6.111 Project Proposal

Dominik Martinez, Olek Peraire

October 2018

1 Overview

Autonomous robots that interact with people are becoming more capable and common. These robots often use FPGAs for navigation as well as communication controls. Two distinct advantages of FPGAs for autonomous mobile devices are that they can be made to be power and weight efficient.

Our project is a self-balancing, 2-wheeled, robot that can take voice commands through a microphone. While being a very simple robot compared to any robot that can be of practical use, our robot incorporates non-trivial dynamics and controls and simple voice recognition procedures, both key ingredients in more sophisticated robots. To develop out robot, we plan on using 2 FPGAs, one mounted on the robot to control it, and another external one that will communicate with the robot via bluetooth. By doing this, we are creating a scenario similar to that of an autonomous robot, where the robot is on its own, required to perform certain tasks, while while receiving commands from an outside source.

The controls FPGA will be a CMOD A7-35T due to its small size and low weight, as it will be mounted on the robot. At first, we plan to use a Teensy micro-controller in order to read the data transmitted in I^2C from the MPU-9250 IMU and transmit it to the FPGA via UART. We will implement a PID controller to stabilize the robot and keep it in an upright position. Upon receiving commands, the robot will execute them for a given amount of time, then it will stop and wait for a new command. The different parts of the controller will be added sequentially, starting with the proportional controller, followed by the derivative, and finally, the integral controller (as well as wheel encoders) will be added in order to track drift.

Voice recognition is a process that is increasing being explored in the software world, but for our purposes, we want to implement voice recognition in hardware. In practice, hardware based voice recognition is significantly cheaper than software based voice recognition and for certain applications may be preferable. Our voice controller module will use a common feature extraction method for audio, the Mel-frequency cepstral coefficients, and a common inference model used in speech-to-text, the hidden Markov model, to determine if a command is being spoken and what the command is.

2 Design

2.1 Goals

2.1.1 Baseline Goals

- Proper reading of data from the IMU into the controls FPGA with noise filter to get accurate attitude readings
- Proportional feedback controller, able to provide a proportional feedback loop to stabilize the robot
  - Generate PWM waves for given duty cycle
  - Output to a motor driver in order to move motors in proper direction.
- Communicate with voice command FPGA via bluetooth module
- Voice command module can recognize very simple commands such as "Go" and "Stop"

2.1.2 Expected Goals

- Derivative feedback controller, able to use ω reading from IMU
- Ability to rotate the robot while maintaining stability
  - Alpha-blending rotation input commands with commands from feedback controller
- Voice commands recognize multiple word commands such as "Go forwards", "Go backwards", "Turn right", etc.

2.1.3 Stretch Goals

- Integral feedback controller, able to track and correct for drift over time
  - Read data in from wheel encoder to know how far the robot has moved
- Read IMU data without use of a Teensy
- Voice module can recognize specific people's voices and react differently, such as only having one authorized voice for control

2.2 Block Diagrams

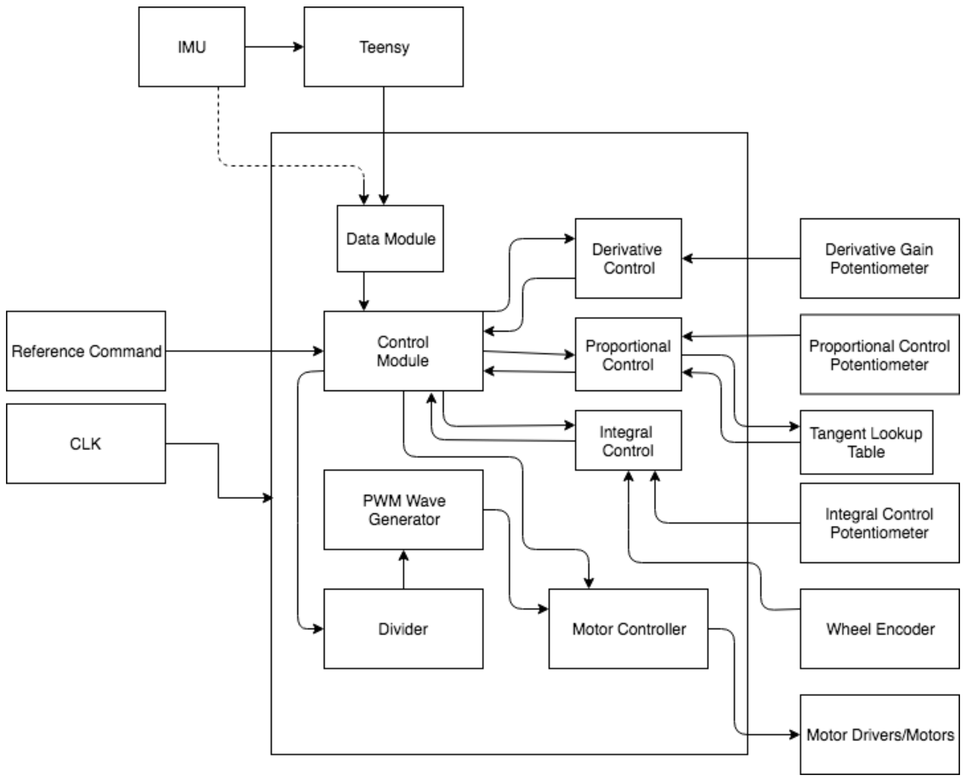

Figure 1: Block Diagram of Modules for Controller FPGA

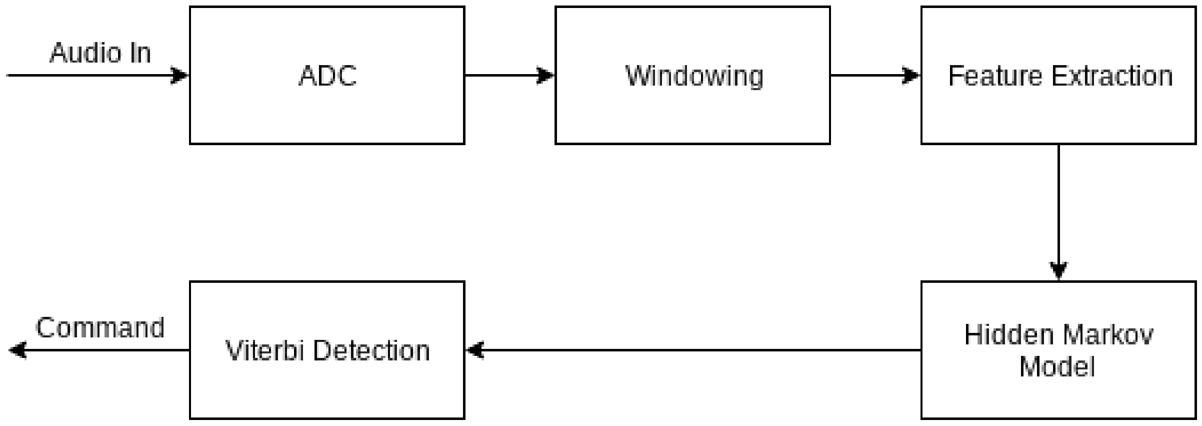

Figure 2: Block diagram for voice control

3 Module Implementation and Testing

3.1 Controls (Olek)

- **Control Module:** The control module will be the module at the top of the hierarchy on the robot's FPGA. It will first take in the gryoscope and accelerometer readings from the data module, as well as the commanded reference, and provide them to the proportional, derivative, and integral feedback controllers. It will then take in the readings from the proportional, derivative, and integral controllers and combine them to generate a direction and duty cycle for each motor. It will then aloha-blend these balancing commands with any turning commands that have been given. From here, the direction will be sent to the motor controller module, and the duty cycle will be sent to the divider module, which will then make its way to the PWM wave generator module.

  The only outputs this module creates are the duty cycle and direction of each motor. So, in order to test this module, we can use the Vivado ILA and simulate different inputs from the PID controllers. Then, we will be able to see if the direction and duty cycles for each motor are correct.

- **Data Module:** The data module will read in the data from the Teensy via UART and will output the necessary values to the control module. This module will be very similar to part of Lab 5c, so it will not require too much testing. However, because we will be reading in more values than in Lab 5c, we may need to test it using an LCD screen and code from that lab, along with a NEXYS4 FPGA. We will then be able to use our readings on the digital bubble level to see if they are correct.

- **Proportional Controller:** The proportional controller provides a feedback based on the angle θ offset from the reference command (vertical when no command is given). Because a gryoscope is not viable to use in the long term, and an accelerometer is not viable to use in the short term, we will use a form of sensor fusion, where we will get our θ from the following equation described in the lecture notes:

  $$\theta_y[n] = \beta(\theta_y[n-1] + Tg_y[n-1]) + (1-\beta)\tan^{-1}\left(\frac{a_z[n-1]}{a_x[n-1]}\right)$$

  Where β is a constant we can vary, and T is the length of a timestep. With this equation, we will be able to provide a feedback command based on the proportional gain, which will be set by a potentiometer and an analog input on the A7-35T. In order to find the values for the inverse tangent function, we will use a lookup table.

  In order to test this module, we will first make sure we are getting the correct readings in the Control Module, and we will then hold the IMU at different angles and can use the ILA in Vivado to see what commands the module is returning and confirm that they are correct.

- **Derivative Controller:** The derivative controller module will be used to provide feedback control based on ω from the IMU. Instead of differentiating θ, in order to preserve accuracy, we will be using the reading directly from the gryoscope. In order to filter the low frequency noise a gryoscope creates, we will be using some sort of high pass filter.We can then use a potentiometer and analog input on the CMOD A7-35T to vary the derivative gain, and provide derivative feedback control.

  To test this module, we can move our IMU at different angular velocities and use the ILA in Vivado to see if the module is returning reasonable values.

- **Integral Controller:** The integral controller module will be used to provide integral feedback control to the robot to prevent drift. Because the gyroscope and accelerometer readings lose fidelity over time, we will use wheel encoders in order to know how far the wheel has moved. By knowing how many unintentional revolutions have occurred, we can calculate how far we have unintentionally travelled, and return commands that will correct for that. This module is most useful over an extended period of time and requires the use of new hardware, the wheel encoder, so it is included in our stretch goals.

  Because this module will only be implemented once we have a self-balancing robot, we will be able to test it by forcing the robot forward while maintaining a vertical angle, and seeing if it corrects for the drift. If we have to test more in depth, we can manually turn the robot wheels and use the Vivado ILA to see what outputs the module is providing.

  Something else to note is that the CMOD A7-35T only has 2 analog inputs for the potentiometers, so we will be tuning the integral gain after the proportional and derivative gains have been set to work properly. At that point, the gains will only require fine tuning, so we hope to not have to vary them by hand too much once the integral controller is implemented.

- **Divider:** The divider module is used to provide a value to count to in the PWM wave generator module. We will be using the divider provided in the lecture notes, so we presume it will not require much testing. However, we can use the ILA in order to make sure it outputs the values corresponding to the duty cycle of the wave we want to generate.

- **PWM Wave Generator:** This module will take in a value from the divider module and will then generate a square wave with a duty cycle based on that value. This module will be similar to a clock, in how it is implemented, so we will be able to test it with the ILA.

- **Motor Controller:** The motor controller module will take take a PWM wave from the PWM wave module, as well as some information from the control module (such as direction) and output it to the motor drivers. To test this module, we can input constant values and check to see if the motors are turning accordingly.

3.2 Voice Commands (Dominik)

- **ADC** The ADC module is used to input data from a microphone into the FPGA, sample it at an appropriate frequency for the system, and appropriately filter out frequencies to avoid aliasing. We will use the labkit's LM4550 to sample audio at 48 kHz and then use a low-pass filter and down sample that signal to 16 kHz. This is basically identical to lab 5 and will require minimal if any testing.

- **Windowing** In order to properly process the audio signal we need to break the audio stream into smaller lengths of audio to properly process. The windowing module applies a Hamming window to the signal with interval of around 10-20 ns. We use a Hamming window to avoid introducing discontinuities in the signal, as we will be running a Fourier transform on this signal later. Testing for this will involve simulating the expected results on Matlab and then verifying that the signal from the FPGA setup is the same.

- **Feature Extraction** There are a few ways to do feature extraction on an audio signal for voice detection. For our purpose we will be using Mel-frequency cepstral coefficients (MFCC) as these are commonly used as features in voice recognition. In order to calculate the MFCCs we first take the Fourier transform of the windowed signal, map the FT onto the mel scale, take the logs of those powers, take the discrete cosine transform of the log powers, and the MFCCs are the amplitudes of the resulting spectrum. Similar to the previous module, testing for this involves simulating the function with Matlab and verify equivilent output on the FPGA.

- **Hidden Markov Model** A simple model for speech-to-text systems is using a Hidden Markov Model, where the features from the previous model are the observed sequence of events and the potential text of the spoken word is the underlying true chain. For our purposes, we can develop models for all the potential voice commands that we would want to use. This module takes the inputs from the feature extraction module, which are vectors, and inserts them into a HMM data structure for later processing. Testing for this involves writing test-benches with controlled inputs and comparing the output of the module to manually verified HMMs.

- **Viterbi Algorithm** Finally, we need a way to figure out what is the most likely command based off of the observed states as processed by the previous module. The Viterbi algorithm is a straight forward algorithm for computing the most likely sequence of hidden states in the HMM. It's commonly used in speech-to-text applications. The output of this module is a command signal to be sent to the controls module. Similarly to the previous module we can test this by hand writing tests with controlled values and verify against manually computed outputs.

4 Parts Needed

- CMOD A7-35T
- Segway Robot kit from 6.302
- Bluetooth Modules
- Potentiometers, wires, breadboard
- Battery to power teensy, CMOD A7-35T
- Wheel encoders
- MPU-9250 IMU
- Teensy
- Labkit