

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Emory University, I agree that the Library of the University shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was prepared, or, in his absence, by the Dean of the Graduate School when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from, or publication of, this thesis that involves potential financial gain will not be allowed without written permission.

Alexander C. Shkolnik

NOTICE TO BORROWERS

Unpublished theses deposited in the Emory University Library must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Alexander C. Shkolnik
77 Kirkwood Rd
West Hartford, CT 06117

The director of this thesis is:

Dr. James Lu
Department of Mathematics and Computer Science
400 Dowman Drive
Atlanta, GA 30322

Users of this thesis not regularly enrolled as students at Emory University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

Name of use	Address	Date	Type of use (Examination only or copying)
-------------	---------	------	---

Neurally Controlled Simulated Robot:
Applying Cultured Neurons to Handle an Approach / Avoidance Task in Real Time
and a Framework for Studying Learning in Vitro

By

Alexander C. Shkolnik
Master of Sciences

Department of Mathematics and Computer Science

James Lu
Adviser

Steve Potter
Committee Member

Thomas DeMarse
Committee Member

Phil Hutto
Committee Member

Accepted:

Dean of the Graduate School

Date

Neurally Controlled Simulated Robot:
Applying Cultured Neurons to Handle an Approach / Avoidance Task in Real Time
and a Framework for Studying Learning in Vitro

by

Alexander C. Shkolnik

Thesis Adviser: Dr. James Lu
Mathematics and Computer Science, Emory University

Research Adviser: Dr. Steve Potter
Laboratory for Neuroengineering, Georgia Institute of Technology

An Abstract of
a thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements for the degree of
Master of Sciences

Department of Mathematics and Computer Science

2003

Abstract

Little is known about how information is encoded in the brain, and even less is known about how computation and useful data manipulation occurs in living neural networks. The goal of this project was to construct a simulated robot to explore data encoding and processing in living neuronal networks. Information was encoded by varying timings between neuronal input stimulations. Encoding information in this way resulted in a non-linear neural response. This response, if interpreted as a “computation” can be used to emulate any logic gate, and thus a Universal Turing Machine. This neural response was used to control a simulated robot in real-time to approach an object if it was too far away, or to avoid an object if it was too close. The animat provides a framework for studying living neural networks at the behavioral level. Such an animat may also be useful to study learning in living neural networks, as expressed by changes in the animat’s behavior.

Neurally Controlled Simulated Robot:
Applying Cultured Neurons to Handle an Approach / Avoidance Task in Real Time
and a Framework for Studying Learning in Vitro

by

Alexander C. Shkolnik

Thesis Adviser: Dr. James Lu
Mathematics and Computer Science, Emory University

Research Adviser: Dr. Steve Potter
Laboratory for Neuroengineering, Georgia Institute of Technology

A thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements of the degree for
Master of Sciences

Department of Mathematics and Computer Science

2003

Acknowledgements

It is a pleasure to thank the many people who made this thesis possible.

In no particular order, special thanks to: my research adviser at the Georgia Institute of Technology Laboratory for Neuroengineering, Dr. Steve Potter for giving me the opportunity to work in his lab and for the insights I have gained under his direction; Former post-doc in the Potter lab, now professor of biomedical engineering at University of Florida, Dr. Thomas DeMarse for giving me training and for helping me with the project; my Emory thesis adviser and professor in Math / CS at Emory University, Dr. James Lu for giving me material support and direction; The faculty of the Math / CS department at Emory University for giving me a broad education in computer science; the faculty of the Neuroscience and Behavioral Biology department at Emory University for giving me a broad education in neuroscience; Dr. Pat Marsteller, Dr. David Edwards and the SURE program for exposing me to research and pushing me forward; and my parents, Dr. Nikolay Shkolnik and Valentina Shkolnik for their support and help.

CONTENTS

Chapter	Page
1. Introduction	1
2. A Brief Review of Neuroscience and Living Neural Networks	3
2.1 Bottom up approach to studying the brain: Molecular and cellular neuroscience.....	4
2.2 Top down approach to studying the brain: psychology, psychobiology ..	11
2.3 Living Neural Networks	12
3. Neural Computation, MEA's and Animats	13
3.1 Information encoding and processing.....	13
3.2 Multielectrode array, recording activity, learning	16
3.3 Animats and robots	16
4. Neurally Controlled Animat Problem	18
5. Effect of dual channel probing on neural activity	22
5.1 Background.....	22
5.2 Method	24
5.3 Data Analysis.....	26
5.4 Results.....	27
6. Demonstrating the computational power of the IPI neural effect	38
6.1 Schema for emulation of digital logic gates	38
6.2 Input mappings for emulation of a NOT gate.....	41
6.3 Emulation of an AND or NOR gate.....	42
6.4 Emulation of an OR / NAND gate	43
6.5 To emulate an XOR gate:	44
6.6 Results.....	45
7. Construction of an animat to track and follow a reference object at a given distance	47
7.1 Method.....	47
7.1.1 Neural culture, hardware and software to interface with neurons	47

7.1.2	Animat.....	48
	Encoding sensory information (distance and direction from robot to object) to neural stimulation (IPI):.....	49
	Decoding Neural activity to animat movement:	50
7.2	Behavioral Testing.....	59
7.3	Software Implementation.....	59
7.4	Animat Behavioral Testing Results	61
8.	Learning expressed as changes in animat behavior.....	67
8.1	Background.....	67
8.2	Method	68
8.3	Results.....	70
9.	Conclusions and Future Directions	76
10.	References	78
	Appendix A – Data Analysis Code.....	81
A.1	getSpikeData.m.....	82
A.2	compMeans.m.....	84
A.3	plotrastall.m.....	88
A.4	plotSpkRaster.m.....	89
A.5	sortSpikeData.m.....	91
A.6	plot_hist.m.....	92
	Appendix B – animat Code	93
B.1	run.m	94
B.2	animat.m.....	95
B.3	printObj.m	99
B.4	printRobot.m	100
B.5	decodeNeur.m	101
B.6	histTrain.m	103
B.7	simstimIPI.m	105
B.8	trainMappings.m	106
B.9	testDecode2.m.....	112
B.10	stimIPI.cpp.....	113

ILLUSTRATIONS

Figure	Page
1. NEURONAL SCHEMATIC	5
2. ACTION POTENTIAL	6
3. ELECTRICAL CIRCUIT MODEL OF A NEURON	9
4. HISTOGRAM OF RESPONSE OF CHANNEL C TO PROBING ON CHANNEL A (75 TRIALS) .	24
5. MEA DISH.....	26
6. BLOWUP OF MEA DISH WITH CULTURED NEURONS, SHOWING WIRES AND ELECTRODES USED FOR BOTH RECORDING AND STIMULATION.....	26
7. FULL IPI RESPONSE CURVE (DISH 4819, CH 38 AND 25).....	28
8. FULL IPI RESPONSE CURVE, DATA FROM EXPERIMENT IN CHAPTER 7	28
9. EFFECT OF TWO PROBES WITH LONG IPI (TOP) OR SHORT IPI (BOTTOM).....	30
10. EFFECT OF PROBING TWO CHANNELS AT ONCE (BLUE) VS. ONE CHANNEL ALONE (RED)	30
11. POST-STIM IPI RESPONSE CURVE (DISH 4819, CH 38 AND 25)	32
12. (BLOW UP OF FIGURE 11)	33
13. POST IPI RESPONSE CURVE, DATA FROM EXPERIMENT IN CHAPTER 7	33
14. RASTER PLOT DEPICTING SPIKES ON GIVEN CHANNELS FOLLOWING 75 TRIALS OF PROBING AT IPI = 250 MS.....	35
15. HISTOGRAM DEPICTING RESPONSE OF CHANNEL 44 TO THE SECOND PROBE OF AN IPI PAIR, DISH 4819, 1/13/03.....	37
16. POST IPI RESPONSE CURVE.....	39
17. ANIMAT IN ITS WORLD; ANIMAT SHOWN HAS COORDINATES { X = 15; Y = 20; $\theta = \pi / 6$ }	48

18. ANIMAT INPUT MAPPING; ENCODING SENSORY INFORMATION (DISTANCE OF ANIMAT TO REFERENCE OBJECT) INTO MAGNITUDE OF IPI PAIR PULSE STIMULATION	50
19. SCHEMATIC DIAGRAM DEPICTING LOCK AND KEY DECODING.....	53
20. SAMPLE FULL IPI RESPONSE CURVE; FORWARD / BACKWARD MOVE-MENT IS DECODED BY A THRESHOLD CORRESPONDING TO IPI OF DESIRED DISTANCE	54
21. UNIT MOVEMENT VECTOR	55
22. POST IPI RESPONSE CURVE.....	56
23. OUTPUT MAPPING: POST-IPI AVERAGED NEURAL RESPONSE TO VECTOR MAGNITUDE	57
24. DIAGRAM OF INPUT / IPI RESPONSE / OUTPUT CURVES: DISTANCE BETWEEN ANIMAT AND OBJECT TO ANIMAT MOVEMENT	58
25. SAMPLE PATH TAKEN BY ANIMAT DURING TRIAL	63
26. ANIMAT DISTANCE TO OBJECT – TOP: FULL CURVE FOR ALL 40 STEPS; BOTTOM: BLOWUP OF STEPS 10 THROUGH 30.....	64
27. RASTER PLOT OF 3 SETS OF POST-IPI TRAINING CURVES	71
28. LOWESS SMOOTHED PLOTS OF 3 SETS OF POST-IPI TRAINING CURVES	72
29. DISTANCE CURVES, PRE-TETANUS AND POST-TETANUS.....	73
30. DISTANCE CURVES (BLOW UP).....	73

TABLES

Table	Page
1. DIGITAL OUTPUT MAPPING: AVERAGED NEURAL RESPONSE TO A SET OF PROBES → DIGITAL OUTPUT	40
2. LOGICAL INPUT → PROBE DELAY, MAPPING FOR NOT GATE EMULATION	41
3. NOT GATE EMULATION TRUTH TABLE	41
4. LOGICAL INPUT → PROBE DELAY MAPPING FOR AND GATE EMULATION	42
5. AND GATE EMULATION TRUTH TABLE	42
6. LOGICAL INPUT → PROBE DELAY MAPPING FOR OR GATE EMULATION	43
7. OR GATE EMULATION TRUTH TABLE	43
8. LOGICAL INPUT → PROBE DELAY MAPPING FOR XOR GATE EMULATION	44
9. XOR GATE EMULATION TRUTH TABLE	44

CHAPTER 1

Introduction

Information in digital computers is encoded in binary, and this information is processed mainly in a serial fashion using Boolean logic gates. The brain utilizes a highly distributed parallel architecture for encoding information and can manipulate this information efficiently to allow an animal to process sensory information in real-time. Little is known about how information is actually encoded in the brain, and even less is known about how computation and useful data manipulation occurs in living neural networks. The goal of this project was to explore information encoding and computation in neural networks. Information was encoded through varying timings between inputs to a living neuronal network. Neural computation was explored through the construction of a simulated robot.

Encoding information by varying the time delay between inputs to a living neural network produces a non-linear “IPI probing response” in the neurons. Interpreting this response as a computation, I show that living neural networks are capable of emulating a Universal Turing Machine, and can thus execute any computer program with polynomial slow-down. The non-linear response to information encoding is used to construct a simulated robot which behaves as a simulated animal, or animat. The animat is constructed in such a way as to approach an object that is far away, and to avoid objects that are very close. Thus, a neurally controlled simulated robot is built to handle a type of approach / avoidance problem. The constructed animat can be a useful tool for studying properties of individual neurons in a population setting at a behavioral level of testing.

Of great interest to neuroscientists is the ability of synapses, or connections between neurons, to change strength based on activity within those neurons. This feature is thought to provide the brain with memory functions, but may also be a useful computational tool employed by the brain. Changes in neuronal properties or synapses can change the nonlinear dynamics of the IPI probing response. Given that the animat's behavior is constructed around this response, changes at the neuronal level may be expressed at the behavioral level of the animat. I show the feasibility of using the animat to study learning in neurons, by looking at changes in the animat's behavior.

CHAPTER 2

A Brief Review of Neuroscience and Living Neural Networks

Given the highly interdisciplinary nature of this work, a basic (and somewhat oversimplified) introduction to neuroscience and living neural networks will be presented. This chapter will provide useful definitions of terms and review certain assumptions I have made regarding the nature of the biological component used in completing this project.

The mammalian nervous system is comprised primarily of neurons and glial cells. Neurons, which are the primary computing blocks in the brain, are analogous in some ways to small logical circuits within a CPU. There are significant differences however. Unlike the logic gates of a CPU, a neuron is an analog device which receives analog inputs through its dendretic tree and outputs an analog signal through neurotransmitters delivered across a synapse to neighboring neurons. Glial cells, which outnumber neurons by 2:1, provide structural support for the neurons (the word glia means glue), are important in the conversion and production of substances the neurons need [Tsacopoulos, 2002], and may play a role in computation as well [Hansson and Ronnback, 2003]. Although the focus in this work will be on neurons, it is important to recognize the possible contributions of glial cells as well.

The neocortex in the adult human, a region of the brain thought to be responsible for most higher level processing, is estimated to have over 20 billion neurons [Pakkenberg et al. 2003]. There are roughly $.15 \times 10^{15}$ (.15 quadrillion) synapses (connections) between these neurons. Furthermore, there is an estimated 150,000 km of

myelinated fibers (insulated “wiring”) in the neocortex. Although each neuron is functionally slow – it takes a few milliseconds to achieve a response in an individual neuron, and each synapse can add a few milliseconds as well for interneuronal communication – the parallel and distributed processing of information by billions of highly interconnected neurons allows the brain to achieve computational results that are unparalleled by the relatively serial data processing capabilities of a digital computer.

The study of the brain has generally been broken down into two approaches: bottom-up, or looking at how the brain works by looking at how individual cells work, and top-down, or looking at systems in the brain at the macro-scale. Though each approach has made considerable progress, they are far from meeting anywhere on middle ground; we still have no idea how collections of individual neurons produce the effects seen at the macro level.

2.1 Bottom up approach to studying the brain: Molecular and cellular neuroscience

A typical neuron is divided into three sections: the dendrites (inputs), the soma or cell body (processing center) and the axon, which can split and communicate with thousands of other neurons (the output).

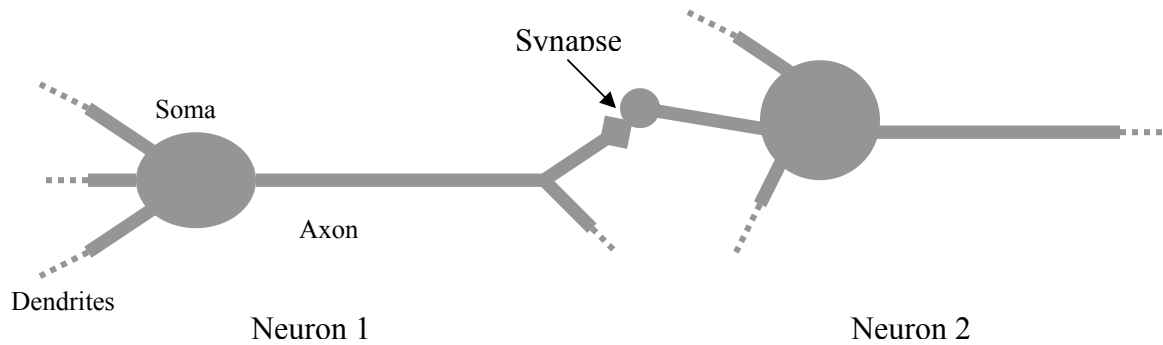


Figure 1: Neuronal schematic

Neurons, like all other cells, are bound by a membrane, which is only permeable to a few elements [Hodgkin and Huxley, 1952]. A neuron consumes energy to actively maintain a steady state voltage difference across this membrane. It does so by pumping Na^+ (sodium) ions out of the cell while also pumping K^+ (potassium) into the cell at a 3:2 ratio. The pumping produces significant concentration gradients, which are coupled with a flow of Na^+/K^+ ions that passively travel down their concentration gradients through “leak channels.” Since more positive ions are actively expelled from the cell than are taken up, the cell membrane maintains a steady state negative voltage potential of approximately -70mV . Thus, two forces are important: the voltage potential across the membrane, and the differences in ionic concentrations of various individual ion types.

In addition to the Na^+/K^+ pump, and the “leak” currents, additional processes play key roles in the activity of the neuron. Embedded within the neural membrane are a large number of proteins which act as ion channels. The channels are basically pores which are normally closed, and can open if something happens to manipulate their shape. Some channels can open when a chemical messenger binds to them. In other cases these

ion channels are sensitive to voltage changes in the membrane. When a cell depolarizes (i.e. its voltage increases from -70mV towards 0), a certain type of fast-acting voltage sensitive channels can open, which allow Na^+ to enter the cell because of electrical and chemical gradients at work. The influx of Na^+ further drives the membrane voltage positive, resulting in a positive feed-forward cascade which initiates the beginning of an action potential (or spike). After a short delay, the channels that opened to allow Na^+ to enter close via automatic processes. A slower channel then opens which allows K^+ to move down its concentration gradient, which functions to again hyperpolarize the cell. The result is an action potential as shown below:

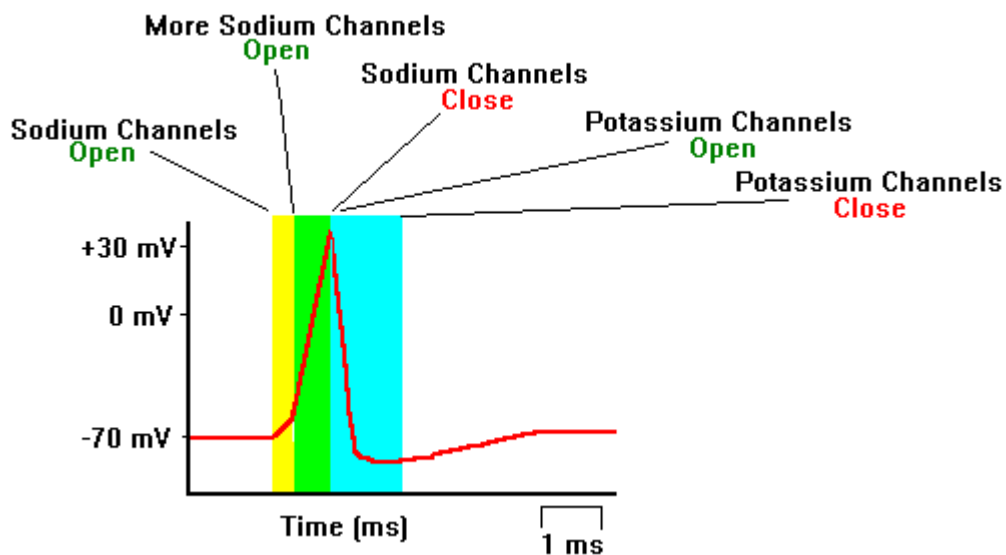


Figure 2: Action Potential

Figure reproduced with permission of Dr. Eric H. Chudler, Department of anesthesiology, University of Washington

To review: Na^+ channels with a very quick response time open, and close shortly after.

The influx of positive current opens slower K^+ channels, which lets K^+ out, thus

hyperpolarizing the cell. The voltage at which the membrane opens enough Na^+ channels

to allow more Na^+ to come in to depolarize the cell in a feed forward process is called the action potential voltage threshold, and is typically ~ -40 mV. Using this property of neurons, it is possible to artificially inject positive current into the cell (or negative current outside the cell), to depolarize the membrane enough to induce an action potential.

Given the electrical nature of the neuron, it is possible to “listen” to the neurons simply by recording electrical activity on an electrode inside or near the neuron. Further, it is possible to “talk” to the neurons by supplying a large current on an electrode inside or near a neuron, thus inducing an action potential. In typical in-vitro studies, Neurons are often punctured by electrodes to examine their electrical properties. Such a procedure is accurate and useful for many purposes, but kills the cell rather quickly and limits the number of cells that can be recorded from at one time. The experimental setup in this project (described below) utilizes large networks of living neurons cultured in-vitro atop an external grid of electrodes. External electrodes are more sensitive to background electrical noise than punctured electrodes, but are still sufficient to pickup nearby action potentials. Because the cells are not being punctured, such a system allows us to effectively communicate with large networks of living neurons without damaging them in the process.

An important question to consider is how do neurons normally communicate? Typically, thousands of synapses connect to a neuron’s dendrites and soma. At these synapses, if the presynaptic (input) neuron fires an action potential it releases a chemical messenger (neurotransmitter) which binds to the membrane of the postsynaptic neuron’s dendritic tree. The neurotransmitter induces changes which may slightly depolarize the

cell dependant on the type of neurotransmitter sent, the type of receptor on the dendritic tree, and also the location of that receptor. When the membrane is depolarized sufficiently and the voltage reaches a critical threshold value, it will fire an action potential which is then propagated along the axon. At this point, the propagation is “all or nothing” in that once initiated, the action potential will conduct along the axon to its terminal synapses.

In order to understand how these ionic compositions produce these characteristic action potentials, Hogkin and Huxley, using the nerve of a squid, characterized the relative concentrations responsible. Formally, this can be modeled as an electrical circuit where the Na^+/K^+ pump and ion concentration gradients act as batteries. Given the difference in charge between the inside and outside of a cell, the membrane acts as a capacitor. Finally, the various channels which allow ions through can be modeled as variable resistors. A simplified version of the neural electrical model looks something like this:

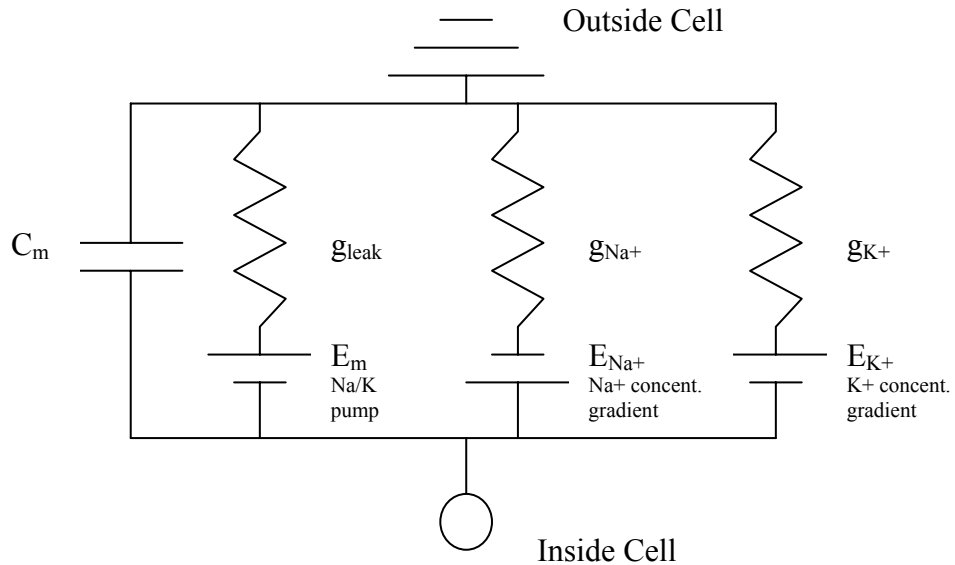


Figure 3: Electrical circuit model of a neuron

The above electrical circuit offers a good approximation of neural electrical behavior. I will not go much further into the details of how the equations from this circuit represent neural activity, but just about any book on computational neuroscience will review these equations [for example, see O'Reilly and Munakata, 2000]. If one factors in hundreds of other types of channels, and includes models for how the resistance of each channel changes (i.e. models of how individual channels open and close), one can see that this model for an individual section of neuronal membrane is very complex. Further, different regions of the neuron may have differing properties, so it becomes necessary to compartmentalize the model. I point this out only to demonstrate the inherent complexity within a single neuron. It is worthwhile to note that individual neurons may therefore have much more computational ability than a simple logic circuit. The description above is very much oversimplified, but enough to give some insight as to the workings of a

neuron. A great deal of literature, including several books, has been devoted to the topic of processing capability of single neurons [for example, see Koch, 1998].

The field of neuroscience has recently made significant advances in our understanding of how individual neurons work. Many researchers have built near complete models, capable of imitating the inputs and outputs of varying types of neurons. Such realistic models of neurons can be built in specialized computer applications such as GENESIS, NEURON or PDP++ [for model examples, see O'Reilly and Munakata, 2000 or Bower and Beeman, 1998]. These models typically involve compartments for each piece of dendrite / soma / axon, where each compartment has an electrical model as shown above as well as equations to represent the change of resistance for each type of channel. Although extremely useful for studying neuronal behavior, these realistic models of individual neurons utilize a significant amount of the processing capability of a digital computer, which is used to emulate the living neuron.

Given our computational restraints, realistic modeling of large living neural networks of neurons is difficult, and dependant on the complexity of the model, is sometimes not feasible with today's computing power. Those models that do try to emulate a few hundred cells working in a network require many minutes or even hours to compute the activity that would normally be done in those neurons in just a few seconds. If we are aiming to understand how the brain can process real-world data, then such a model is inefficient. Further, models inherently leave out details. Because we have so little information about how the brain computes information, it becomes very difficult to know which parts of a model are sufficient, and what may still be missing from the model. Thus computational modeling has at least two major drawbacks: first, modeling a

few dozen or even a few hundred cells (which is the limit of what a single computer may be able to handle in reasonable time) does not approach the computational capacity of our brain. Second, real neurons are inherently more complex than a model, and we simply can not say how much of this complexity may or may not be important for computation. Although computational models have proven to be extremely useful, they are non-optimal for studying activity in large neuronal networks. One solution to this problem involves the use of cultured living neuronal tissue, through which researchers can test hypotheses in a controlled environment.

2.2 Top down approach to studying the brain: psychology, psychobiology

In a top-down approach to studying the brain, we find psychologists studying behavior, and psychobiologists studying the underlying neuronal mechanisms of behavior. A great deal of information about how the brain is wired, what areas are associated with particular capacities, and how some regions are interconnected has been learned. Although it is not known at this time how the brain actually performs its computation, what is known is that computation occurs not solely at the level of the single neuron. The computation of the brain occurs across a population of perhaps millions of neurons.

2.3 Living Neural Networks

Given how much we know about the brain at various levels, can we put together a model for intelligence or even basic sensory / information processing? At this point in time, we are, unfortunately, far from answering this question. We understand how some individual neurons work reasonably well, down to the molecular level. We also vaguely understand how millions of neurons in a brain region are responsible for a given behavior. However, there is almost nothing we can say about how individual neurons work together to produce the effects we see at the top (behavioral) level.

One approach to combat this gap in understanding involves studying large networks of living neurons cultured in vitro. These networks can be studied under controlled conditions, allowing the researcher to observe any effects of modifying the activity of small numbers of neurons on the activity in the entire network. Such a system is used in the experiments described below, and is described in further detail in chapter 5.

CHAPTER 3

Neural Computation, MEA's and Animats

3.1 Information encoding and processing

Boolean logic is the mathematical system underlying digital electronics. Any information that a computer processes is stored in binary code, which can be processed by Boolean language. The mathematical system underlying information storage, processing and computation in the brain is unknown. Given the nature of neurons and neural networks, one can speculate about the nature of information encoding and processing.

One possible encoding scheme is the firing rate of an individual neuron averaged over time [Maass and Bishop, 1999]. Many neurons fire at a near constant rate, and this rate can be manipulated to contain information. Examples of this include the firing rate of stretch receptor neurons in a muscle spindle, which varies according to varied force applied to the muscle tissue [Kandel and Schwartz, 1991]. This is a widely used encoding concept mainly due to the ease with which experiments can be carried out. A possible problem with this approach is that animal behavior can change very quickly based on sensory information, and averaging spike firing over time may not be fast enough to code for this processing. In addition to examining the firing rate of individual neurons, one can examine firing rates averaged over groups of neurons. Studies have shown that groups of neurons may have correlated firing rates (see Hubel and Wiesel, 1962 for example of columns in visual cortex).

In addition to firing rate, spike timing patterns may play a crucial role in information coding and processing. These schemes can include time-to-first-spike following some sensory information [Maass and Bishop, 1999]. For example, strong sensory information may lead to shorter latency to a first spike on a subset of neurons. Another scheme is coding by relating spike timing to some periodic signal (phase coding). This has been studied in models [Hopfield, 1995] and experimentally [O'Keefe and Recce, 1993]. Synchrony between neurons or groups of neurons may play a role as well. Neurons firing in near synchrony may mean that data represented by individual or small groups of neurons is being grouped together by the larger collection of synchronous neurons, and can be treated as a unit [Milner 1974, Malsburg, 1981]. Further, timing between neurons that fire in synchrony may contain information relating to those grouped data sets.

Paired-pulse facilitation (PPF) and paired-pulse depression (PPD) are forms of short-term plasticity common to most chemically transmitting synapses, manifested as enhancement or suppression in the amplitude of the second of two rapidly evoked postsynaptic potentials [Zucker, 1989]. PPF/PPD is thought to be plasticity on a very short time scale of < 100 ms, and can be seen in just two neurons. Some speculate that this short latency plasticity is how the brain is actually able to compute information [Maass and Zador, 1998; Buonomano, 2000].

The present study will combine several of the approaches described above. I will induce spikes on two neurons with a varying time delay (a variation in synchrony) and test the effect this has on an averaged neural response firing rate of many neurons.

Perhaps because we do not have a thorough understanding of how information is encoded in neurons, we also can not say how neurons are able to compute. Many theoretical frameworks for neural computation have been proposed (for a review see Arbib, 1999; journals such as Neurocomputing and Neural Computation), yet few have been empirically demonstrated in living tissue. Artificial neural networks are often used to simulate neural processing with varying results. Many of these models are based on highly oversimplified neural models, yet can achieve some success.

Demonstrations of computation within living neurons in-vitro have been very limited. Some theoretical studies have shown that the logical AND (equivalent to binary multiplication of a single bit) operation is crucial ability for some single neurons [Koch, 1999], and this conclusion appears to have some empirical support as well [Shnupp and King, 2001]. Another study has demonstrated theoretical ability of addition in small networks of neurons, backed up by some weak empirical support. In the same study, the authors used the results from the addition experiment (which did not work out so well) to present weak evidence that the neural network could perform NOR operations (and thus all other logical operations through combination) [Garcia et al. 2002].

Emulation of digital gates in non-silicon substrates goes well beyond study in living neuronal networks. This has been shown in theoretical chaotic systems [Sinha, Ditto 1999], cellular biocomputing [Simpson et al. 2001], DNA [Lie et al, 2000], proteins [Deonarine et al. 2002], and quantum mechanics [Steane 1999].

3.2 Multielectrode array, recording activity, learning

The use of multielectrode array recording was developed independently by Thomas (1972), Pine (1974) and Gross (1974). The use of the MEA is made possible by several recent technological advances, and allows researchers to study cultured networks of neurons in-vitro [For a recent review see Potter 2001]. Neurons are cultured over a grid of non-invasive external electrodes which allow the researcher to monitor activity in a large number of neurons simultaneously. These same electrodes can provide a means of electrical stimulation.

MEAs have been used successfully to study information coding in systems such as the retina [Warland et al. 1997]. Much of the current research with MEAs involves studying input/output relations in cultured networks of neurons, and neural brain slices. Demonstrations of synaptic plasticity in culture are also of great interest. Several studies have effectively shown plasticity in cultured neuronal networks [for an excellent review see Marom and Shahaf, 2002].

[Tateno and Jimbo, 1998; Bi and Poo, 1999; Jimbo et al. 1999].

3.3 Animats and robots

The term *animat* [Wilson, 1991] was coined to describe a simulated animal in a virtual or robotic form. These animats can be used to perform behavioral studies in the real (or simulated) world. Complex patterns of behavior can arise from a collection of very simple rules [Braitenberg, 1984]. Such behavior is called emergent [Watts, 1998], and results largely from the fact that even a simple system in a complex environment can

behave unpredictably. Because of this unpredictability, the animat offers a nice empirical alternative to test the dynamics of these systems in the real world. Several mechanisms have typically been used to control an animat including artificial neural networks and genetic algorithms [see Watts, 1998 for review].

Over the past decade some prosthetic type robots or simulated embodiments have been developed using live animals or even humans. In one case, a “*ratrobot*” [Nicolelis, 2002] was developed, whereby the researchers could give a rat hints through electrical stimulation of particular brain regions which corresponded to a “recommended direction” for the animal to turn while it tried to complete a maze [Talwar, 2002]. In several other studies, robots are controlled by recording from multiple electrodes in the motor cortex; these studies generally attempt to mimic the animal’s behavior (i.e. arm movement) in a robotic form [Laubach et al., 2000; Wessberg et al., 2000; Chapin et al., 1999]. One may note that in these studies, information in the subject’s brain has already been processed substantially, and the motor cortex is typically a short step away from muscle control. Thus these breakthroughs represent developments in our understanding of how information is decoded from the brain to control muscle movement, but may not shed light on neural computation and information processing.

In the prosthetic-type robots described above, we generally only have monitoring abilities near the output level of the information flow in the brain. The central question remains, how does the brain actually process information? What does this information look like as it is being processed? Recently living neurons in-vitro culture have been used to control animats [DeMarse et al., 2001]. However, the behavior of the animat under such control was arbitrary, and not related to any known computation of input stimuli.

CHAPTER 4

Neurally Controlled Animat Problem

In order to investigate how living neural networks compute, this project entailed the construction of an artificial animal, which was controlled by a living neural network.

The goals of the project are to answer the following questions:

- Are cultured neuronal networks capable of demonstrating non-linear computation?
- Can such computation be used to control a (simulated) robot to handle a useful task in real time?
- If so, can such a hybrid neuro-robotic system be used to perform reliable behavioral studies, which can be used as a framework to study plasticity (learning) in cultured neural networks?

Computation in the brain is more powerful, reliable / redundant and adaptable than computation in standard computers. In fact, many researchers are attempting to harness that computational power by building silicon chips that emulate neurons, which they hope some day may provide more powerful personal computers. However, this can only truly become reality when we understand how information is encoded and processed by these neurons. As stated in section three, there are several unproven hypotheses about how neurons encode and process information. My goal is to encode information by varying inputs by a given time-delay, and to show a non-linear reaction to this

information. Interpreting this reaction as a computation, I will show how to emulate any logic gate. If all logic gates can be simulated, then we will know that in theory cultured neurons have the ability to emulate a digital computer with polynomial delay [Sipser, 1997]. Though it is seemingly obvious that brain tissue should have computational abilities, as shown in section three, this has been difficult to demonstrate empirically. Thus even theoretical emulation of simple digital logic is an important first step in understanding how to harness the computational ability of the brain.

As discussed in chapter 3, the brain really does not solve problems by executing programs using Boolean logic circuitry. Instead, information is processed in a highly distributed, parallel manner, which is highly statistical. The inherent statistical nature of computation in the brain gives tolerance for error. In the scheme of things, emulation of a single logic gate is not very interesting because it is not biologically realistic. Thus, in addition to emulating simple digital logic, I will attempt to “solve” a more interesting problem through neural computation. I will construct a neurally controlled simulated robot which will approach an object, and maintain a certain distance away from it.

The simulated robot body and virtual world is implemented in software, and software is also used to control the hardware for neural stimulation and recording. Linear mappings are used to convert sensory information (location of a reference object) to neural input (electrical stimulation). The neurons react non-linearly to this stimulation. This reaction is then linearly mapped to an output behavior (robotic movement). No sensory information (input) is directly passed to the software modules coding for robotic movement (output). Thus, most of the interesting (non-linear) processing in this system will be done by living neurons.

The animat constructed in this project is the first successful neurally controlled animat to solve a “real world” problem. In chapter three I discussed prosthetic type robots that implement an animal’s desired behavior. This project differs in that the animat constructed attempts to solve a specific problem with no control by an animal (or software). Living neurons are utilized as a “computational element”, kind of an external processor.

The environment of the animat constructed is simple, but could easily be made more complex in future experiments. As shown in section three, an animat with simple rules can demonstrate complicated behavior when placed in a complicated environment. Given this property, the animat constructed in this project may have potential for demonstrating complex behavior. The animat may also have potential for exhibiting adaptation and change over time based on input stimulation. Such change is worth exploring and provides a new approach to studying topics of interest to neuroscientists.

A significant amount of research in neuroscience has focused on studying the role of synapses, and synaptic plasticity. Synaptic plasticity is thought to represent memory and adaptation of the brain to input information. If the robot described above is successful in handling its task, then it could potentially be used as a framework for studying plasticity. Behavioral experiments, similar to those performed by psychologists / neuroscientists on mice or other animals, could instead be performed with a neurally controlled robot. This can provide a tool for researchers to use techniques in psychology (a top-down approach) on a system where we have control of variables from the bottom level.

Ideally, any change in neuronal behavior would be expressed as a change in robotic behavior. Following a behavioral framework, the researcher could get a “baseline” representation of robotic behavior. Then neural behavior could be changed, for example by inducing synaptic plasticity, or adding a drug, or creating a lesion. The effect on neural behavior might then show up as a change in robotic behavior as compared to baseline behavior. When we learn how to manipulate the robotic behavior, we could potentially control the behavior to handle more difficult tasks.

CHAPTER 5

Effect of dual channel probing on neural activity

Though no one knows how neurons encode information, it is speculated that timing between inputs plays a key role for neural computation. Prior work in cultured neural networks usually involved probing on a single channel, i.e. stimulating on some channel, and then measuring the results of that stimulation on the rest of the network. To integrate the concept of time coding, I test the effect of probing on two channels with a given time delay, or Inter-Probe Interval (IPI), between the two probes. This IPI probing schema contains at least 3 units of information: which two of the 60 channels are to be probed, the sequence of probing, and the time delay between these probes. In this experiment I attempt to discern how the neural network reacts to IPI probing of varying IPI. The goal of this experiment is to produce an input/output neural mapping function which may represent some sort of neural computation.

5.1 Background

Dissociated neurons begin forming connections within a few hours in culture, and within a few days establish an elaborate and spontaneously active living neural network. After one month in culture, development of these networks become relatively stable and is characterized by spontaneous bursts of activity.

Effect of single channel probing:

In response to short application of an electric current to a single electrode (a probe), the neural dish typically responds with spikes detected over the entire MEA. Stimulation on a single channel initially (within 15 ms) produces precisely timed spikes as measured on individual channels (different channels than the channel used for stimulation). For example, if we apply a short input pulse, or probe, to channel A, we may measure three or four spikes on channel B that occur within 15 ms. If we repeat this stimulation many times, we can compare each trial and build a histogram representing the statistical likelihood that a spike following a probe will fall within a given time bin. For an example, see Figure 4; one can notice three precisely timed spikes occurring at ~ 2.5 ms, 2.7 ms, and 5.0 ms post-stimulation.

The short lasting precisely timed activity gives way to longer lasting “bursting” activity with little or no precision. These burst spikes can fall anywhere within a given time window following the probe, and their histogram appears more smoothed out. Thus I often break down the response of a probe into a precisely timed component, as depicted by histograms of repeated trials on individual channels lasting between 0 – 20 ms, and a “bursting” component of non-precise spikes, typically averaged over all 60 channels, lasting 20 – 100+ ms. A sample histogram showing number of spikes out of 75 trials total that fell into .2 ms time bins following a probe is shown below:

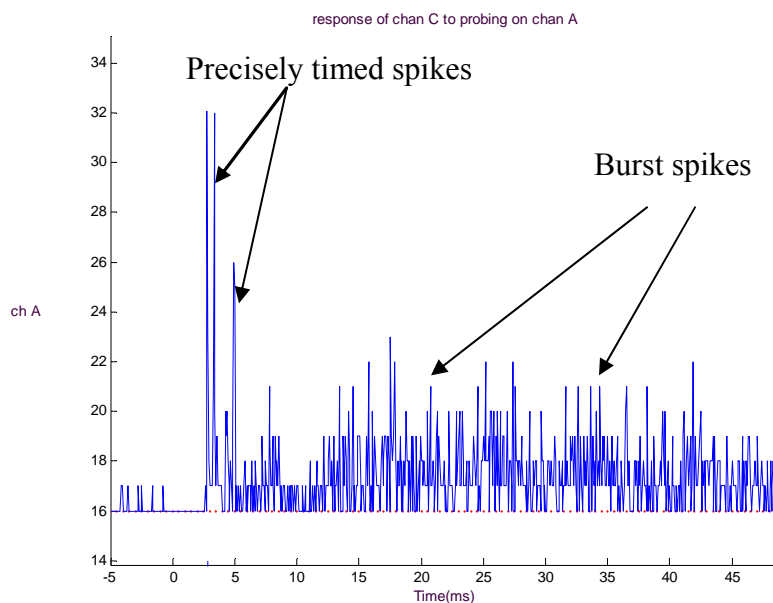


Figure 4: Histogram of response of channel C to probing on channel A (75 trials)

One should note that “bursting” activity occurs spontaneously in cultured neural networks, and to perhaps a lesser extent in in-vivo brain recordings. The nature of this bursting is not known. In some cultures, the bursting rate is fast ($\sim 1\text{hz}$), and in some it is slow or even not present. Though synchronized activity is present in-vivo, the activity in cultures is sometimes likened to in-vivo epileptic recordings. I point this out because many of the properties that arise as a result of IPI probing may stem from bursting dynamics within cultured dishes.

5.2 Method

Cultured cortical neurons from E-18 rats were dissociated and cultured on 60 channel multi-electrode arrays (MEAs) from MultiChannel Systems (see Figure 5). Each channel (electrode) provides a means for recording electrical activity within its vicinity, which may include several neurons (see Figure 6). These electrodes also provide a means

for electrical stimulation of neurons by passing a current through nearby neuronal membranes.

Two channels for stimulation were selected (from 60 available channels) by probing on various active channels until two were found that each produce a moderate effect in the dish. A probe consists of a +/- 600mV, 400 uS biphasic pulse applied to a given channel [DeMarse et al., 2001; Jimbo, 1999]. Once selected, the two channels were probed with a varying IPI ranging over +/-750 ms every 3 seconds. A total of 29 data points (of varying IPI) were collected serially in this way, and these 29 IPI probes were repeated over 75 iterations. This experiment was repeated on two dishes, using differing sets of electrodes in each dish. These two dishes came from the same batch of plated neurons, however later experiments use different sets of dishes.

The initial experiment utilized an 8 channel STG stimulator from Multichannel Systems for electrical stimulation. The nature of this stimulation hardware requires manual programming, and could not be used for randomized stimulation patterns. This is why 29 data points were serially probed over and over again. Later, Dr. Thomas DeMarse (formerly of the Potter lab) developed a 64 Channel Neural Stimulator (64CNS) board, with which one could dynamically stimulate the neurons. This hardware was used in the animat experiment described in Chapter 7. A neural response curve for setting animat parameters was obtained by dynamically probing with random IPI. This data is also presented here as an alternative as it demonstrates the same effect.

Neural data was recorded through an amplifier provided by Multichannel systems, and interfaced with a computer in the lab. Meabench software developed by the Potter group was used to manipulate and save raw data. Given the external nature of the

electrodes, the electrical signals, both in stimulating and in recording, deteriorate quickly. Thus to produce action potentials, electrodes are stimulated with $\sim 10^4$ times more voltage than is actually picked up by spiking neurons. Because of this difference, an electrical artifact is present following stimulation. An artifact suppression software utility developed in the Potter lab [Wagennar and Potter, 2002] was used to remove this stimulation artifact, and also to remove any 60hz noise that may be present. A spike detector software utility, trained to ignore electrical noise on individual channels, picked up electrical activity that broke a threshold ($5.5 \times \text{RMS}$), and produced a stream of spike data which was recorded to a file and later used for data analysis.



Figure 5: MEA Dish

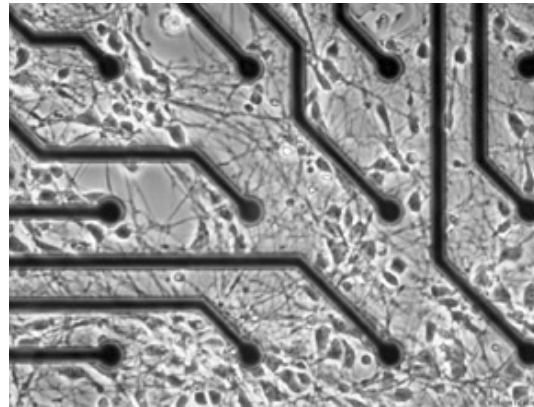


Figure 6: Blowup of MEA dish with cultured neurons, showing wires and electrodes used for both recording and stimulation

5.3 Data Analysis

Spike data from the experiment was analyzed in Matlab utilizing the scripts in appendix A to determine the effect of IPI probing. Data was imported and stored in matlab data structures from raw spike files by utilizing 'getSpikeData.m'. Averaged

activity was measured in two ways: either by examining X ms following the second probe (sometimes referred to as a “Post IPI Response curve”), or examining the duration of the probe + X ms following the second probe (sometimes referred to as “Full IPI Response curve”). Both of these charts are plotted by compMeans.m. This matlab function also prints a corresponding color coded spike intensity graph representing the number of trials that fall into a given bin on the Y axis (Spikes / ms / Channel).

I further analyzed the data by creating raster plots of activity on individual channels by using plotrastall.m, and corresponding histograms by using plot_hist.m.

5.4 Results

Full IPI Response Curve:

Sample computed IPI response curves showing the averaged effects of probing on channels A and B with a given IPI are shown below (Figure 7 - Figure 13). Figure 7 and Figure 8 show the full-IPI response curve, where the response is measured as spikes / channel over a period of time equal to the given IPI + 100 ms. This curve is shaped as a ‘V’. With higher IPI values, the ‘V’ shaped curve eventually tops off and has a minimal rising slope (this is best illustrated in Figure 8).

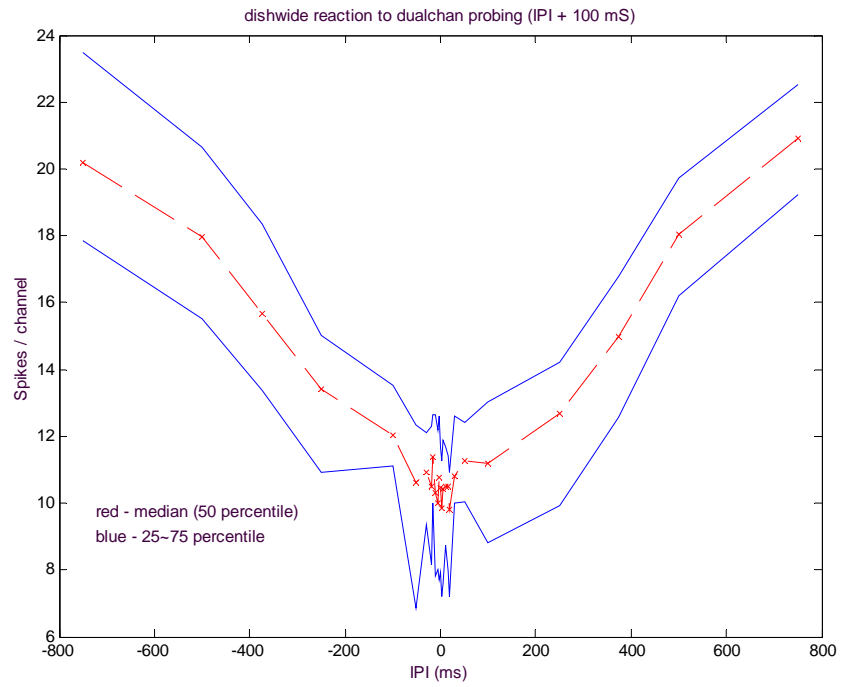


Figure 7: Full IPI Response curve (dish 4819, ch 38 and 25)

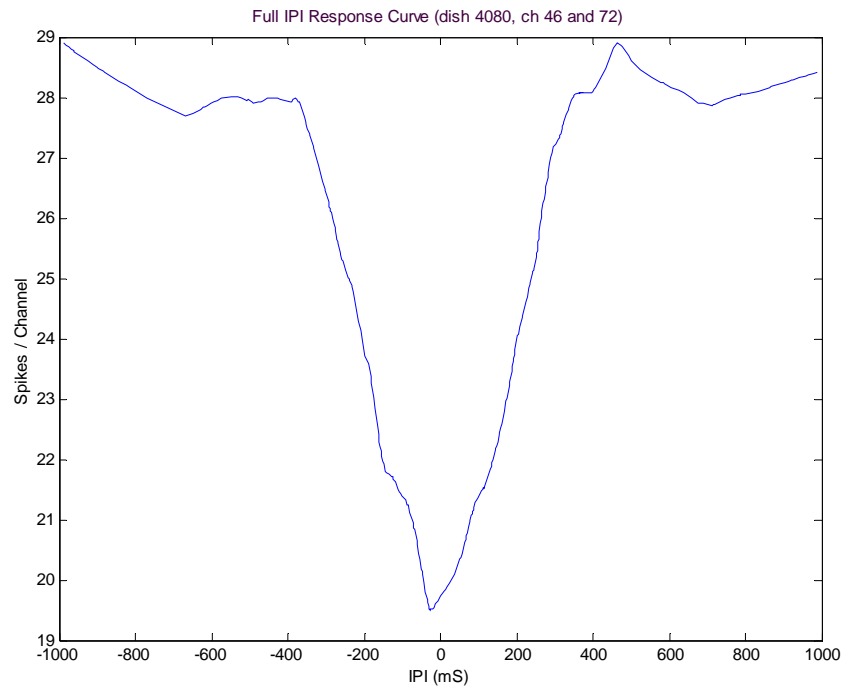


Figure 8: FULL IPI response curve, data from experiment in chapter 7

This curve implies that the bursting behavior of the dish has a relative refractory period, where a “full” reaction to an IPI probing pair is reduced monotonically as IPI is reduced. The source of this refractory effect is unknown, and may be the result of changing ionic concentrations (e.g. calcium), depleted neurotransmitters (presynaptic), or habituation (postsynaptic). This effect is characterized by the schematic shown in Figure 9.

Interestingly, if we probe with IPI very close to zero ($IPI < 30$ ms), it appears that the effect of a second probe is not as strong as it may be alone, but there does appear to be something similar to paired-pulse facilitation (PPF) occurring, as the effect of the two probes together seems to induce a longer lasting burst than a probe on a single channel. This is depicted in a histogram of real data in Figure 10. When probing two channels at the same time, the initial time bins go up slightly (indicating some possible effect of stimulating a different set of neurons initially), but the effect seems to last much longer (indicating facilitation).

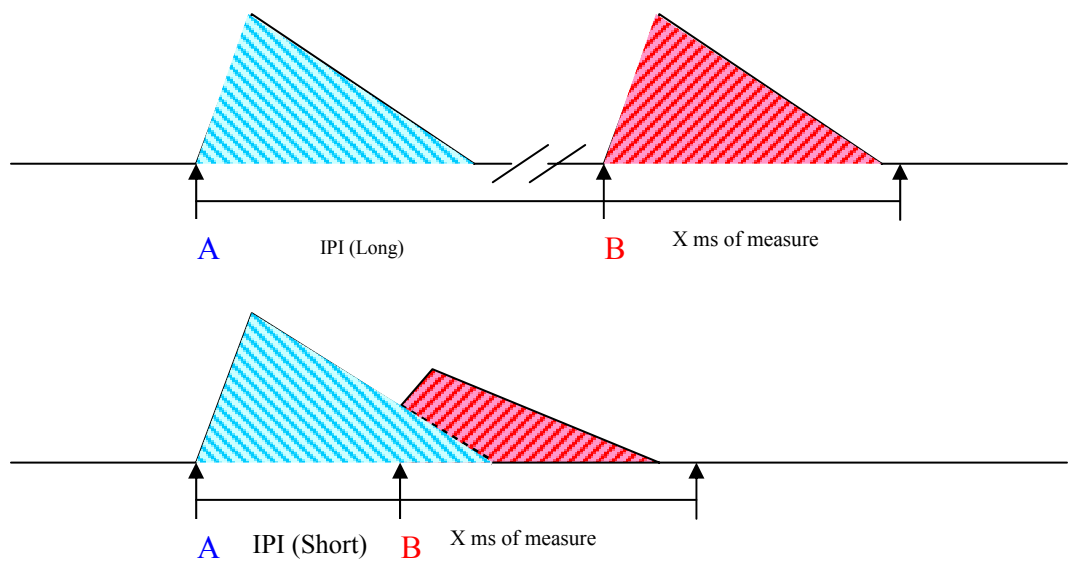


Figure 9: effect of two probes with long IPI (top) or short IPI (bottom)

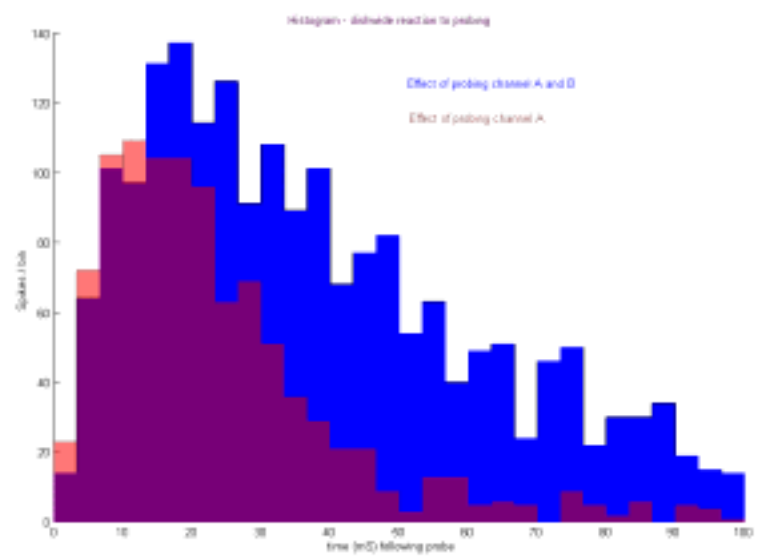


Figure 10: effect of probing two channels at once (blue) vs. one channel alone (red)

This figure compares a single probe (red) against two probes at the same time (blue). The overlapping region is depicted in maroon. When probing two channels, one can see a slight increase in the peak of the histogram, indicating that perhaps a slightly different set of neurons are activated by the two probes; however, the fact that the burst seems to last much longer (the decay slope is less steep), it also seems that when probing on two channels very close to each there is potentiation or something similar to paired-pulse facilitation.

The major data points of the Full reaction to IPI probing activity may be characterized approximately as follows:

$$1.1) \lim_{IPI \rightarrow 0} Full(IPI) \approx \text{Max}(probe(A) + probe(B - A), probe(B) + probe(A - B))$$

(where $probe(x)$ represents the number of spikes per channel following a probe on only channel x ; and $probe(x-y)$ represents the net number of spikes per channel resulting from probing x directly following a probe on y)

Assuming $probe(A)$ is approximately equal to $probe(B)$, we can approximate:

$$1.2) \lim_{IPI \rightarrow 0} Full(IPI) \approx probe(A) + probe(B - A)$$

Further:

$$2) \lim_{IPI \rightarrow \infty} Full(IPI) \approx probe(A) + probe(B) \approx 2 * probe(A)$$

Post IPI Response Curve:

In Figure 11 and Figure 12 we ignore the response directly following the first probe, and measure how the first probe effects the reaction of the second probe. Instead of estimating a sum of spikes as done above, in this case a firing rate is obtained. Given the information above, one might expect a single ‘U’ curve, rather than a double ‘U’ curve. The peak seen near zero IPI in Figure 11 partially represents a residual effect of probing A (we can not ignore the effect of A if our IPI is close to zero because A happened so shortly before). However, as shown in Figure 13, the peak at zero can be twice as high as stimulating a single channel alone (represented by $IPI \rightarrow \infty$). Thus, when stimulating with short IPI, groups of neurons may experience paired-pulse facilitation, thus prolonging the effect of the burst. However, the MIN seen at ~ 100 IPI is significantly less than the effect of probing on a single channel. As mentioned above, this may be the effect of a bursting refractory period leading to depression on the second pulse of an IPI pair.

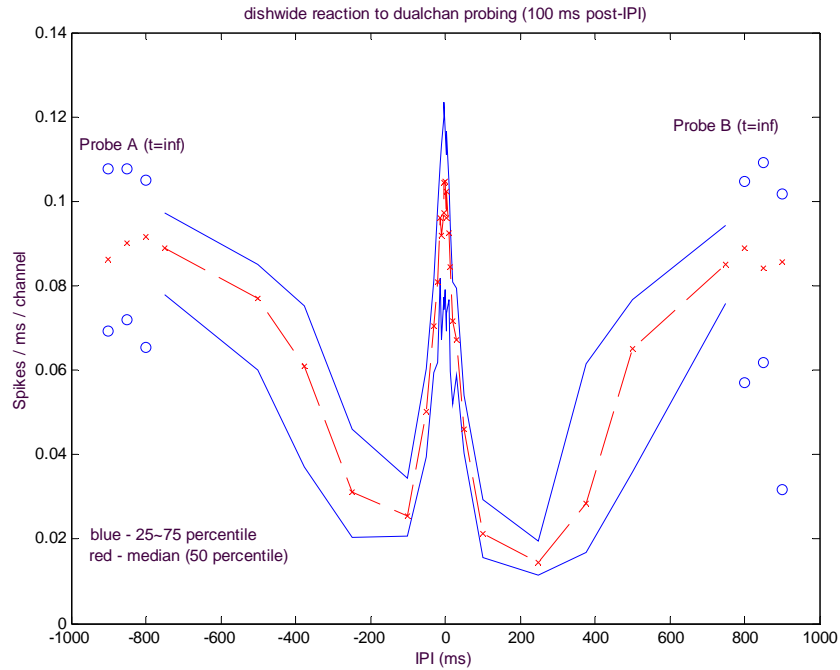


Figure 11: Post-stim IPI Response curve (dish 4819, ch 38 and 25);

29 data points (varied IPI) are probed serially. This is repeated over 75 trials. The median value at each IPI is shown in red, and the middle 50% of trials fall between the blue lines. The separate points at the far right and left indicate data probes of A alone, or B alone (which can be thought of as $IPI = \pm \infty$). One can see that when the IPI pair pulse is $> \sim 30$ ms, the effect of a second probe is decreased, indicating something similar to paired-pulse depression.

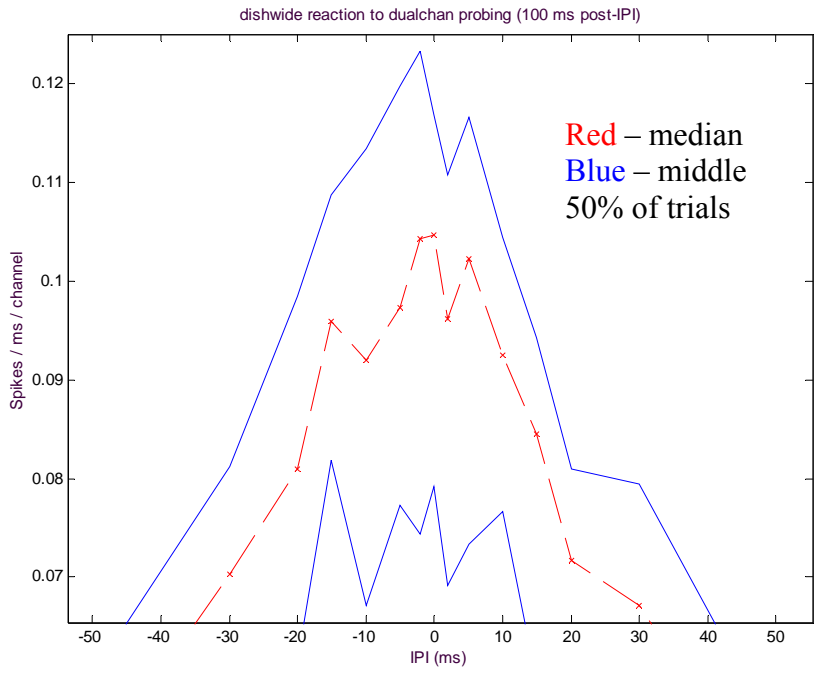


Figure 12: (blow up of Figure 11)

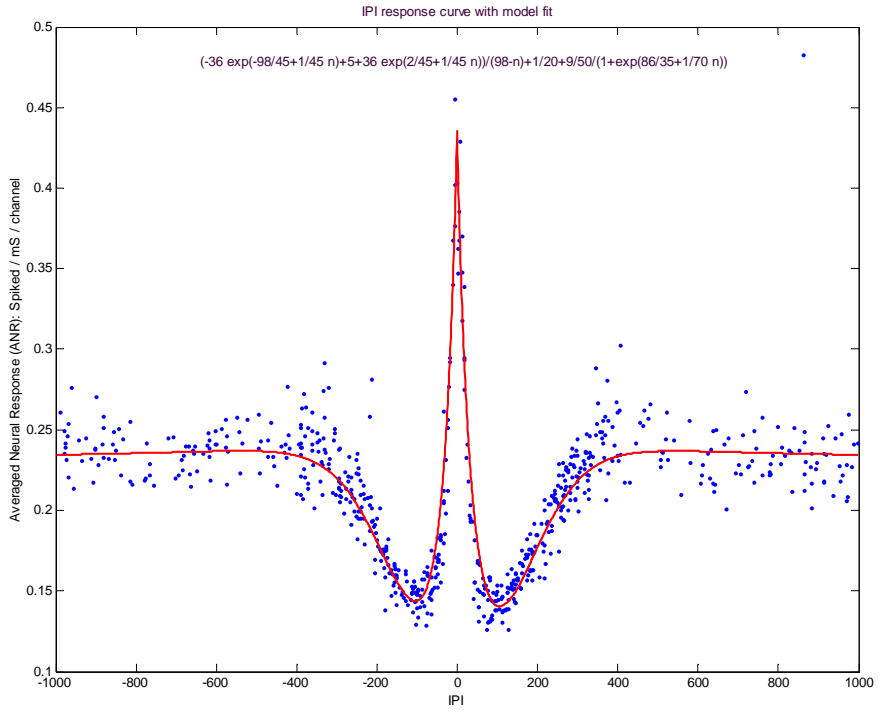


Figure 13: Post IPI response curve, data from experiment in chapter 7

A raster plot of random IPI probes shown in blue; this figure very clearly shows facilitation when IPI < 30ms, and depression when IPI > 30ms.

Also, there may be a slight local peak at $\sim \pm 400$ IPI. This peak may represent a “burst reset”: After probing on the first channel, spontaneous activity is depressed for the first 400ms. However, this depressive effect wears off within 400ms, and by this time the neural network is well “rested”, thus a slightly increased reaction to the second probe at 400ms ensues.

The major data points in the Post IPI response curve can be characterized as follows:

$$3) \lim_{IPI \rightarrow 0} Post(IPI) \approx \text{Max}((pr(A) + pr(B - A)), (pr(B) + pr(A - B))) \approx pr(A) + pr(B - A)$$

(Where $pr(x)$ is a rate of firing following a probe on x alone, and $pr(x-y)$ is the net firing rate of probing channel x directly following a probe on channel y .)

At $IPI = 0$, the firing rate is \sim equal to probing A alone plus the net effect of probing B given a simultaneous probe on A

and thus It follows that:

$$4) \text{Min}(Post(IPI)) \approx \text{Min}(pr(B - A), pr(A - B)) \approx pr(B - A)$$

and:

$$5) \lim_{IPI \rightarrow \infty} (Post(IPI)) \approx \text{Max}(pr(A), pr(B)) \approx pr(A)$$

Reaction of individual channels to IPI probing:

Now that I have determined what is going on at a macro (averaged) level, can I say anything about activity on individual channels following IPI probing? Figure 14 shows a raster plot of activity on individual channels following 75 trials of an IPI probe of 250 ms. In this figure, at each channel, the 75 trials are shown “stacked” on top of each other.

The figure demonstrates several noteworthy properties of IPI probing. If one looks slightly past time = 0 (probe A) or time = 250 (probe B), one may notice that vertical lines appear at several channels. This indicates “precisely timed spikes” that reoccur with nearly each trial. Following 15 ms after a probe (in this case Time > 15 &

Time > 265) we no longer see any precisely timed spikes, and instead see a blur of dots indicating non-precise bursting behavior following a probe. On several channels (i.e. 1,10,13,16, etc.), the precisely timed spikes following a probe on channel A (at Time = 0) occur at clearly different latencies than the precisely timed spikes following a probe on channel B (at Time = 250).

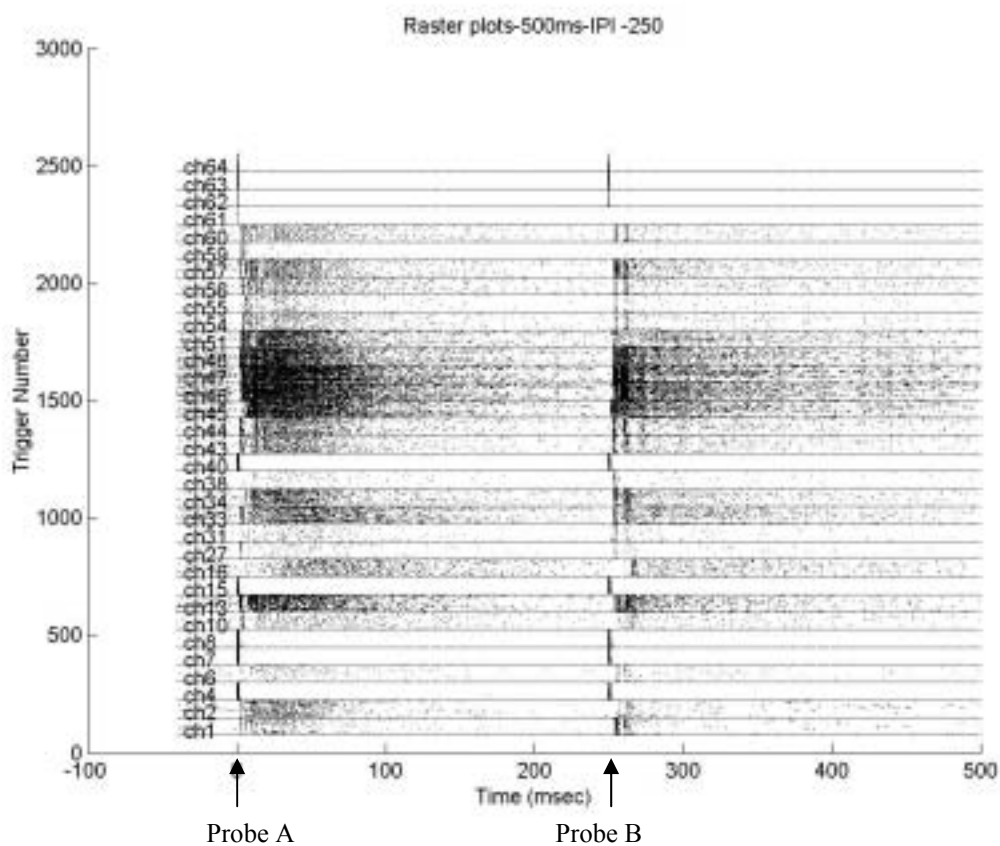


Figure 14: Raster plot depicting spikes on given channels following 75 trials of probing at IPI = 250 ms

Each row depicts the response of a single channel to a given probe; data from 75 trials are “stacked” from bottom up to produce the above graph. One can see vertical lines on some channels shortly after a probe; these lines are “precisely timed spikes” that reoccur multiple times across the 75 trials.

Additionally, if one compares the bursting behavior following a probe on channel B (at 250 ms after the probe on A) to the bursting following the initial probe on channel A, the spike density is notably less. This is yet another demonstration reflecting the IPI response curves shown above.

To sum up, two effects are noticed here:

- 1) One can see differing latencies, or appearance/disappearance of precisely timed spikes depending on which channel is probed
- 2) Decreasing spike density of bursting is observed following the second probe of an IPI pair

The effects described above are consistent for any given IPI. I already described above how the bursting behavior changes with varied IPI. To examine how precisely timed spikes may change over time, one may examine histograms of spike data on a given channel following varied IPI probes. This is illustrated in an example histogram in Figure 15; Across all such histograms, it is apparent that precisely timed spikes remain relatively consistent across varying IPI, and seem to override most preceding behavior, particularly when $IPI > 20$ ms.

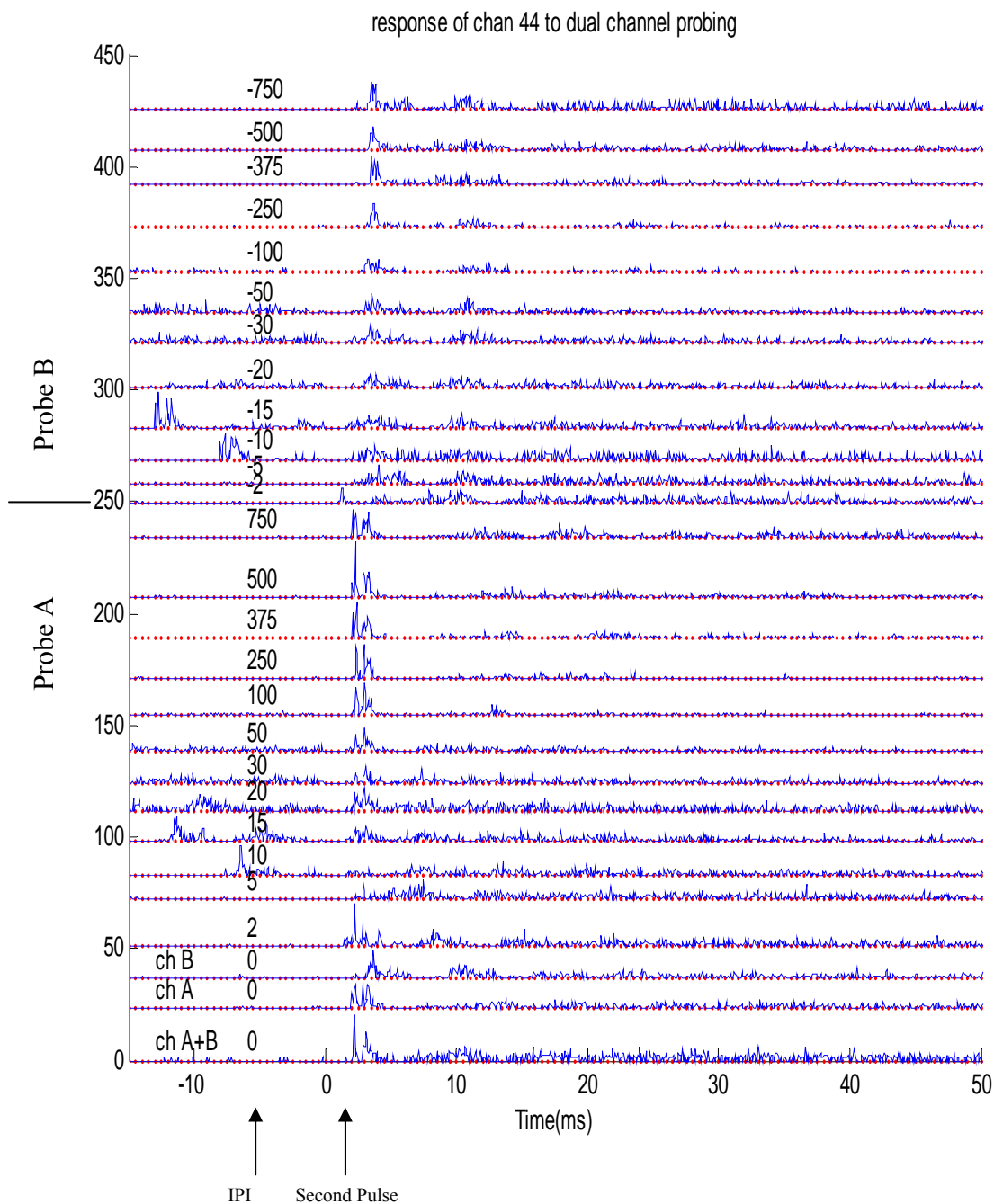


Figure 15: Histogram depicting response of channel 44 to the second probe of an IPI pair, Dish 4819, 1/13/03;

Note that each row represents a histogram that is lined up so that the second probe of an IPI pair occurs at time 0. The bottom half of the chart contains probes with positive IPI, thus the response is similar to probing A alone; the top half has probes with negative IPI, thus the resemblance to probing B alone. Generally speaking, only probe order effects the precisely timed spikes (peaks < 15ms), whereas IPI duration effects bursting (activity > 15ms).

CHAPTER 6

Demonstrating the computational power of the IPI neural effect

Certain nonlinear dynamics are represented by the IPI curves (described in chapter 5), which may be interpreted as some sort of computation. In this chapter I present a thought experiment to show that cultured neural networks, using the IPI effect, can emulate any logic gate, and therefore can emulate a standard digital computer with polynomial slow-down. In the next chapter, I will present a more interesting application of the IPI effect, to show how it can be used to control a robot to handle an AI task in real-time.

6.1 Schema for emulation of digital logic gates

A sample response of a living neural network to two probes with varying time delay between them is shown in Figure 16. One can think of this response as a sort of computation. Given an input (IPI) the network produces an output (averaged firing rate). Thus, the neurons act as a type of analog gate. By simple mappings, this response curve can emulate any and all digital gates.

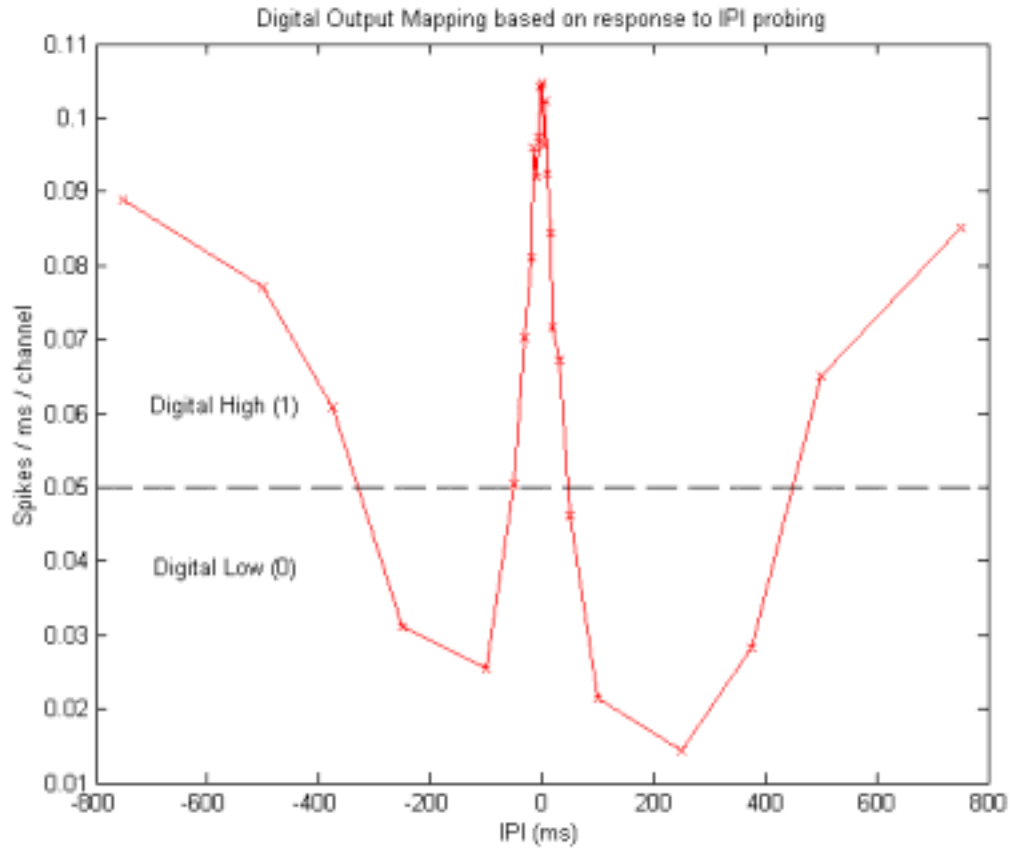


Figure 16: Post IPI response curve

Mappings:

Here I present sample mappings from logic inputs (consisting of high 1 and low 0) to time delay in reference to a clock tick. The clock tick is simply a marker in time to help calculate IPI based on latency to stimulations around the clock tick. One can probe the neural network with that given IPI, inducing a predictable reaction in the network based on the IPI curve shown in Figure 16. The averaged neural response is mapped to a logical output as shown below in Table 1.

Table 1: Digital Output Mapping: averaged neural response to a set of probes → digital output

<.05 spikes/ms/channel → logic output 0
 >.05 spikes/ms/channel → logic output 1

The mapping of logic inputs to IPI varies depending on the type of gate one attempts to emulate. When emulating a gate with two logic inputs, P and Q, the values on each logic input are converted to a delay from the tick of a clock to create an IPI probe pair. In the below mappings, logical input P maps to a stimulation delay from clock on channel 1, and logical input Q maps to a stimulation delay from clock on channel 2.

For example, a mapping of (P=1) → +250 means that if our logic input P is high (1), we will stimulate channel 1 at clock+250 ms. Similarly, if given a mapping of (Q=1) → -500, this means that if our logic input Q is high (1), we will stimulate channel 2 at clock-500 ms. Thus, if our two digital inputs, P and Q are both high (1), then we will stimulate channel 1 and 2 with an IPI of $250 - (-500) = 750$ ms, which according to the IPI response curve and Table 1, results in a logic output of 1.

The following tables represent mappings of inputs (P,Q) to IPI, and the resulting truth tables:

6.2 Input mappings for emulation of a NOT gate

(note that a NOT gate has only one logic input. Ch2 is always stimulated at clock+0 ms.)

Table 2: Logical input \rightarrow probe delay, mapping for NOT gate emulation

P \rightarrow delay Ch1	Q \rightarrow delay Ch2
1 \rightarrow 250	* \rightarrow 0
0 \rightarrow 750	* \rightarrow 0

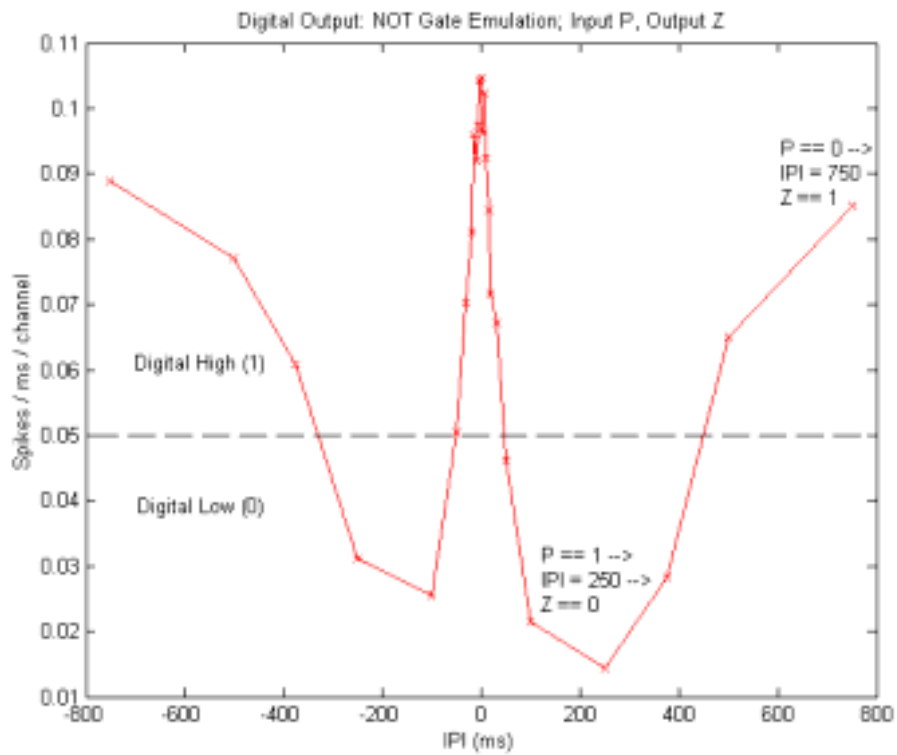


Table 3: NOT gate emulation truth table

Logic Input		IPI probe input			Digital Output
P	Q	Delay Ch1	Delay Ch2	IPI = delay(Ch1)-delay(Ch2)	Z
1	*	250	0	250	0
0	*	750	0	750	1

6.3 Emulation of an AND or NOR gate

(a NOR gate is obtained if input mappings are reversed) :

Table 4: Logical Input \rightarrow probe delay mapping for AND gate emulation

P \rightarrow delay Ch1	Q \rightarrow delay Ch2
1 \rightarrow 500	1 \rightarrow -250
0 \rightarrow 0	0 \rightarrow 250

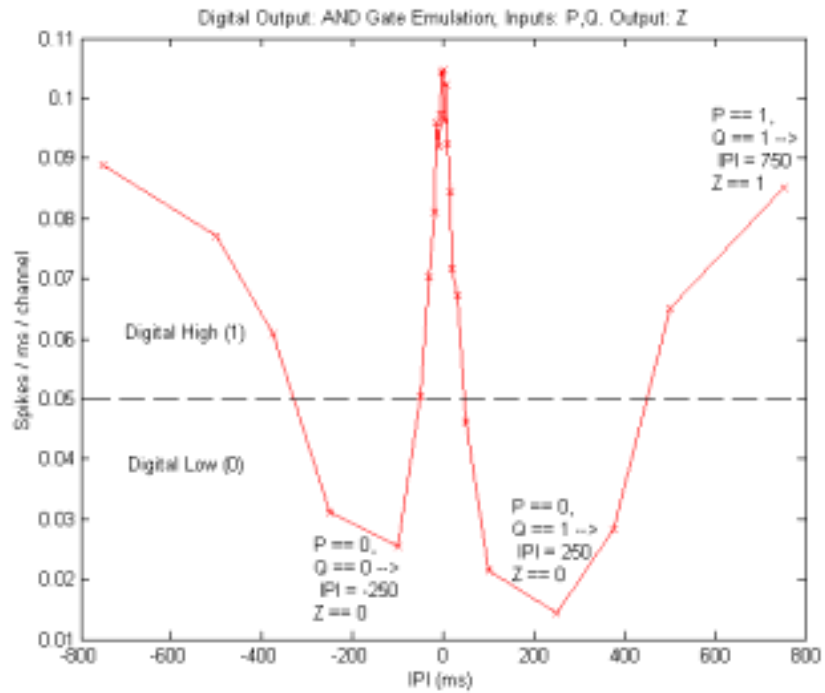


Table 5: AND gate emulation truth table

Logic Input		IPI probe input			Digital Output
P	Q	Ch1	Ch2	IPI= (Ch1-Ch2)	Z
1	1	500	-250	750	1
1	0	500	250	250	0
0	1	0	-250	250	0
0	0	0	250	-250	0

6.4 Emulation of an OR / NAND gate

(a NAND gate is obtained if input mappings are reversed) :

Table 6: Logical Input → probe delay mapping for OR gate emulation

P → delay Ch1	Q → delay Ch2
1 → 500	1 → -500
0 → 250	0 → 500

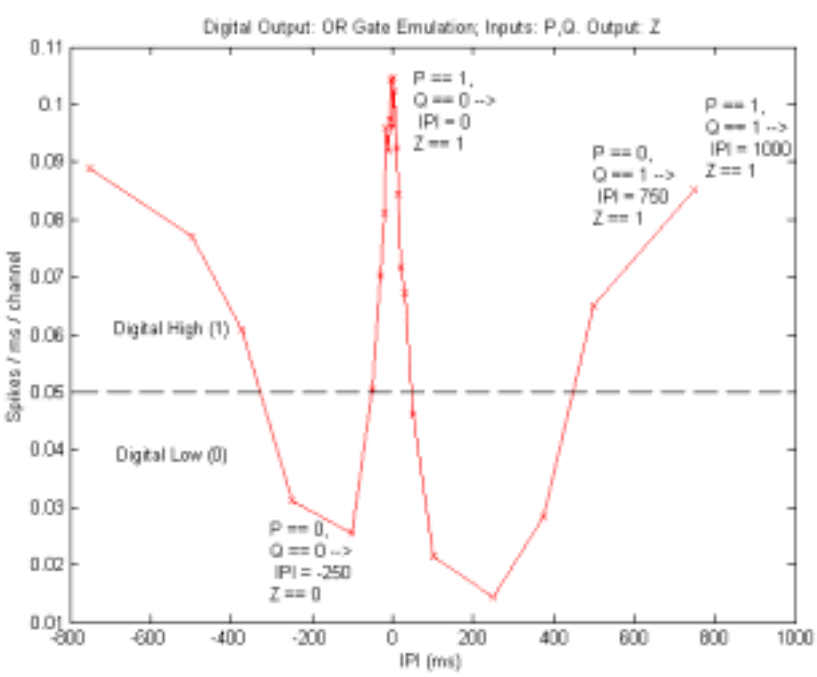


Table 7: OR gate emulation truth table

Logic Input		IPI probe input			Digital Output
P	Q	Ch1	Ch2	IPI = (Ch1-Ch2)	Z
1	1	500	-500	1000	1
1	0	500	500	0	1
0	1	250	-500	750	1
0	0	250	500	-250	0

6.5 To emulate an XOR gate:

Table 8: Logical Input → probe delay mapping for XOR gate emulation

P → delay Ch1	Q → delay Ch2
1 → -250	1 → -500
0 → 250	0 → 500

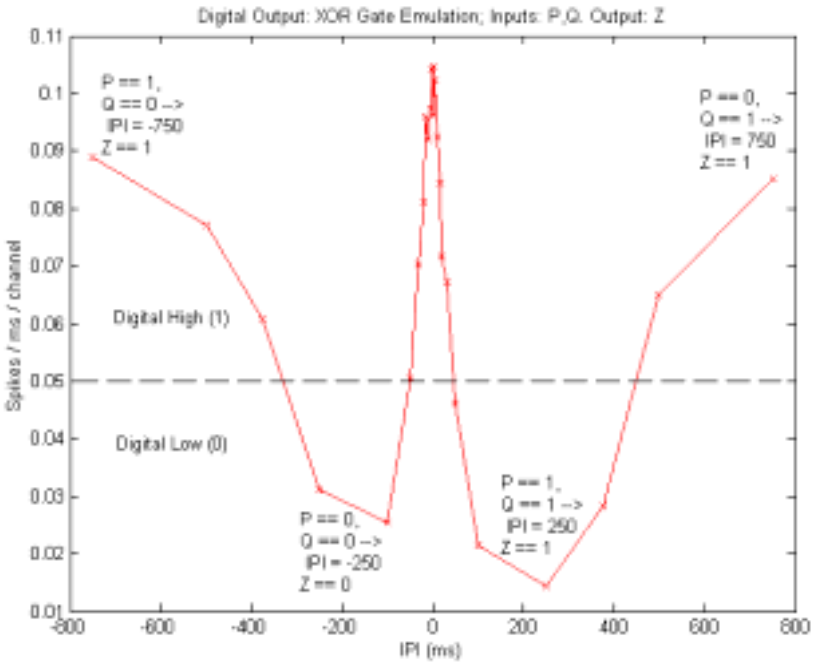


Table 9: XOR gate emulation truth table

Logic Input		IPI probe input			Digital Output
P	Q	Ch1	Ch2	IPI = (Ch1-Ch2)	Z
1	1	-250	-500	250	0
1	0	-250	500	-750	1
0	1	250	-500	750	1
0	0	250	500	-250	0

6.6 Results

As shown by the sample truth tables above, neurons may perform the computation of any logic gate. In chapter three I reviewed prior work which weakly demonstrated that a set of neurons could emulate a NOR gate, from which any other gate could be indirectly constructed. In this chapter I showed how a cultured network of neurons can directly emulate any binary logic gate. Gates with more than 2 inputs can be made by combining several binary gates, thus (at least in theory) neurons are capable of demonstrating Boolean logic.

Since PC's are made up of nothing more than combinations of logic gates, the results of this chapter indicate that cultured neural networks can emulate a digital computer, and execute any computer program with polynomial slow down over standard computers [Sipser, 1997]. The results presented here are important as a preliminary "proof of concept" indicating that cultured neurons can manipulate information to perform computations. Emulation of digital gates in non-silicon substrates goes well beyond study in living neuronal networks. Such emulation has been shown in theoretical chaotic systems [Sinha, Ditto 1999], cellular biocomputing [Simpson et al. 2001], DNA [Lie et al, 2000], proteins [Deonarine et al. 2002], and quantum mechanics [Steane 1999]. It is demonstrated here in neurons simply to point out that cultured neurons in-vitro are capable of robust computation.

As reviewed in chapter 2 and 3, It is not likely that the brain actually computes information using logic gates as computers do. The IPI response curve above, which we can think of as a gate with an analog response, has significantly more interesting

properties than a standard digital gate. In the next chapter I show how the IPI response curve can be used to control a robot to handle a task in real time.

CHAPTER 7

Construction of an animat to track and follow a reference object at a given distance

The most natural way to explore the computational ability of living cultured neurons is to provide some type of embodiment to be controlled by the neurons, and to study the behavior of this embodiment in its environment in real-time. In this experiment, I create a virtual animal, or animat controlled by living neurons. The animat embodiment can provide the neurons with a controlled way of expressing behavior and reacting to sensory stimuli. The animat will be constructed to track and follow a reference object, and maintain a certain distance from that object. Such a task may be compared to a car with “autopilot” that needs to follow the car in front of it at a given distance.

7.1 Method

7.1.1 Neural culture, hardware and software to interface with neurons

See chapter 5 for a description of the living neural network and the hardware and software used to interface with it. The only modification from data collected and analyzed in chapter 5 is that when running this animat experiment new stimulation hardware was made available thanks to Dr. Thomas DeMarse, formerly in the Potter group. The 64 channel neural stimulator (64 CNS) allows one to stimulate any channel of the dish dynamically, meaning that the board could be programmed and executed using C++ code in real-time. Thus using this hardware, instead of probing 29 IPI data points, I

could now probe random IPI points dynamically. The results of this were partially shown in chapter 5, and are also shown below.

7.1.2 Animat

The animat provides a simulated embodiment for our cultured neurons. The Animat, shown in Figure 17, has three numbers associated with it at all times: X,Y coordinates in planar space, and a heading, θ , the direction the animat is currently facing (direction is usually given in radians, $\theta = \text{zero}$ points directly to the right). The virtual world has no actual units of distance, but for convenience I will always label distance as cm. The “arena” size is +/- 100cm in both the X and Y directions. In any following graphs, the animat is shown as a blue triangle with heading theta.

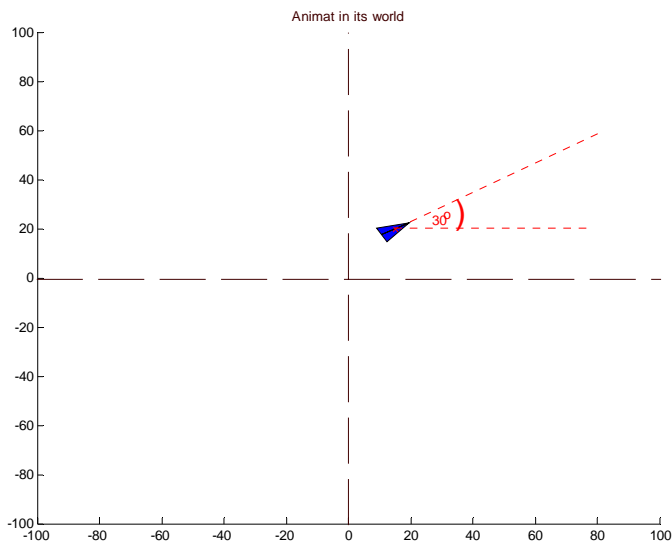


Figure 17: Animat in its world; animat shown has coordinates { $x = 15$; $y = 20$; $\theta = \pi / 6$ }

The reference object has X,Y coordinates in planar space. Depending on the experiment setup, the reference object may be stationary (fixed X,Y coordinates through

the entire experiment or for a given number of trials) or mobile. The reference object is shown in any graphs as a red square.

In the following sections, I present the encoding and decoding schema utilized to control the animat. Encoding refers to mapping sensory information (relative location of the reference object to the animat) to neural stimulation patterns. Decoding refers to mapping neural response to an animat movement. A step refers to a movement at a given point in time. In real-time, a step takes 3 seconds.

Encoding sensory information (distance and direction from robot to object) to neural stimulation (IPI):

The input encoding was relatively simple. Direction was encoded into IPI polarity, which is the sequence of stimulation (i.e. : -1 is stimulation of ch A followed by ch B, and +1 is ch B then ch A), as -1 if the reference object is to the left of the current animat heading, or +1 if the reference object is to the right of the animat heading. Distance from animat to reference object was mapped to an IPI duration based on a linear function with two (non continuous) components, as shown in Figure 18. When stimulating, actual IPI = IPI polarity * IPI duration.

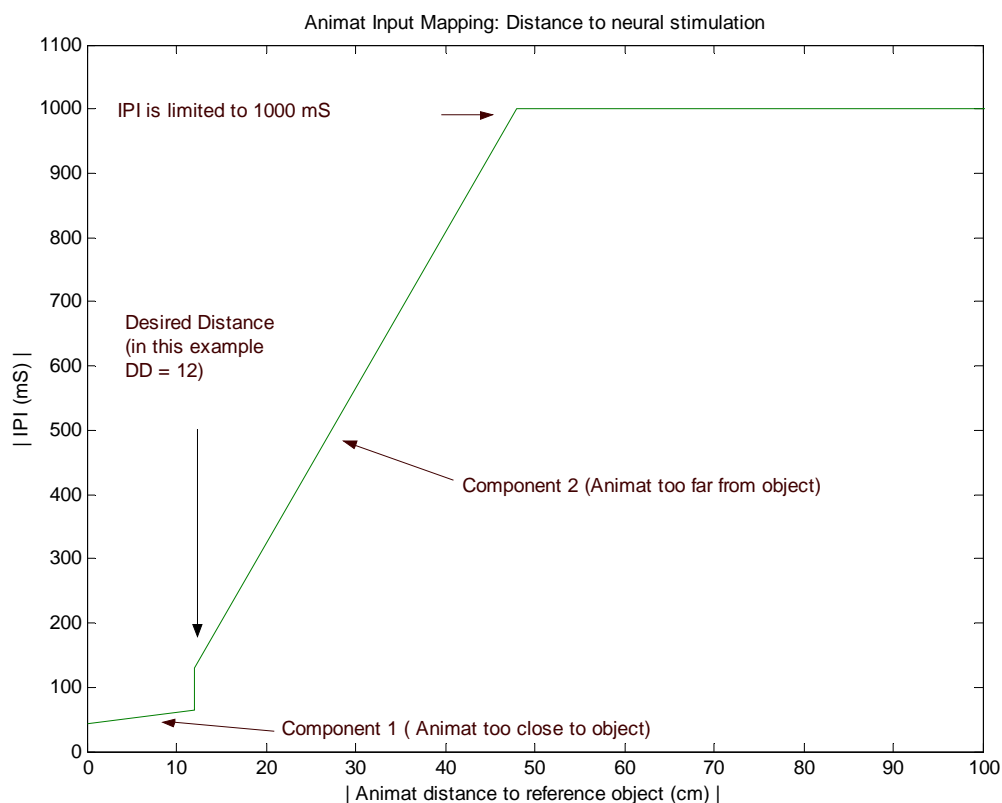


Figure 18: Animat Input Mapping; encoding sensory information (distance of animat to reference object) into duration of IPI pair pulse stimulation

Decoding Neural activity to animat movement:

In chapter 5 I broke down the effect of probing a cultured neural network into two responses: short latency (spikes < 15 ms) “precisely timed” spiking, and longer latency ($15 \text{ ms} < \text{spikes} < 100 \text{ ms}$) “bursting”. I used these two components to decode information from the neural activity following a dual channel IPI probe pair. Neural response was decoded to create a movement vector. As explained below, direction of the movement vector (or next animat’ heading) was decoded based on both precisely timed spikes, and averaged neural activity over the course of the IPI stimulation (utilizing the Full IPI response effect). Directional decoding corresponded to a unit vector pointing in

one of four directions (northeast, southeast, northwest, southwest). The magnitude of the movement vector was decoded from the averaged neural activity following the IPI stimulation (utilizing the Post IPI response effect).

Directional vector decoding:

As described in chapter 5, varying IPI latency has little effect on precisely timed spikes following the second probe of the pair; precisely timed spikes generally reflect the most recent channel to be stimulated. This effect was used to determine the ordering of input stimulation (IPI polarity). As stated in the above section on sensory encoding, the input ordering of an IPI pair corresponded to the relative direction of the reference object to the robot. Thus when decoding the stimulus sequence, if the precisely timed spikes best matched the response we expected from channel B, the animat should turn right, or if they matched A, the animat should turn left. To keep things simple, the animat turned in every step by exactly +30 degrees, or -30 degrees. Although this introduced error into the system (for example, if the animat was facing the reference object, it was forced to turn away from the object), the error averaged out over time. Given the statistical nature of neurons, and even muscle cell contraction, such “error averaging” may be biologically realistic.

I refer to this decoding scheme as a “lock and key approach”. Before running the animat experiment, parameters of the system are tuned. One parameter was a set of histograms reflecting the spikes that occurred within 15 ms on individual channels in response to varying IPI polarity probing. I obtained two sets of histograms, or two “locks”; each lock set corresponded to the given channel being most recently stimulated

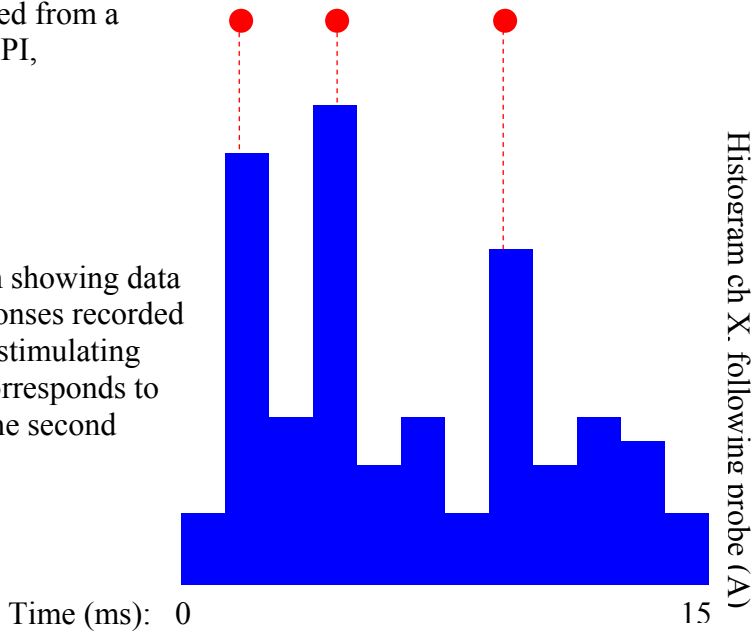
(either A or B). During the actual animat experiment, the neurons were probed with some IPI. The spiking in response to a single IPI probe was treated as a “key”, and typically only matched one of the “locks”. Matching was done by examining each spike following the IPI probe; each spike fell in a given time bin and corresponded to a (normalized) histogram value for a certain channel. The histogram values for all spikes following an IPI probe were squared and summed. Each sum can be considered a score for the given lock. The result was two such scores, one corresponding to likeliness of a probe of channel ‘A’ being most recent, the other corresponding to a probe on ‘B’. The scores were compared and the higher value was selected as the winning lock, which was thus unlocked by the key.

A schematic of the lock and key decoding mechanism is shown in Figure 19. In the schematic, spikes produced following a sample single probe with positive IPI best correspond to the histogram corresponding to positive IPI probing on channel X, which was based on many trials of positive IPI stimulations. 2×60 ($\{+IPI, -IPI\} \times \{\text{channel \#}\}$) such histograms are available for fitting. These histograms were obtained during parameter tuning (described below).

Good fit (key fits lock):

Key: Sample spikes collected from a single probe with **positive** IPI, recorded on channel X.

Lock: Schematic histogram showing data from 75 trials of spike responses recorded on channel X as a result of stimulating with **positive** IPI (time 0 corresponds to stimulation of channel A, the second probe of the IPI pair)



Poor fit (key does NOT fit lock):

Key: Sample spikes (same as above) collected from a single probe with **positive** IPI, recorded on channel X.

Lock: Schematic histogram showing data from 75 trials of spike responses recorded on channel X as a result of stimulating with **negative** IPI (time 0 corresponds to stimulation of channel A, the second probe of the IPI pair)

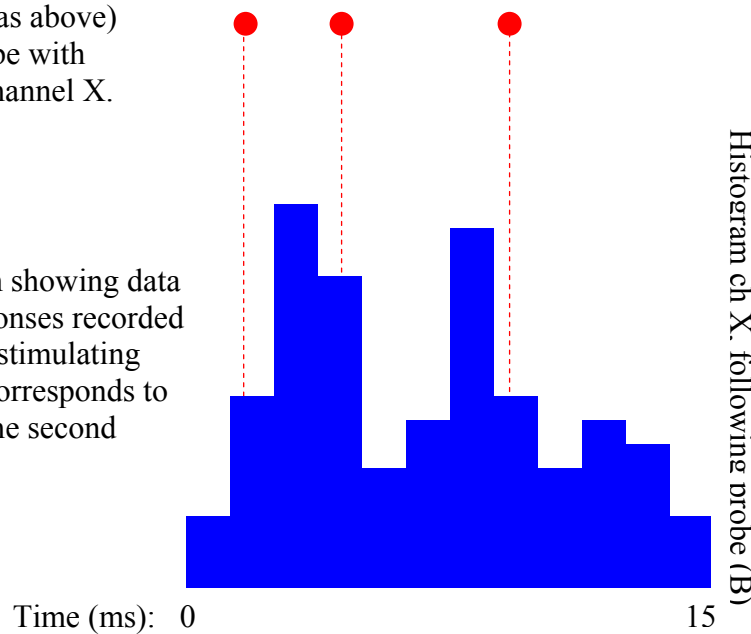


Figure 19: Schematic diagram depicting lock and key decoding

Decoding through the lock and key mechanism allowed the animat to choose a direction, left or right, to move in. The animat must also choose to move forward or backward. The Post-IPI response curve (discussed in chapter 5), which is shaped as a double-‘U’ has two local mins. The IPI at these mins was mapped to the desired distance during parameter tuning (the reason for this is discussed in the next section). These IPI values, which correspond to desired distance between animat and object, were used to set a threshold in the Full-IPI response curve (see Figure 20). In the Full-IPI response curve, neural response was averaged over a period during the IPI probe and 100 ms following the IPI probe. In order to move forward, the Full IPI response must have broken the threshold.

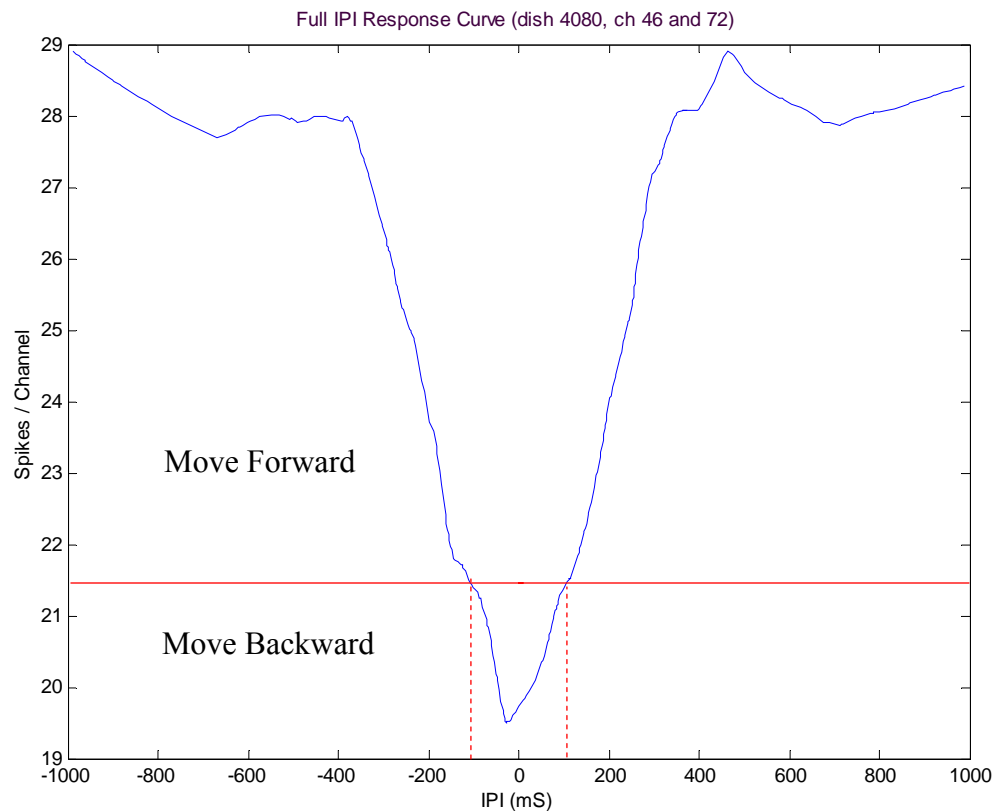


Figure 20: Sample Full IPI response curve; Forward / Backward movement is decoded by a threshold corresponding to IPI of desired distance

In this example, desired distance is encoded to 100 IPI. Thus the threshold is set to the Full averaged neural reaction corresponding to +/- 100 ms IPI stimulation.

The result of the directional vector decoding discussed thus far produces one of four unit vectors:

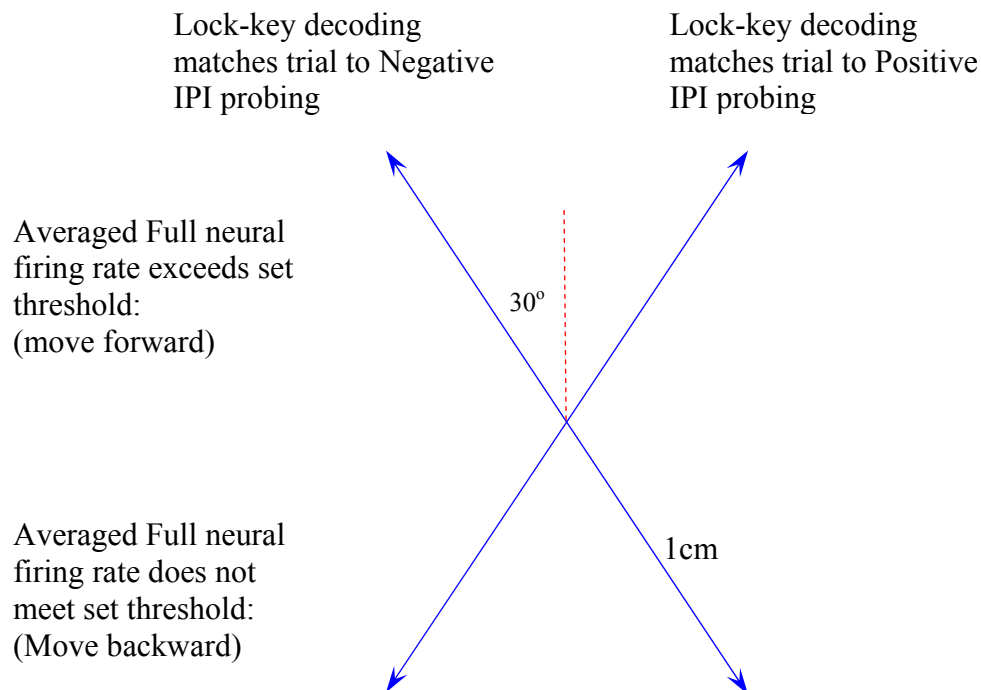


Figure 21: Unit movement vector

Movement vector magnitude decoding:

The above determined directional vector was assigned a magnitude based on a simple linear mapping (see Figure 23) of averaged neural activity following the second probe of the IPI pair to distance (vector magnitude). The Post IPI response curve is described in chapter 5, and shown in Figure 22 for reference. In effect this output mapping is simply a multiplication of the post-IPI averaged neural response by a

coefficient. The implication here is that more neurons active following an IPI probe result in a stronger movement (corresponding to more motor neuron' action potentials striking a target muscle spindle). One should note that the “desired distance” to IPI mapping mentioned above was based on the MIN values of the post-IPI curve (which correspond to ~ +/- 100 ms).

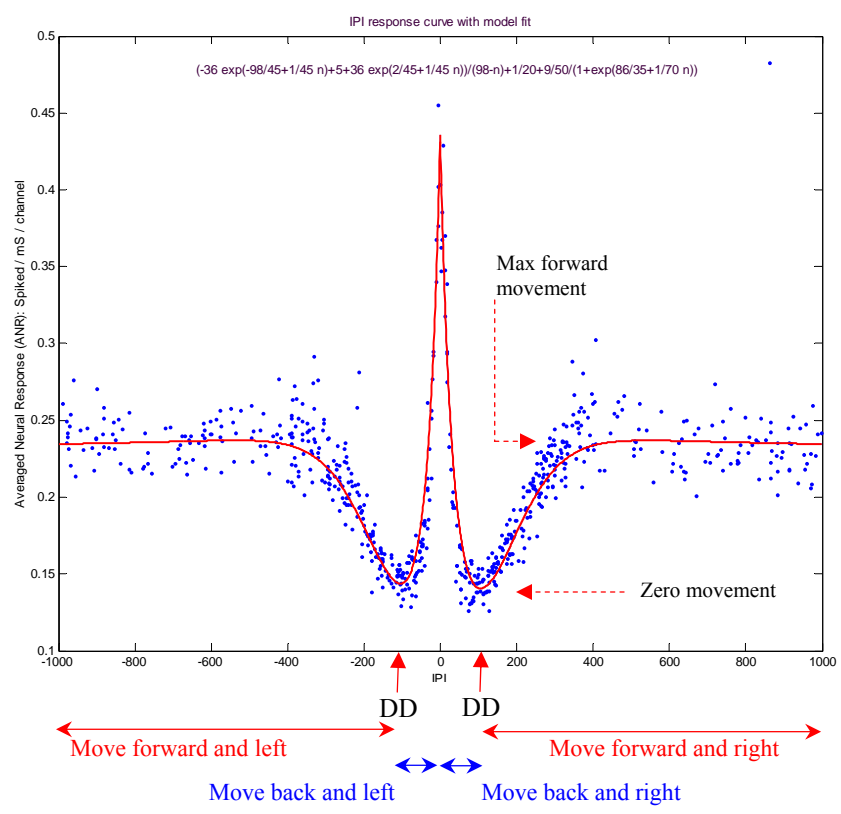


Figure 22: Post IPI response curve

A sample “output” mapping looks like this:

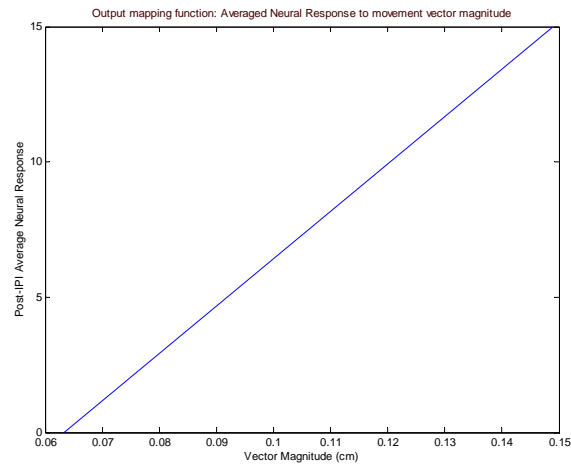


Figure 23: Output mapping: Post-IPI averaged neural response to vector magnitude

To sum up: animat' movement was a three step process: sensory information was encoded into an IPI in software, neurons reacted to this encoding (and computed something), and then the neural response was decoded into a movement in software. One dimension of this (movement vector magnitude) is shown in Figure 24.

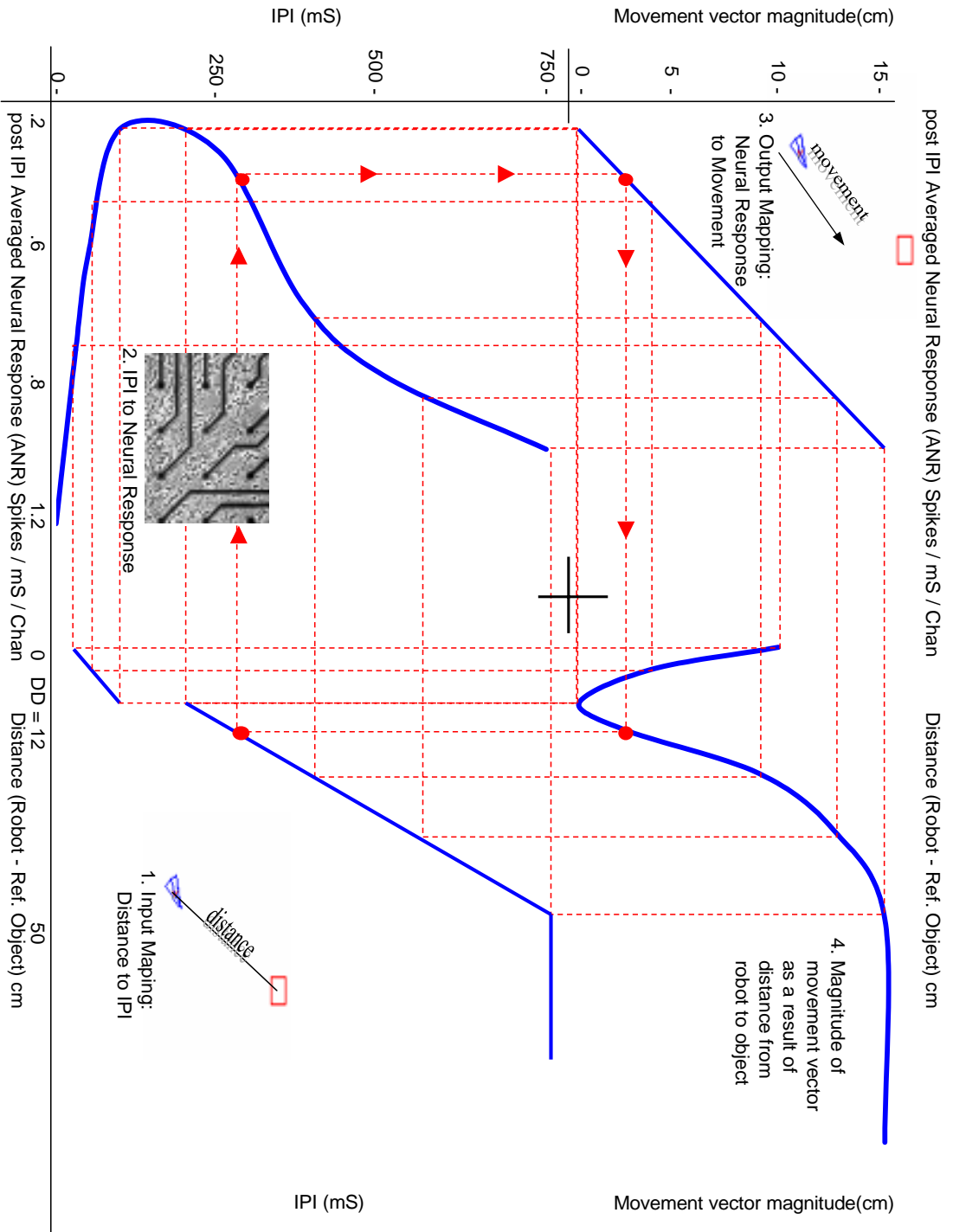


Figure 24: Diagram of input / IPI response / output curves: distance between animal and object to animal movement

7.2 Behavioral Testing

At “step 0” the animat was placed at (-80,-80), and a stationary reference object was placed at (0,90). Thus, the initial distance was ~ 188 cm. The animat took a “step” every three seconds. Sensory information was encoded into an IPI as described above. The animat moved according to a directional vector constructed by the procedure outlined above. In a given “trial”, the animat made 40 steps. At the start of each trial, the system was “reset” to “step 0” (so the distance is reset to ~ 188). The animat took 17 steps with a stationary object. In the remaining 23 steps, the object began to move down and leftwards with some random noise added to make the exact movement unpredictable. The experiment consisted of a set of 20 behavioral trials. Each trial took ~ 160 seconds, and the entire set took just under an hour.

7.3 Software Implementation

Software described in this chapter can be found in Appendix B. Most of the software was implemented in Matlab. Matlab was a natural choice for implementation given the built in graphical capabilities and powerful real-time data analysis tools. Because each step took 3 seconds, the relative inefficiency of Matlab, compared to C++ for example, was negligible. The interface between the Matlab animat code and the neural recording hardware/software and neural stimulation hardware was implemented as a C/C++ Mex function (stimIPI.cpp). The neural recording software is not provided here, but is available online at (<http://www.its.caltech.edu/~pinelab/wagenaar/meabench.html>).

A simulated animat (using real data recorded earlier) was first developed using the data obtained in chapter 5. This simulation will not be discussed in detail because

discussion of the real-time system is more interesting. I mention it only because it was implemented successfully using a different neural culture than the culture discussed in this chapter.

The real-time animat is broken down into several phases. Phase 1 is data collection and “parameter tuning”, and is implemented in `trainMappings.m`. In this phase, the cultured neurons are probed with randomly varying IPI (normal distribution) 400 times. Half of these trials used IPI ranging ± 1000 ms, the other half used a range of ± 200 ms. The purpose of such a distribution was to focus the testing in the ± 200 ms range because this is where the most interesting nonlinear dynamics are expressed.

After probing, linear input and output mappings (described above) are constructed separately for + IPI and for – IPI. Desired Distance (DD) = 12 is mapped to the min point of the Post-IPI response curve. The bottom 5% of this curve is disregarded if it is too flat (hence the discontinuity of the input mapping curve). A threshold is set in the Full IPI response curve at the IPI corresponding to DD as part of the output decoding. Lastly, using `histTrain.m`, short latency (15ms) histograms are created for each channel for each IPI polarity to be used for lock-key decoding.

Following phase 1, `testDecode2.m` can be used to analyze the data obtained in phase 1. This function displays the percent of accurate decodings for turning left/right, and for moving forward/back.

Phase 2 is behavioral testing, which as described above, is the actual animat working in real time to accomplish its task of maintaining distance from an object. If the animat is farther from the object than desired, it should approach. If it is too close, it should avoid (step away). A behavioral testing set is initiated by calling `run.m`, which sets

a few parameters and calls animat.m 20 times. Animat.m produces a trial of 40 steps as described in the above ‘behavioral’ section. It returns the movement history (x,y coordinates), distance history, turn accuracy (number of steps that turned properly towards the reference object), and approach accuracy (number of steps that properly moved forward when true distance > desired distance, and moved backward when true distance < desired distance). These values are used for behavioral analysis following the 20 probes.

7.4 Animat Behavioral Testing Results

The animat experiment (consisting of parameter tuning and 20 behavioral trials) was run three times in simulation mode (using real data from the “tuning parameters” phase, but not in real-time). The simulation was first done with dish #'s 4819 using data collected 1/13/03; the second simulation followed by a real-time trial was done on dish # 4080 using data collected 4/1/03; both simulation and real-time experiments were again conducted on 4/13/03 on dish 4080. In simulation mode, parameters could be tuned to achieve maximal performance. In the first real-time experiment, all of the parameters in the system (the IPI response curves, input/output mappings, and the histograms) seemed to change by the time the parameter tuning phase was completed and the behavioral testing phase was begun. These changes may be evidence of synaptic plasticity or some other type of adaptation, perhaps to constant stimulation. This will be discussed more in chapter 8. The result of the first real-time experiment was an animat whose behavior was erratic. This was strange given that in simulation mode, it ran almost perfectly. The

problem was that the turning accuracy dropped substantially, indicating that perhaps the “lock” was no longer valid. In other words, the major peaks in the lock histograms shifted over the first hour during parameter tuning (a more in depth discussion of plasticity is brought up in chapter 8). The second time running the experiment in real time, the neural dish was stimulated every three seconds for one hour prior to tuning parameters. The results from this second real-time experiment were much better, and will be presented in depth here.

Several quantifiable measures allow the performance of the animat to be evaluated. One must keep in mind the “goal” of the system, which is to approach an object that is far away, or to avoid (move away from) an object that is too close. If the object is moving, it should follow it. If the object is not moving, and the animat is at the desired distance, it should maintain that distance. For reference of what the animat’s world looked like, and how the animat behaved in a sample trial, see Figure 26 below.

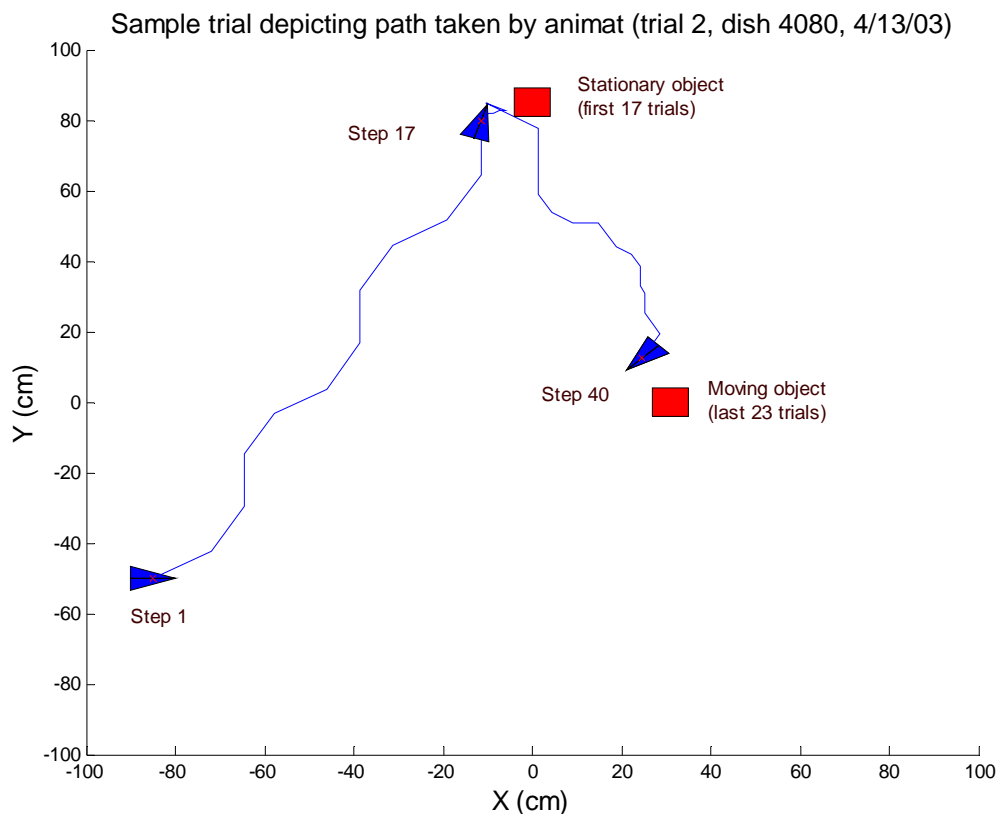
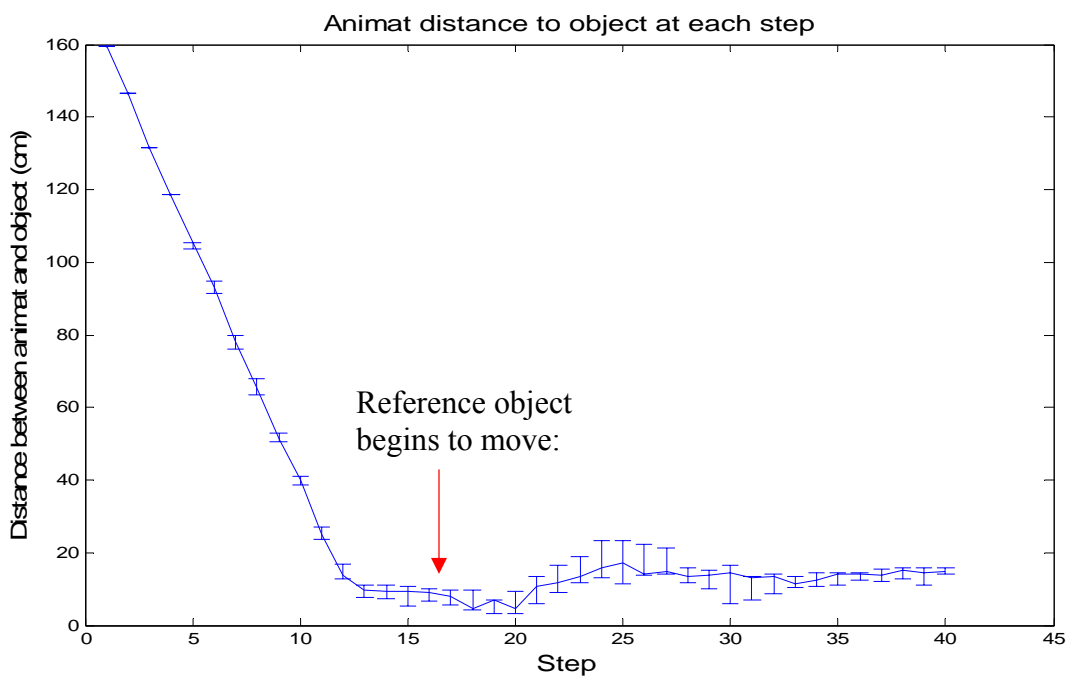
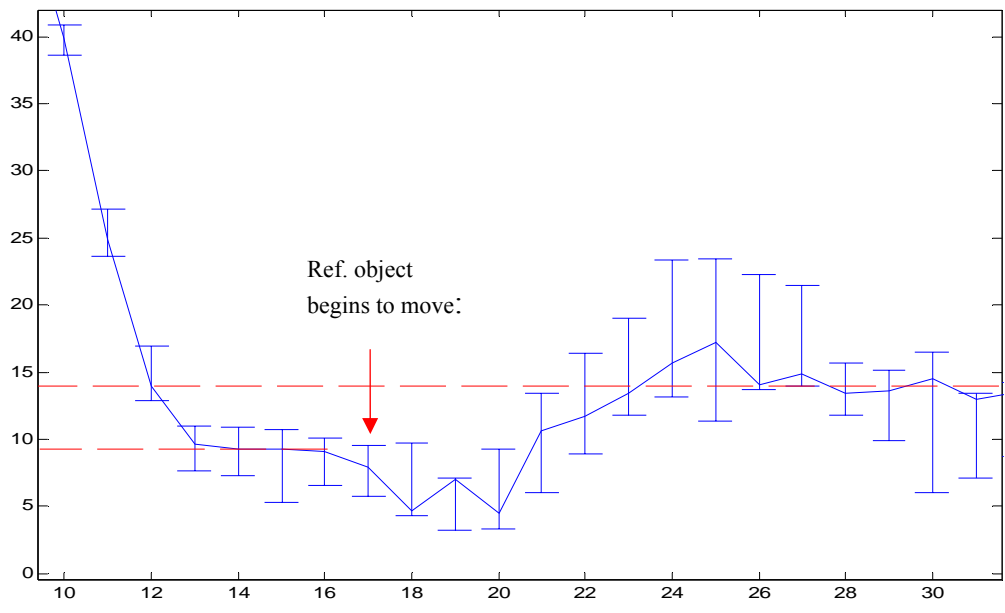


Figure 26: Sample path taken by animat during trial

From the top level, we can ask the question, did the animat approach and maintain distance from a stationary object? Figure 27 depicts the median distance (among 20 trials) between animat and object over the 40-step trial. The object is stationary the first 17 steps. One can see that clearly the animat decreases the distance with each subsequent step, and reaches the “desired distance” within 13 steps. The distance is then maintained relatively constant at ~ 9.5 cm (though it was trained to maintain a distance of 12 cm). The error bars in the figure represent the distance of the median 50% of trials. These error bars are reasonably tight, indicating that the animat’s behavior is repeatable.



During behavioral testing, the animat takes 40 steps over 20 trials; the blue line is the median distance at a given step. Error bars represent the median 50% of trials;



Blowup of above distance curve; Shows mean “desired distance” values maintained for stationary / moving ref. object

Figure 27: Animat distance to object – top: full curve for all 40 steps; bottom: blowup of steps 10 through 30

In steps 18 through 40 of each trial, the reference object begins to move down and tends to move rightward, though this rightward movement is randomized. One may notice an initial dip in the distance graph at ~ step 19. This is due to the fact that the animat initially approaches the object from below, and is facing the object as the object begins to move towards the animat. Thus it is the object moving that decreases the distance. The animat typically backs up, or turns to the side to avoid the object coming towards it. It then proceeds to turn until it is facing the object again. Interestingly, as the animat is turning it may be facing the wrong direction for several steps. However the object continues to move further and further from the animat, which forces the animat to take larger and larger steps (in the wrong direction). This accounts for the slight peak seen at ~ step 25. The animat's behavior then stabilizes, and it is able to follow the object appropriately. The median distance maintained to the moving object is ~ 14 cm, which is ~ 4.5cm more than the median distance maintained between the animat and the stationary object. It is interesting that this is approximately how much the object is moving in each step. One should recognize, however, that the distance curve does not have an upward slope, which would indicate that the animat could not keep up with the object. Rather, the animat settles to a new "desired distance" when the object is moving at the given speed.

Decoding accuracy (the percentage of steps in which the animat turned in the correct direction, or correctly chose to move forward/backwards) plays a large role in the animat's behavior. In simulation mode, decoding the turn values at each step reached as high as 100%. In real time, the turning accuracy remained high at 95.3% ($\sigma = 3.2\%$). Approach accuracy (decoding forward/backward movement) was a bit lower, but still

high enough for proper functioning of the animat, at 87.6% ($\sigma = 11.4\%$). It seems the threshold used for approach based on the Full IPI curve was a bit too low, but may be a result of plasticity (see chapter 8).

CHAPTER 8

Learning expressed as changes in animat behavior

8.1 Background

One of the main benefits of living neural networks as opposed to digital computing is a built in ability to learn based on experience. When learning occurs, synaptic weights adjust (often referred to as synaptic plasticity) and thenceforth the system's behavior changes even if given the same input conditions as before the learning had occurred. Finite State Automata theory breaks down when trying to emulate neural learning, because the neurons can "rewire" themselves automatically, thus changing the possible states the system may enter on a given input. Any change in synaptic weights may therefore be considered a doubling of the states in the automaton. If one tries to emulate learning to infinite precision (since the synaptic weights are analog), one may realize that the living neural network may actually have an infinite number of states. Though neural networks are chaotic systems, infinite precision is probably not required to model an analog synapse. Even so, the number of finite states that a network of neurons may enter may be unreasonably many to consider with standard automata theory.

Given that learning may increase the processing power of living neuronal networks, it is unfortunate that thus far, we still know very little about how the brain learns. Studying learning in-vitro has typically involved synaptic plasticity between two or maybe three cells. Learning in large neural networks in-vitro has proved to be difficult to characterize. Even when attempts are made to characterize any changes in neural firing

patterns that may occur following a given type of input, one still may be left wondering “so what does that mean?” A tremendous gap remains between our limited understanding of learning at the neuronal (bottom) level and how this learning may influence an animal at the behavioral (top) level.

The animat described in chapter 7 may allow researchers to close this gap by studying neuronal mechanisms of learning in combination with behavioral mechanisms of learning. Thus, changes in behavior of the animat can be compared to changes in neuronal behavior, and vice versa. The animat system can be used to assign meaning to neuronal learning if it alters its behavior in some way as a result of the neuronal changes.

In the experiment presented in this chapter, I hope to demonstrate the feasibility of using the previously described animat as a framework for studying learning. I will do this by showing that changes at the neural level can be expressed by changes in the IPI response curves (both the Post IPI and the Full IPI response curves); further, I will attempt to characterize how changing the IPI response curves may influence behavior. I provide theoretical (thought experiments) as well as limited empirical evidence of such feasibility.

8.2 Method

Train Parameters (1), and obtain baseline behavior:

This portion of methodology was actually described in chapter 7. To recap, parameter tuning (1) was conducted within a few minutes of placing the dish into the environmental chamber. The neurons were stimulated every 3 seconds for one hour.

Parameters were trained again (2). To determine “baseline behavior”, a 20 trial behavioral experiment was conducted using the animat.

Induced learning (tetanic stimulation):

Following the baseline behavioral testing, a strong tetanic pulse train was applied on a third stimulation channel. The tetanic train consisted of 10 pulses at 20 Hz, applied 20 times at 5 second intervals

Tetanic pulses of this sort have been shown to induce plasticity within cultured networks (Jimbo et al, Tateno et al). If the tetanus is effective, it may induce some lasting changes in the network; mainly, when probing on single channels, the effect is a dish-wide channel specific enhancement or depression. Thus, probing on channel A following tetanus may result in increased neural response on all other channels, when compared to baseline. At the same time, stimulating on channel B may have an opposite effect, whereby neural activity is suppressed on all channels when B is stimulated, compared to baseline. Thus, if plasticity is induced, 4 combinations of single-probe effects are possible. The effect of tetanus on a paired-pulse IPI probe is unknown, but one might expect changes in the effect of such probing following tetanic stimulation.

Experiment-proper:

Following the tetanic pulse, 20 trials of behavioral testing was repeated. The aim was to compare these behavioral trials with those before the tetanus. Following the second set of behavioral testing the parameters were trained again (3).

8.3 Results

Changes in Post IPI Response Curve:

There were 3 sets of IPI curves to examine during data analysis: (1) initial; (2) 1 hour later, pre-tetanus (baseline); and (3) post-tetanus (experiment-proper). The IPI response curves are shown in Figure 28 (raw data) and Figure 29 (Lowess smoothed data).

A quick note regarding the Lowess smoothing, is that it does linear regression using 10% of all data points to calculate each smoothed data point. Near IPI = 0, the data is so steep that the lowess smoothing truncates the peak, so it would not be accurate to make inferences about why the green (post-tetanus) peak at IPI=0 is smaller than the red and blue peaks.

When comparing post vs pre tetanus IPI curves (green and red respectively), one may notice a lower neural response at the min when probed with -100 IPI; one may also notice the slope (from -100 to -300 IPI) is steeper in post-tetanus. Aside from these minor variations, the post and pre tetanus curves appear very similar. Repeated experiments are necessary to properly conclude what effect tetanus stimuli may have, if any, on the IPI response curve. Despite this inconclusiveness, there appears to be a drastic change from the initial IPI response curve (blue) compared to the other curves. Apparently, after the first hour of stimulation the entire IPI effect seems to have shifted down by roughly 10% - 15%. This change may occur due to some habituation in response to the probing every 3 seconds in the first hour, or it may be that the dish is acclimating from the incubator environment, to the environmental chamber' environment. The very fact that the IPI response curve changed in the first hour indicates that the IPI response curve is indeed

capable of reflecting some kind of neural change, and may thus be useful in changing the behavior of the robot.

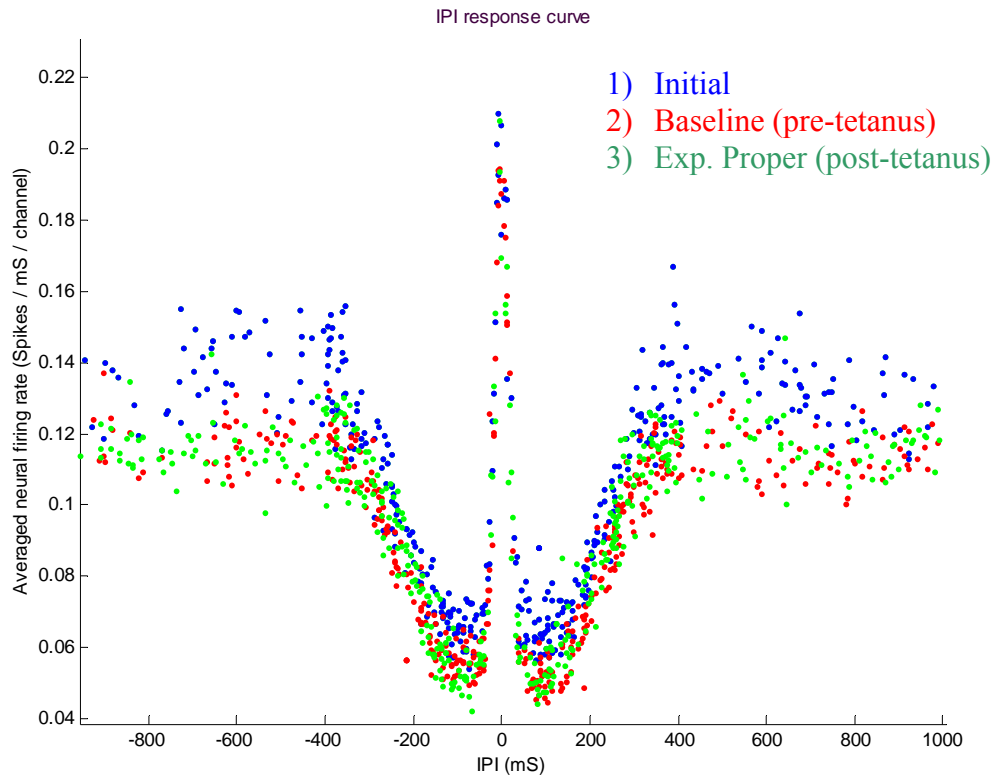


Figure 28: Raster plot of 3 sets of Post-IPI response curves

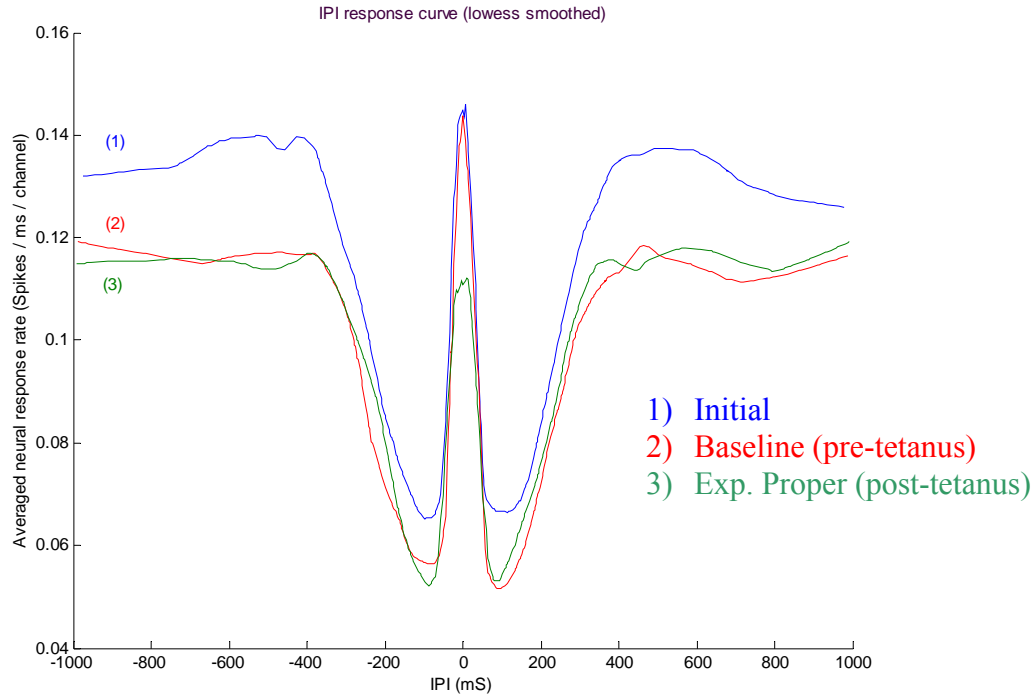


Figure 29: Lowess smoothed plots of 3 sets of Post-IPI response curves

Changes in Behavior:

There were 2 sets of behavioral results to examine: Pre-tetanus (baseline) and Post-tetanus (experiment proper). The behavioral comparison is made by examining the distance curves (described in chapter 7) for each set of behavioral testing. The compared distance curves are shown in Figure 30 and Figure 31. The pre-tetanus curve (shown in red) is identical to the one shown in chapter 7 (because it is the same data). The post-tetanus curve (shown in green) may show some slight differences, namely a slightly less steep approach curve (steps 0 through 13), and a decreased maintained “desired distance” of 7cm (post tetanus), as opposed to 9.5 (pre tetanus). Conclusions are difficult to draw here, however, given that the error bars are overlapping.

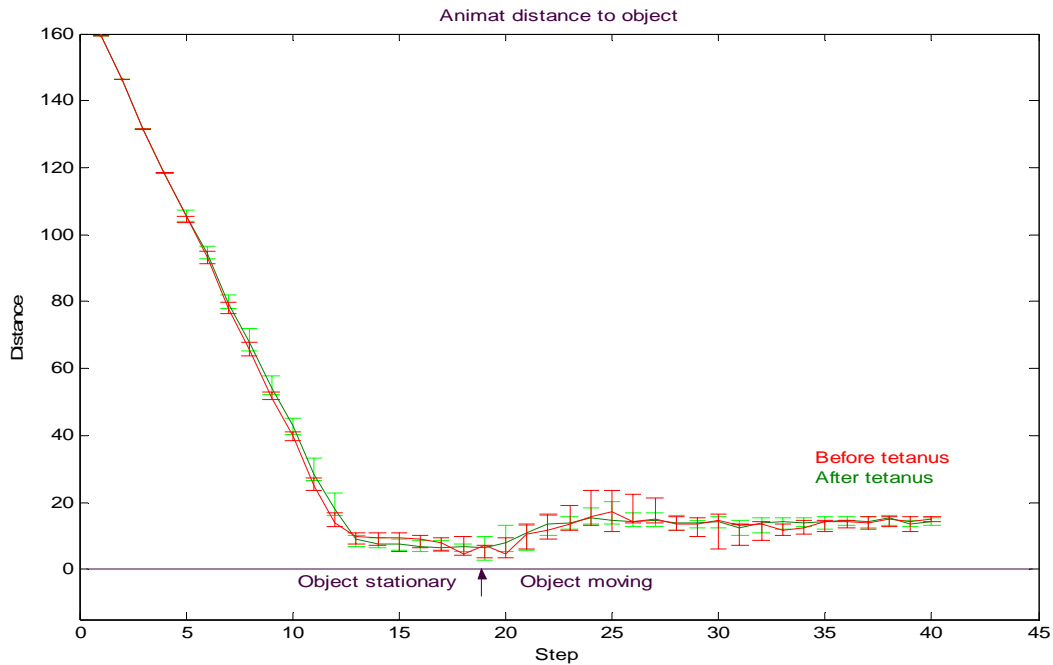


Figure 30: Distance curves, pre-tetanus and post-tetanus

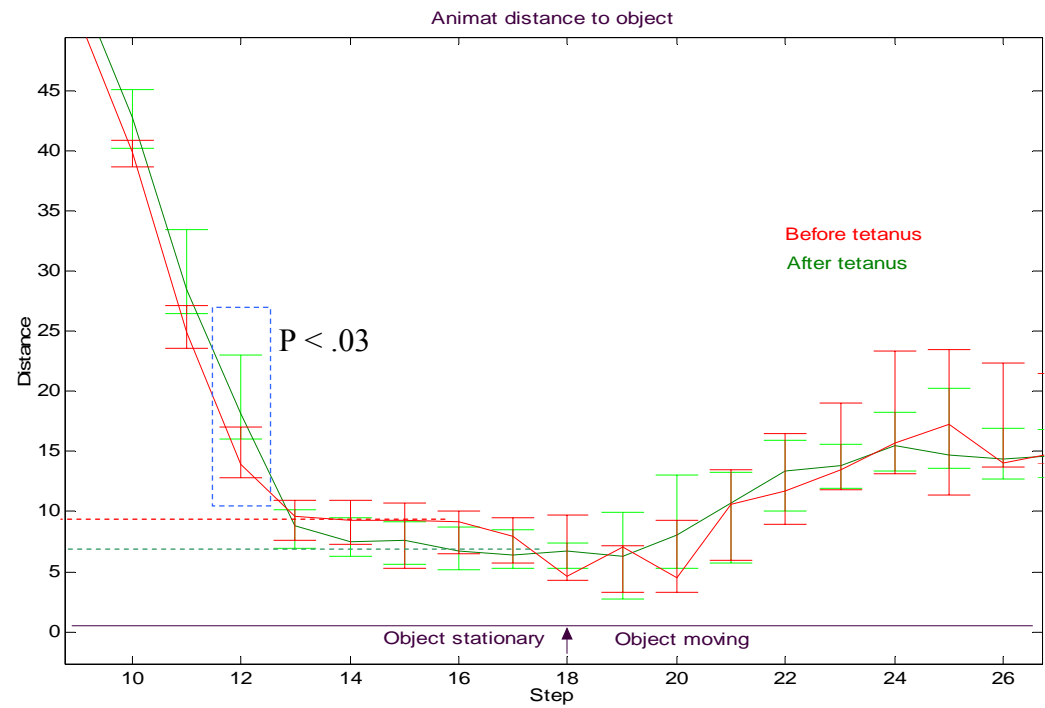


Figure 31: distance curves (blow up)

Some of the change in behavior may be statistically significant (p-value = .027 at step 12)

Changes in precisely timed spikes

Recall that precisely timed spikes were used to build lock's (histograms of data from many IPI probes), which could be opened with a key (a single IPI probe). The lock and key approach resulted in a 95% turning accuracy at each step, before and after tetanus. Because the locks were not retrained following tetanus, the fact that turning accuracy remained high indicates that no change was induced in the precisely timed spikes. Data from the Potter lab indicates that precisely timed spikes may be a direct result of stimulating a certain set of neurons, and is not a synaptic process. Thus, it is reasonable to assume that these precisely timed spikes may not change with stimulation designed to induce synaptic plasticity.

Further Discussion:

The experiment presented here suggests that the IPI response curves may change as a result of induced synaptic plasticity. Though the evidence favors the suggestion, further experiments are required to prove that the behavior of the animat can be directly modified through tetanus (or any other type of) stimulation. However, I can speculate as to the nature that a more effective tetanus stimulation than the one shown in this experiment may have on the behavior of the animat. For example, excitation on the latter of an IPI probe pair may cause the 'U' shaped IPI response curve to shift upwards. Such an effect would mainly influence the step size of the robot, causing it to approach the reference object quicker, but may decrease the maintained desired distance, because the neurons will tend to fire over the forward/backward threshold set in the Full IPI response curve. As a second example, excitation on the prior of an IPI probe pair may induce a

longer burst, and therefore increase the relative burst' refractory period, thus broadening the curve. The exact effect on behavior is difficult to predict (because the linear mappings do not change, but the non-linear dynamics of the neural response does), though one can say for sure that the approach speed would change, as would the desired distance.

CHAPTER 9

Conclusions and Future Directions

In this project, I showed that probing with varying delay between the probes produces a predictable non-linear response. I showed how to emulate digital logic using such a response, thus proving that cultured neurons can theoretically execute a computer program with polynomial slow-down. I then applied the living neurons to handle a more interesting real-world problem in real-time. In this project an animat was created that combines a living neural network with a virtual body in an effort to create a system where the living neural network could be studied. The animat was successful at tracking and maintaining distance from a reference object, which can be considered both an approach and avoidance task. Part of the robustness of the animat in the current project is that it reacts more strongly when necessary to correct for error. Thus, if it has an error on some trial, the error will not be fatal because in the next trial it will make up for it.

The animat in this project is unique because I am replacing algorithmic components of control with neural computation. In the animat, sensory information is encoded into stimulus information, which induces a given reaction in the neural network. The behavior of the animat is determined solely on this neural response. There is no algorithmic component converting sensory information into animat movement. Thus, the animat demonstrates some of the computational power of cultured cortical neurons.

Furthermore, the animat provides a scheme for testing the effects of plasticity in cultured neurons, and these effects are visible through quantifiable measurements of robotic behavior. The animat opens the door for additional experiments, both to determine interesting robotic behavior, such as tracking and following a moving

reference object, and to determine the effects of varying types of plasticity that may be induced.

One may ask the question, “how much intelligence does the animat display?” An in-depth philosophical discussion of this topic is beyond the scope of this thesis, but one should consider that the animat is handling a relatively difficult approach / avoidance task in real-time. A truly intelligent machine should of course be able to handle a variety of tasks, including tasks that it has never faced before. To accomplish such a goal, more complex sensory information needs to be encoded (rather than simply the direction and distance of a single object within the environment). Thus, an important future direction in the development of this animat may involve the utilization of additional channels for stimulation.

I hope that the animat built in this project is at the beginning of its development cycle. In the near future, the animat can be converted into a real robot fairly simply. Small changes, such as improvements to the mapping schemes to make them more dynamic can improve the performance of the animat. For example, with each step the animat takes, the individual channel histograms used for lock/key decoding may be slightly modified. This will take into account any slight “drift” in precisely timed spikes over time.

References

1. Arbib, M.A., ed. (2002). *The Handbook of Brain Theory and Neural Networks: Second Edition*. Cambridge, MA: MIT.
2. Bi, G., Poo, M. (1999). Distributed synaptic modification in neural networks induced by patterned stimulation. *Nature*. 401(6755):792-6.
3. Bower, J.M., Beeman, D. (1998). *The Book of Genesis*. Santa Clara, CA: Springer-Verlag publishers.
4. Braitenberg, V. (1984) *Vehicles*. Cambridge, MA: MIT.
5. Buonomano, D.V. (2000). Decoding Temporal Information: A Model Based on Short-Term Synaptic Plasticity. *J. Neurosci*. 20:1129–1141.
6. Damper, R.I, French, R.L.B. and Scutt, T.W. (2001). The Hi- neural simulator and its applications. *Microelectronics Reliability*. 41(12) 2051-2065.
7. DeMarse, T.B., Wagenaar, D.A., Blau, A.W. and Potter, S.M. (2001). The neurally controlled animat: biological brains acting with simulated bodies. *Autonomous Robots*. 11:305–310.
8. Deonarine, A.S., Clark, S.M, and Konermann, L. (2003). Implementation of a multifunctional logic gate based on folding/unfolding transitions of a protein. *Future Generation Computer Systems*. 19:87-97.
9. Fry, R.L. (2003). A theory of neural computation. *Neurocomputing*. In Press, Corrected Proof. Available online at <http://www.sciencedirect.com/science/article/B6V10-47RYWNX-D/2/48ac18d74eca18ea005961511d493eae>
10. Garcia, P.S., Calabrese, R.L., DeWeerth, S.P., and Ditto, W. (2002). Simple Arithmetic with Firing Rate Encoding in Leech Neurons: Simulation and Experiment (Pre-print). Available online at <http://www.neuroengineering.com/publications/111180.pdf>.
11. Gerstner, W. (1988) Spiking Neurons. In *Pulsed Neural Networks* (Maass, W. and Bishop C., eds), pp 3-53. Cambridge, MA: MIT.
12. Gholmieh, G., Courellis, S., Marmarelis, V., Berger, T. (2002). An efficient method for studying short-term plasticity with random impulse train stimuli. *J Neurosci Methods*. 121(2):111-27.

13. Hansson, E., Ronnback, L. (2003). Glial neuronal signaling in the central nervous system. *FASEB J.* 17(3):341-8.
14. Hodgkin, A.L. and Huxley, A.F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology.* 117:500.
15. Jimbo, Y., Tateno, T., Robinson, H.P.. (1999). Simultaneous induction of pathway-specific potentiation and depression in networks of cortical neurons. *Biophys J.* 76(2):670-8.
16. Kandel, E.R., Schwartz, J.H., Jessell, T.M. (1991). *Principles of Neural Science.* Norwalk, CT: Appleton & Lange.
17. Koch, C. (1998). *Biophysics of Computation: Information Processing in Single Neurons.* Oxford, UK: Oxford University Press.
18. Kralik, J.D., Dimitrov, D.F., Krupa, D.J., Katz, D.B., Cohen, D., Nicolelis, M.A. (2001). Techniques for long-term multisite neuronal ensemble recordings in behaving animals. *Methods.* 25(2):121-50.
19. Maass, W. and Zador, A. (1998). Computing and learning with dynamic synapses. In *Pulsed Neural Networks* (Maass, W. and Bishop C., eds), pp 321-336. Cambridge, MA: MIT.
20. Marom, S., Shahaf, G. (2001). Learning in networks of cortical neurons. *J. Neurosci.* 21(22):8782-8788.
21. Marom, S., Shahaf, G. (2002). Development, learning and memory in large random networks of cortical neurons: lessons beyond anatomy. *Quarterly Reviews of Biophysics.* 35:63-87.
22. Mass, W. (1998) Computing with Spiking Neurons. In *Pulsed Neural Networks* (Maass, W. and Bishop C., eds), pp 54-85. Cambridge, MA: MIT.
23. Nicolelis, M.A., Ribeiro, S. (2002). Multielectrode recordings: the next steps. *Curr Opin Neurobiol.* 12(5):602-6.
24. Nicolelis, MA. (2002). The amazing adventures of robotrat. *Trends Cogn Sci.* 6(11):449-450.
25. O'Reilly, R.C. and Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience.* Cambridge, MA: MIT.
26. Pakkenberg, B., Pelvig, D., Marnier, L., Bundgaard, M.J., Gundersen, H.J.G., Nyengaard, J.R., Regeur, L. (2003). Aging and the human neocortex. *Experimental Gerontology.* 38(1-2):95-99.

27. Potter SM. Distributed processing in cultured neuronal networks. *Prog Brain Res.* 130:49-62.
28. Potter, S.M. and DeMarse, T.B. (2001). A new approach to neural cell culture for long-term studies. *Journal of Neuroscience Methods*, 110(1-2):17-24.
29. Reger, B.D., Fleming, K.M., Sanguineti, V., Alford, S. and Mussa-Ivaldi, F.A. (2000). Connecting brains to robots: an artificial body for studying the computational properties of neural tissues. *Artif Life.* 6(4):307-24.
30. Sanger, T.D. (2003). Neural population codes, *Current Opinion in Neurobiology.* In Press, Corrected Proof. Available online at <http://www.sciencedirect.com/science/article/B6VS3-487V2KN-3/2/9366dd3e64b8f3d33473c5bb461032d1>
31. Schnupp, J.W., King, A.J. (2001). Neural processing: the logic of multiplication in single neurons. 11(16):R640-2.
32. Simpson, M.L., Sayler, G.S., Fleming, J.T. and Applegate B. (2001). Whole-cell biocomputing. *Trends in Biotechnology*, 19:317-323.
33. Sinha, S., Ditto, W.L. (1999). Computing with distributed chaos. *Phys Rev E.* 60(1):363-77.
34. Sipser, M. (1997). *Introduction to the theory of computation.* Boston, MA: PWS Publishing.
35. Talwar, S.K., Xu, S., Hawley, E.S., Weiss, S.A., Moxon, K.A., Chapin, J.K. (2002). Rat navigation guided by remote control. *Nature.* 417(6884):37-38.
36. Tateno, T., Jimbo, Y. (1999). Activity-dependent enhancement in the reliability of correlated spike timings in cultured cortical neurons. *Biol Cybern.* 80(1):45-55.
37. Tsacopoulos, M. (2002). Metabolic signaling between neurons and glial cells: a short review. *Journal of Physiology.* 96(3-4):283-288.
38. Wagenaar, D.A., and Potter, S.M. (2002). Real-time multi-channel stimulus artifact suppression by local curve fitting. *J. Neurosci. Meth.* 120:113-120.
39. Watts, J.M. (1998). Animats: computer-simulated animals in behavioral research. *J Anim Sci.* 76(10):2596-604.
40. Zucker, R.S. (1989). Short-term synaptic plasticity. *Annu. Rev. Neurosci.* 12:13–31.

Appendix A – Data Analysis Code

Effect of dual channel probing on neural activity

The code in this appendix provides sample Matlab scripts used for data analysis, which were mostly utilized in chapter 5.

- A.1) `getSpikeData.m` converts raw spike data (from Meabench) into the spike data structures used in the rest of the matlab scripts in this project.
- A.2) `compMeans.m` computed post-IPI and full-IPI response curves.
- A.3) `plotrastall.m` plots a raster plot of all channels for a given latency during and following IPI probing
- A.4) `plotSpkRaster.m` plots a raster plot for an individual channel
- A.5) `sortSpkData.m` returns histogram data for an individual channel
- A.6) `plot_hist.m` plots histograms for all channels

A.1 getSpikeData.m

```

function y = getSpikeData();

% converts raw spike data (from Meabench) into the spike data structures used in the rest
% of the matlab scripts in this project.

format long;

numProbes = 29;
numChans = 65;
numTrials = 75;

%initialize y data structure
for(i = 1:numProbes)
    for (j=1:numChans)
        eval(sprintf('y.probe%d.ch%d = zeros(numTrials,25);',i,j));
    end
end

% open data file
fid = fopen('c:\research\1-13-03-4819.spike','rb');
if (fid<0)
    error('Cannot open the specified file');
end

probe=1;
trial=1;
ch = -1;
prevtime = 0;
time = 0;

[time,ch] = rspike(fid);
eval(sprintf('y.probe%d.ch%d(%d,%d) = time;',probe,ch,trial,1));
while (trial ~= 76)
    % read information for 75 trials
    prevtime = time;
    [time,ch] = rspike(fid);

    while ((time-prevtime) < 2900)
        % stimulations separated by ~ 3000ms
        eval(sprintf('index = nnz(y.probe%d.ch%d(%d,:))+1;',probe,ch,trial));
        eval(sprintf('arSize = size(y.probe%d.ch%d,2);',probe,ch));
        if index > arSize
            eval(sprintf('y.probe%d.ch%d =
doubleArr(y.probe%d.ch%d,2,[]);',probe,ch,probe,ch));
        end
        eval(sprintf('y.probe%d.ch%d(%d,%d) = time;',probe,ch,trial,index));
        [time,ch] = rspike(fid);
    end
end

```

```
end
if (probe == 29)
    probe = 1
    trial = trial+1
else
    probe = probe+1
end
%eval(sprintf('y.probe%d.ch65(%d,1) = time+100;',probe,trial));
end
```

A.2 compMeans.m

```

function compMeans(Spk_matrix,T_Total)
% compute means and medians and plot
% ave spikes computed from time of latter probe + T_Total
% results in spikes per ms
% plot of 25/50/75 percentiles

% use setprodtype before calling this function
global probdtype;

% set graph boundaries for x values near used IPI vals
xArr = zeros(21,2);
xArr(1,:) = [-1,1]; xArr(2,:) = [850,900]; xArr(3,:) = [-850,-900]; xArr(4,:) = [1,3.5];
xArr(5,:) = [3.5,7.5]; xArr(6,:) = [7.5,12.5]; xArr(7,:) = [12.5,17.5];
xArr(8,:) = [17.5,25]; xArr(9,:) = [25,40]; xArr(10,:) = [40,75]; xArr(11,:) = [75,175];
xArr(12,:) = [175,300]; xArr(13,:) = [300,400]; xArr(14,:) = [400,600];
xArr(15,:) = [600,800]; xArr(16,:) = [-1,-3.5]; xArr(17,:) = [-3.5,-7.5];
xArr(18,:) = [-7.5,-12.5]; xArr(19,:) = [-12.5,-17.5]; xArr(20,:) = [-17.5,-25];
xArr(21,:) = [-25,-40]; xArr(22,:) = [-925,-975]; xArr(23,:) = [925,975];
xArr(24,:) = [-40,-75]; xArr(25,:) = [-75,-175]; xArr(26,:) = [-175,-300];
xArr(27,:) = [-300,-400]; xArr(28,:) = [-400,-600]; xArr(29,:) = [-600,-800];

numBins = 25;

Paves = zeros(29,75);
close all;
figure;
hold on;

for stimch = [1:29]
    % for each IPI duration

    A1=63;
    for ch = [1:60]
        % for each channel

        Pool1=[];
        Pool2=[];

        eval(sprintf('Spk2 = Spk_matrix.probe%d.ch%d;',stimch,ch));
        eval(sprintf('Trigger2=Spk_matrix.probe%d.ch63(:,1);',stimch));

        for j=1:75
            % for each trial

            Spk_T2=Spk2(j,:)-Trigger2(j);

```

```

    Spk_T2=Spk_T2(find(Spk_T2<(abs(probetype(stimch))+T_Total) &
    Spk_T2>(0.5+abs(probetype(stimch)))));
    Paves(stimch,j) = Paves(stimch,j) + size(Spk_T2,2);
end;
end
Paves(stimch,:) = Paves(stimch,:)/(T_Total*60);
a(stimch) = mean(Paves(stimch,:));
d(stimch) = median(Paves(stimch,:));
b(stimch) = std(Paves(stimch,:));
e(stimch) = prctile(Paves(stimch,:),25);
f(stimch) = prctile(Paves(stimch,:),75);

cmin = min(Paves(stimch,:));
cmax = max(Paves(stimch,:));
csize = (cmax-cmin)/numBins;

% build color coded histogram
c = hist(Paves(stimch,:),numBins)/nnz(Paves(stimch,:));

cX = xArr(stimch,:);
fill([cX(1),cX(1),cX(2),cX(2),cX(1)], [cmin,cmax,cmax,cmin,cmin], [1,1,1]);

for i = 0:numBins-1

    if(c(i+1)>0)
        cCol = [min(1,5*c(i+1)),.3,max(0,(1-5*c(i+1)))];
        fill([cX(1),cX(1),cX(2),cX(2),cX(1)], [cmin +
csize*i,cmin+cszize*(i+1),cmin+cszize*(i+1),cmin+cszize*i,cmin+cszize*i],cCol);
    else
        cCol = [1,1,1];
    end
end
end

Paves1 = zeros(29,75);
for stimch = [12,26]
    for ch = [1:60]
        Pool1=[];
        Pool2=[];

        eval(sprintf('Spk2 = Spk_matrix.probe%d.ch%d;',stimch,ch));
        eval(sprintf('Trigger2=Spk_matrix.probe%d.ch63(:,1);',stimch));

        for j=1:75
            Spk_T2=Spk2(j,:)-Trigger2(j);
            Spk_T2=Spk_T2(find(Spk_T2<(T_Total) & Spk_T2>.5));
            Paves1(stimch,j) = Paves1(stimch,j) + size(Spk_T2,2);
        end;
    end
end
Paves1(stimch,:) = Paves1(stimch,:)/(T_Total*60);

```

```

a1(stimch) = mean(Paves1(stimch,:));
d1(stimch) = median(Paves1(stimch,:));
b1(stimch) = std(Paves1(stimch,:));
e1(stimch) = prctile(Paves1(stimch,:),25);
f1(stimch) = prctile(Paves1(stimch,:),75);

cmin = min(Paves1(stimch,:));
cmax = max(Paves1(stimch,:));
csize = (cmax-cmin)/numBins;
c = hist(Paves1(stimch,:),numBins)/nnz(Paves1(stimch,:));

if stimch == 12
    cX = [-1000,-1050] ;
else
    cX = [1000,1050] ;
end

fill([cX(1),cX(1),cX(2),cX(2),cX(1)], [cmin,cmax,cmax,cmin,cmin],[1,1,1]);

for i = 0:numBins-1

    if(c(i+1)>0)
        cCol = [min(1,5*c(i+1)),.3,max(0,(1-5*c(i+1)))];
        fill([cX(1),cX(1),cX(2),cX(2),cX(1)], [cmin +
csize*i,cmin+csize*(i+1),cmin+csize*(i+1),cmin+csize*i,cmin+csize*i],cCol);
    else
        cCol = [1,1,1];
    end

end

end

end

yl=yylim;
yl = yl(2);

fill([100,100,110,110,100],[.9*yl,.95*yl,.95*yl,.9*yl,.9*yl],[1,1,1]);
text(100,.875*yl,'0 %')
text(190,.875*yl,'>= 20 %')
for i = 110:10:200
    cCol = [min(1,5*(i-100)/500),.3,max(0,(1-5*(i-100)/500))];
    fill([i,i,i+10,i+10,i],[.9*yl,.95*yl,.95*yl,.9*yl,.9*yl],cCol);
end

singchm1 = [e([2,3,23,22]),e1([26,12])];
singchm2 = [f([2,3,23,22]),f1([26,12])];
singchmeds = [d([2,3,23,22]),d1([26,12])];

a = a([1,4:21,24:29]);

```

```

b = b([1,4:21,24:29]);
d = d([1,4:21,24:29]);
e = e([1,4:21,24:29]);
f = f([1,4:21,24:29]);
[c,i] = sort(probetype([1,4:21,24:29]));

xlabel('IPI (ms)')
ylabel('Spikes / ms / channel')
title(sprintf('dishwide spike intensity by trial (%d ms post-IPI)',T_Total))
text(-250,.95*yl,'Each column represents 50')
text(-250,.925*yl,'trials at given IPI. Columns')
text(-250,.9*yl,'are divided into 25 bins, color')
text(-250,.875*yl,'intensity represents number of')
text(-250,.85*yl,'trials that fall in given bin.')
saveas(gcf,sprintf('dishwide spike intensity by trial (%d ms post-IPI).jpg',T_Total))

figure
plot(c,e(i),'b-');
hold on
plot(c,f(i),'b-');
plot(c,d(i),'rx--');

plot([800,-800,850,-850,900,-900],singchm1,'bo');
plot([800,-800,850,-850,900,-900],singchm2,'bo');
plot([800,-800,850,-850,900,-900],singchmeds,'rx');

yl=yylim;
text(700,.9*yl(2),'Probe B (t=inf)')
text(-900,.9*yl(2),'Probe A (t=inf)')
text(-850,.8*yl(2),'red - median (50 percentile)')
text(-850,.75*yl(2),'blue - 25~75 percentile')
title(sprintf('dishwide reaction to dualchan probing (%d ms post-IPI)',T_Total))
xlabel('IPI (ms)')
ylabel('Spikes / ms / channel')
saveas(gcf,sprintf('dishwide reaction to dualchan probing (%d ms post-IPI).jpg',T_Total))

```

A.3 plotrastall.m

```

function plotrastall(Spk_matrix,Time,stimchs,reduce)
% Plots raster plot for a given spike matrix, for a given time period in msecs
% this is designed to work with load_mchan_spks.m or similar

%plots raster plot for all channels on one plot, stacked with hw channel 1 on bottom.
%reduce = 0 plots all; reduce = 1 only plots channels with significant activity

global probetype;

%close all;
for stimch=stimchs
    figure;
    hold on;

    count = 1;

    for i=1:64
        [Spk_trigger,plotted] = plotSpkRaster(Spk_matrix,i,stimch,Time,1,count,reduce);
        if( plotted == 1)
            count = count+1;
        end
    end;

    if stimch == 2
        title(sprintf('Raster plots-%dms-A only',Time));
        saveas(gcf,sprintf('Raster plots-%dms-A only.jpg',Time));
    elseif stimch == 3
        title(sprintf('Raster plots-%dms-B only',Time));
        saveas(gcf,sprintf('Raster plots-%dms-B only.jpg',Time));
    else
        title(sprintf('Raster plots-%dms-IPI %d',Time,probetype(stimch)));
        saveas(gcf,sprintf('Raster plots-%dms-IPI %d.jpg',Time,probetype(stimch)));
    end
end

end

```

A.4 plotSpkRaster.m

```

function
[Pool,plotted,n]=plotSpkRaster(Spk_matrix,ch,probe,T_Total,allchans,count,reduce)

%Rearrange the spike data(Spk_matrix) of certain channel(ch) by trigger order.
%And plot the RASTER PLOT.

%INPUT:
% Spk_matrix = array with dimensions (65,4,75,x);
% Spk_matrix (ch,stim channel of prob,iter,spikecount)
%ch: hardware channel number.
%stimch: 1:4 value of probe channel
%T_Total: time length (msec) after trigger.
%allchans: 1 if plotting all channels, 0 otherwise
%count = 1 if stand alone plot, or row number of stacked plots
%reduce: 0 plots anything; 1 kill the plot if no activity shown

%OUTPUT:
% Plotted: 0 if not plotted, 1 if plotted (based on reduce)

Pool=[];
reduce_sensitivity = 5;
%(min number of spikes in a histogram bin required for a plot if reduce is enabled)

    eval(sprintf('Spk = Spk_matrix.probe%d.ch%d;',probe,ch));

    % get trigger times from A1 firings from previous set probe data
    eval(sprintf('Trigger=Spk_matrix.probe%d.ch63(:,1)-.5;',probe));

    for j=1:75
        Spk_T=Spk(j,:)-Trigger(j);
        Spk_T=Spk_T(find(Spk_T<T_Total & Spk_T>-40 & Spk_T ~= 0));

        eval(sprintf('plotX.iter%d = (Spk_T(:,:));',j));
        eval(sprintf('Spk_trigger.trig%d=plotX;',j));
        eval(sprintf('Pool=[Pool,plotX.iter%d];',j));
    end;

    n = hist(Pool,T_Total*6);
    plotted = 0;

    if( (reduce == 0) | (max(n)>reduce_sensitivity))

        plot(plotX.iter2,2*ones(1,size(plotX.iter2,2))+allchans*count*75,'k.','Markersize',2);
        hold on;

        text(-35,allchans*count*75+18,sprintf('ch%d',ch));
        for i=1:75

```


A.5 sortSpikeData.m

```

function
[Pool,plotted,n]=sortSpikeData(Spk_matrix,ch,probe,T_Total,allchans,count,reduce)

%Rearrange the spike data(Spk_matrix) of certain channel(ch) by trigger order.
%And plot the RASTER PLOT.

%INPUT:
% Spk_matrix = array with dimensions (65,4,75,x);
% Spk_matrix (ch,stim channel of prob,iter,spikecount)
%ch: hardware channel number.
%stimch: 1:4 value of probe channel
%T_Total: time length (msec) after trigger.
%allchans: 1 if plotting all channels, 0 otherwise
%count = 1 if stand alone plot, or row number of stacked plots
%reduce: 0 plots anything; 1 kill the plot if no activity shown

%OUTPUT:
% Plotted: 0 if not plotted, 1 if plotted (based on reduce)

global probetype;

A1=63;
Pool=[];
reduce_sensitivity = 5;
%(min number of spikes in a histogram bin required for a plot if reduce is enabled)

    eval(sprintf('Spk = Spk_matrix.probe%d.ch%d;',probe,ch));

% get trigger times from A1 firings from previous set probe data
eval(sprintf('Trigger=Spk_matrix.probe%d.ch63(:,1);',probe));

for j=1:50
    Spk_T=Spk(j,:)-Trigger(j) + .002*abs(probetype(probe));
    Spk_T=Spk_T(find(Spk_T<T_Total & Spk_T>-40 & Spk_T ~= 0));

    eval(sprintf('plotX.iter%d = (Spk_T(:,:));',j));
    eval(sprintf('Spk_trigger.trig%d=plotX;',j));
    eval(sprintf('Pool=[Pool,plotX.iter%d];',j));
end;

n = hist(Pool,T_Total*10);
plotted = 0;

```

A.6 plot_hist.m

```

function plot_hist (Spk_matrix,Time,Channel)
% plot raster plot and histogram for single channel (or an array of channels).
% One plot is made for each channel, and for of the 4 probes.
% input spike matrix, time in msecs, and hw channel #
% note, only raster plots where bins have more than 5 points will be shown;

global probetype;

figure
a = gcf;
hold on
histmax(1) = 0;

for stimch=1:29
    for i=1:size(Channel,2)

        Ch=Channel(i);
        [Spk_trigger,count]=sortSpikeData(Spk_matrix,Ch,stimch,Time,0,1,0);
        [n,xout] = hist(Spk_trigger,Time*15);
        histmax(stimch+1) = histmax(stimch) + max(n) + 2;
        plot(-300:.5:400,histmax(stimch),'r');
        plot(xout-abs(probetype(stimch)),n+histmax(stimch),'b-');
        text(-150,histmax(stimch)+8,int2str(probetype(stimch)));

    end;
end;

title(sprintf('response of chan %d to dual channel probing',Channel))
xlabel('Time(ms)')
text(-240,histmax(1)+8,'ch A+B');
text(-240,histmax(2)+8,'ch A');
text(-240,histmax(3)+8,'ch B');
saveas(gcf,sprintf('histograms-ch%d full.jpg',Channel));

xlim([-15 50])
text(-13,histmax(1)+8,'ch A+B');
text(-13,histmax(2)+8,'ch A');
text(-13,histmax(3)+8,'ch B');
for stimch=1:21
    text(-6,histmax(stimch)+8,int2str(probetype(stimch)));
end
saveas(gcf,sprintf('histograms-ch%d blow up.jpg',Channel));

```

Appendix B – animat Code

Construction of an animat to track and follow a reference object at a given distance

The code in this appendix provides sample Matlab scripts used for the implementation of the animat, which were mostly utilized in chapters 7 and 8.

- B.1) run.m starts and runs a set of 20 animat trials
- B.2) animat.m is a behavioral trial of the animat (this is the main animat function)
- B.3) printObj.m prints the reference object when making animat graphs
- B.4) printRobot.m prints the robot when making animat graphs
- B.5) decodeNeur determines a directional vector for a given set of IPI data, specifying the direction the animat should move
- B.6) histTrain.m builds the lock histograms for lock-key decoding
- B.7) simstimIPI.m in simulation mode, this function returns data obtained during parameter tuning
- B.8) trainMappings.m tunes all parameters; builds input output mappings
- B.9) testDecode2.m tests the accuracy of the lock and key against tuning data
- B.10) stimIPI.cpp is a matlab mex function to interface matlab with both, the neural stimulation hardware, and meabench spike reading software. This code allows matlab functions to stimulate the dish with an IPI and returns real-time spike data during and following the IPI stimulation

B.1 run.m

% matlab script to tune parameters and execute a behavior trial of the animat

trainMappings

```
numtrials = 20;  
global NUMSTEPS;  
NUMSTEPS = 40;
```

```
histX = zeros(numtrials,NUMSTEPS);  
histY = zeros(numtrials,NUMSTEPS);  
dist = zeros(numtrials,NUMSTEPS);
```

```
for ( i = 1:numtrials)  
    [a,b,c,d,e] = animat;  
    histX(i,:) = a;  
    histY(i,:) = b;  
    dist(i,:) = c;  
    turnAccuracy(i) = d;  
    approachAccuracy(i) = e;  
end
```

B.2 animat.m

```

function [robotHistoryX,robotHistoryY,distHistory,turnAccuracy,approachAccuracy] =
animat()

% Simulated Animat

%Output:
%   robotHistoryX -> x coordinate at each step
%   robotHistoryY -> y coordinate at each step
%   distHistory -> distance at each step
%   turnAccuracy -> % of steps in which animat turned left / right properly
%   approachAccuracy -> % of steps in which animat moved forward / back properly

% initial params
global Spk_matrix;
global IPldiff;
global minLeftIPI;
global minRightIPI;
global COUNTTIME;
global MAXMOVEMENT;
global CH1;
global CH2;
global NUMSTEPS; % number of steps

ObjX = 0; % initial object X coord
ObjY = 85; % initial object Y coord
DesiredDist = 12; % desired distance to maintain

RobotX = -85; % initial X of robot
RobotY = -50; % initial Y of robot
RobotTheta = 0; % initial heading of robot;

arenaSize = 100; % X = +/- 100, Y = +/- 100

turnAccuracy = []; %counters for accuracy ratings
approachAccuracy = [];

%%%%%%%%%
close all
figure

% Plot initial animat situation
hold off
printRobot(RobotX,RobotY,RobotTheta);
axis([-arenaSize,arenaSize,-arenaSize,arenaSize])
hold on
printObj(ObjX,ObjY);

```

```

for step = 1:NUMSTEPS

    relX = ObjX-RobotX;
    relY = ObjY-RobotY;

    ObjDist = ((relX)^2 + (relY)^2)^.5;

    distHistory(step) = ObjDist;
    robotHistoryX(step) = [RobotX];
    robotHistoryY(step) = [RobotY];

    ObjTheta = atan2(relY,relX); % real angle of robot to object
    if ObjTheta < 0
        ObjTheta = ObjTheta + 2*pi
    end

    %relative angle of robot heading to object
    relTheta = RobotTheta - ObjTheta;

    if relTheta < -pi
        % set -pi < relTheta < pi
        relTheta = relTheta + 2*pi;
    end
    if relTheta > pi
        % set -pi < relTheta < pi
        relTheta = relTheta - 2*pi;
    end

    % Map object location to IPI
    LorR = (relTheta < 0) + 1; % 1 for L, 2 for R
    global mapInX;
    global mapInY;
    IPI = interp1(mapInX(LorR,:),mapInY(LorR,:),ObjDist,'linear','extrap')

% comment this block to run in simulation mode
    %Stimulate with IPI
    testdata = stimIPI(CH1,CH2,IPI);
    str = 'trueIPI = min(testdata.time( find(';
    findstr = 'testdata.ch == 60';
    findstr = [findstr,'& testdata.time > (abs(IPI)-2) '];
    findstr = [findstr,'& testdata.time < (abs(IPI)+2) )]);';
    eval([str,findstr]);
    if (IPI < 0)
        trueIPI = trueIPI * -1;
    end
    testdata.time = testdata.time - abs(trueIPI);
    testdata.IPI = trueIPI
    Spk_data = testdata;

% uncomment these to run in simulation mode
%     IPI = IPI + round(rand(1)*100)

```

```

% Spk_data = simstimIPI(Spk_matrix,IPI);

%Measure effect of stimulation, and decode into "motor" output
b = decodeNeur(Spk_data,IPIdiff);

if ((b(1) == 1 & IPI > 0) | (b(1) == -1 & IPI < 0))
    turnAccuracy(end+1) = 1
else
    turnAccuracy(end+1) = 0
end
if (((IPI<minLeftIPI) & b(2) ==1) | ((IPI>minRightIPI) & b(2) ==1) )
    approachAccuracy(end+1) = 1;
elseif ( (IPI>minLeftIPI) & (IPI<minRightIPI) & (b(2) == -1) )
    approachAccuracy(end+1) = 1;
else
    approachAccuracy(end+1) = 0;
end

%Turn left or right? (turn by +/- pi/6 at each iteration)
RobotTheta = RobotTheta + b(1)*pi/6

%Move forward or back?
dirVecY = b(2);

%How much do we move ?
global mapOutX;
global mapOutY;
trialPost = length(Spk_data.time(find(Spk_data.time>0))) / (60*COUNTTIME)
vecMagnitude = interp1(mapOutX(LorR,:),mapOutY(LorR,:),trialPost,'linear','extrap')
if (vecMagnitude < 0)
    vecMagnitude = 0;
elseif (vecMagnitude > MAXMOVEMENT)
    vecMagnitude = MAXMOVEMENT;
end

dirVecY = dirVecY*vecMagnitude;

RobotX = RobotX + dirVecY*cos(RobotTheta);
RobotY = RobotY + dirVecY*sin(RobotTheta) ;

hold off
printRobot(RobotX,RobotY,RobotTheta);
axis([-arenaSize,arenaSize,-arenaSize,arenaSize])
hold on

% move object for later trials
if (step>17)
    ObjY = ObjY - 4;
    ObjX = ObjX + rand*6-2
end

printObj(ObjX,ObjY);

```

```
        % saveas(gcf,sprintf('animat%d.bmp',step))
    pause(3);
end

turnAccuracy = mean(turnAccuracy);
approachAccuracy = mean(approachAccuracy);
```

B.3 printObj.m

```
function printObj(x,y)
% prints the object position in the animat display

objSize = 8;

o2 = objSize/2;

ptX = [-o2,-o2,o2,o2,-o2];
ptY = [-o2,o2,o2,-o2,-o2];

fill(ptX+x,ptY+y,'r')
axis([-100,100,-100,100])

end
```

B.4 printRobot.m

```
function printRobot(x,y,theta)
%prints the robot in the animat display

robSize = 10;
ptY(1) = robSize*sin(theta);
ptX(1) = robSize*cos(theta);
ptY(2) = robSize*sin(theta+(pi/2))/3;
ptX(2) = robSize*cos(theta+(pi/2))/3;
ptY(3) = robSize*sin(theta-(pi/2))/3;
ptX(3) = robSize*cos(theta-(pi/2))/3;
ptY(4) = ptY(1);
ptX(4) = ptX(1);
ptY(5) = 0;
ptX(5) = 0;

offsetX = .5*robSize*cos(theta+pi);
offsetY = .5*robSize*sin(theta+pi);
ptX = ptX+offsetX+x;
ptY = ptY+offsetY+y;

close all
figure
hold on
fill(ptX,ptY,'b')
plot(x,y,'rx')
axis([-100,100,-100,100])

end
```

B.5 decodeNeur.m

```

function predicted = decodeNeur (Spk_data,IPIdiff)

global chHist;
global chList;
global minLeftFULL;
global minRightFULL;
minFULL = [minLeftFULL,minRightFULL];
global HISTBINSIZE;
global HISTTIME;
global COUNTTIME;

% Function decodeNeur attempts to decode data from current trial, to classify
% into 1 of 4 directional vector quadrants

% input:
% Spk_data: structure representing current trial data.
%   ch contains the channel of each spike,
%   time contains time of each spike.
% chHist: trained histograms for ind. chans to compare current data to
% chList: list of channels to pay attention to when looking at histograms
% minFull: minFull has averaged reaction including during IPI at min of ANR
%   minFull(1) is for left (-) side, and minFull(2) is for right (+) side
% histBinSize: we assume hists are divided into histBinSize*Time bins
% Time: # of ms to use for histogram analysis (usually ~ 20)
% fullTime: # of ms to use for minFull analysis (usually ~ 100)

% output:
% predicted(1) == 1 -> -IPI (turn left)
% predicted(1) == -1 -> +IPI (turn right)
% predicted(2) == -1 -> move back
% predicted(2) == 1 -> move forward

% get trial data
trialTotal = 0;

trueIPI = Spk_data.IPI;

% get full spike count
trialFull = length(Spk_data.time) / (60);

% get post IPI spike rate
trialPost = length(Spk_data.time(find(Spk_data.time>0))) / (60*COUNTTIME);

% find spike times for hist analysis
findIndex = find(Spk_data.time > 0 & Spk_data.time < HISTTIME);
Spk_data.time = Spk_data.time(findIndex);
Spk_data.ch = Spk_data.ch(findIndex);

```

```

for ch = chList
    trialData(['ch',int2str(ch)]) = Spk_data.time(find(Spk_data.ch==ch));
end

%compare trial data to trained probes

probeSum = 0;

for ch = chList
    histo = chHist.group1(['ch',int2str(ch)]) - chHist.group3(['ch',int2str(ch)]);
    histo = sign(histo) .* ((100*histo).^2);

    t = trialData(['ch',int2str(ch)]);
    t(end+1) = 0;
    t(end+1) = HISTTIME;
    t = hist(t,HISTBINSIZE*HISTTIME);

    val = t * (histo');
    probeSum = probeSum + sum(val);
end
end

if (probeSum > IPIdiff)
    predicted(1) = -1;
else
    predicted(1) = 1;
end

if(predicted(1) == 1)
    predicted(2) = trialFull > minFULL(1);
else
    predicted(2) = trialFull > minFULL(2);
end

if (predicted(2))
    predicted(2) = 1;
else
    predicted(2) = -1;
end

predicted(3) = probeSum

```

B.6 histTrain.m

```

function [chHist,chList] = histTrain()
global Spk_matrix;
global HISTBINSIZE;
global HISTTIME
% Develop histograms of spike data on single channels, following either very negative
(group 1)
% or very positive (group 3) IPI stimulation.

% Input:
% Spk_matrix: spike data
% Spk_matrix.group[1,3].trial[i] is an array of spike times for a given trial following
% the first stimulation of an IPI pair. The first value of this array is the assumed
% IPI in ms. Note that the actual IPI may vary slightly (< 1 ms)
% HISTBINSIZE: factor to divide bins for histogram. We will use HISTBINSIZE*Time
bins for the
% histogram which spans Time ms.
% Time: # of ms following second probe in IPI pair for which we construct a histogram

% Output:
% chHist: histogram data
% chHist.group[1,3].ch[i] contains number of spikes that fall in a given time bin for a
% given channel for a given group
% chList: these are the only channels that have reasonable activity on them, and are
thus
% worth looking at later

totalSpikes = zeros(1,29);
chList = [];

% rearrange data into more usefull form, by channel
for group=[1,3] % Do left IPI, and then right IPI
    group = ['group',int2str(group)];
    %store normalized data for each channel - 0 to Time ms post second IPI
    for(ch = 1:60)
        a = [Spk_matrix.(group).time];
        b = [Spk_matrix.(group).ch];
        c = a(find(b == ch & a>0 & a< (HISTTIME)));
        c(end+1) = 0;
        c(end+1) = HISTTIME;

        aHist = hist(c,HISTTIME*HISTBINSIZE);
        if (max(aHist(2:HISTTIME*HISTBINSIZE-2)) > 10) % only keep channels with
activity on them
            % figure;
            % bar(aHist);
            if (isempty(find(chList == ch)))
                chList = [chList,ch];
            end
        end
    end
end

```

```
end

    % Normalize by dividing hist values by # of spikes on that channel

end
if (length(c) > 0)
    chHist.(group).(['ch',int2str(ch)]) = aHist ./length(c);
end
end
end

chList = chList(find(chList<=60)); % ignore analog chans 61-64
```

B.7 simstimIPI.m

```
function y = simstimIPI(Spk_matrix, IPI)
    % function to allow execution in simulation mode; returns real data at
    % approximately (randomized) value near IPI we are trying to stimulate with

    GROUPECUTOFF = 400;

    if (IPI < -1*GROUPECUTOFF)
        group = 'group1';
    elseif (IPI < GROUPECUTOFF)
        group = 'group2';
    else
        group = 'group3';
    end

    z = [Spk_matrix.(group).IPI];
    a = find(z>IPI);

    if ~isempty(a)
        [c,d] = min(z(a));
        i = a(d);
        testdata = Spk_matrix.(group)(i);
    else
        [c,d] = max(z);
        testdata = Spk_matrix.(group)(d);
    end

    y = testdata;
```

B.8 trainMappings.m

```

% script for training the animat
% Run this before doing behavioral animat experiments
% This will set all the training parameters

% This script takes ~ 30 minutes to execute

% Script builds:
  %input mappings (distance from object -> IPI)
  %output mappings (ANR -> distance to move)
  %Histograms charecterising precisely timed spikes
  % to help distinguish + from - IPI stimulations

%
% wait for CLOCK milliseconds between stimulations, stimulate on two channles
% with varying IPI at each clock tick
% repeat this n_trials times, then create input/output mappings

global CH1; CH1 = 46;
global CH2; CH2 = 72;
global NUMTESTS; NUMTESTS = 400; % Number of times to test IPI response curve
global OUTMIN; OUTMIN = .05; % percent of ANR to map to min movement
    %(skip flat area where ANR < OUTMIN% of max)
global BACKMAX; BACKMAX = .50; % percent of ANR to map to maximum backward
movement
global MAXMOVEMENT; MAXMOVEMENT = 15; % Largest possible robotic movement
in a single step
global DESIRED_DIST; DESIRED_DIST = 12; % Desired distance to keep between
robot and
    %reference object
global SENSITIVITY; SENSITIVITY = 4;
global MAPSIZE; MAPSIZE = 200; % coordinates go from X = Y = 0:MAPSIZE
global MAXIPI; MAXIPI = 1000; % Maximum IPI that can be applied
global HISTTIME; HISTTIME = 20; %number of ms to examine spikes on single
channels to
    %distinguish +/- inputs
global HISTBINSIZE; HISTBINSIZE = 5; % histogram of ind chans split into
    %HISTBINSIZE * HISTTIME bins
global GROUPECUTOFF; GROUPECUTOFF = 300;

global COUNTTIME; COUNTTIME = 100; % ms to count

global L; L = 1;
global R; R = 2;

global Spk_matrix;
global chHist;
global chList;

```

```

goodtest = 1;

for (i=[401:NUMTESTS])
    i
    time_remaining = 3.4*(NUMTESTS-i)/60

    pause(3);

    % randomly (evenly) assign IPI
    % half the trials will be IPI between +/- 1000
    % and half will be between +/- 500. This will place 2x more
    % trials in the "important areas" of our IPI curve (<500 ms)
    if mod(i,2) == 0
        IPI = round(rand(1) * 2000 - 1000);
    else
        IPI = round(rand(1) * 800 - 400);
    end

    if IPI < -1*GROUPECUTOFF
        group = 'group1';
    elseif IPI < GROUPECUTOFF
        group = 'group2';
    else
        group = 'group3';
    end

    % Stimulate with given IPI
    testdata = stimIPI(CH1,CH2,IPI);

    % Measure spikes to COUNTIME ms post IPI

    % precise IPI (should be ~= IPI)
    str = 'trueIPI = min(testdata.time( find(';
    findstr = 'testdata.ch == 60';
    findstr = [findstr,'& testdata.time > (abs(IPI)-2) '];
    findstr = [findstr,'& testdata.time < (abs(IPI)+2) )]);'];
    eval([str,findstr]);
    if (IPI < 0)
        trueIPI = trueIPI * -1;
    end
    testdata.IPI = trueIPI;

    if (~isempty(trueIPI))
        testdata.IPI = trueIPI
        if (~isempty(testdata.time))
            testdata.time = testdata.time - abs(trueIPI);
        end

        Spk_matrix.(group)(end+1) = testdata;

```

```

%Store IPI value
spkIPI(i) = IPI;

% store number of spikes/ms/chan including during IPI
spkFull(i) = length(testdata.time) / 60;

% store number of spikes/ms/chan following second pulse of IPI
spkPost(i) = length(find(testdata.time > 0)) / (COUNTTIME*60);
end
if (mod(i,100) == 0)
    eval(['save Animat-%s',datestr(now,30)]);
end
end

%save data
eval(['save Animat-%s',datestr(now,30)]);

% y is a matrix with all test data for NUMTESTS training points
% Use this data to develop linear input-output mappings

spkFull = spkFull(find(spkPost ~= 0));
spkIPI = spkIPI(find(spkPost ~= 0));
spkPost = spkPost(find(spkPost ~= 0));

% measure response following IPI
yPost = smooth(spkIPI,spkPost,round(NUMTESTS*.1));

% measure response during and following IPI
yFull = smooth(spkIPI,spkFull,round(NUMTESTS*.1));

figure;
plot(spkIPI,spkPost, '.');
figure
plot(spkIPI,spkFull, '.');

[spkIPI,i] = unique(spkIPI);
yPost = yPost(i);
yFull = yFull(i);

figure;
plot(spkIPI,yPost);
figure;
plot(spkIPI,yFull);

% c(i) = interp1(x,y2,i)

%ANFR = averaged neural response
%These should be at ~ IPI 1000
maxLeftANR = max(yPost(find(spkIPI<100)));
maxRightANR = max(yPost(find(spkIPI>100)));

```

```

% These should be at ~ IPI = 200
[minLeftANR,minLeft_i] = min(yPost(find(spkiPI<0)));
[minRightANR,minRight_i] = min(yPost(find(spkiPI>0)));

global minLeftIPI;
global minRightIPI;
minLeftIPI = spkiPI(minLeft_i);
rightSpkiPI = spkiPI(find(spkiPI>0));
minRightIPI = rightSpkiPI(minRight_i);

% output - Mapping from ANR to Movement
% For each polarity (+/-) find linear coefficient to decode averaged
% neural activity into a movement vector.

% Map ANR to movement by: interp1([x1,x2],[y1,y2],x)z
global mapOutX;
global mapOutY;

mapOutX = zeros(2,2);
mapOutY = zeros(2,2);

%point 1
mapOutX(L,1) = (maxLeftANR - minLeftANR) * OUTMIN + minLeftANR;
mapOutX(R,1) = (maxRightANR - minRightANR) * OUTMIN + minRightANR;
mapOutY(L,1) = 0;
mapOutY(R,1) = 0;

%point 2
mapOutX(L,2) = maxLeftANR;
mapOutX(R,2) = maxRightANR;
mapOutY(L,2) = MAXMOVEMENT;
mapOutY(R,2) = MAXMOVEMENT;

% all output function points
% interpolate these points to get ANR to distance movement mapping
mapLeftOutX = mapOutX(L,:);
mapLeftOutY = mapOutY(R,:);
mapRightOutX = mapOutX(L,:);
mapRightOutY = mapOutY(R,:);

% input - Mapping from Distance (cm) to IPI
% split IPI in to 4 quadrants. We will use 4 separate linear mappings of
% distance from robot to object to IPI, each mapping corresponds to a
% quadrant. Quadrants are split so that +/- are divided, and there is a
% division where x is less than min or more than min

global mapInX;
global mapInY;

```

```

mapInX = zeros(2,5);
mapInY = zeros(2,5);

% point 1
% Maximum to backup is BACKMAX*MAXMOVEMENT
% points 1 and 2 correspond to backwards movement
findL = find(spkiPI<0 & spkiPI>=minLeftIPI);
findR = find(spkiPI>0 & spkiPI<=minRightIPI);
maxBackANR(L) = (maxLeftANR-minLeftANR) * BACKMAX + minLeftANR;
maxBackANR(R) = (maxRightANR-minRightANR) * BACKMAX + minRightANR;

mapInX(L,1) = 0;
mapInX(R,1) = 0;
mapInY(L,1) = interp1(yPost(findL),spkiPI(findL),maxBackANR(L));
mapInY(R,1) = interp1(yPost(findR),spkiPI(findR),maxBackANR(R));

% point 2
findL = find(spkiPI<0 & spkiPI>=minLeftIPI);
findR = find(spkiPI>0 & spkiPI<=minRightIPI);
minBackANR(L) = (maxLeftANR-minLeftANR) * OUTMIN + minLeftANR;
minBackANR(R) = (maxRightANR-minRightANR) * OUTMIN + minRightANR;

mapInX(L,2) = DESIRED_DIST;
mapInX(R,2) = DESIRED_DIST;
mapInY(L,2) = interp1(yPost(findL),spkiPI(findL),minBackANR(L));
mapInY(R,2) = interp1(yPost(findR),spkiPI(findR),minBackANR(R));

% point 3
% points 3 and 4 correspond to forward movement
findL = find(spkiPI<=minLeftIPI);
findR = find(spkiPI>=minRightIPI);

% allow for the function to be continuous by adding .01 to X
mapInX(L,3) = DESIRED_DIST+.01;
mapInX(R,3) = DESIRED_DIST+.01;
mapInY(L,3) = interp1(yPost(findL),spkiPI(findL),minBackANR(L));
mapInY(R,3) = interp1(yPost(findR),spkiPI(findR),minBackANR(R));

global minLeftFULL;
global minRightFULL;
minLeftFULL = interp1(spkiPI,yFull,mapInY(L,3));
minRightFULL = interp1(spkiPI,yFull,mapInY(R,3));
minFull = [minLeftFULL,minRightFULL];

% point 4
mapInX(L,4) = SENSITIVITY * DESIRED_DIST;
mapInX(R,4) = SENSITIVITY * DESIRED_DIST;
mapInY(L,4) = -1*MAXIPI;
mapInY(R,4) = MAXIPI;

% point 5 (limit - flat horizontal line, so same as p 4)
mapInX(L,5) = 1.5*MAPSIZE;

```

```
mapInX(R,5) = 1.5*MAPSIZE;
mapInY(L,5) = mapInY(L,4);
mapInY(R,5) = mapInY(R,4);

% Train IPI polarity identification by examining single channels and
% building histograms on these chans
'analyzing histograms'
[chHist,chList] = histTrain;
'done'
% fine tune parameter of histogram analysis
''

'testing data against histograms, fine tuning params'
global IPIdiff;
IPIdiff = testDecode2;
'done'
```

B.9 testDecode2.m

```

function IPIdiff = testDecode2()

    % decodes all training responses to make sure lock-key was effective; IPI diff can
    % be used to fine tune lock-key

    global Spk_matrix;
    global chHist;
    global chList;
    global minLeftFULL;
    global minRightFULL;
    global HISTBINSIZE;
    global HISTTIME;
    global COUNTTIME;

    rSum = [];
    ISum = [];
    for (group = [1,3])
        group = ['group',int2str(group)]
        corr.(group)(1) = 0;
        b = length(Spk_matrix.(group));
        for ( i = 1:b )
            a = Spk_matrix.(group)(i);
            predicted = decodeNeur2 (a,0);

            if (a.IPI > 0)
                rSum(end + 1) = predicted(3);
            else
                ISum(end + 1) = predicted(3);
            end
        end
    end

    correct(1) = mean(ISum)
    correct(2) = mean(rSum)
    IPIdiff = mean(correct);

```

B.10 stimIPI.cpp

```

/* matlab mex-function */
/* matlab requires interface to a C program, so wrappers are used */

/* stimulate neurons with a given IPI duration, at 2 given channels, then return data
during IPI period + 100 ms following */

#define MCSHARDWARE 1
#include "64CNS.H"
extern "C" {
#include "mex.h"
#include "matrix.h"
}

#include <stdio.h>
#include <unistd.h> /* close */

#include <stdlib.h>
#include <time.h>
#include <math.h>

#include <common/EasyClient.H> //Blocking version
#include <common/ChannelNrs.H>

/*****/

extern "C" static void getSpikes( mxArray *plhs[], int IPI);

class Trigwin {
private:

public:

    ~Trigwin(){

    }

    void getSpkwin(double *spikeArr[], int trigch, int IPI);
    //variables
    int *trigbin;
    int nbins;
    int spikecount[64];
    int binflag;
    int trigch;

};

```

```

void Trigwin::getSpkwin(double *spikeArr[], int trigch, int IPI)
{
    int ch, i=0;
    int totspikes=0;
    timeref_t winsize,offsetT;
    timeref_t t,spiket;
    timeref_t msec = 100;
    int count = 0;
    //Open spike client with no start signal need to begin data collection
    StreamClient<Spikeinfo,SpikeAux> source("spike", false);

    Spikeinfo buffer[1];

    if (IPI < 0)
        IPI *= -1;

    //wait for a trig event on Analog channel
    do {
        source.read(buffer,1);
        ch=buffer->channel;
        t=buffer->time;
    } while (ch!=trigch);

    winsize = t + (((timeref_t)IPI)+msec)*25;
    offsetT = t;

    //Read until next spike after window size
    do { // Window

        ch=buffer->channel; //HW order offset 0
        spiket=buffer->time; //in samples
        if (spiket >= offsetT) {
            spikeArr[0][i] = (double)ch;
            spikeArr[1][i] = (double)(spiket - offsetT)/25.0;
            (spikeArr[2][0])++;
            i++;
        }
        source.read(buffer,1);

    } while (spiket < winsize);

    return;
}

Trigwin *Spkwin;

static

```

```

void getSpks(double *spikeArr[], int trigch, int IPI)
{
    (*Spkwin).getSpkwin(spikeArr, trigch, IPI);
}

static
void stimIPI( int ch1, int ch2, int IPI, mxArray *plhs[])
{
    CNS64 Stim ;

    //Create connection to 64CNS neural stim board
    Stim.connect("/dev/ttyS0");
    Stim.reset();
    Stim.xsrc(0);
    Stim.sync(1);
    Stim.immediate(1);

    Spkwin = new Trigwin;

    Stim.IPIProbe(ch1,ch2,600,400,IPI);
    Stim.disconnect();

    getSpikes(plhs,IPI);

    return;
}

extern "C" {
    static void getSpikes( mxArray *plhs[], int IPI)
    {

        double *spikeData[3];
        double *Spkch;
        double *Spktime;
        double numspikes = 0;
        mxArray *tmp, *tmp2;

        spikeData[0] = (double*)mxMalloc(16000,sizeof(double));
        spikeData[1] = (double*)mxMalloc(16000,sizeof(double));
        spikeData[2] = (double*)mxMalloc(2,sizeof(double));

        getSpks(spikeData, 60, IPI);

        Spkch = spikeData[0];
        Spktime = spikeData[1];
        numspikes = (int)spikeData[2][0];
    }
}

```

```

    tmp = mxCreateDoubleMatrix(1,(int)numspikes,mxREAL);
    mxSetPr(tmp,Spkch);
    mxSetField(plhs[0],0,"ch",tmp);

    tmp2 = mxCreateDoubleMatrix(1,(int)numspikes,mxREAL);
    mxSetPr(tmp2,Spktime);
    mxSetField(plhs[0],0,"time",tmp2);

    return;
}

void mexFunction(
    int          nlhs,
    mxArray      *plhs[],
    int          nrhs,
    const mxArray *prhs[]
)
{
    int  ch1, ch2, IPI;
    const char *structstr[] = {"ch", "time"};

    /* Check for proper number of arguments */

    if (nrhs != 3) {
        mexErrMsgTxt("stimIPI requires three input arguments.");
    } else if (nlhs != 1) {
        mexErrMsgTxt("stimIPI requires one output arguments.");
    }

    /* Setup for stimulation */
    ch1 = (int)mxGetScalar(prhs[0]);
    ch2 = (int)mxGetScalar(prhs[1]);
    IPI = (int)mxGetScalar(prhs[2]);
    mexPrintf("\nStimulating with %d ms IPI . . .",IPI);

    /* Setup to return spike data */
    plhs[0] = mxCreateStructMatrix(1,1,2,structstr);
    nlhs = 1;

    stimIPI(ch1,ch2,IPI,plhs);

    mexPrintf(" done");
    return;
}
/* end of extern "C" */

```