only has the transients in it which start and end in this file.
```

```
% compute the best-fit line to the steady-state portion by flipping each
    transient
% upside down (to get rid of the initial inrush portion of the waveform), and
     then
% looking for the place where the derivative of the waveform starts to grow
    beyond
% a threshold.
                                                                          220
if (mod(length(ind_out),2) ~= 0)
   warning('the length of ind_out is not even.');
end

for (k = 1:2:length(ind_out))

   transient = y(ind_out(k):ind_out(k+1));

   ndx1 = (1:length(transient))';
   ndx1a = flipud(ndx1);                                                  230
   transient_rev = transient(ndx1a);
   transient_out = zeros(size(transient));
   transient_out(1) = transient_rev(1);

   for pt = 2:length(transient_rev);

      db = transient_rev(pt) - transient_rev(pt-1);

      if ((db > 500) && (length(ndx1)-pt < 100))
         break;                                                           240
      else
         transient_out(pt) = transient_rev(pt);
      end

   end

   transient_proc = flipud(transient_out(find(transient_out)));

   if (length(transient_proc) > 120*60*60)
                                                                          250
      for (pt = 1:floor(length(transient_proc)/(120*60*60)))

         chunk = transient_proc(120*60*60*(pt-1)+(1:(120*60*60)));

         A = [ones(size(chunk)) (1:length(chunk))'];
         mu = A \ chunk;
         chunk_approx = A*mu;

         chunk_start = ind_out(k) + 120*60*60*(pt-1);
                                                                          260
         output_line = [Ts*chunk_start;
                        Ts*length(chunk);
                        chunk_approx(1);
                        chunk_approx(length(chunk_approx))]';

         out_matrix = [out_matrix;
                       output_line];
```

```
        end

        chunk = transient_proc(120*60*60*pt:length(transient_proc));
        A = [ones(size(chunk)) (1:length(chunk))'];
        mu = A \ chunk;
        chunk_approx = A*mu;

        chunk_start = ind_out(k) + 120*60*60*pt;

        output_line = [Ts*chunk_start;
                       Ts*length(chunk);
                       chunk_approx(1);
                       chunk_approx(length(chunk_approx))]';

        out_matrix = [out_matrix;
                      output_line];

     else

        A = [ones(size(transient_proc)) (1:length(transient_proc))'];
        mu = A \ transient_proc;
        transient_approx = A*mu;

        % output matrix: each line corresponds to one transient;
        % the makeup of each line: [(start time) (start power) (end time) (end
            power)]

        output_line = [Ts*ind_out(k);
                       Ts*(ind_out(k+1)-ind_out(k));
                       transient_approx(1);
                       transient_approx(length(transient_approx))]';

        out_matrix = [out_matrix;
                      output_line];

     end

  end


  % fix output

  if (isempty(outcode));
     outcode = [1 0 0];
  end


  out_matrix = [out_matrix;
                post_matrix];
```

C.2.2 *syncdata.m*

```
function [datet, powerdata] = syncdata(compfile, func, offset)

% compfile is the name of the file that has the data to process.
```

```
% sync is what purpose we want this function to perform.
%   func = 0: generate a vector which has 1s when the compressor is on, and 0s
%            when it is off.
%   func = 1: generate a vector which has the linear approximation to the power
%            level during steady state operation.

   fid = fopen(compfile, 'r');
   A = textscan(fid, '%s%f%f%f', 'delimiter', '|');                          10
   fclose(fid);

   startvec = datenum(A{1}, 'yyyy-mm-dd HH:MM:SS');
   lengthvec = A{2};
   startpower = A{3};
   endpower = A{4};

   % get the right time to start the time vector, and create said vector.

   stime = datevec(startvec(1));                                             20
   etime = datevec(startvec(size(startvec,1)));

   if ( (etime(5) > 0) || (etime(6) > 0) )
      dtime = datevec(datenum([etime(1:3) etime(4)+1 0 0]) - datenum([stime(1:4)
          0 0]));
   else
      dtime = datevec(datenum([etime(1:4) 0 0]) - datenum([stime(1:4) 0 0]));
   end

   Ts = 1/120;                                  % for the nilm
   ind = (1:(dtime(3)*120*60*60*24 + dtime(4)*120*60*60))';                  30
   t = Ts*(ind-1);

   % create a series of datenums with the samples incremented in seconds.
   datet = datenum([stime(1)*ones(size(t,1),1) stime(2)*ones(size(t,1),1) stime
       (3)*ones(size(t,1),1) stime(4)*ones(size(t,1),1) zeros(size(t,1),1) t]);

   % create a vector of off times (rather than run times, which is what
       lengthvec is.)
   endvec = datenum(datevec(startvec) + [zeros(size(startvec,1),5) lengthvec]);


   % sync = 0: just have ones when the compressor turns on, and zeros when it's   40
       off.

   if (func == 0)

      ndx = 1;
      powerdata = zeros(size(t));

      for (k = 1:length(ind))

         if (datet(k) >= startvec(ndx))
                                                                             50
            if (datet(k) >= (endvec(ndx)))
               ndx = ndx + 1;
               continue;
            end
```

```
            powerdata(k) = 1;

         end

      end                                                                    60

      % for testing.
      % ind1 = 12*120*3600+(1:120*60*60)';
      % plot(datet(ind1), powerdata(ind1))
      % datetick('x', 15)

   elseif (func == 1)

      % now try to actually generate the approximation to the power when it's on
          .
                                                                             70
      ndx = 1;
      powerdata = zeros(size(t));

      for (k = 1:length(ind))

         if (ndx > size(startvec,1))
            break;
         end

         if (datet(k) >= startvec(ndx))                                      80

            if (datet(k) >= (endvec(ndx)))
               ndx = ndx + 1;
               continue;
            end

            powerdata(k) = startpower(ndx) + ((endpower(ndx)-startpower(ndx))/(
                endvec(ndx)-startvec(ndx)))*(datet(k)-startvec(ndx));

         end
                                                                             90
      end

      % for testing.
      % ind1 = 12*120*3600+(1:120*60*60)';
      % plot(datet(ind1), powerdata(ind1))
      % datetick('x', 15)

   else
      warning('func not equal to 0 or 1.\n');
                                                                            100
   end

   clear endvec;

   datet = datenum(datevec(datet) + [zeros(size(datet,1),5) offset*ones(size(
       datet,1),1)]);
```

C.2.3 postprocess.m

```
function outmatrix = postprocess(inmatrix, timevector, offset, filename)

   outmatrix = zeros(size(inmatrix));
   FID = fopen(filename, 'w');

   yr = timevector(1);
   mo = timevector(2);
   day = timevector(3);
   hr = timevector(4);
   min = timevector(5);
   sec = timevector(6);

   offsetmin = offset(1);
   offsetsec = offset(2);

   for (k = 1:size(inmatrix,1))

      outmatrix(k,1) = datenum(yr, mo, day, hr, min+offsetmin, sec+offsetsec+
          inmatrix(k,1));
      outmatrix(k,2) = inmatrix(k,2);
      outmatrix(k,3) = inmatrix(k,3);
      outmatrix(k,4) = inmatrix(k,4);

      fprintf(FID, '%s|%.5e|%.5e|%.5e\n', ...
            datestr(outmatrix(k,1), 31), ...
            outmatrix(k,2), ...
            outmatrix(k,3), ...
            outmatrix(k,4));

   end

   fclose(FID);
```

C.3 Method Testing

```
% generate the detection plots for the histogram representation.

clear;
clear all;

% state 4: normal charge: 08/09, 1pm thru 09/05, 12pm

runinfo4 = load('~/static/charge-diag/cycling-data/state4/runinfo4_char.proc');

% the 2-d representation.

tempvals4 = (min(runinfo4(:,2)):0.05:max(runinfo4(:,2)))';
out4 = zeros(length(tempvals4), 2);

for (k = 1:length(tempvals4))
   temp = tempvals4(k);
   ndx = find(abs(runinfo4(:,2)-temp) < 0.05);
```

```
   if (isempty(ndx))
      continue;
   end
   out4(k,1) = mean(runinfo4(ndx,1));
   out4(k,2) = mean(runinfo4(ndx,2));
end

% the standard deviations do not provide much additional information, since we
% don't have multiple datapoints at the same dT at this resolution.

for (k = 1:size(out4,1))
   if (out4(k,1) > 3600)
      out4(k,1) = 3600;
   end
end

% get rid of the zeros and interpolate.
ind4 = find(out4(:,1));
temp4ind = tempvals4(ind4);
out4m = interp1(temp4ind, out4(ind4,1), tempvals4, 'spline');



% state 5: undercharge.

runinfo5 = load('~/static/charge-diag/cycling-data/state5/runinfo5_char.proc');

% the 2-d representation.

tempvals5 = (min(runinfo5(:,2)):0.05:max(runinfo5(:,2)))';
out5 = zeros(length(tempvals5), 2);

for (k = 1:length(tempvals5))
   temp = tempvals5(k);
   ndx = find(abs(runinfo5(:,2)-temp) < 0.05);
   if (isempty(ndx))
      continue;
   end
   out5(k,1) = mean(runinfo5(ndx,1));
   out5(k,2) = mean(runinfo5(ndx,2));
end

for (k = 1:size(out5,1))
   if (out5(k,1) > 3600)
      out5(k,1) = 3600;
   end
end


% get rid of the zeros and interpolate.
ind5 = find(out5(:,1));
temp5ind = tempvals5(ind5);
out5m = interp1(temp5ind, out5(ind5,1), tempvals5, 'spline');

for (k = 1:size(out5m,1))
   if (out5m(k) > 3600)
```

```
      out5m(k) = 3600;
   end
end


% plot both sets of data.

plot(out4m, tempvals4, out5m, tempvals5)

ind4 = (1:length(out4))';
ind5 = (1:length(out5))';
plot(ind4, out4m, ind5, out5m)

% these must be filtered.

w = 'db5';
N = 12;

[c4,l4] = wavedec(out4m, N, w);
o4filt = wrcoef('a', c4, l4, w, 4);
[c5,l5] = wavedec(out5m, N, w);
o5filt = wrcoef('a', c5, l5, w, 4);

plot(ind4, o4filt, ind5, o5filt)


% now do the validation

runinfo4v = load('~/static/charge-diag/cycling-data/state4/runinfo4_valid.proc')
    ;
runinfo5v = load('~/static/charge-diag/cycling-data/state5/runinfo5_valid.proc')
    ;

runinfo4v = [runinfo4v(find(runinfo4v(:,1)),1) runinfo4v(find(runinfo4v(:,1)),2)
    ];

plot((1/60)*o4filt, tempvals4, (1/60)*o5filt, tempvals5, (1/60)*runinfo4v(:,1),
    runinfo4v(:,2), 'o')

plot((1/60)*o4filt, tempvals4, (1/60)*o5filt, tempvals5, (1/60)*runinfo5v(:,1),
    runinfo5v(:,2), 'o')


% compute the distances for a detection metric.

% regular charge dataset.

valid4a = zeros(size(runinfo4v,1),2);
valid4b = zeros(size(runinfo4v,1),2);

for (k = 1:length(runinfo4v))
   ndx4 = find(min(abs(abs(runinfo4v(k,2)-abs(tempvals4))))==abs(abs(runinfo4v(k
       ,2)-abs(tempvals4))));
   ndx5 = find(min(abs(abs(runinfo4v(k,2)-abs(tempvals5))))==abs(abs(runinfo4v(k
       ,2)-abs(tempvals5))));
   valid4a(k,:) = [o4filt(ndx4) tempvals4(ndx4)];
```

```
   valid4b(k,:) = [o5filt(ndx5) tempvals5(ndx5)];
end

training = [valid4a; valid4b];
group = [ones(size(valid4a,1),1); 2*ones(size(valid4b,1),1)];

svmstruct1 = svmtrain(training, group);
class1b = svmclassify(svmstruct1, runinfo4v);
length(find(class1b==1));

% undercharge dataset.

valid5a = zeros(size(runinfo5v,1),2);
valid5b = zeros(size(runinfo5v,1),2);

for (k = 1:length(runinfo5v))
   ndx4 = find(min(abs(abs(runinfo5v(k,2)-abs(tempvals4))))==abs(abs(runinfo5v(k
       ,2)-abs(tempvals4))));
   ndx5 = find(min(abs(abs(runinfo5v(k,2)-abs(tempvals5))))==abs(abs(runinfo5v(k
       ,2)-abs(tempvals5))));
   valid5a(k,:) = [o4filt(ndx4) tempvals4(ndx4)];
   valid5b(k,:) = [o5filt(ndx5) tempvals5(ndx5)];
end

training = [valid5a; valid5b];
group = [ones(size(valid5a,1),1); 2*ones(size(valid5b,1),1)];

svmstruct2 = svmtrain(training, group);
class2b = svmclassify(svmstruct2, runinfo5v);
length(find(class2b==2))/length(class2b)
```

C.4 Building Simulation Code

C.4.1 house_real.m

```
% write a simple simulation of the house as cooled by an air-conditioner, to
% try to characterise its dynamics.

clear;
clear all;

global Ch1 Rh1 Ch2 Rh2 Rh3;
global Qac Qload;
global compstate;
global ac_on;

ac_on = [];

% parameters
Ch1 = 2e7; Ch2 = 1.2e5;
Rh1 = 0.01; Rh2 = 0.003; Rh3 = 0.006;
Qac = 3596; Qload = 1580; % regular charge

compstate = 0;
```

```
t = (0:5:3*24*3600)';
omega = 2*pi/(24*3600);
To = 24-6*sin(omega*t);

Ti = rfeuler('rhouse', [24; 24], t);
plot((1/3600)*t, To, (1/3600)*t, Ti(1,:)')


% now make the same plots.

N = 17280;
start = 2400; % regular charge
state = [ac_on(N,2); N];
output = zeros(size(ac_on));

for k = N+start+(1:2*N-start)

   if ((ac_on(k,2) == state(1)) && (state(1) == 1) && (k == (state(2)+720)))

      output(k,1) = t(k)-t(state(2));

      To_m = mean([To(k) To(state(2))]);
      Ti_m = mean([Ti(1,k) Ti(1,state(2))]);
      output(k,2) = To_m - Ti_m;

      state = [ac_on(k,2); k];

      continue;

   end

   if (ac_on(k,2) ~= state(1))

      if (state(1) == 1)

         output(k,1) = t(k)-t(state(2));

         To_m = mean([To(k) To(state(2))]);
         Ti_m = mean([Ti(1,k) Ti(1,state(2))]);
         output(k,2) = To_m - Ti_m;

         state = [ac_on(k,2); k];

      elseif (state(1) == 0)

         state = [ac_on(k,2); k];

      end

   end

end

ndx = find(output(:,1));
ndx2 = find(output(ndx,1) > 500);
```

```
ndx = ndx(ndx2);
ndx2 = find(output(ndx,1) < 3601);
ndx = ndx(ndx2);

save -mat-binary ~/bucket/crlaugh/charge-diag/2008-08-04/regdata2.mat output
                                                                          80
% first cycling diagnostic.

tempvals = (min(output(ndx,2)):0.05:max(output(ndx,2)))';
out = zeros(length(tempvals), 2);

for (k = 1:length(tempvals))
   temp = tempvals(k);
   ndxr = find(abs(output(ndx,2)-temp) < 0.05);
   if (isempty(ndxr))
      continue;                                                           90
   end
   out(k,1) = mean(output(ndx(ndxr),1));
   out(k,2) = mean(output(ndx(ndxr),2));
end

ind = find(out(:,1));
out_m = interp1(tempvals(ind), out(ind,1), tempvals, 'spline');

w = 'db5'; N = 12;
[c,l] = wavedec(out_m, N, w);                                             100
o_filt = wrcoef('a', c, l, w, 4);

plot((1/60)*o_filt, tempvals)

save -mat-binary ~/bucket/crlaugh/charge-diag/2008-08-04/regdata.mat out
    tempvals
```

C.4.2 rfeuler.m

```
function y = rfeuler(func, x0, t)                                          0

   y = zeros(2, length(t));
   y(:,1) = x0;
   dt = t(2)-t(1);

   for (k = 2:size(y,2))
      y(:,k) = y(:,k-1) + dt.*eval('rhouse(y(:,k-1), t(k-1))');
   end
```

C.4.3 rhouse.m

```
function Tidot = rhouse(Ti,t)                                              0

   Tidot = zeros(2,1);

   global Ch1 Ch2 Rh1 Rh2 Rh3;
   global Qac Qload
   global compstate;
```

```
global ac_on;

omega = 2*pi/(24*3600);
To = 24-6*sin(omega*t);                                                    10

uthresh = 23.5;
bthresh = 23.4;

Tidot(1) = (1/(Rh1*Ch1))*(To-Ti(1)) - (1/(Rh2*Ch1))*(Ti(1)-Ti(2)) + Qload/Ch1
    ;
Tidot(2) = (1/(Rh3*Ch2))*(To-Ti(2)) + (1/(Rh2*Ch2))*(Ti(1)-Ti(2));

if ( (compstate == 0) && (Ti(1) >= uthresh) )
   compstate = 1;
   Tidot(2) = Tidot(2) - Qac/Ch2;                                          20
   ac_on = [ac_on; t 1];
   return;
elseif ( (compstate == 1) && (Ti(1) >= bthresh) )
   compstate = 1;
   Tidot(2) = Tidot(2) - Qac/Ch2;
   ac_on = [ac_on; t 1];
   return;
elseif ( (compstate == 1) && (Ti(1) < bthresh) )
   compstate = 0;
   ac_on = [ac_on; t 0];                                                   30
   return;
end

ac_on = [ac_on; t 0];
```

Bibliography

[1] New one-family houses completed with central air conditioning, number of houses. AHRI website, as obtained from U.S. Census Bureau.

[2] Th. Afjei, P. Suter, and D. Favrat. Experimental analysis of an inverter-driven scroll compressor with liquid injection. In *Proceedings of the International Compressor Conference at Purdue*, volume 2, pages 541–550, July 1992.

[3] P.L. Alger. *Induction Machines: Their Behavior and Uses*. Gordon and Breach, 2nd edition, 1970.

[4] V.R. Anderson and J.R. Tobias. Comfort control for central electric heating systems. *IEEE Transactions on Industry Applications*, IA-10(6):741–745, November 1974.

[5] P.P. Angelov, R.A. Buswell, V.I. Hanby, and J.A. Wright. A methodology for modeling HVAC components using evolving fuzzy rules. In *26th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 247–252, Nagoya, 28 October 2000.

[6] P. R. Armstrong. *Model Identification with Application to Building Control and Fault Detection*. PhD thesis, Massachusetts Institute of Technology, September 2004.

[7] P.R. Armstrong, C.R. Laughman, S.B. Leeb, and L.K. Norford. Fault detection based on motor start transients and shaft harmonics measured at the RTU electrical service. In *International Refrigerigation and Air Conditioning Conference at Purdue*, number R137, pages 1–10, 12 July 2004.

[8] P.R. Armstrong, C.R. Laughman, S.B. Leeb, and L.K. Norford. Detection of rooftop cooling unit faults based on electrical measurements. *HVAC+R Research Journal*, 12(1):151–175, January 2006.

[9] ASHRAE. *ASHRAE Handbook: Fundamentals*. ASHRAE, Atlanta, GA, 2001.

[10] ASHRAE. *ASHRAE Handbook: HVAC Systems and Equipment*. ASHRAE, Atlanta, GA, 2004.

[11] K. Assawamartbunlue and M.J. Brandemuehl. Refrigerant leakage detection and diagnosis for a distributed refrigeration system. *HVAC&R Research*, 12(3):389–405, July 2006.

[12] C. Aydin and B. Ozerdem. Air leakage measurement and analysis in duct systems. *Energy and Buildings*, 38:207–213, 2006.

[13] Shawket Ayub, James W. Bush, and David K. Haller. Liquid refrigerant injection in scroll compressors operating at high compression ratios. In *1992 International Compressor Engineering Conference at Purdue*, volume 2, pages 561–567, July 1992.

[14] M. Backstrom. *Keltetechnik*. Verlag und Druck G. Braun, Karlsruhe, 1953.

[15] A. Bejan. *Advanced Engineering Thermodynamics*. Wiley-Interscience, second edition, 1997.

[16] I. Bell, V. Lemort, J.E. Braun, and E. Groll. Analysis of liquid-flooded compression using a scroll compressor. In *International Compressor Engineering Conference at Purdue University*, 1263, pages 1–9, 2008.

[17] M.E.H. Benbouzid and G.B. Kliman. What stator current processing-based technique to use for induction motor rotor faults diagnosis? *IEEE Transactions on Energy Conversion*, 18(2):238–244, June 2003.

[18] M.E.H. Benbouzid, M. Veiera, and C. Theys. Induction motors' faults detection and localization using stator current advanced signal processing techniques. *IEEE Transactions on Power Electronics*, 14(1):14–22, January 1999.

[19] M. Benouarets, A.L. Dexter, R.S. Fargus, P. Haves, T.I. Salsbury, and J.A. Wright. Model-based approaches to fault detection and diagnosis in air-conditioning systems. In *Proceedings of the Fourth International Conference on System Simulation in Buildings*, pages 529–547, Liege, Belgium, 1995.

[20] F. Bin and J. Churgay. A study on migration phenomenon in automotive A/C systems. In W. Soedel, editor, *1998 International Compressor Engineering Conference at Purdue*, volume 1, pages 225–230. Purdue University, 14 July 1998.

[21] J.E. Braun. Automated fault detection and diagnostics for vapor compression cooling equipment. *ASME Journal of Solar Energy Engineering*, 125:266–274, August 2003.

[22] M.S. Breuker and J.E. Braun. Common faults and their impacts for rooftop air conditioners. *International Journal of HVAC+R Research*, 4(3):303–318, June 1998.

[23] M.S. Breuker and J.E. Braun. Evaluating the performance of a fault detection and diagnostic system for vapor compression equipment. *International Journal of HVAC+R Research*, 4(4):401–425, October 1998.

[24] F.R. Carrie, A. Bossaer, J.V. Andersson, P. Wouters, and M.W. Liddament. Duct leakage in european buildings: status and perspectives. *Energy and Buildings*, 32(3):235–243, September 2000.

[25] B. Checket-Hanks. Compressor problem: 'No Defect Found'. *Air Conditioning, Heating, and Refrigeration News*, pages 10,12, 19 May 2003.

[26] B. Chen and J.E. Braun. Simple rule-based methods for fault detection and diagnostics applied to packaged air conditioners. *ASHRAE Transactions*, 107(1):847–857, 2001.

[27] M.C. Comstock and J.E. Braun. Literature Review for Application of Fault Detection and Diagnostic Methods to Vapor Compression Cooling Equipment. Technical Report HL 99-19, Report #4036-2, Purdue University, December 1999. Sponsored by ASHRAE Deliverable for Research Project 1043-RP.

[28] G. Cooper. *Air-Conditioning America: Engineers and the Controlled Environment, 1900-1960*. Johns Hopkins University Press, 1998.

[29] Copeland Corp. *Copeland Refrigeration Manual - Part 5: Installation and Service*, 1970.

[30] Copeland Corp. *Copeland Application Engineering Bulletin AE-1182-R24*, Apr 1993. Application Note 22-1182.

[31] A. Cowan. Review of recent commercial rooftop unit field studies in the Pacific Northwest and California. Technical report, New Buildings Institute, PO Box 653, White Salmon, WA, 98672, 8 October 2004.

[32] R.W. Cox. *Minimally intrusive strategies for fault detection and energy monitoring*. PhD thesis, Massachusetts Institute of Technology, September 2006.

[33] R.W. Cox, S.B. Leeb, and L.K. Norford. A minimally intrusive, low cost system for determining indoor air flow patterns. In *Proceedings of the 2004 IEEE Workshop on Computers in Power Electronics*, pages 63–68, August 2004.

[34] J.B. Cummings and C.R. Withers Jr. Problems related to air handler leakage. *ASHRAE Journal*, 50(1):36–46, January 2008.

[35] J.B. Cummings, C.R. Withers Jr., J. McIlvaine, J.K. Sonne, and M.J. Lombardi. Air handler leakage: field testing results in residences. *ASHRAE Transactions*, 109(1):496–502, 2003.

[36] M. Cunniffe, R.W. James, and A. Dunn. An analysis of fault occurrence in refrigeration plant and the effect on current practice. *Australian Refrigeration, Air Conditioning, and Heating*, 40(7):36–43, July 1986.

[37] M.H.F. De Salis and D.J. Oldham. The development of a rapid single spectrum method for determining the blockage characteristics of a finite length duct. *Journal of Sound and Vibration*, 243(4):625–640, June 2001.

[38] T. DeNucci, R.W. Cox, S.B. Leeb, J. Paris, T.J. McCoy, C.R. Laughman, and W.C. Greene. Diagnostic indicators for shipboard systems using non-intrusive load monitoring. In *IEEE Electric Ship Technologies Symposium*, Philadelphia, PA, July 2005.

[39] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[40] J.R. Dormand and P.J. Prince. Practical runge-kutta processes. *SIAM Journal on Scientific and Statistical Computing*, 10(5):977–989, September 1989.

[41] D.G. Dorrell, W.T. Thomson, and S. Roach. Analysis of airgap flux, current, and vibration signals as a function of the combination of static and dynamic airgap eccentricity in 3-phase induction motors. *IEEE Transactions on Industry Applications*, 33(1):24–34, January 1997.

[42] T. Downey and J. Proctor. What can 13,000 air conditioners tell us? In *2002 ACEEE Summer Study on Efficiency in Buildings*, pages 53–68, 2002. Panel 1.

[43] D. Dragomir-Daescu, A.A. Al-khalidy, M. Osama, and G.B. Kliman. Damage detection in refrigerator compressors using vibration and current signatures. In *SDEMPED 2003 – Symposium on Diagnostics for Electric Machines, Power Electronics, and Drives*, pages 355–360, Atlanta, GA, 24 August 2003.

[44] A.K. Dutta, T. Yanagisawa, and M. Fukuta. An investigation of the performance of a scroll compressor under liquid refrigerant injection. *International Journal of Refrigeration*, 24:577–587, 2001.

[45] D.K. Eads. Establishing a centrifugal-fan performance curve. *Chemical Engineering*, 88(6):201–208, 1981.

[46] B. Eck. *Fans*. Pergamon Press, Oxford, New York, 1973. trans. Ram S. Azad and David R. Scott.

[47] H.W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Springer, 2000.

[48] M. Farzad and D. L. O'Neal. System performance characteristics of an air conditioner over a range of charging conditions. *International Journal of Refrigeration*, 14:321–328, November 1991.

[49] Michael Y. Feng, Kurt W. Roth, Detlef Westphalen, and James Brodrick. Packaged rooftop units: automated fault detection and diagnostics. *ASHRAE Journal*, 47(4):68–72, April 2005.

[50] A.E. Fitzgerald, C. Kingsley Jr., and S.D. Umans. *Electric Machinery*. Mc-Graw Hill, sixth edition, 2003.

[51] P.W. Francisco and L. Palmiter. Field evaluation of a new device to measure air handler flow. *ASHRAE Transactions of the AIEE*, 109(2):403–412, 2003.

[52] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *The GNU Scientific Library Manual*, version 1.9 edition, February 2007.

[53] A.S. Glass, P. Gruber, M. Roos, and J. Todtli. Preliminary evaluation of a qualitative model-based fault detector for a central air-handling unit. In *Proceedings of the Third IEEE Conference on Control Applications*, volume 3, pages 1873–1882, Glasgow, Scotland, U.K., 24 August 1994.

[54] A.S. Glass, P. Gruber, M. Roos, and J. Todtli. Qualitative model-based fault detection in air-handling units. *IEEE Control Systems Magazine*, 15(4):11–22, August 1995.

[55] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[56] H. Goodfellow and E. Tahti. *Industrial Ventilation Design Guidebook*. Academic Press, 2001.

[57] D.Y. Goswami, G. Ek, M. Leung, C.K. Jotshi, S.A. Sherif, and F. Colacino. Effect of refrigerant charge on the performance of air-conditioning systems. *International Journal of Energy Research*, 25:741–750, 2001.

[58] I.N. Grace, D. Datta, and S.A. Tassou. Sensitivity of refrigeration performance to charge levels and parameters for on-line leak detection. *Applied Thermal Engineering*, 25(4):557–566, March 2005.

[59] H.T. Grimmelius, J.K. Woud, and G. Been. On-line failure diagnosis for compression refrigeration plants. *International Journal of Refrigeration*, 18(1):31–41, 1995.

[60] F. Gustafsson. Determining the initial states in forward-backward filtering. *IEEE Transactions on Signal Processing*, 44(4):988–992, April 1996.

[61] D. Hales, A. Gordon, and M. Lubliner. Duct leakage in new washington state residences: findings and conclusions. *ASHRAE Transactions of the AIEE*, 109(2):393–402, 2003.

[62] A.K. Halm-Owoo and K.O. Suen. Applications of fault detection and diagnostic techniques for refrigeration and air conditioning: a review of basic principles. *Proceedings of the Institution of Mechanical Engineers Part E - Journal of Process Mechanical Engineering*, 216(3):121–132, August 2003.

[63] C.Y. Han, Y. Xiao, and C.J. Ruther. Fault detection and diagnosis of HVAC systems. *ASHRAE Transactions*, 105(1):568–578, 1999.

[64] T.M. Harms, E.A. Groll, and J.E. Braun. Accurate charge inventory modeling for unitary air conditioners. *HVAC&R Research*, 9(1):55–77, January 2003.

[65] A. Hasan. The occurrence of faults in heating and air conditioning equipment. *The Heating & Ventilating Engineering and Journal of Air Conditioning*, 47(561):447–455, April 1974.

[66] P. Haves. Fault modelling in component-based HVAC simulation. In *Building Simulation '97; Fifth International IBPSA Conference*, volume 1, pages 119–127, 8 September 1997.

[67] P. Haves, T.I. Salsbury, and J.A. Wright. Condition monitoring in HVAC subsystems using first principles models. *ASHRAE Transactions*, 102(1):519–527, 1996.

[68] J.M. House, K.D. Lee, and L.K. Norford. Controls and diagnostics for air distribution systems. *Journal of Solar Energy Engineering*, 125:310–317, August 2003.

[69] J.M. House, W.Y. Lee, and D.R. Shin. Classification techniques for fault detection and diagnosis of an air-handling unit. *ASHRAE Transactions*, 105(1):1087–1097, 1999.

[70] K.D. Hurst and T.G. Habetler. Sensorless speed measurement using current harmonic spectral estimation in induction machine drives. *IEEE Transactions on Power Electronics*, 11(1):66–73, January 1996.

[71] K.D. Hurst and T.G. Habetler. A comparison of spectrum estimation techniques for sensorless speed estimation in induction machines. *IEEE Transactions in Industry Applications*, 33(4):898–905, July 1997.

[72] R. Isermann. Process fault detection based on modeling and estimation methods – a survey. *Automatica*, 20(4):387–404, July 1984.

[73] R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer, 2006.

[74] M. Ishida and K. Iwata. A new slip frequency detector of an induction motor using rotor slot harmonics. *IEEE Transactions on Industry Applications*, IA-20(3):575–582, May 1984.

[75] M. Kim, S.H. Yoon, P.A. Domanski, and W.V. Payne. Design of a steady-state detector for fault detection and diagnosis of a residential air conditioner. *International Journal of Refrigeration*, 31:790–799, 2008.

[76] Minsung Kim and Min Soo Kim. Performance investigation of a variable speed vapor compression system for fault detection and diagnosis. *International Journal of Refrigeration*, 28:481–488, 2005.

[77] W. Kim, J.E. Braun, and H. Li. Evaluation of a virtual refrigerant charge sensor. In *International Refrigeration and Air-Conditioning Conference at Purdue*, number 2314, pages 1–8, 2008.

[78] R.J. Kind and M.G. Tobin. Flow in a centrifugal fan of the squirrel-cage type. *Transactions of the ASME Journal of Turbomachinery*, 112:84–90, January 1990.

[79] P.C. Krause, O. Wasynczuk, and S.D. Sudhoff. *Analysis of Electric Machinery*. McGraw-Hill, 1986.

[80] C. R. Laughman, R. LaFoy, W. Wichakool, P.R. Armstrong, S.B. Leeb, L.K. Norford, J.R. Rodriguez, K. Goebel, A. Patterson-Hine, and E.C. Lupton. Electrical and mechanical methods for detecting liquid slugging in reciprocating compressors. In *International Compressor Engineering Conference at Purdue*, page C1437, July 2008.

[81] C.R. Laughman, P.R. Armstrong, L.K. Norford, and S.B. Leeb. The detection of liquid slugging phenomena in reciprocating compressors via power measurements. In *International Compressor Engineering Conference at Purdue*, page C084. Purdue University, 20 July 2006.

[82] C.R. Laughman, K.D. Lee, R.W. Cox, S.R. Shaw, S.B. Leeb, L.K. Norford, and P.R. Armstrong. Power signature analysis. *IEEE Power and Energy Magazine*, 1(2):56–63, March 2003.

[83] T.M. Lawrence, J.D. Mullen, D.S. Noonan, and J. Enck. Overcoming barriers to efficiency. *ASHRAE Journal*, 47:S40–S47, September 2005.

[84] S. Lee. Don't bury the compressor before it's dead. *Air Conditioning, Heating, and Refrigeration News*, pages 01,10, 4 April 2005.

[85] W.Y. Lee, J.M. House, and D.R. Shin. Fault diagnosis and temperature sensor recovery for an air-handling unit. *ASHRAE Transactions*, 103(1):621–633, 1997.

[86] W.Y. Lee, C. Park, and G.E. Kelly. Fault detection in an air-handling unit using residual and recursive parameter identification methods. *ASHRAE Transactions*, 102(1):528–539, 1996.

[87] S.B. Leeb, S.R. Shaw, and J.L. Kirtley. Transient event detection in spectral envelope estimates for nonintrusive load monitoring. *IEEE Transactions in Power Delivery*, 10(3):1200–1210, July 1995.

[88] H. Li and J.E. Braun. On-line models for use in automated fault detection and diagnosis for HVAC&R equipment. In *Proceedings of the 2002 ACEEE Conference*, 7.147-7.158, Monterey, CA, 2002.

[89] H. Li and J.E. Braun. An improved method for fault detection and diagnosis applied to air conditioners. *ASHRAE Transactions*, 109(2):683–692, 2003.

[90] H. Li and J.E. Braun. Application of automated fault detection and diagnostics for rooftop air conditioners in california. In *Proceedings of the 2004 ACEEE Conference*, pages 3.190–3.201, Monterey, CA, 2004.

[91] H. Li and J.E. Braun. An economic evaluation of automated fault detection and diagnosis for rooftop air conditioners. In *Proceedings of the International Refrigeration and Air Conditioning Conference at Purdue*, R145, pages 1–10, 12 July 2004.

[92] H. Li and J.E. Braun. A methodology for diagnosing multiple-simultaneous faults in rooftop air conditioners. In *Proceedings of the International Refrigeration and Air Conditioning Conference at Purdue*, R131, pages 1–10, 12 July 2004.

[93] H. Li and J.E. Braun. Decoupling features and virtual sensors for diagnosis of faults in vapor compression air conditioners. *International Journal of Refrigeration*, 30(3):546–564, March 2007.

[94] H. Li and J.E. Braun. A methodology for diagnosing multiple simultaneous faults in vapor-compression air conditioners. *International Journal of HVAC+R Research*, 13(2):369–395, March 2007.

[95] Haorong Li. *A Decoupling-Based Unified Fault Detection and Diagnosis Approach for Packaged Air Conditioners*. PhD thesis, Purdue University, August 2004.

[96] Z. Liu and W. Soedel. An investigation of compressor slugging problems. In *1994 International Compressor Engineering Conference at Purdue*, volume 2, pages 433–440, July 1994.

[97] Z. Liu and W. Soedel. A mathematical model for simulating liquid and vapor two-phase compression processes and investigating slugging problems in compressors. *HVAC+R Research Journal*, 1(2):99–109, April 1995.

[98] F. McQuiston, J. D. Parker, and J.D. Spitler. *Heating, Ventilating, and Air Conditioning : Analysis and Design*. John Wiley and Sons, sixth edition, 2005.

[99] V.C. Mei, F.C. Chen, and Z. Gao. Development of a nonintrusive refrigerant charge indicator. *ASHRAE Transactions*, 111(1):276–281, 2006.

[100] R.J. Mowris, A. Blankenship, and E. Jones. Field measurements of air conditioners with and without txvs. In *2004 ACEEE Summer Study on Energy Efficiency in Buildings*, pages 212–227, 2004. Panel 1.

[101] S. Mukhopadhyay and S. Chaudhuri. A feature-based approach to monitor motor-operated valves used in nuclear power plants. *IEEE Transactions on Nuclear Science*, 42(6):2209–2220, December 1995.

[102] W. Murphy. Mobile home air conditioning: An analysis of seasonal performance. Master's thesis, Purdue University, May 1977.

[103] W. Murphy. Basic air conditioner diagnostics using a simple digital thermostat. In *International Refrigeration and Air-Conditioning Conference at Purdue*, 2424, pages 1–8, 2008.

[104] S. Nandi. Modeling of induction machines including stator and rotor slot effects. *IEEE Transactions on Industry Applications*, 40(4):1058–1065, July 2004.

[105] S. Nandi, S. Ahmed, H.A. Toliyat, and R.M. Bharadwaj. Selection criteria of induction machines for speed-sensorless drive applications. *IEEE Transactions on Industry Applications*, 39(3):704–712, May 2003.

[106] S. Nandi, H.A. Toliyat, and X. Li. Condition monitoring and fault diagnosis of electrical machines - a review. *IEEE Transactions on Energy Conversion*, 20(4):719–729, December 2005.

[107] J. Navarro-Esbri and R. Torrella, E.and Cabello. A vapour compression chiller fault detection technique based on adaptative algorithms. application to on-line refrigerant leakage detection. *International Journal of Refrigeration*, 29:716–723, 2006.

[108] L.W. Nelson and J.L Magnussen. Analytical predictions of residential electric heating system performance. *IEEE Transactions on Industry Applications*, IA-10(6):746–750, November 1974.

[109] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 2nd edition, 1999.

[110] R.H. Park. Two-reaction theory of synchronous machines: Generalized method of analysis, part i. *Transactions of the AIEE*, 48:716–727, 1929.

[111] J. Phelan, M.J. Brandemuehl, and M. Krarti. In-situ performance testing of fans and pumps for energy analysis. *ASHRAE Transactions*, 3(1):318–332, 1997.

[112] P. Pillay and Z. Xu. Motor current signature analysis. In *Conference Record of the 1996 IEEE Industry Applications Conference*, volume 1, pages 587–594, 1996.

[113] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[114] R. Radhakrishnan, D. Nikovski, K. Peker, and A. Divakaran. A comparison between polynomial and locally weighted regression for fault detection and diagnosis of HVAC equipment. In *32nd IEEE Annual Conference on Industrial Electronics*, pages 3668–3673, November 2006.

[115] T. M. Rossi and J.E. Braun. A statistical, rule-based fault detection and diagnostic method for vapor compression air conditioners. *International Journal of HVAC+R Research*, 3(1):19–37, January 1997.

[116] K. Roth, D. Westphalen, and J. Brodrick. Residential central ac fault detection and diagnostics. *ASHRAE Journal*, 48(5):96–97, May 2006.

[117] A. Salkin. Shivering for luxury. *The New York Times*, 26 June 2005.

[118] R.R. Schoen and T.G. Habetler. Effects of time-varying loads on rotor fault detection in induction machines. *IEEE Transactions on Industry Applications*, 31(4):900–906, July 1995.

[119] D.A. Schrank, J.P. Vaccaro, and R.G. Lewis. The design and analysis of a relief plate to reduce cylinder pressures in a reciprocating compressor under liquid slugging conditions. In *1998 International Compressor Conference at Purdue*, volume 2, pages 479–487, July 1988.

[120] G.A.F. Seber and C.F. Wild. *Nonlinear Regression*. Probability and Mathematical Statistics. Wiley, 1989.

[121] L.F. Shampine. *Numerical Solution of Ordinary Differential Equations*. Chapman & Hall Mathematics, 1994.

[122] L.F. Shampine and M.W. Reichelt. The matlab ode suite. *Siam Journal of Scientific Computational Optimization and Applications*, 18(1):1–22, January 1997.

[123] S.R. Shaw. *System Identification Techniques and Modeling for Nonintrusive Load Diagnostics*. PhD thesis, Massachusetts Institute of Technology, February 2000.

[124] S.R. Shaw, C.B. Abler, R.F. Lepard, D. Luo, S.B. Leeb, and L.K. Norford. Instrumentation for high performance nonintrusive electrical load monitoring. *Transactions of the ASME Journal of Solar Energy Engineering*, 120(3):224–229, August 1998.

[125] S.R. Shaw, M. Keppler, and S.B. Leeb. Pre-estimation for better initial guesses. *IEEE Transactions on Instrumentation and Measurement*, 53(3):762–769, June 2004.

[126] S.R. Shaw and C.R. Laughman. A method for nonlinear least squares using structured residuals. *IEEE Transactions on Automatic Control*, 51(10):1704–1708, October 2006.

[127] S.R. Shaw and C.R. Laughman. A Kalman-filter spectral envelope preprocessor. *IEEE Transactions on Instrumentation and Measurement*, 56(5):2010–2017, October 2007.

[128] S.R. Shaw and S.B. Leeb. Identification of induction motor parameters from transient stator current measurements. *IEEE Transactions on Industrial Electronics*, 46(1):139–149, February 1999.

[129] S.R. Shaw, S.B. Leeb, L.K. Norford, and R.W. Cox. Nonintrusive load monitoring and diagnostics in power systems. *IEEE Transactions on Instrumentation and Measurement*, 57(7):1445–1454, July 2008.

[130] S.R. Shaw, L.K. Norford, D. Luo, and S.B. Leeb. Detection and diagnosis of HVAC faults via electrical load monitoring. *International Journal of HVAC+R Research*, 8(1):13–40, January 2002.

[131] B.G. Shiva Prasad. Effect of liquid on a reciprocating compressor. *Journal of Energy Resources Technology*, 124:187–190, September 2002.

[132] J. Shugars, P. Coleman, C. Payne, and L.V.W. McGrory. Bridging the efficiency gap: commercial packaged rooftop air conditioners. In *2000 ACEEE Summer Study*, 2000.

[133] H.G. Siewert. Compressor tolerance to liquid refrigerant. In W. Soedel, editor, *Proceedings of the 1972 Compressor Technology Conference*, pages 245–249. Purdue University, 25 July 1972.

[134] F. Simpson and G. Lis. Liquid slugging measurements in reciprocating compressors. In *International Compressor Conference at Purdue*, volume 2, pages 479–487, July 1988.

[135] H. Singh. Refrigerant monitoring for hvac applications. *Heating, Piping, and Air-Conditioning (HPAC) Engineering*, 73(8):29–32, August 2001.

[136] R. Singh, J.J. Nieter, and G. Prater Jr. An investigation of the compressor slugging phenomenon. *ASHRAE Transactions*, 92(4):250–258, 1986.

[137] R. Singh, G. Prater Jr, and J.J. Nieter. Prediction of slugging-induced cylinder overpressure. In *1986 International Compressor Engineering Conference at Purdue*, volume 2, pages 444–459, August 1986.

[138] R. Singh, G. Prater Jr., and J.J. Nieter. Slugging conditions in refrigeration compressors. In *International Multiphase Fluid Transients Symposium*, pages 63–74, 1986.

[139] K. Sinnamohideen. Discrete-event diagnostics of heating, ventilation, and air-conditioning systems. In *Proceedings of the American Control Conference*, pages 2072–2076, Arlington, VA, 25 June 2001.

[140] K. Srinivasan. Measurement of air leakage in air-handling units and air conditioning ducts. *Energy and Buildings*, 37:273–277, 2005.

[141] M. Stocks and A. Medvedev. Estimation of induction machine parameters at start-up using current envelope. *Conference Record of the Industry Applications Conference*, pages 1163–1170, 13 October 2002.

[142] D.E. Stouppe and T.Y.S. Lau. Air conditioning and refrigeration equipment failures. *National Engineer*, 93:14–17, 1989.

[143] M. Stylianou. A response for performance degradation in rooftop units. Technical Report TR 1999-19, CEDRL - Natural Resources Canada, 1615 Lionel-Boulet, Varennes, Quebec, Canada, J3X 1S6, 1999.

[144] R. Supangat, N. Ertugrul, W.L. Soong, D.A. Gray, C. Hansen, and J. Grieger. Broken rotor bar fault detection in induction motors using starting current analysis. In *2005 European Conference on Power Electronics and Applications*, 11 September 2005.

[145] R. Supangat, N. Ertugrul, W.L. Soong, D.A. Gray, C. Hansen, and J. Grieger. Detection of broken rotor bars in induction motor using starting-current analysis and effects of loading. *IEEE Proceedings on Electric Power Applications*, 153(6):848–855, November 2006.

[146] TASC Technical Staff. *Applied Optimal Estimation*. MIT Press, 1974.

[147] S.A. Tassou and I.N. Grace. Fault diagnosis and refrigerant leak detection in vapour compression refrigeration systems. *International Journal of Refrigeration*, 28(5):680–688, August 2005.

[148] C. Thybo and R. Izadi-Zamanabadi. Development of fault detection and diagnosis schemes for industrial refrigeration systems - lessons learned. In *Proceedings of the 2004 IEEE International Conference on Control Applications*, pages 1248–1253, 2 September 2004.

[149] C. Thybo, R. Izadi-Zamanabadi, and H. Niemann. Toward high performance in industrial refrigeration systems. In *Proceedings of the 2002 IEEE International Conference on Control Applications*, pages 915–920, Glasgow, Scotland, U.K., 18 September 2002.

[150] J. Tomczyk. *Troubleshooting and Servicing Modern Air Conditioning and Refrigeration Systems*. ESCO Press, 1995.

[151] C.W. Ueberhuber. *Numerical Computation*, volume 1. Springer, 1997.

[152] P.B. Usoro, I.C. Schick, and S. Negahdaripour. An innovation-based methodology for HVAC system fault detection. *Journal of Dynamic Systems, Measurement, and Control*, 107(4):284–289, December 1985.

[153] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 2000.

[154] P. Vas. *Parameter Estimation, Condition Monitoring, and Diagnosis of Electrical Machines*. Clarendon Press, 1993.

[155] J. Wagner and R. Shoureshi. Failure detection diagnostics for thermofluid systems. *Journal of Dynamic Systems, Measurement, and Control*, 114(4):699–706, December 1992.

[156] M.L. Wald. New math for summer: the cost of chill. *The New York Times*, 23 June 2005.

[157] Shengwei Wang and Jin-Bo Wang. Robust sensor fault diagnosis and validation in HVAC systems. *Transactions of the Institute of Measurement and Control*, 24(3):231–262, 2002.

[158] J. Wen and T.F. Smith. Development and validation of online parameter estimation for HVAC systems. *Journal of Solar Energy Engineering*, 125:324–330, August 2003.

[159] D. Westphalen, K. Roth, and J. Brodrick. Duct leakage fault detection. *ASHRAE Journal*, 47(8):56–58, August 2005.

[160] W.C. Whitman, W.M. Johnson, and J. Tomczyk. *Refrigeration and Air-Conditioning Technology*. Delmar/Thomson Learning, 2000.

[161] Gerald J. Williams. Specifying fans. *HPAC Engineering*, pages 20–26, November 2003.

[162] E.L. Winandy and J. Lebrun. Scroll compressors using gas and liquid injection: experimental analysis and modelling. *International Journal of Refrigeration*, 25:1143–1156, 2002.

[163] T. Yanagisawa, T. Shimizu, and M. Fukuta. Foaming characteristics of an oil-refrigerant mixture. *International Journal of Refrigeration*, 14:132–136, May 1991.

[164] H. Yoshida, H. Yuzawa, T. Iwami, and M. Suzuki. Typical faults of air-conditioning systems and fault detection by ARX model and extended Kalman filter. *ASHRAE Transactions*, 102(1):557–564, 1996.

[165] M. Yoshimura and N. Ito. Effective diagnosis methods for air-conditioning equipment in telecommunications buildings. In *11th Annual Telecommunications Energy Conference*, pages 21.1/1–21.1/7, Florence, 15 October 1989.