2017

Portable Wireless Measurement and Control System

Bryan Salisbury
University of Rhode Island, bryan_salisbury@uri.edu

PORTABLE WIRELESS MEASUREMENT AND CONTROL SYSTEM

BY

BRYAN SALISBURY

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING AND APPLIED MECHANICS

UNIVERSITY OF RHODE ISLAND

2017

MASTER OF SCIENCE THESIS

OF

BRYAN SALISBURY

APPROVED:

Thesis Committee:

Major Professor Musa Jouaneh

D.M.L. Meyer

Jason Dahl

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2017

ABSTRACT

Described within is an approach for a low cost data acquisition and a control platform which utilizes a custom Android application, Bluetooth wireless data-link, and Arduino development board. Scientists, students, and electronic hobbyists often need to conduct experiments outside of a laboratory environment. Scientists and engineers may be required to collect data in the field. Students often conduct experiments to satisfy the requirements of coursework. Hobbyists will often not have access to a well-equipped electronics lab. The commercially available data acquisition and control devices which perform the required functions for mechanical experimentation are often priced in the hundreds to thousands of dollars range. An Arduino based system could be used to measure analog signals between 0-5 volts, and control both digital and pulse width modulated outputs with the correct programming. The low cost of Arduino Boards and ubiquity of Bluetooth enabled computing devices make the combination an attractive option for use outside a laboratory environment. The addition of a wireless Bluetooth serial connection to the Arduino Board enables communication with an Android based measurement and control application. This work distinguishes itself from other Bluetooth control applications available on the Android Marketplace by prioritizing the retrieval and storage of data from the Arduino Board and by providing functionality to review data received. A program was developed for data collection and control which runs on the Arduino development board. A second program was developed which runs on an Android powered computing device, and acts as a client for the Arduino data collection and control program. Data was collected at rates up to 5000 samples per second, and a collection of simple sensor and motor circuits were controlled and measured using the developed programs. This study shows that data can be collected utilizing an Arduino Board, Bluetooth communication,

and Android mobile computing devices, of sufficient resolution to be useful for mechanical experimentation.

ACKNOWLEDGMENTS

I would like to acknowledge my Major Professor, Musa Jouaneh for his guidance and advice throughout the course of my graduate studies. Additionally, I extend thanks and gratitude to my family and friends for their support as this study was completed.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This study seeks to develop a general purpose tool for wireless measurement of sensors and control of simple mechanical devices. A primary goal is to provide capabilities measuring 0-5v analog sensors, and then accessing or reviewing the data on an Android mobile phone. The proposed tool should also be capable of controlling analog outputs in either an open loop or closed loop fashion. These capabilities would combine to form a flexible platform for data acquisition and control, that is both inexpensive and highly portable. Such a platform could be used to supplement academic pursuits, such as the URI Take-Home Kit for System Dynamics [1], but more importantly is valid for any home experimentation requiring analog inputs and outputs between 0-5v. The tool developed by this study would assist experimentators in gaining understanding of the characteristics presented by various mechanical systems. The application developed for the mobile computing device should allow review of collected data, as well as configuration of all operating parameters. The ability to review collected data immediately after testing will enable making small changes in experimental setup to review the change in system behavior. A tool meeting the goals for portable experimentation is presented within this work as a flexible and general purpose data acquisition and control toolkit, which is configurable to support a wide range of future applications.

Current approaches for experimental data acquisition require the use of expensive equipment which may not be suitable for use by students, or financially accessible to electronic hobbyists. Commercially available solutions attempt to solve the problem presented in this work, but do so at a greater cost and provide limited software for use on the Android device. Measurement Computing

Corporation (MCC) is one such company which sells a Bluetooth capable DAQ, the BTH-1208LS for $199.00 [2], a price nearly six (6) times that of the $34.90 Bluetooth capable Bluno [3]. A strong advantage of using the Arduino platform is compatibility with a number of shields such as motor controllers, and ultrasonic distance sensors which can extend out-of-the-box capabilities, and provide value to educators, students, and hobbyists beyond data acquisition [4].

The purpose of the Take-Home Experimentation kit is to increase student engagement and increase accessibility to coursework related laboratory experiments. The original Take-Home Kit consisted of three major components: control software, a custom data acquisition board, and a number of experimental setups [1]. The research proposed within this document replaces the control software with a variant which runs on the Android mobile phone or tablet system rather than a Window PC. An experimentation platform that operates on mobile devices is a valuable development for students, as computing is shifting from PC to mobile computers according to a February 2014 article in TIME Magazine which notes a shift from fixed PCs to mobility [5]. Other related works such as Kaufmann and Buechley's Amarino toolkit have been developed after recognizing a clear and growing need for interfacing mobile devices with Arduino based development kits [6]

This study replaces the second component of the URI Take-Home Kit: the custom PCB, with a commercially available Arduino compatible board. This project will specifically investigate use of the Bluno board as the onboard Bluetooth radio simplifies experimental setup. The custom-developed PCB for the original URI Take-Home Kit included 32KB of additional RAM, status LEDs, 5A H-Bridge driver chip, MAX232 serial communication chip, various connectors for experimental setups [1]. The Bluno (Arduino compatible) has only 2KB of onboard

RAM, includes the required status LEDs, a serial communication chip, and header pins. While the Bluno does not include an H-Bridge capable of driving the motor experimental setup, motor driver shields are commerically available. The limited onboard RAM represents a major limitation for designing the Arduino software. Since each analog to digital conversion is done with 10 bits of precision, the available 2KB of RAM will be capable of storing fewer than 1000 measurements to RAM. Care will be taken while programming the Arduino software to efficiently use the limited available memory while also developing the required programming logic. The Bluno and Android experimentation system would benefit from the development of a custom Arduino shield to provide the required experiment setup connections, otherwise the use of short wires to make jumper connections between the Bluno header pins and the experiment setup connectors would be required.

The Arduino is a flexible, low cost platform for development and testing which is well suited to use in an academic environment. The work proposed herein is significant as no existing community works, research, or commercially available products adequately fit the requirement of measurement and control in a single low cost package. This work will provide a platform for students to conduct laboratory experiments wirelessly with the use of their mobile phones. Wireless data collection to a mobile computer represents a significant advancement in the field from both an educational and a hobbyist perspective. Educators will be able to combine the developed software and methodologies with off the shelf hardware to record inputs and control voltage outputs. Since the hardware is inexpensive and available commercially more students can be supplied with experimental setup kits. The kits would include the necessary equipment and hardware to conduct a number of predefined laboratory experiments with the goal of supplementing educational efforts. This work extends a work already completed by the University of Rhode

Island as a teaching aide for Systems Dynamics and Vibrations courses, the Take-Home Experimentation Kit [1].

Serial communication on the Arduino Board Uno makes use of the built in USART capabilities of the ATmega328 chip. For the Arduino Board Uno digital pins 0,1 may be used for serial communication, or the USB to serial interface may be utilized.

Serial data for this project will be transmitted in an asynchronous fashion, since both the Arduino Board Uno and the Android mobile device rely on separate clock signals. Data is transmitted over the serial connections by breaking data to be transmitted into a series of 10-bit packets or data frames. Each of the 10-bit serial data packets, includes one start bit, 7 or 8 data bits, an optional parity bit if sending only 7 data bits, and one stop bit. Additionally, the number of data packets transmitted or received per second is controlled by configuring the baud rate which is specified in bits per second. Both sending and receiving device must be configured to use the same baud rate.

The Arduino Board Uno is documented to support serial communication at a maximum baud rate of 115200 bits per second. However, it is possible to specify any baud rate within the Arduino software. Arduino defaults to a serial protocol known as 8N1, which consists of a start bit, 8 data bits, and a stop bit without parity. Passing an integer of value '1023' (maximum analog sensor reading) to the Arduino serial print command will result in the transmission of four ASCII encoded characters, each in a single data packet and a new line data packet. Likewise, passing an integer of value '0' to the serial print command will result in one data packet being sent for the sensor reading, and one new line character [7].

List of References

[1] M. K. Jouaneh and W. J. Palm III, "Control systems take-home experiments," *IEEE CONTROL SYSTEMS MAGAZINE*, vol. 31, pp. 44–53, Aug. 2013.

[2] Measurement Computing Corporation. "Wireless multifunction daq device with android and windows support." Accessed: Dec 2, 2016. [Online]. Available: http://www.mccdaq.com/wireless-data-acquisition/BTH-1208LS.aspx

[3] DFRobot. "Bluno - an arduino bluetooth 4.0 (ble) board." Accessed: Dec 2, 2016. [Online]. Available: https://www.dfrobot.com/index.php?route=product/product&product_id=1044

[4] Arduino. "Hardware & related initiatives." Accessed: Dec 2, 2016. [Online]. Available: http://playground.arduino.cc/Main/SimilarBoards#goShie

[5] B. Bajarin. Time Inc. "The pc has been decentralized mobile is the new center." Accessed: December 10, 2016. [Online]. Available: http://time.com/4040/the-pc-has-been-decentralized-mobile-is-the-new-center/

[6] B. Kaufmann and L. Buechley, "Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing," in *12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10)*, New York, NY, USA, 2010.

[7] M. K. Jouaneh, *Fundamentals of mechatronics*. Stamford, CT: Cengage Learning, 2013.

CHAPTER 2

REVIEW OF LITERATURE

A full review of available literature was conducted prior to beginning this study. The review provided valuable insight into current approaches for Arduino to mobile phone communication, as well as identified limitations. Firstly, existing Bluetooth support libraries place an emphasis on communication from mobile phone to Arduino Board, and provide limited ability to return data to the mobile phone. Second, the hardware limitations of the Arduino Board concerning data-link bandwidth and memory present a signification design challenge.

A number of existing projects utilize Arduino, Android, and Bluetooth serial communication. The majority of these works have functionality developed only for sending data to the Arduino Board from the Android device, and do not provide a convenient mechanism for bidirectional communication.

The first work evaluated was the Amarino toolkit. Amarino is a comprehensive toolkit designed to enable Bluetooth communication between Arduino and Android computing platforms. This toolkit consists of an Android application, and Android API, and an Arduino Library called *MeetAndroid*. The Amarino Toolkit makes it easy for Arduino developers without any experience in Android programming, to obtain values from the Android device. Amarino also has a plug-in library which defines a number of Android events for which data can be sent back to the Arduino Board, for example: phone ringing, battery level, light sensor value, compass value, device orientation. The Amarino toolkit extends an excellent feature set to Arduino for retrieving data from the Android OS, however does not provide any functionality for saving data to the file system. Since Amarino doesn't provide a mechanism to save the received data, a custom Android application is

still required. [1]

Figure 1. Amarino Application [1]

One such example of a custom Android application is the Amarino example Sensorgraph. This Android application makes use of the Amarino Toolkit Plugin to transmit real time measurements from an analog input pin to the Android application, over a Bluetooth serial connection. The Android Sensorgraph application graphically displays the transmitted measurements on a plot versus time to show variation over time of the measured input. This application does not provide any functionality for saving recorded data to storage on the Android device, but is capable of viewing real time data. This project was designed and tested with the BlueSMiRF Gold modem for Bluetooth. [2]

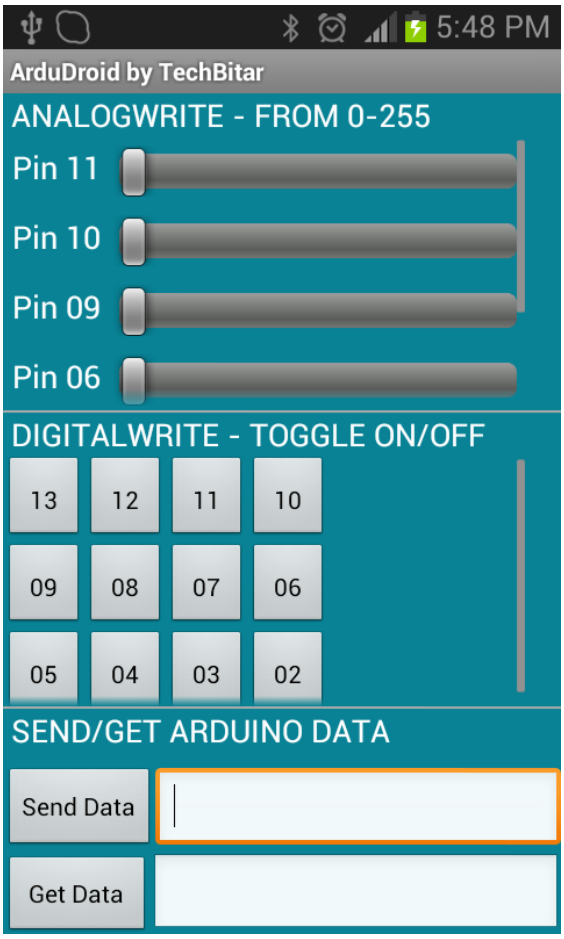

Figure 2. Ardudroid [3]

Ardudroid is an example of another Android application created with the intention of controlling an Arduino Board from an Android device. Ardudroid allows control of all analog and digital outputs. Ardudroid also makes it possible to retrieve a single string of data from the Arduino Board. [3]

One project found was functionally close to the requirements of this study. Practical Arduino - Oscillioscope / Logic Analyzer uses an Arduino Board for data acquisition. The Arduino Board is connected to a computer, using a USB connection and data is sent to the computer application. The website provides three (3) Arduino Sketch files which can be uploaded to change the behavior of the Scope: Analog, Digital, Optimized Digital. The Arduino sketch files are available

in the Practical Arduino Github repository, and are licensed under the GPLv3 software license. The Practical Arduino Oscilloscope is reported to work near 5KHz sampling frequency divided between all channels being monitored. Since the Arduino sketches are provided, and licensed under an open source license, the code provided could be utilized for this project. However, this project is not compatible with mobile computing devices and does not provide control functionality. [4]

Figure 3. Practical Arduino - Oscillioscope [4]

In reviewing the existing solutions and projects available for use it was apparent that to use the combination of Arduino Board and mobile computer for data acquisition and control, the creation of a specialized Android application and Arduino Sketch would be required.

List of References

[1] B. Kaufmann and L. Buechley, "Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing," in *12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10)*, New York, NY, USA, 2010.

[2] G. Richardson. "Sensorgraph." Accessed: April 10, 2015. [Online]. Available: http://bygriz.com/sensor-graph/#more-1344

[3] H. Bitar. "Ardudroid: A simple 2-way bluetooth-based android controller for arduino." Accessed: Feb 10, 2015. [Online]. Available: http://www.techbitar.com/ardudroid-simple-bluetooth-control-for-arduino-and-android.html

[4] Practical Arduino. “Oscilloscope / logic analyzer.” Accessed: April 15, 2015. [Online]. Available: http://www.practicalarduino.com/projects/scope-logic-analyzer

CHAPTER 3

METHODOLOGY

The first task required for completion of this study was selection of Bluetooth capable hardware. This considered multiple options for achieving Bluetooth connectivity, but ultimately settled on using the Bluno Board, an Arduino compatible development board with integrated Bluetooth Low Energy radio. The Bluno had an advantage over other methods of achieving Bluetooth connectivity as it was the only Arduino compatible board available which would not require add-on circuitry for wireless communication.

The study consisted of two software development efforts as well as an initial testing phase to characterize the data acquisition performance of the system. The first development effort was Arduino code for the measurement of analog inputs and subsequent code for the control of the analog outputs. A protocol for transmitting data from the Arduino platform to the Android mobile computing device was developed. The second software development effort was an Android application to act as a client for sending commands to the Arduino board. The Android application receives Bluetooth transmissions from the Arduino board and parses this data in order to save test results.

3.1 Theoretical Performance

Streaming ASCII characters requires one byte to be transmitted per digit sent over serial connection. This results in a variable, and unstable communication rate. The equations given below represent theoretical streaming bandwidth for a 115,200 baud rate serial connection.

$$[11520(packets/second)/5(packets/sample) = 2304(samples/second)] \quad (1)$$

It is possible to optimize the communication rate by sending the data as bytes rather than encoding the integer values of a 10-bit sensor reading into ASCII encoded text. Since each data packet may contain 8 bits of data (1 byte), it is possible to send the 10 bit measurement as only two packets, rather than the four packets required by ASCII encoded serial communication. Since there will be 6 unused bits in one of the two packets, those can be used to identify the measurement. An optimized communication protocol was developed which uses only 2 bytes for data packets, and 1 byte for new line separation.

The optimized communication format proposed has consistent bandwidth characteristics, and due to more efficient usage of available bandwidth will allow measurements to be sent to the Android phone at a much higher average sample rate:

$$[11520(packets/second)/3(packets/sample) = 3840(samples/second) \quad (2)$$

3.2 Arduino Sketch

The Arduino Sketch developed for this study is responsible for all measurement and control tasks as well as serializing the data and transmitting it back to the connected client. The Arduino Sketch takes a series of text based configuration parameters from a connected client. The sketch has distinct modes of operation: stateNull, stateSendBuf, stateMeasure, stateControl. Each state has a mode of operation named modeProgram which is used for configuring running parameters. This study has developed an Android client for use with the Arduino Sketch, but any client capable of sending the appropriate commands is compatible.

The application has four main states of execution which are configurable by the user to control behavior of the program, given by Figure 4. Each state contains the logic required for performing a distinct set of tasks. Transitions between the states are controlled by commands sent from the connected client application. Each state performs only a single task which is fully configurable by the connected client in order to create a flexible framework for general purpose data acquisition and control. Execution state and all running parameters are controlled by sending commands to the Arduino Board.

Valid case sensitive top level commands are given in Table 1. The top level commands are used to control execution state, as well as designate serial input as configuration data. Commands A, M, C, S correspond to states of program execution. Sending one of the state control commands will request that the Arduino application switch to the requested state as soon as possible. Serial input following the modeProgram command is returned to the current execution state for parsing. Each state has a unique routine for parsing configuration strings. Command K is used to implement a rudimentary flow control and retransmission mechanism for the Bluetooth serial connection. Command A, for stateNull is used as a stop command.

stateNull returns no data and stops all actions performed by the Arduino Code. This state accepts no configuration parameters, but does listen for user commands requesting state changes.

stateSendBuf returns all measured samples as a compressed packed array of bytes to the listening client. Each byte must be acknowledged by sending an ACK command to the Arduino Sketch, but accepts no configuration parameters

or control commands.

stateMeasure collects data points until the data buffer is full. This state listens for user commands requesting state changes and it accepts the parameters given in Table 2.

stateControl runs the open and closed loop control routines and returns the current value of the input channel every 100 milliseconds. This state listens for user state change requests and takes the parameters given in Table 3.

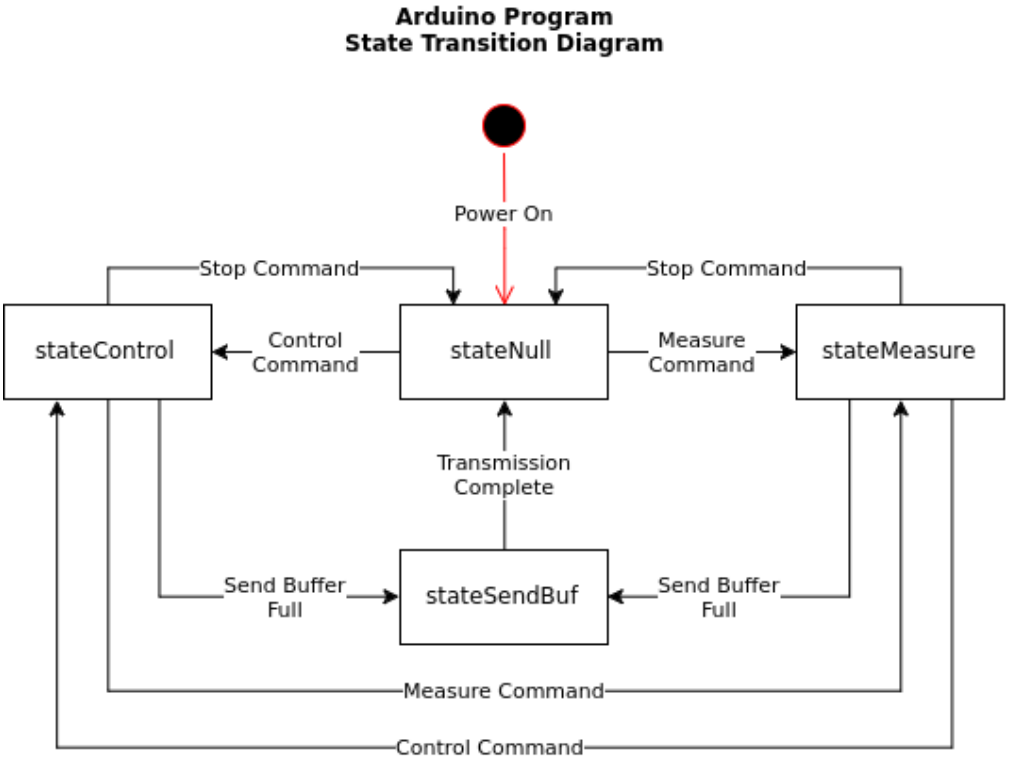

Figure 4. Arduino State Transition Diagram

Command	Action
A	stateNull
M	stateMeasure
C	stateControl
S	stateSendBuf
P	modeProgram
K	Sets ACK flag used by stateSendBuf

Table 1. Generic Commands

Command	Action
C	(int) Compression - 0=Off / 1=On
D	(int) preCount for timer overflow
S	(byte) measureMask. Bits 0-5 mapped to A0-A5. Set high to enable.

Table 2. stateMeasure Parameters

Command	Action
A	(float) Kp Gain
B	(float) Ki Gain
C	(float) Kd Gain
D	(int) Desired / Target value
E	(int) preCount for timer overflow
F	(int) Open or Closed loop control - Open=0 / Closed=1
H	(int) Max output value
L	(int) Min output value
O	(int) Designated output pin
I	(int) Designated input pin
M	(bool) Turn measurement on / off
S	Sets start flag

Table 3. stateControl Parameters

3.3 Android Application

An Android application was developed to interface with the developed Arduino. The Android application is capable of switching the Arduino code between the different execution modes as well as configuring the respective parameters. This Android application receives and saves data samples. The application additionally graphically displays the received data.

The Android Application consists of one screen to control each operating mode of the Arduino Sketch. Additionally, there is one activity present which is used for configuring default options. Source code for these activities is included in appendices B, C, D, E.

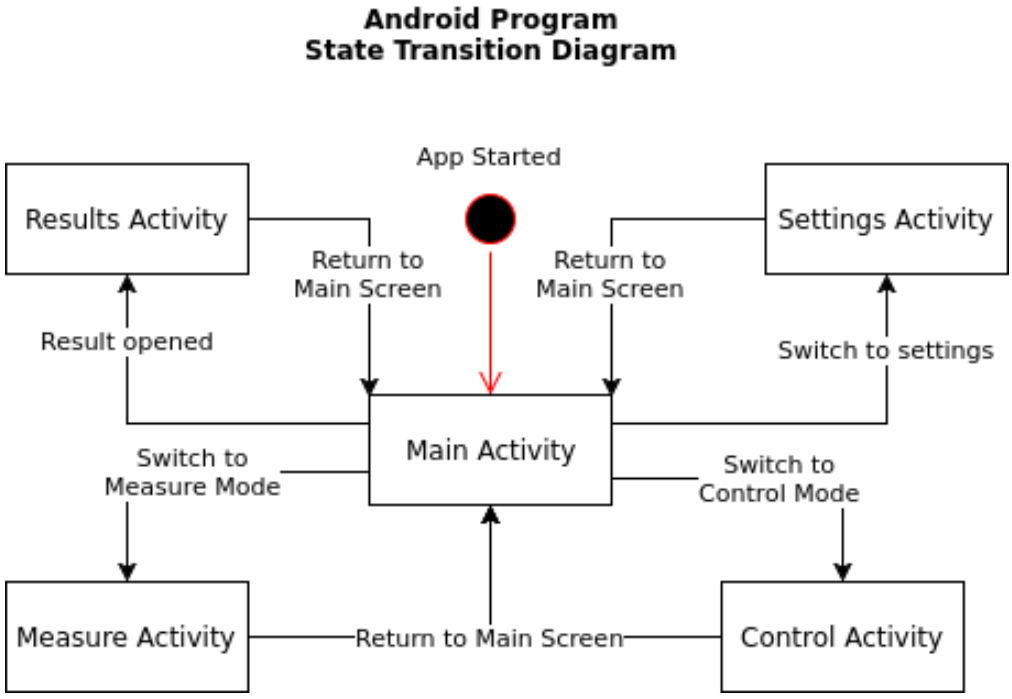

Figure 5. Android State Transition Diagram

The Android application is structured with an activity which correlates loosely with the execution states of the Arduino program. The application opens to a Main Activity screen which presents a listing of all available data collections. The Main Activity contains navigation providing access to Measure Mode, Control Mode, and a Settings Activity. MeasureMode connects to the Bluno Bluetooth Board and allows the user to indicate the appropriate configuration parameters for this data collection. MeasureMode sends configuration strings to the Bluno Board to configure measureMode, as well as listens for data received. MeasureMode contains logic for unpacking compressed measurements and saving data to an application inter-

nal database. ControlMode Activity has options for configuring the relevant PID control parameters, as well as choosing between open loop or closed loop control modes. Open loop control outputs signal of user specified duty cycle, while closed loop uses user supplied PID parameters to calculate required controller output. SettingsActivity is used to specify relatively constant parameters such Bluetooth device address, controlMode sampling frequency, and controlMode input/output pins. The resultsActivity graphically displays collected datasets and provides access to export data to any application capable of opening CSV data files on the Android system. Any dataset opened in the resultsActivity screen can be zoomed into for basic on phone analysis, but the exported data files can be brought into a scientific mathematics package such as Matlab or Octave any required computer analysis, including plotting or experimental system performance characterization.

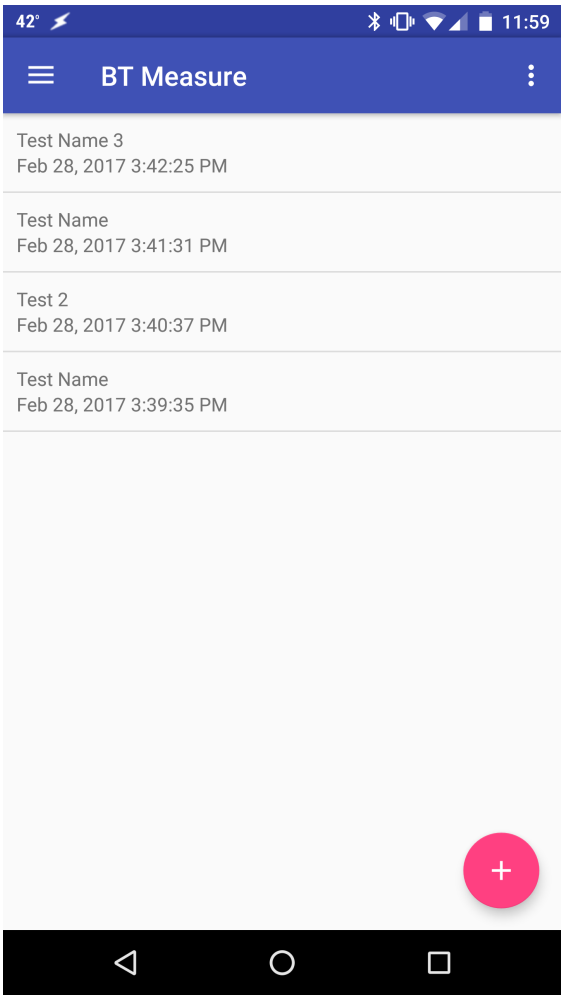

Figure 6. Android Application

The main screen of the developed Android application gives the user a view of all collected test data, as well as access to the various modes of operation and settings.

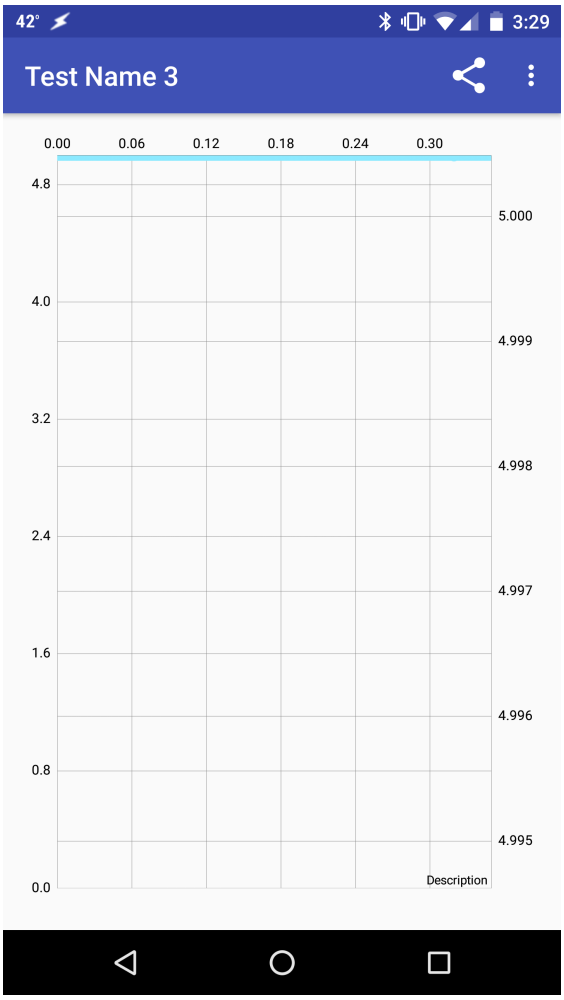

Figure 7. Results Activity

The Results activity gives the user access to collected data, as well options to delete or export any given dataset.

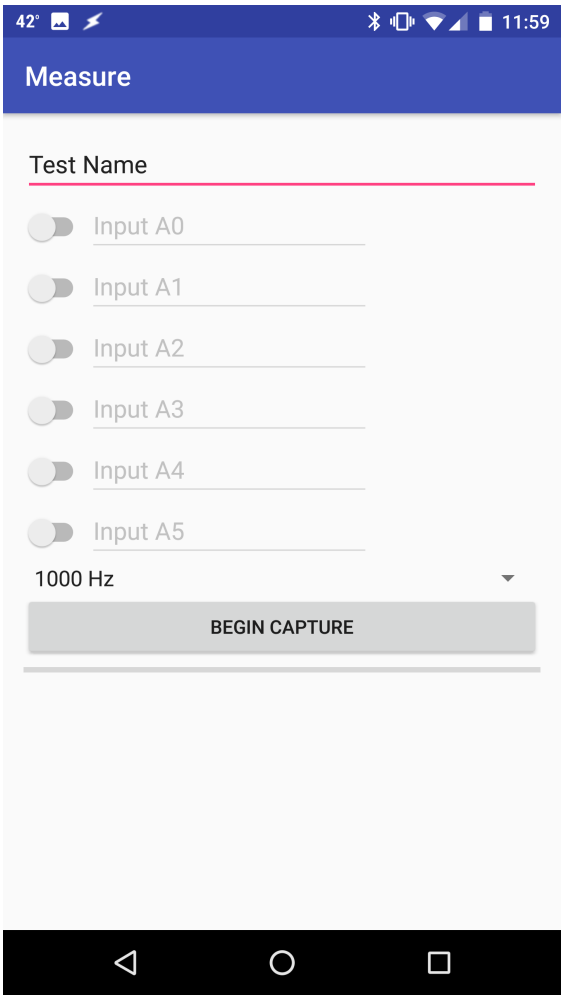

Figure 8. Measure Activity

The Measure Activity provides a user interface for configuring data collection options, and processes the incoming data stream to save samples.

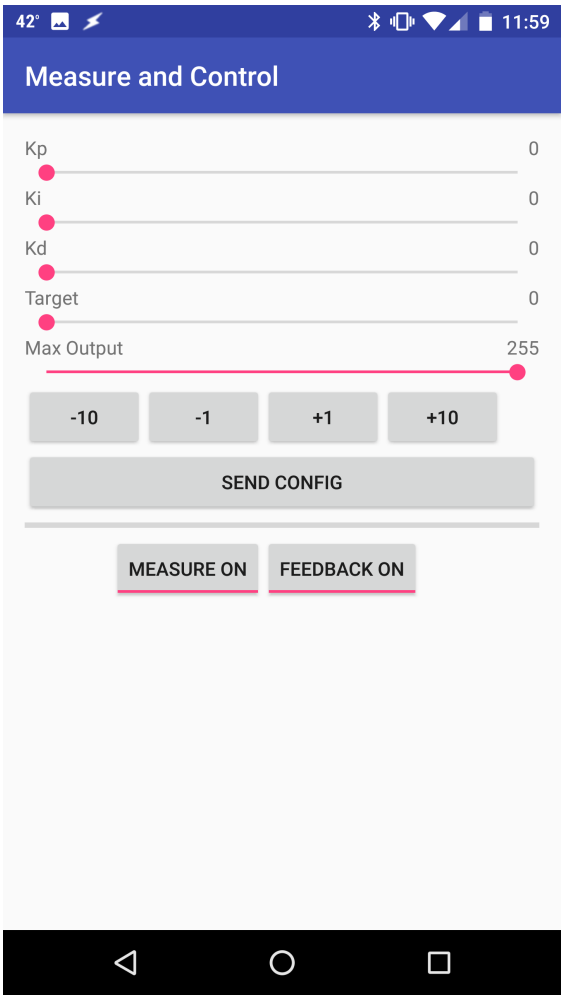

Figure 9. Control Activity

The Control Activity provides a user interface for configuring open and closed loop control options. The *Measure On* option is used for recording sensor data. The *Feedback On* option indicates Closed Loop control mode, versus Open Loop.

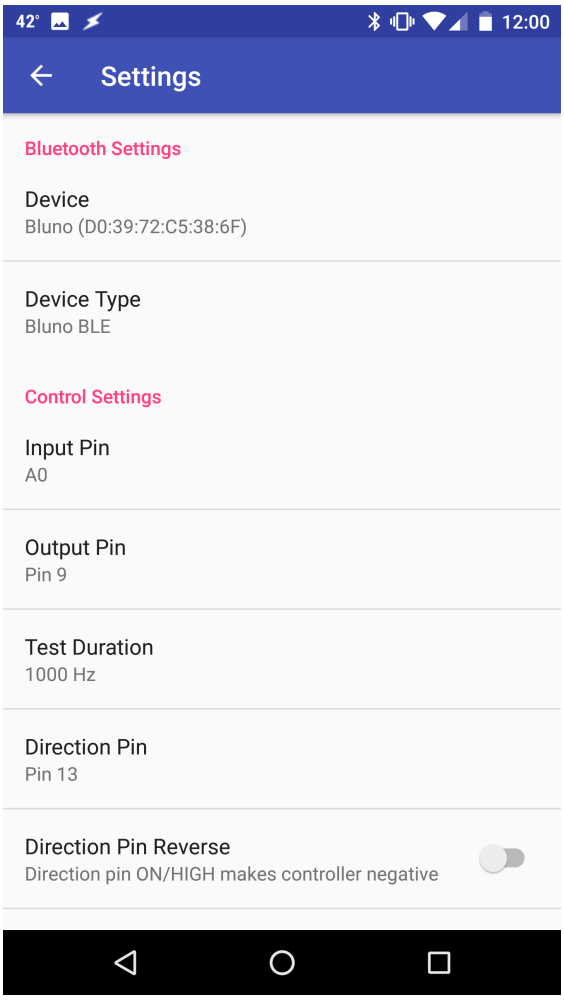

Figure 10. Settings Activity

The Settings Activity is used to configure various application defaults and control mode defaults.

CHAPTER 4

FINDINGS

4.1 Serial Connection Performance

Testing was completed in order to validate the theoretical performance expectations. In order to control as many variables as possible, the first stage of testing was completed using a USB serial connection, rather than introduce bluetooth at this early testing stage. A testing client was developed using VB.net in order to record the performance characteristics of the specific test. The developed testing interface is shown in Figure 11 below, and source code is provided in Appendix F.

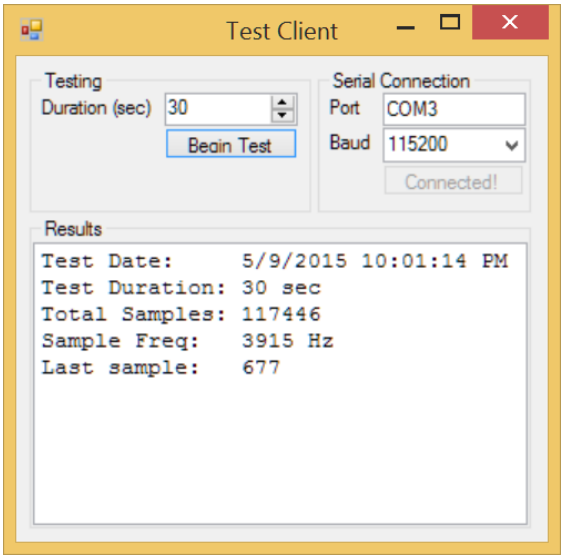

Figure 11. Test Client Screenshot

The test apparatus consisted of an Arduino Board Uno, connected by USB to a Microsoft Surface Pro 2 tablet which was running the test client application.

The first tests were completed with an Arduino Sketch designed only to test the performance of the serial communication protocol, by sending a single fixed value repeatedly. This data was used to support the theoretical throughput calculations and determination that data acquisition is bandwidth limited.

	Duration (sec)	Samples Received	Sample Frequency (hz)
Test 1 - "1023" Only	30	50340	1678
Test 2 - 0.0v input	30	117458	3915
Test 3 - 3.3v input	30	58827	1961
Test 4 - 5.0v input	30	50327	1678

Table 4. ASCII Encoded Communication - 115,200 bps

Table 4 also shows the limitations of sending a 10-bit measurement, encoded as a variable number of packets. Since the sample size is variable, the time required to transmit varies as a function of measured voltage.

	Duration (sec)	Samples Received	Sample Frequency (hz)
Test 5 - 0.0v input	30	117458	3915
Test 6 - 3.3v input	30	117452	3915
Test 7 - 5.0v input	30	117448	3915

Table 5. Bandwidth Optimized Communication - 115,200 bps

	Duration (sec)	Samples Received	Sample Frequency (hz)
Test 8 - 0.0v input	30	169628	5654
Test 9 - 3.3v input	30	169577	5653
Test 10 - 5.0v input	30	169647	5655

Table 6. Bandwidth Optimized Communication - 230,400 bps

	Duration (sec)	Samples Received	Sample Frequency (hz)
Test 11 - 0.0v input	30	162144	5405
Test 12 - 3.3v input	30	162178	5406
Test 13 - 5.0v input	30	162117	5404

Table 7. Bandwidth Optimized Communication - 460,800 bps

These tests show the potential performance of the Arduino Uno Board as a data acquisition platform at various serial baud rates. Tests 5-7 were performed at the highest documented baud rate supported by the Arduino Uno Board. Tests

8-10 show performance at a doubled transmission data rate of 230,400 bps. Tests 11-13 show transition to processing limitations on the Arduino Uno Board, rather than being bandwidth limited by the serial communications port.

4.2 Application Development

Once theoretical bandwidth capacity was confirmed work began on developing the Android user interface, Arduino Sketch, and introducing the Bluetooth connection. Performance of the Bluetooth serial connection could not be characterized until proof of concept application was developed. Development and testing of the application showed that the serial connection implemented over Bluetooth introduced significant latency to the communication. The additional latency reduced streaming sample throughput to approximately 300 samples per second. At this point it was clear that streamed samples would not be a reliable communication strategy when implemented over the Bluetooth serial link.

Since each analog to digital measurement requires 10 bits, two 700 byte (byte = 8 bits) arrays were allocated for data storage. This strategy allows samples to be stored to the Arduino Board memory and transmitted to the Android client application at a later time. This strategy proved effective for collecting data samples and transmitting the samples to the Android client for permanent storage. However, caution was exercised with future memory allocation as sample storage utilized (1400 / 2048) bytes memory, leaving little for program variables.

Once the Android application and Arduino Sketch were fully developed, additional testing was performed to validate the laboratory experiments presented in System Dynamics Experimentation at Home are possible to run using the tools developed by this study. This includes data collection tests involving a DC Motor with Tachometer, Cantilever Beam with Accelerometer, Plate Temperature Control, and Cantilever Beam Forced Response with Motor and Accelerometer [1].

Collected data from the experimental testing shows two possible sources of error. The most common source of collection error is a missing data point. These errant data points may or may not affect the validity of any collected data. The Android application displays a warning for the user in any case in which a test dataset does not contain the expected 700 measurements. A second source of error is a retransmitted data point which is then saved twice. This error is characterized by an unexpected flat line between the two or more repeated data points.

4.3 DC Motor with Tachometer

Testing was conducted to validate closed loop control of a DC motor with tachometer. The experimental setup consisted of the same items utilized by students using the URI System Dynamics Experimentation kit.

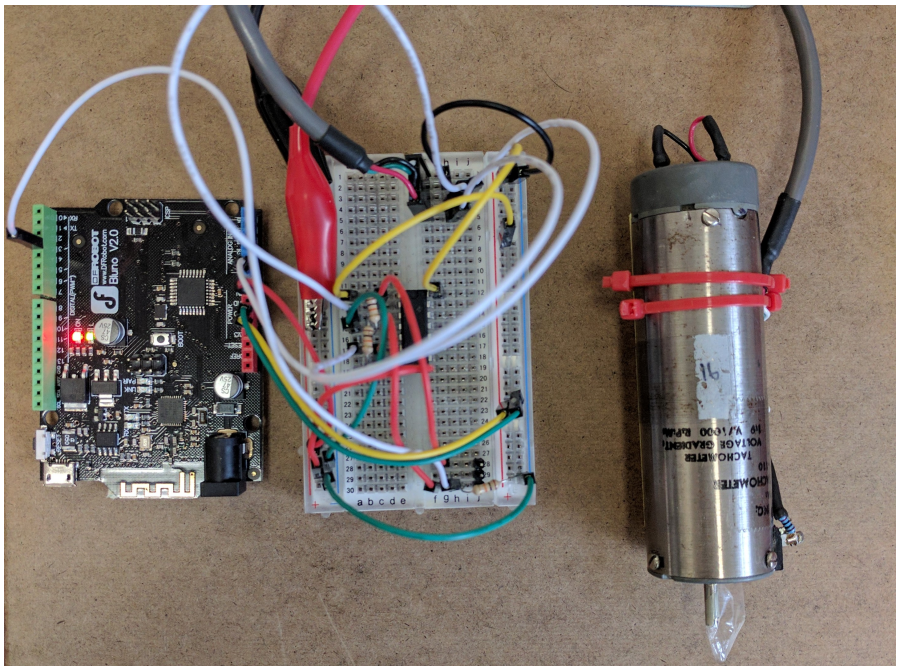

Figure 12. DC Motor with Tachometer Experimental Setup

An experimental setup and circuit was developed Bluno Board, 12v power supply, Android phone running the test client. An additional circuit known as an H-Bridge was connected to the Bluno Board setup to function as a motor

controller, as an H-Bridge is not included on the Bluno Board. Figures 13 and 14 show closed loop control of the DC Motor with tachometer in both PI and P control. The results show clear characteristics of both P and PI control. PI control shows windup characteristics and settles to a steady state error of 0v with minimal overshoot. P control shows a large steady state error of approximately 1.85v. The exact steady state error is difficult to determine due to periodic noise in the tachometer voltage signal of amplitude approximately 0.15v.

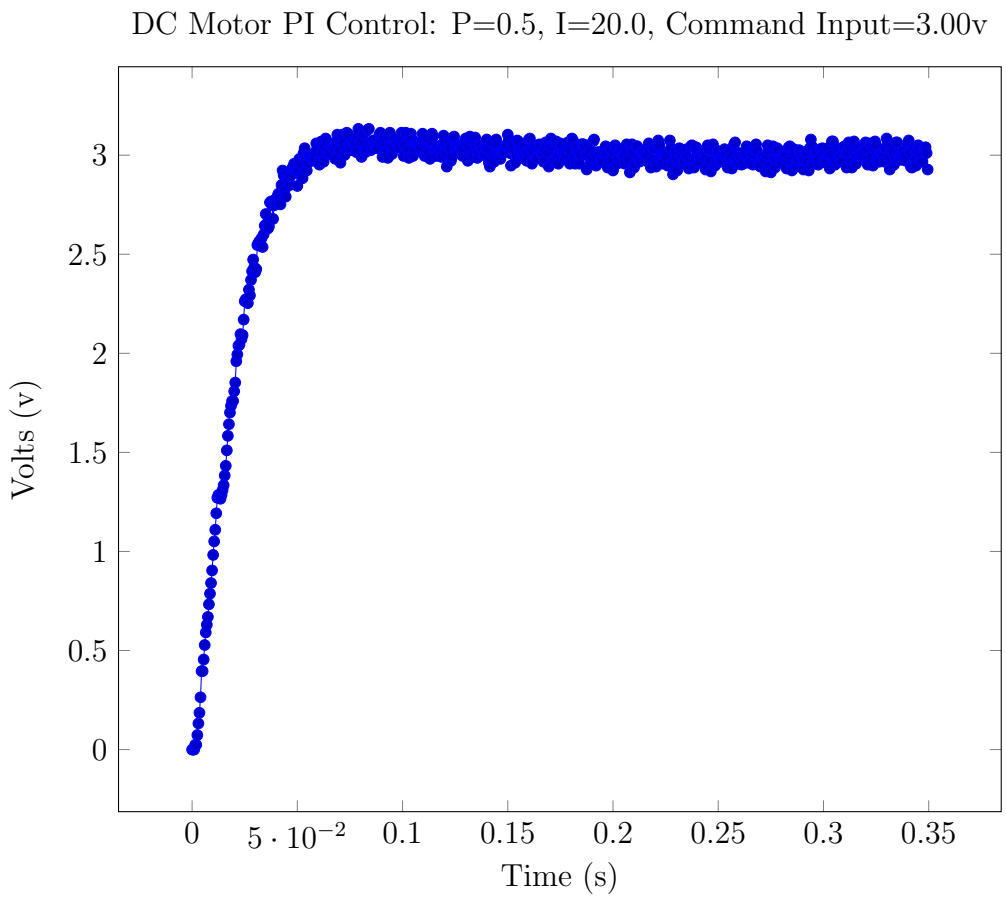

Figure 13. DC Motor with Tachometer PI Control

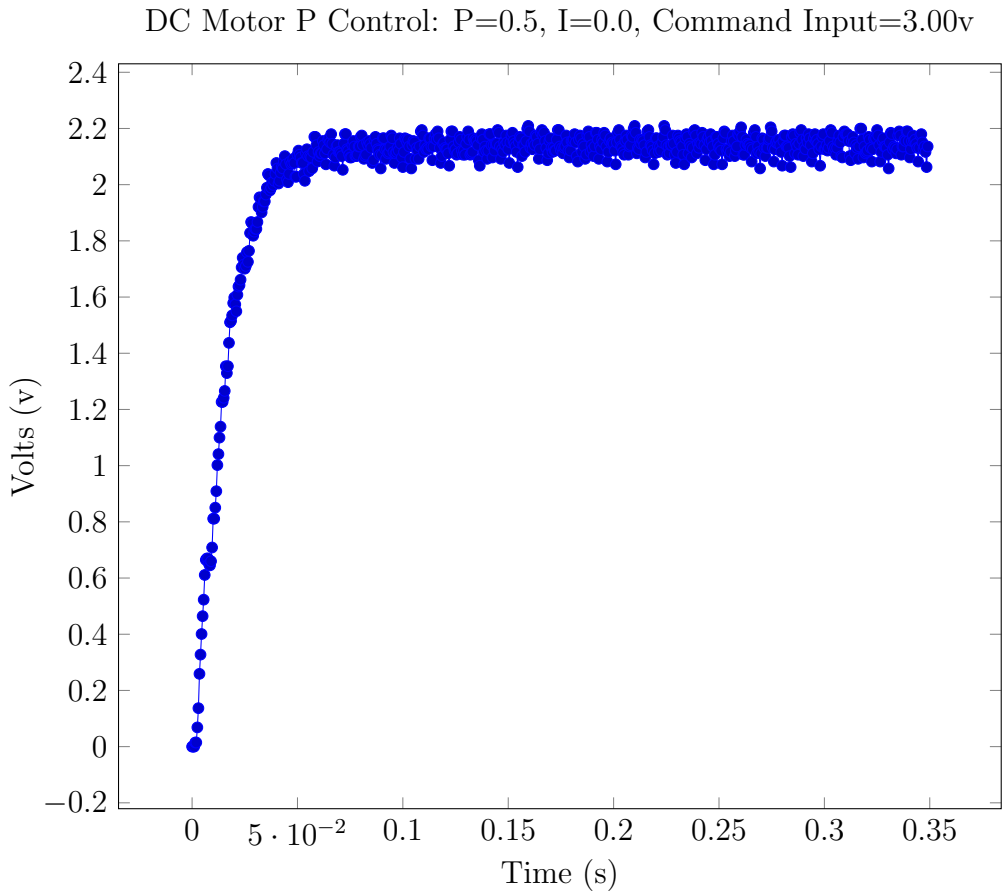

Figure 14. DC Motor with Tachometer P Control

The response appears visually similar to data collected using similar gains for the URI Take-Home Experimentation Kit shown in Figure 15.

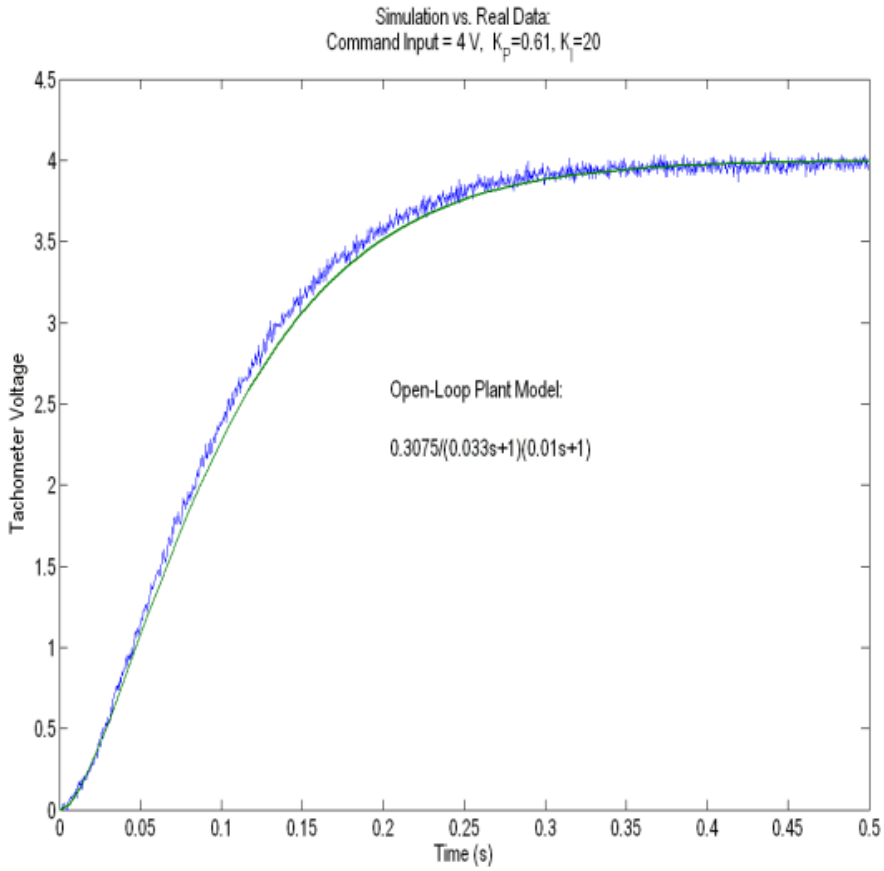

Figure 15. Take-Home Kit Performance [1]

DC motor control showing open loop control of the DC Motor with Tachometer test apparatus depicted in Figure 16 through Figure 19. The tests show clipped sensor data where the input voltage from tachometer was greater than 5.00v which would require voltage reducing circuit to accurately measure. These results would be used by students to determine the time constant τ of the given motor circuit. Time constant τ is determined using time required to reach 63% of open loop steady state response [1] and a Matlab procedure is followed to determine required gains to produce stable control system output.

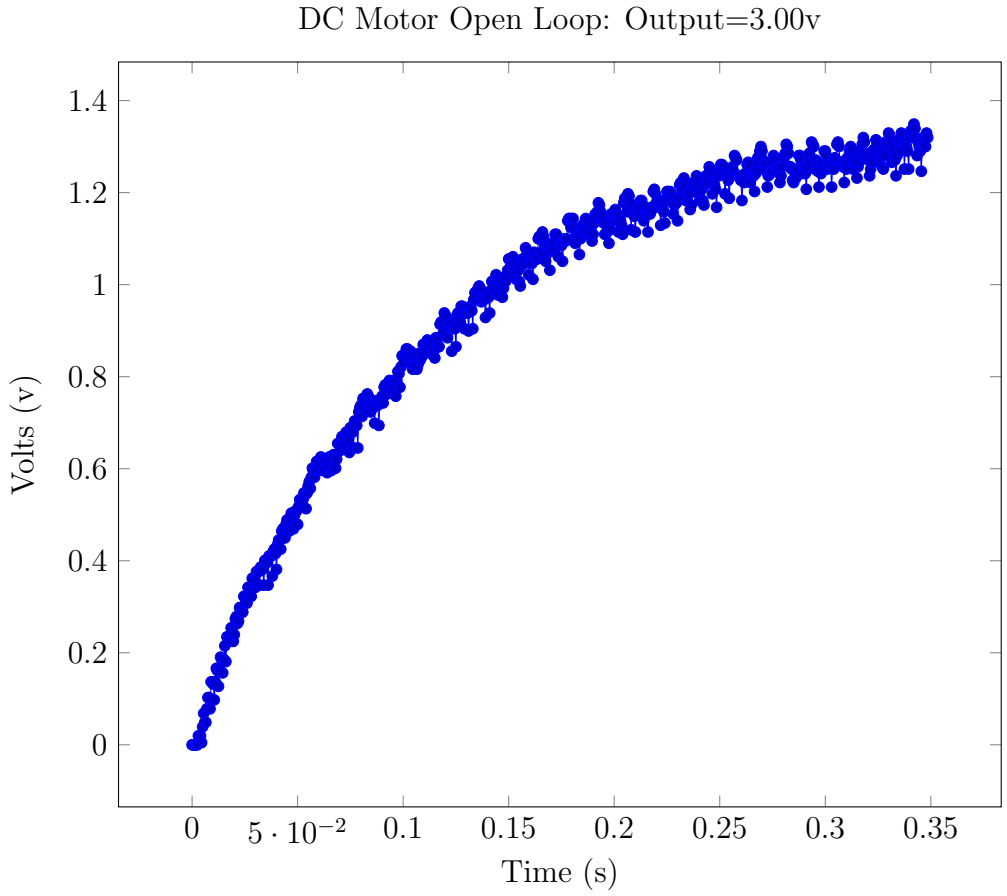

Figure 16. DC Motor Open Loop Test 1

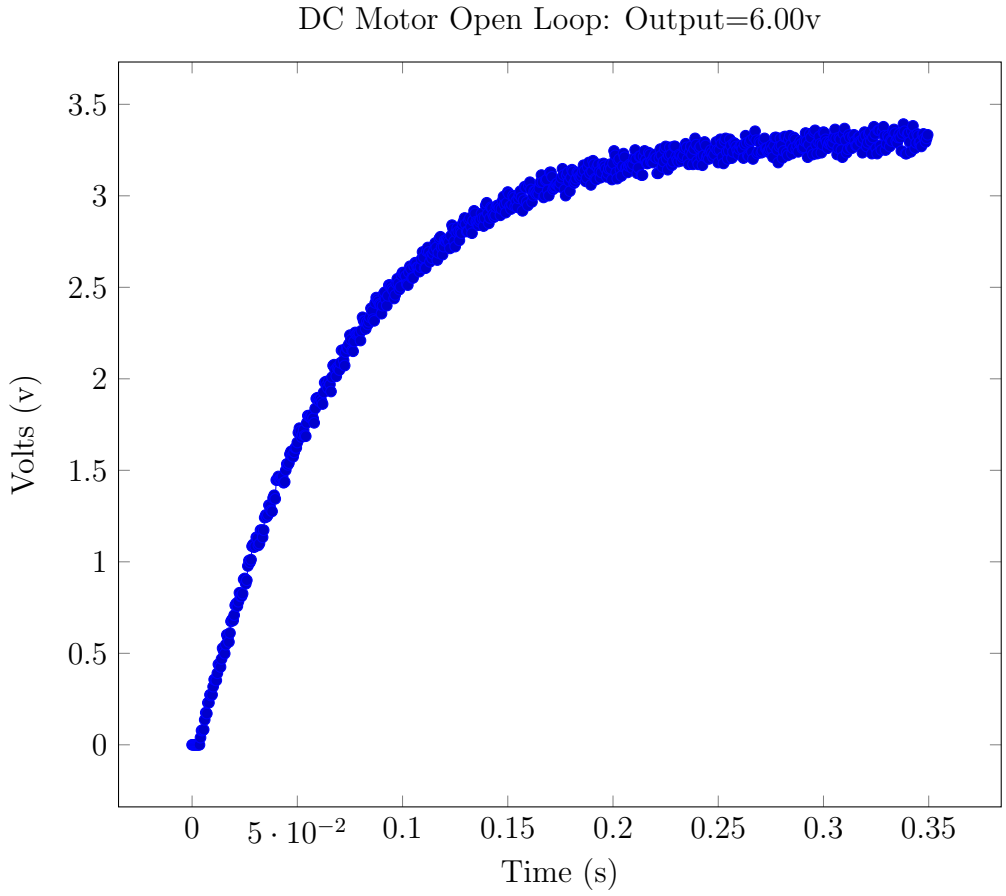

Figure 17. DC Motor Open Loop Test 2

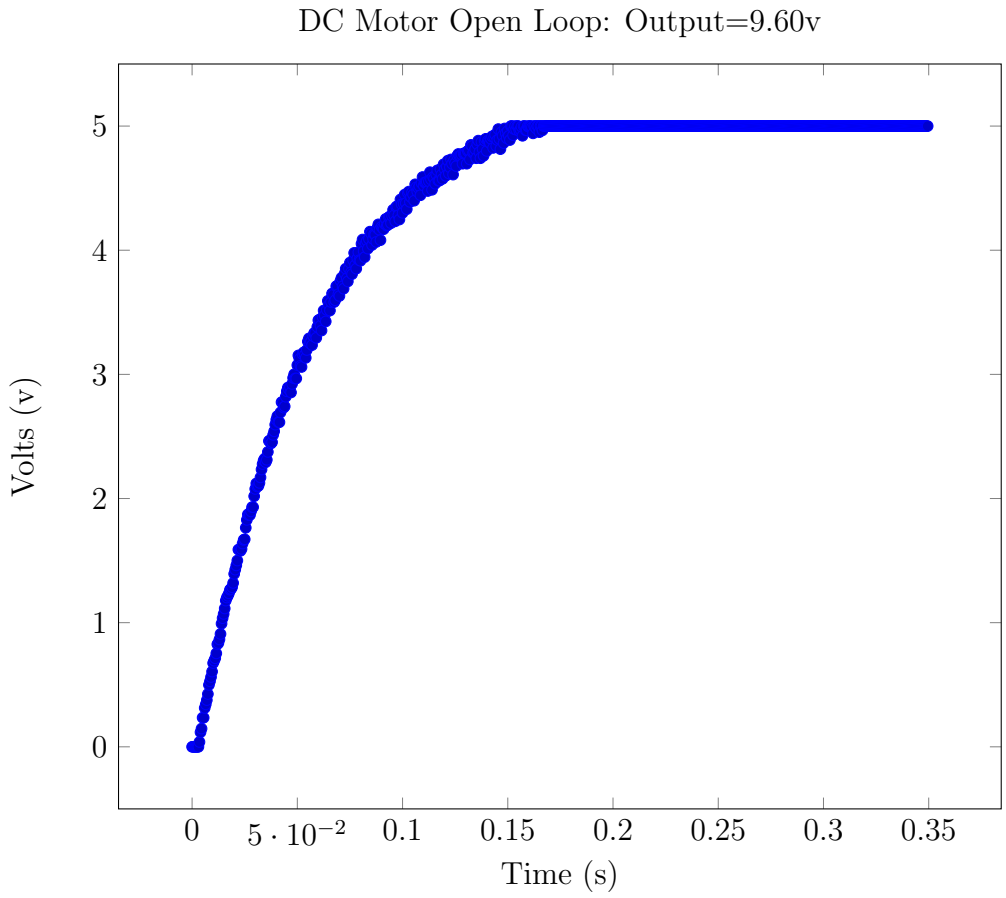

Figure 18. DC Motor Open Loop Test 3

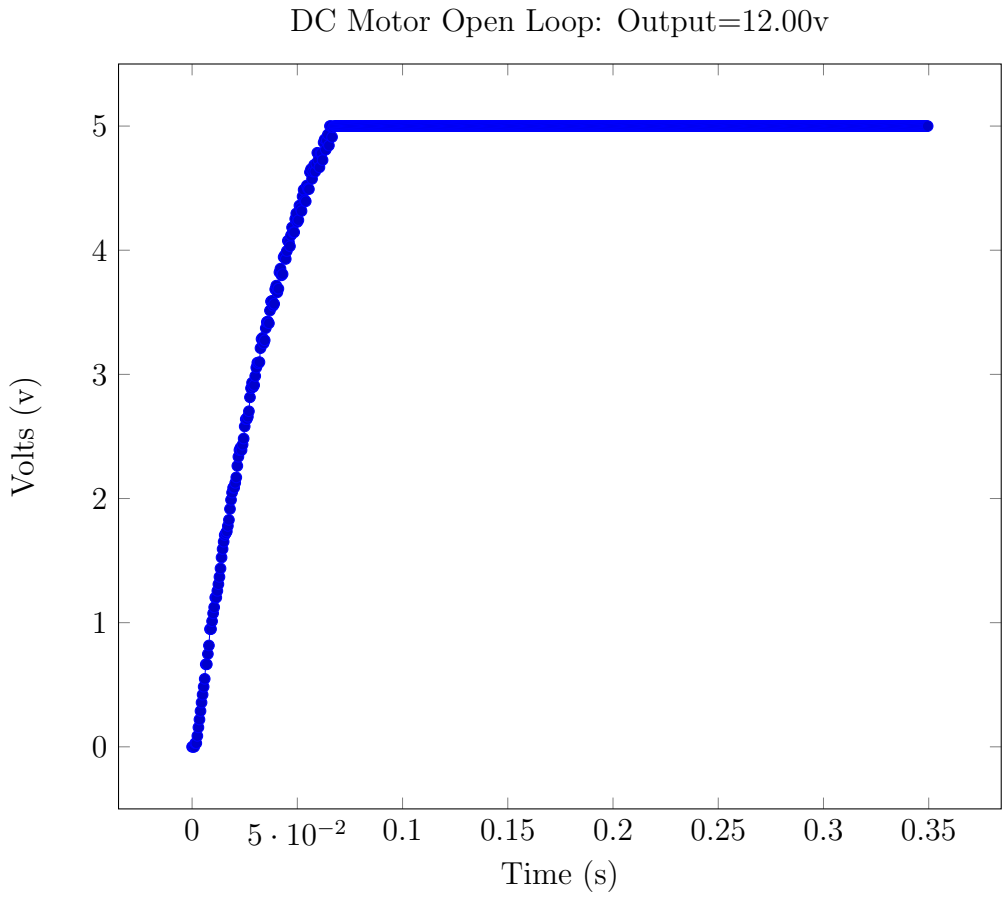

Figure 19. DC Motor Open Loop Test 4

4.4 Cantilever Beam with Accelerometer

Testing was conducted to measure the vibration of a cantilevered beam. This data example was collected at a rate of 2000 samples per second. The experimental setup is shown in Figure 20 and consists of a flat spring, acceleration sensor, Bluno running Arduino Sketch, and Android client application. The collected data is plotted in Figure 21. Students conducting mechanical experimentation can calculate oscillation frequency and approximate damping ratio from the collected data.

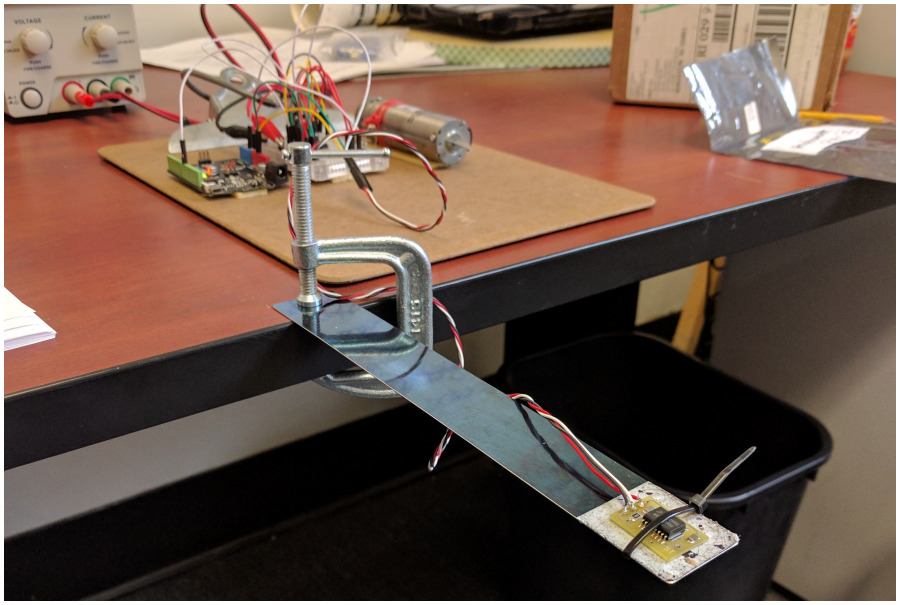

Figure 20. Cantilever Beam with Accelerometer Experimental Setup

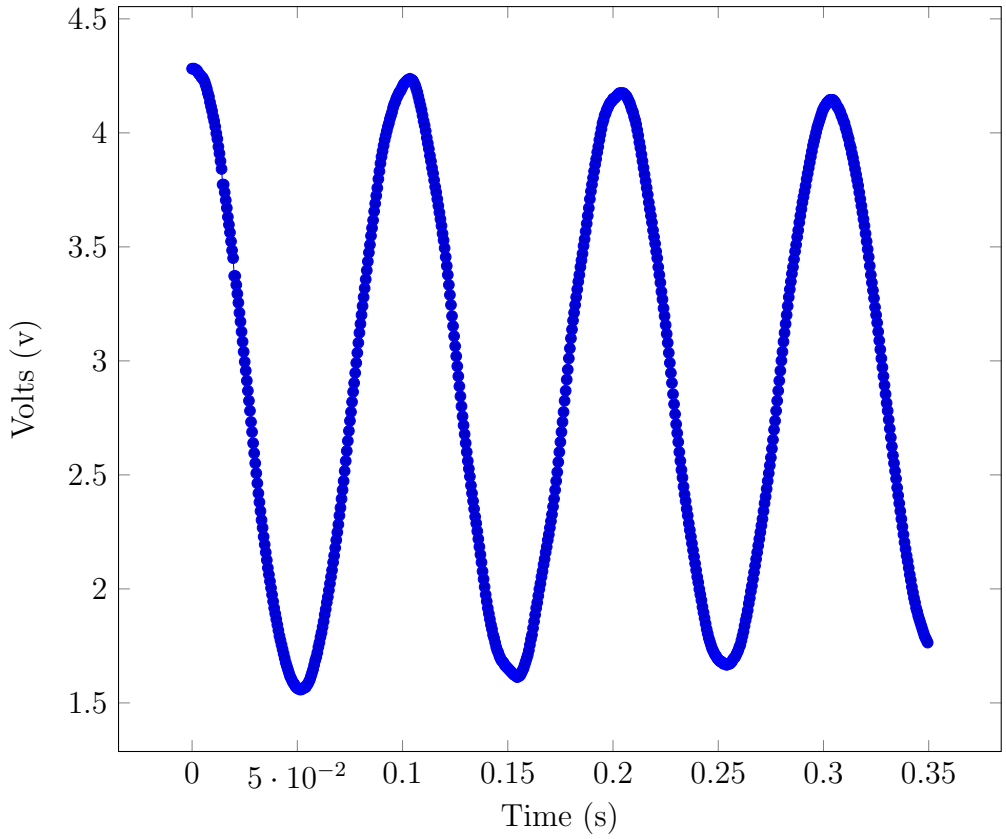

Figure 21. Cantilever Beam with Accelerometer Experimental Data Sample

4.5 Plate Temperature Control

Testing was conducted to determine temperature control performance. This data example was collected at a rate of 4 samples per second. The experimental setup is shown in Figure 22 and consists of a copper block, temperature sensor, resistive heating element, power supply, H-Bridge motor control circuit, Bluno running Arduino Sketch, and Android client application. The collected data is plotted in Figure 23.

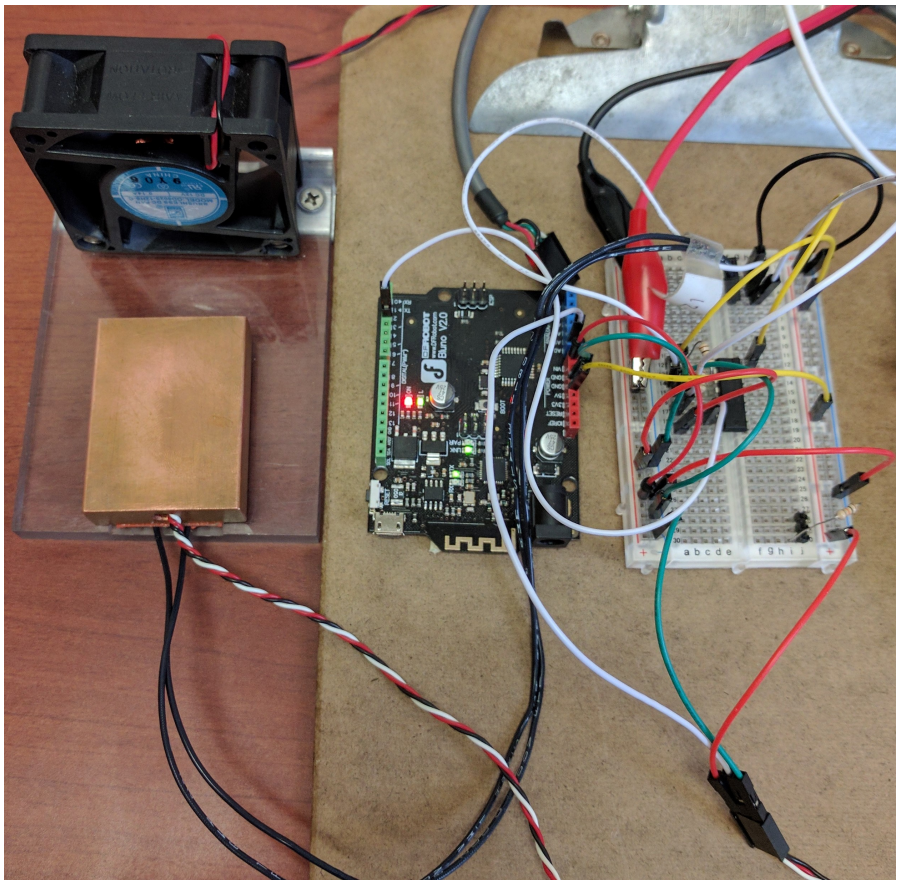

Figure 22. Plate Temperature Control Experimental Setup

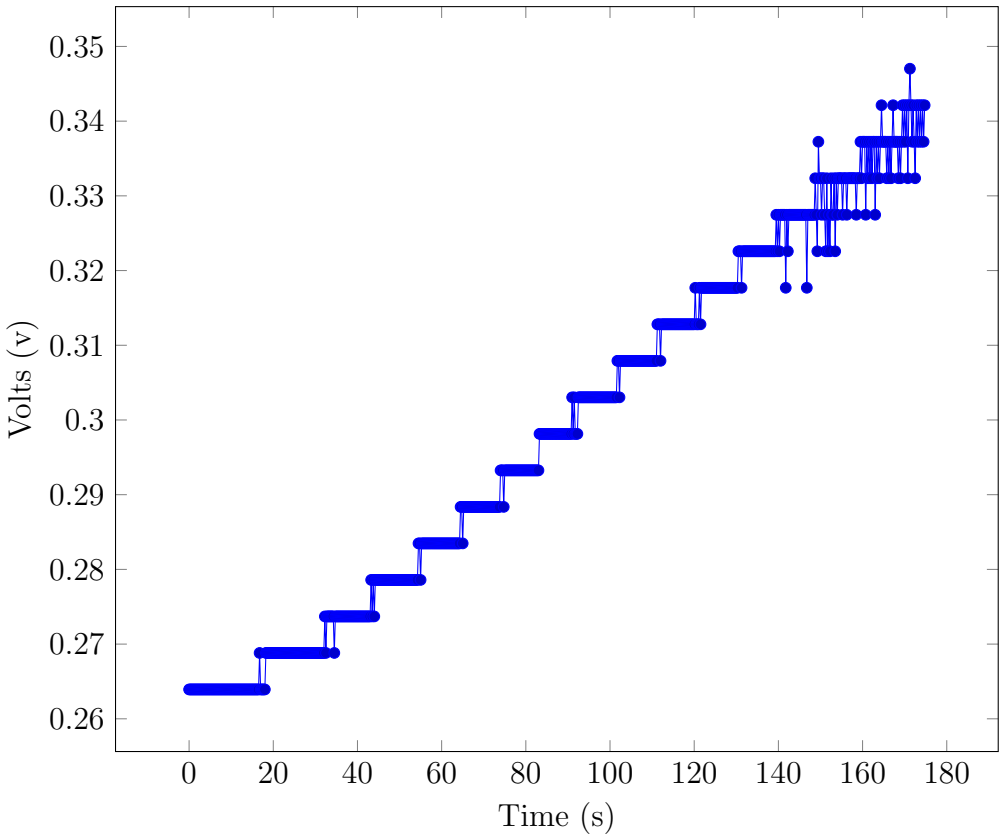

Figure 23. Plate Temperature Control Experimental Data Sample

This sample of collected data shows steps due to the relatively low resolution of the temperature sensor, but otherwise shows a strong linear trend. The board was running in PI control for this test and was set at 0.35v target.

4.6 Cantilever Beam Forced Response

Testing was conducted to determine Cantilever Beam Forced Response performance. This data example was collected at a rate of 2000 samples per second. The experimental setup is shown in Figure 24 and consists of a flat spring, small motor, acceleration sensor, Bluno running Arduino Sketch, and Android client application. Test data was collected under steady state forced oscillation with the motor running in an open loop control mode at varied output voltages. Students calculate frequency $f = 1/T$ and amplitude from the collected steady state response data. This data is then used to plot the frequency response of the cantilever beam system.

The data collected for the cantilever beam forced response shows a small number of repeated data points which are characterized as two or more sequential data points recorded at the same value. This is one of the two possible error types discussed in the Application Development section of this chapter. Manual data cleanup would remove this error, or increased sampling frequency would minimize the affect on resulting plot.

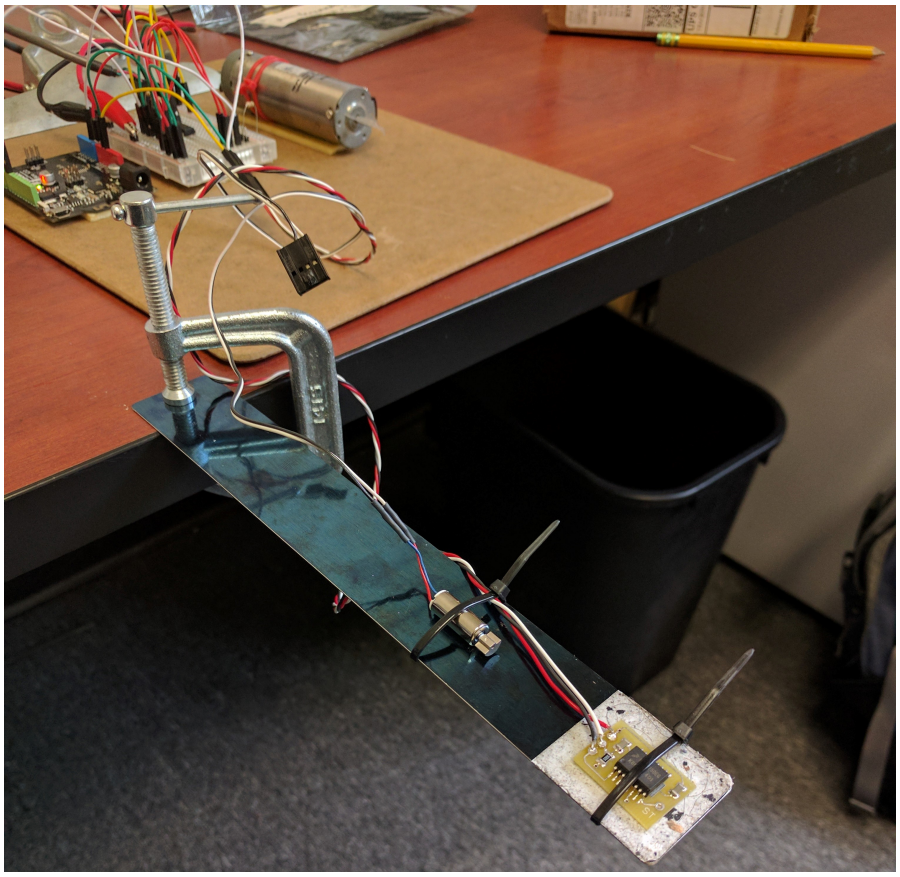

Figure 24. Cantilever Beam Forced Response Experimental Setup

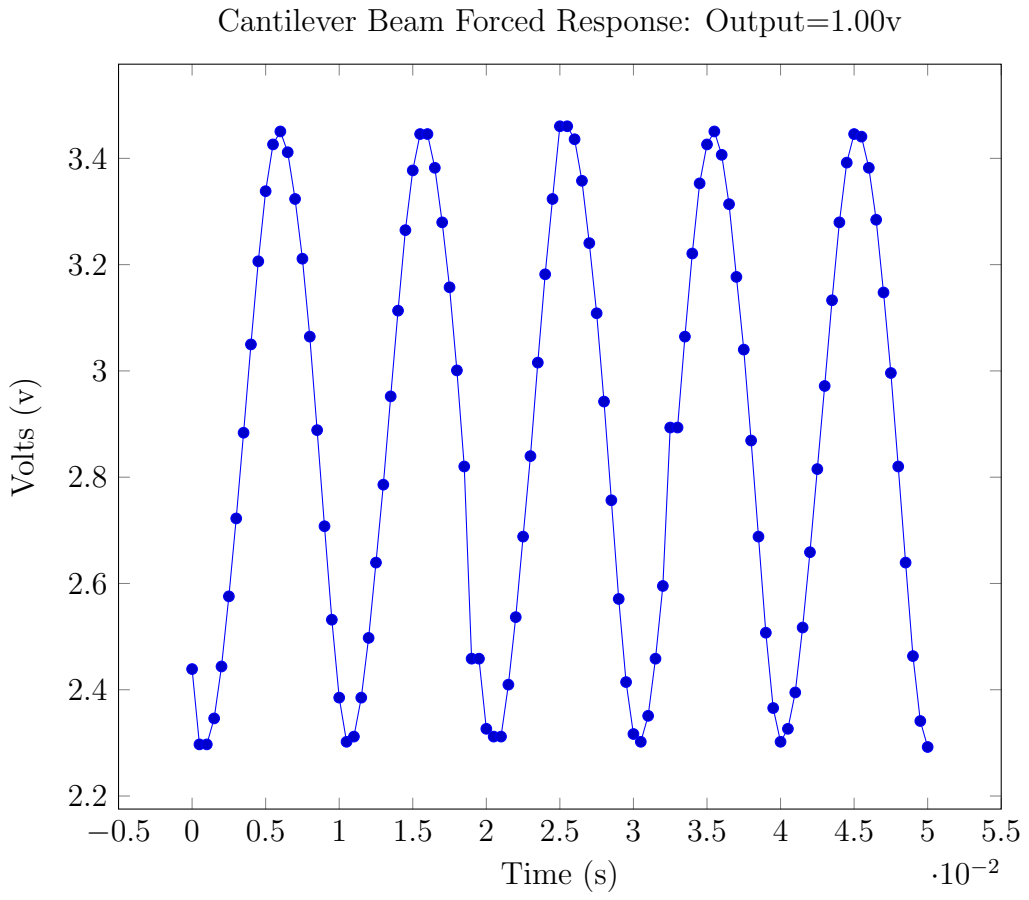

Figure 25. Forced Response Test 1

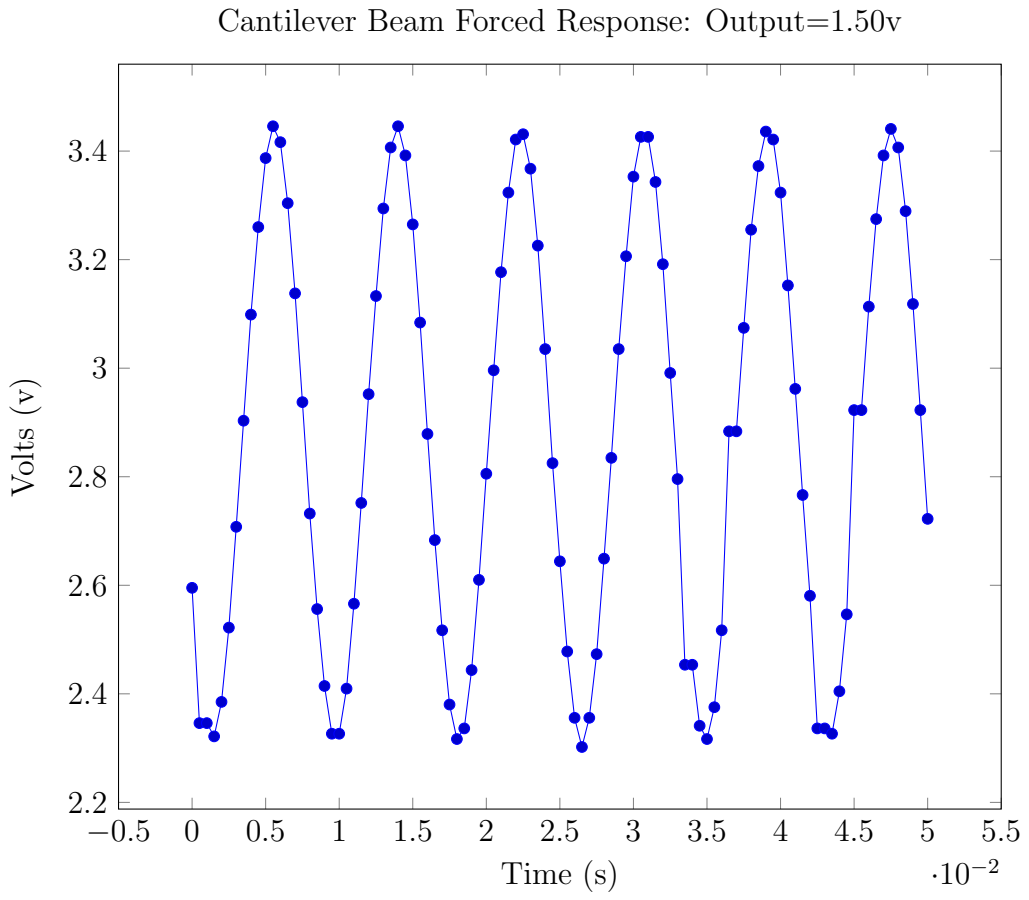

Figure 26. Forced Response Test 2

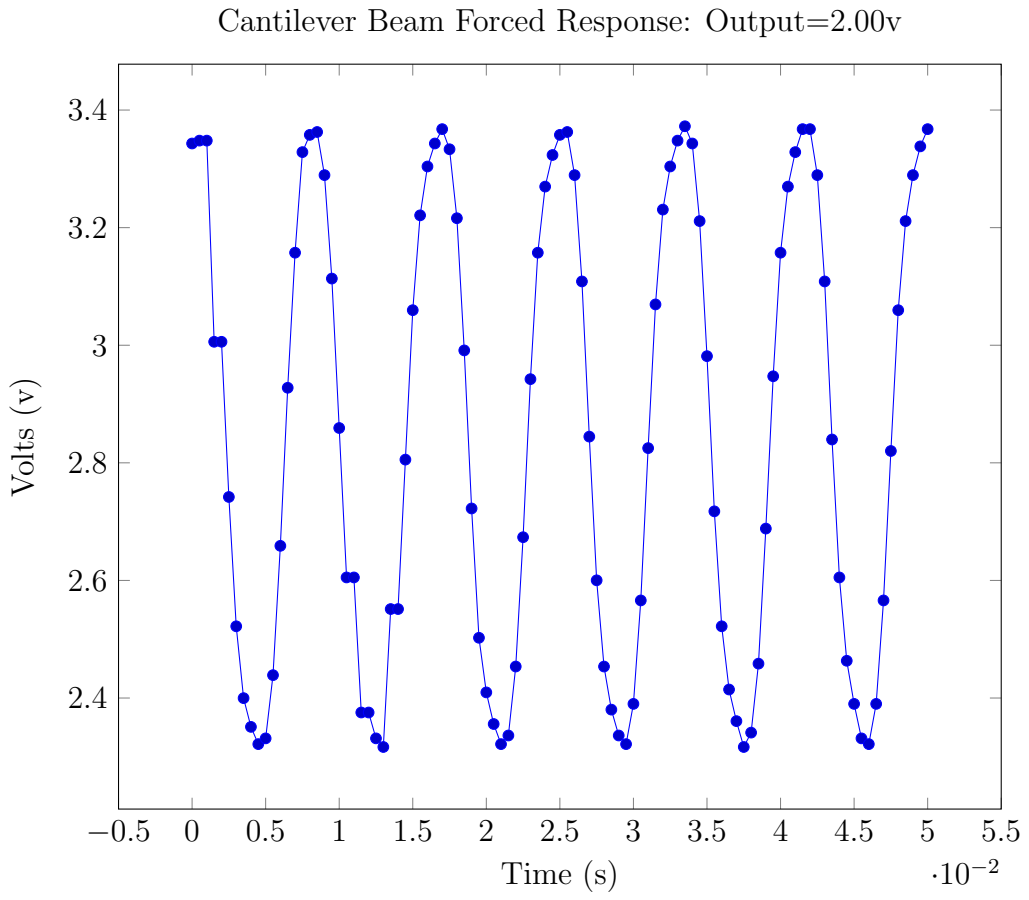

Figure 27. Forced Response Test 3

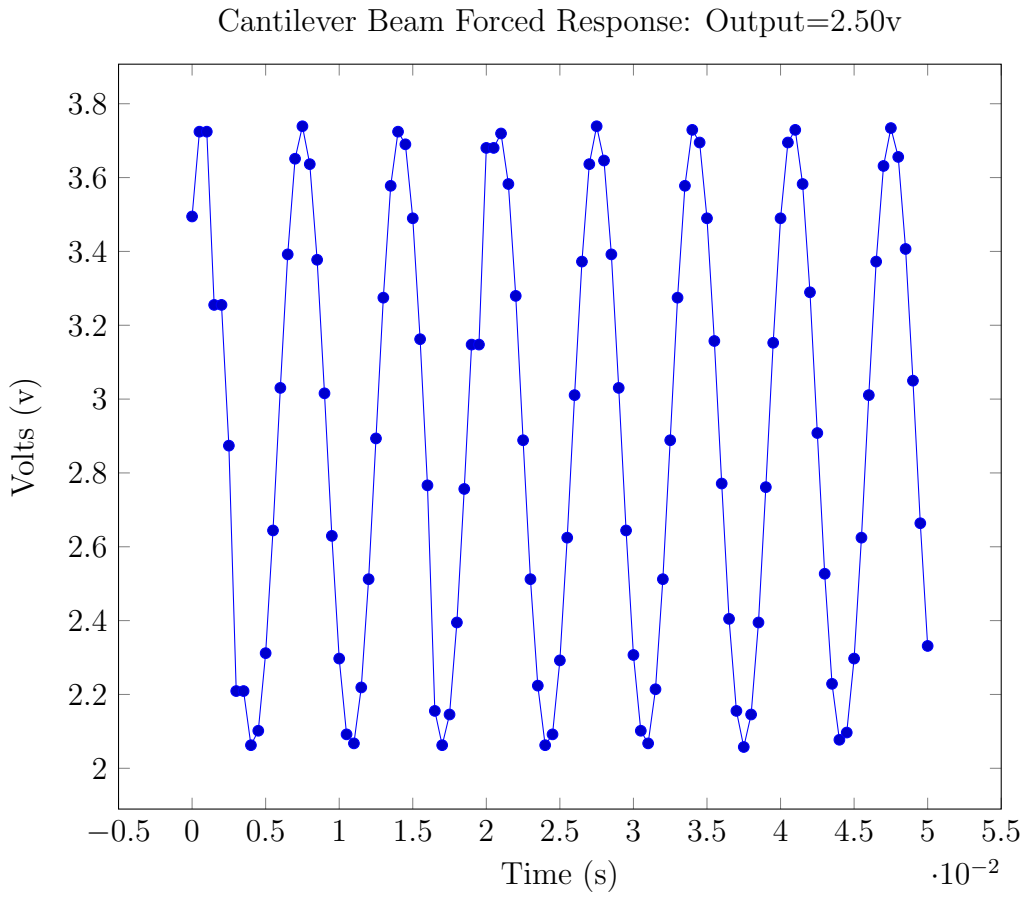

Figure 28. Forced Response Test 4

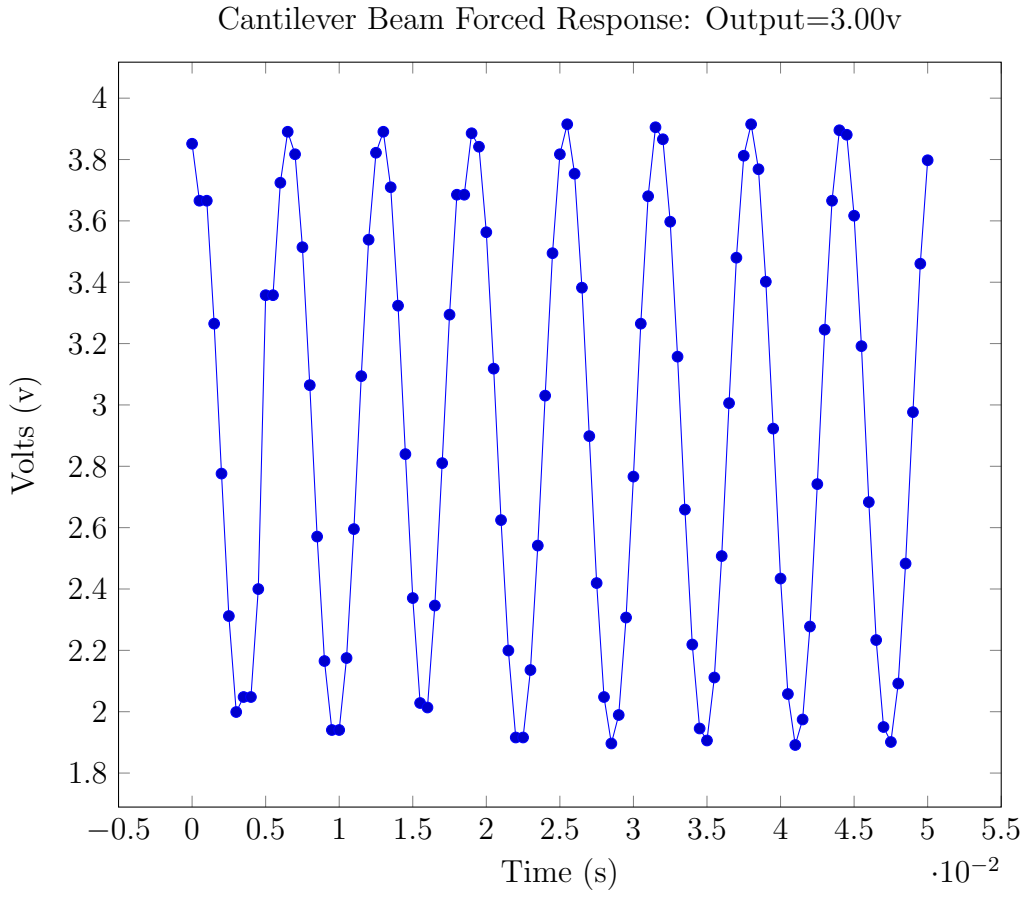

Figure 29. Forced Response Test 5

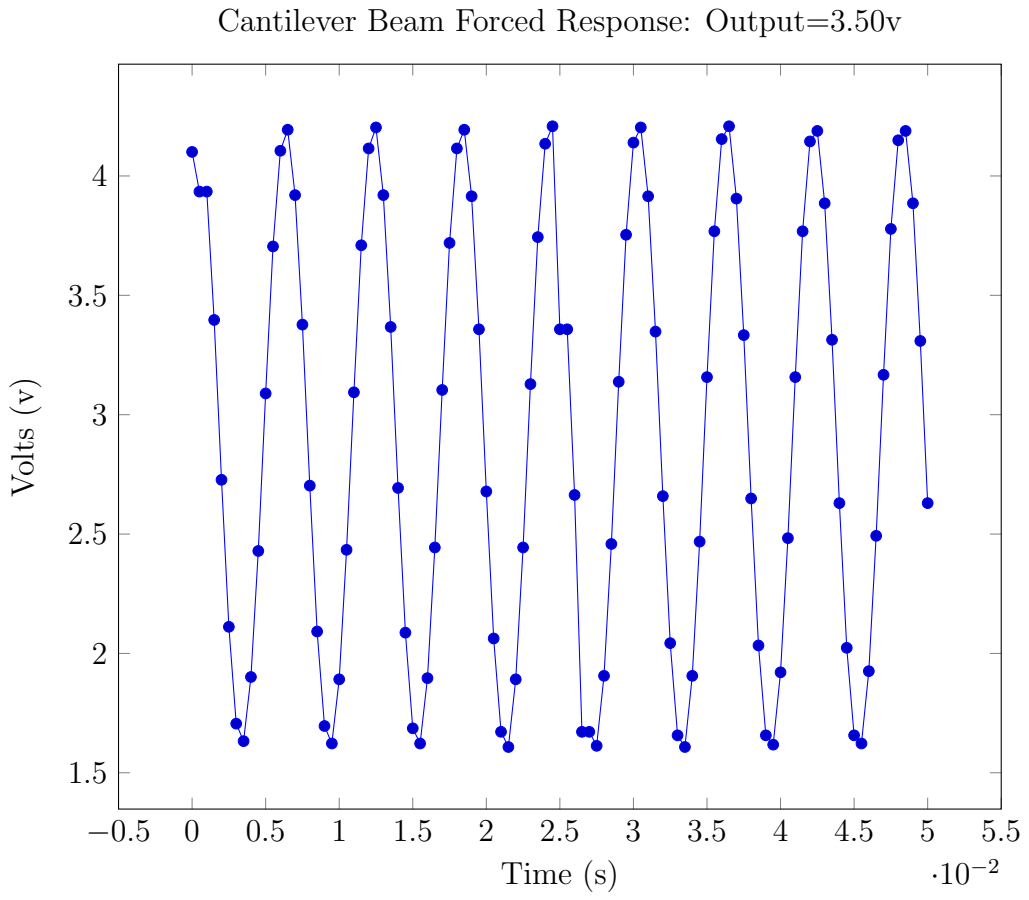

Figure 30. Forced Response Test 6

List of References

[1] M. K. Jouaneh and W. J. Palm III, "Control systems take-home experiments," *IEEE CONTROL SYSTEMS MAGAZINE*, vol. 31, pp. 44–53, Aug. 2013.

CHAPTER 5

CONCLUSIONS

The study presented within shows a promising approach for low cost experimental data capture to a mobile computing device. While there exist more expensive and capable devices, performance of the described wireless measurement and control system is adequate for use by students performing mechanical experimentation.

As noted by Jouaneh and Palm, most Mechanical Engineering programs introduce students to the theory of system dynamics, controls, mechatronics, and vibrations [1]. This study gives those same students the capability to perform mechanical experimentation to study real world behavior of the theoretical systems described in coursework. The tool presented within this study provides a means of increasing accessibility to coursework related laboratory experiments. Additionally, the approach presented within uses general purpose commercially available hardware, which means it can be rapidly scaled to meet the requirements of any given class.

This study designed a tool capable of measuring any analog input between 0v and 5v DC, with a sampling frequency up to 5000 samples per second. Compatibility with any analog input means that all components of the original URI Take-Home Experimentation Kit can be measured or controlled using only a Bluetooth capable Arduino Board, power supply, H-Bridge circuit for motor control, Android mobile computing device, and the developed software for Arduino and Android. Additionally, since this approach is not a turn-key solution like the URI Take-Home Experimentation Kit, the student conducting the experiment gains valuable experience building simple electronic circuits. The Arduino program is

developed to be flexible and fully configurable by the end user. This approach allows a compatible client to configure all running parameters controlling the measurement and control output of the program, from timing to specifying inputs and output pins. The value of such an approach is that future development work only needs to be completed on the client side implementation.

Future work on the client should implement streaming measurements to the Android client for cases where low sampling rates are requested in order to increase effective test length. Such an implementation would enable better measurement performance when using the developed software with experiments requiring more than 700 samples at rates less than 10 samples per second.

The resulting data shows that the developed system can reasonable characterize the performance of instrumented real world mechanical systems. Additionally, the collected data shows that the measurement performance of the developed system is great enough to capture data likely to be useful for students conducting experiments. The sensor data collected using the developed approach for data collection from mechanical systems can be analyzed the same methodology developed for the URI Take-Home Experimentation Kit.

List of References

[1] M. K. Jouaneh and W. J. Palm III, "Control systems take-home experiments," *IEEE CONTROL SYSTEMS MAGAZINE*, vol. 31, pp. 44–53, Aug. 2013.

APPENDIX A

Arduino Sketch

```
// Bryan Salisbury


// State definitions
const byte stateNull      = 0;
const byte stateSendBuf   = 2;
const byte stateMeasure   = 3;
const byte stateControl   = 4;
const byte modeProgram    = 7;

// global
byte stateEntryFlag = -1;       // start execution in
   impossible state
byte nextState = stateNull;     // start state for code
   execution
boolean debug  = true;          // Enable debugging
   information
volatile unsigned int preCount = 0; // Set by programming
   command

// global timing
unsigned long markTemp  = 0; // Mark for temp storage of a
   time
unsigned long markControl  = 0; // Used in controlMode

// output buffer
const int maxBuffer = 700;
volatile unsigned int index = 0; // used in ISR for
   buffering stored samples
volatile byte buffer0[maxBuffer];
volatile byte buffer1[maxBuffer];

// stateMeasure
byte measureMask        = 0; // Measure & Transmit Mask
boolean compressOutput = 1; // Binary encode output data (
   default)

// stateSendBuf
```

```
unsigned int lastIndex = 0; // used to mark last index
    accessed in stateSendBuf
boolean ack = false;        // Set high on ACK signal from
    Android
byte ackTimeout = 100;      // timeout for transmission
    resend
byte errorCount = 0;        // transmission error count
const byte failCount = 255;  // Abort sending after XX
    resends

// modeProgram
const byte maxControlStringLength = 19;
char bufferInput[maxControlStringLength + 1];

// stateControl
const byte tSamp = 1000; // microseconds
double kp = 0.0,ki = 0.0,kd = 0.0;
int inputValue = 0, lastInputValue = 0;
int error = 0, output = 0, desiredPosition = 0, sumError =
    0, minOut = 0, maxOut = 0;
byte outputPin = 0, inputPin = 0, directionPin = 0;
boolean start = false, feedback=true;
boolean directionPinPositive = LOW;
byte currentState;

ISR(TIMER1_OVF_vect)
{
  TCNT1=preCount;

  //PORTB |= (1 << 5); // PIN 13 LED ON
  if(measureMask > 0){
    // voodoo code: shifts one bit over each iteration to
       match the bits set in measureMask for each input
    for(byte i=0; i<5; i++){
      if(measureMask & (1 << i)){ // compares measuremask
         to see if input is enabled

        // Check to make sure we will not overflow buffer
        if(index < maxBuffer){
          int sensorValue = analogRead(i);
          buffer0[index] = sensorValue & 3;
          buffer1[index] = sensorValue >> 2;
          index++;
        }else{
```

```
            break;
          }
        }
      }
    }
    //PORTB &= (0 << 5); // PIN 13 LED OFF
}

void setup() {
    // initialize serial communication at 115200 baudrate
       bits per second
  Serial.begin(115200);
  nextState = stateNull;

  for(int i = 0; i < maxBuffer-1; i++){
    buffer0[i] = 0;
    buffer1[i] = 0;
  }
}

void printState(){
  Serial.print("STATE=");
  Serial.print(currentState);
  Serial.write('\n');
}

byte getNextState(byte state) {
  byte pindex = 0;
  unsigned long markProgram = 0;

  if (Serial.available()) {

    // switch on Serial.read()

    switch (Serial.read()) {
      case 'A':
        return stateNull;
        break;

      case 'M':
        return stateMeasure;
        break;

      case 'C':
```

```
    return stateControl;
    break;

  case 'P':
    // All blocking code
    // Necessary at this time in order to read the
       entire string into buffer
    pindex = 0;
    markProgram = millis();
    while (millis() - markProgram < 250){
      if (Serial.available()) {
        while (Serial.available()) {
          // maxControlStringLength is defined max
             length of command string
          // does not include null character
          // mem overflow if index exceed this value.
          if (pindex > maxControlStringLength) {
            break;
          }
          bufferInput[pindex] = Serial.read();
          if(bufferInput[pindex] == '\n'){
            break;
          }
          pindex++;
        }
        if(bufferInput[pindex] == '\n'){
          break;
        }
      }
      if(bufferInput[pindex] == '\n'){
        break;
      }
    }

    // Add null character to terminate bufferInput
    // index is last empty character or last character
       in array at this point
    bufferInput[pindex] = '\0';

    if (debug) {
      Serial.print("INFO ");
      Serial.print(bufferInput);
      Serial.write('\n');
    }
```

```
        return modeProgram;
        break;

      case 'S':
        return stateSendBuf;
        break;

      case 'K':
        ack = true;
        while (Serial.available()) {
          if(Serial.peek() == 'K'){
            Serial.read(); // discard duplicated ack
               command
          }
        }
        Serial.flush(); // waits for outgoing transmission
           to complete.
        break;

      default:
        Serial.print("ERROR[");
        Serial.print(state);
        Serial.print("]: Command not recognized");
        Serial.write('\n');

        while(Serial.available()){
          Serial.write(Serial.read());
        }
    }

  }

  return state;
}

int getPosition(){
  return analogRead(inputPin);
}

void doControl(){
  output = 0;
  inputValue = getPosition();
  error = (desiredPosition - getPosition());  // Define
     error
```

```
  if(ki > 0){
    // Calculate I term of output
    output = (ki * (sumError * (tSamp / 1000000)));

    // Apply limits to control output
    if(output > maxOut){
      output = maxOut;
    }else if(output < minOut){
      output = minOut;
    }
  }

  // kp gain added to output
  output += (kp * error);
  output -= (kd * (inputValue - lastInputValue));

  // Apply limits to control output
  if(output > maxOut){
    output = maxOut;
  }else if(output < minOut){
    output = minOut;
  }

  if(ki > 0){
    sumError += error;
  }else{
    sumError = 0;
  }

  lastInputValue = inputValue;


  // Set direction output and make output positive
  if(output >= 0){
    digitalWrite(directionPin, directionPinPositive);
  }else if(output < 0) {
    output *= -1; //make output value positive
    digitalWrite(directionPin, !directionPinPositive);
  }

  analogWrite(outputPin, output);  // Send PWM duty rate to
      motor actuator
}
```

```
void makePortsInputLow() {
  PORTD = PORTD & B11; // Set outputs 2-7 LOW
  PORTB = 0;           // Set outputs 8-13 LOW
  PORTC = 0;           // Set A0-A5 LOW

  DDRD = DDRD & B11; // Configure digital pins as inputs
     2-7
  DDRB = 0;          // Configure pins 8-13 as inputs
  DDRC = 0;          // Configure pins A0-A5 as inputs
}

void sendSample(byte b0, byte b1){
  Serial.write(b0);
  Serial.write(b1);
  Serial.write('\n');
}

void loop() {
  byte tmpNextState = 0;

  currentState = nextState;
  switch (currentState) {
    case stateNull:
      /*
      stateNull is the application initial state.
      Application waits for input from user.
      */
      if (stateEntryFlag != stateNull) {
        makePortsInputLow();
        printState();
        stateEntryFlag = stateNull;

      }

      nextState = getNextState(currentState);
      break;

    case stateMeasure:
      /*
      Main execution state
      Controls timing / etc.
      */
      if (stateEntryFlag != stateMeasure) {
```

```
    stateEntryFlag = stateMeasure;
    printState();

    // Reset timing variables
    start=false;
    preCount=0;

    measureMask = 0;
    DDRB |= (1 << 5); // PIN13 SET DIRECTION TO OUTPUT

    // Configure TIMER1 registers
    TCCR1A = 0x00;
    TCCR1B = (1 << CS11) | (1 << CS10); // 64 prescale
        = 16e6/64 hz = 250000 hz
    index=0;
}

// State change and configuration block
tmpNextState = getNextState(currentState);
if (tmpNextState == modeProgram) {
    Serial.print(bufferInput);
    Serial.write('\n');
    char* cmd = strtok(bufferInput, ":");
    while (cmd != 0) {
      switch (cmd[0]) {
        case 'C':
          cmd++; // skip first character
          // set global compression variable
          compressOutput = atoi(cmd);
          Serial.print("INFO compression=");
          Serial.print(compressOutput);
          Serial.write('\n');
          break;
        case 'D':
          cmd++;
          // set delay value
          preCount = atoi(cmd);
          TCNT1 = preCount;
          Serial.print("INFO preCount=");
          Serial.print(preCount);
          Serial.write('\n');
          break;
        case 'S':
          cmd++;
```

```
            // set mask for outputs
            measureMask = atoi(cmd);
            Serial.print("INFO measureMask=");
            Serial.print(measureMask); //must be != 0 for
                samples to buffer
            Serial.write('\n');
            break;
          }
          cmd = strtok(0, ":");
          delay(100);
        }
        bufferInput[0] = '\0';

      } else {
        nextState = tmpNextState;
      }

      // if test not yet started, check to see if required
         params set and start test
      if(!start){
        if(preCount != 0 && measureMask != 0){
          start = true;
          TIMSK1 |= (1 << TOIE0); // Turn on Timer1
              overflow interrupt
        }
      }

      // Switch to sending mode
      if(index >= maxBuffer){
        nextState = stateSendBuf;
      }

      if(nextState != currentState){
        // Exit Case
        TIMSK1 &= (0 << TOIE0); // Turn off Timer1 overflow
            interrupt
        start = false; // probably redundant
      }
      // end configuration block
      break;


    // Control State
    case stateControl:
```

```
if (stateEntryFlag != stateControl) {
  makePortsInputLow();
  // stateControl Defaults
  kp = 0;
  ki = 0;
  kd = 0;
  sumError = 0;
  error = 0;
  output = 0;
  desiredPosition = 0;
  //tSamp = 2;
  minOut = 0;
  maxOut = 255;
  inputPin = 0;
  outputPin = 9;
  directionPin = 13;
  directionPinPositive = LOW;
  start = false;
  feedback = true;

  TCCR1A = 0x00;
  TCCR1B = (1 << CS11) | (1 << CS10); // 64 prescale
     = 16e6/64 hz = 250000 hz
  index=0;

  stateEntryFlag = stateControl;
  printState();
  preCount = 0;
}

if(start){
  if(feedback){
    if((micros() - markControl) >= tSamp){
      markControl=micros();
      doControl();
    }
  }else{
    analogWrite(outputPin, desiredPosition >> 2); //
       shift 2 bits right. 8bits for control
  }

  // we can get away with using markTemp here and in
     modeProgram because neither
  // is used at the same time and the consequences of
```

```
      screwing up the timing here are minimal
  if((millis() - markTemp) >= 100){
    markTemp=millis();
    Serial.print(getPosition());
    Serial.write('\n');
  }
}

// State change and configuration block
tmpNextState = getNextState(currentState);
if (tmpNextState == modeProgram) {
  //Serial.print("INFO input=");
  //Serial.print(bufferInput);
  //Serial.write('\n');
  char* cmd = strtok(bufferInput, ":P");
  while (cmd != 0) {
    switch (cmd[0]) {
      case 'A': //Kp
        cmd++;
        kp = atof(cmd);
        Serial.print("INFO kp=");
        Serial.print(kp);
        Serial.write('\n');
        break;

      case 'B': //Ki
        cmd++;
        ki = atof(cmd);
        Serial.print("INFO ki=");
        Serial.print(ki);
        Serial.write('\n');
        break;

      case 'C': // Kd
        cmd++;
        kd = atof(cmd);
        Serial.print("INFO kd=");
        Serial.print(kd);
        Serial.write('\n');
        break;

      case 'D':
        cmd++;
        desiredPosition = atoi(cmd);
```

```
    Serial.print("INFO desiredPosition=");
    Serial.print(desiredPosition);
    Serial.write('\n');
    break;

  case 'E':
    cmd++;
    preCount = atoi(cmd);
    Serial.print("INFO preCount=");
    Serial.print(preCount);
    Serial.write('\n');
    break;

  case 'F':
    cmd++;
    feedback = atoi(cmd);
    Serial.print("INFO feedback=");
    Serial.print(feedback);
    Serial.write('\n');
    break;

  case 'H':
    cmd++;
    maxOut = atoi(cmd);
    Serial.print("INFO maxOut=");
    Serial.print(maxOut);
    Serial.write('\n');
    break;

  case 'L':
    cmd++;
    minOut = atoi(cmd);
    Serial.print("INFO minOut=");
    Serial.print(minOut);
    Serial.write('\n');
    break;

  case 'O':
    cmd++;
    outputPin = atoi(cmd);
    Serial.print("INFO outputPin=");
    Serial.print(outputPin);
    Serial.write('\n');
    break;
```

```
      case 'I':
        cmd++;
        inputPin = atoi(cmd);
        Serial.print("INFO inputPin=");
        Serial.print(inputPin);
        Serial.write('\n');
        break;

      case 'M':
        cmd++;
        measureMask = atoi(cmd);
        if(measureMask){
          index=0;
          measureMask = 1 << inputPin; //set
              measuremask from desired input pin
        }
        Serial.print("INFO inputPin=");
        Serial.print(inputPin);
        Serial.write('\n');
        break;

      case 'S':
        start = true;
        pinMode(outputPin, OUTPUT);
        Serial.print("INFO start\n");
        break;
    }
    cmd = strtok(0, ":P");
    delay(50);
    if(start){
      TIMSK1 |= (1 << TOIE0); // Turn on Timer1
          overflow interrupt
    }
  }
  bufferInput[0] = '\0';

} else {
  nextState = tmpNextState;

}

if(index >= maxBuffer){
  nextState = stateSendBuf;
```

```
      start = false; // probably redundant
    }

    // Exit Conditions
    if(nextState != currentState){
      TIMSK1 &= (0 << TOIE0); // Turn off Timer1 overflow
          interrupt
      makePortsInputLow();
    }
    break;

  case stateSendBuf:
    if (stateEntryFlag != currentState) {
      stateEntryFlag = currentState;
      printState();
      makePortsInputLow();
      ack = true;
      lastIndex = 0;
      errorCount = 0;
      markTemp = millis();
    }

    if(ack){
      sendSample(buffer0[lastIndex], buffer1[lastIndex]);
      lastIndex++;
      ack = false;
      markTemp = millis();
    }else{
      if((millis() - markTemp) > ackTimeout){
        lastIndex++;
        sendSample(buffer0[lastIndex], buffer1[lastIndex
            ]);
        markTemp = millis();
        errorCount++;
      }
    }

    nextState = getNextState(currentState);
    if(lastIndex >= maxBuffer){
      nextState = stateNull;
    }

    if(errorCount >= failCount){
      nextState = stateNull;
```

```
        }

        if(nextState != currentState){
          // Exit Conditions
          Serial.print("INFO SENT=");
          Serial.print(lastIndex);
          Serial.write('\n');
          Serial.print("INFO ERRORCOUNT=");
          Serial.print(errorCount);
          Serial.write('\n');
          break;
        }
        break;
    }
}
```

APPENDIX B

MainActivity.Java

```
package com.bryansalisbury.btmeasure;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.view.View;
import android.support.design.widget.NavigationView;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.bryansalisbury.btmeasure.models.TestSequence;
import com.orm.query.Select;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity
        implements NavigationView.
            OnNavigationItemSelectedListener {

    // Test List
    ArrayList<TestSequence> Tests = new ArrayList<>();
    TestSequenceAdapter itemsAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Toolbar toolbar = (Toolbar) findViewById(R.id.
    toolbar);
setSupportActionBar(toolbar);

DrawerLayout drawer = (DrawerLayout) findViewById(R
    .id.drawer_layout);
ActionBarDrawerToggle toggle = new
    ActionBarDrawerToggle(
          this, drawer, toolbar, R.string.
              navigation_drawer_open, R.string.
              navigation_drawer_close);
drawer.setDrawerListener(toggle);
toggle.syncState();

NavigationView navigationView = (NavigationView)
    findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(
    this);

itemsAdapter =  new TestSequenceAdapter(this, Tests
    );
ListView lvResults = (ListView) findViewById(R.id.
    listViewResults);

lvResults.setAdapter(itemsAdapter);
lvResults.setOnItemClickListener(new AdapterView.
    OnItemClickListener() {
     @Override
     public void onItemClick(AdapterView<?>
         adapterView, View view, int i, long l) {
          TestSequence item = (TestSequence)
              adapterView.getItemAtPosition(i);
          //Toast.makeText(getBaseContext(),"You
              selected : " + item.testName +
          //          " ["+ item.getId() +"]", Toast.
              LENGTH_SHORT).show();

          Intent intent = new Intent(getBaseContext()
              , ResultsActivity.class);
          intent.putExtra("TEST_SEQUENCE_ID", item.
              getId().intValue());
          startActivity(intent);
```

```
            }
        });
    }

    @Override
    public void onResume(){
        super.onResume();
        Tests.clear();
        try{
            Tests.addAll(TestSequence.findWithQuery(
                TestSequence.class, "SELECT * FROM
                TEST_SEQUENCE ORDER BY TIMESTAMP DESC"));
        }catch(Exception ex){

        }
        itemsAdapter.notifyDataSetChanged();
    }

    @Override
    public void onBackPressed() {
        DrawerLayout drawer = (DrawerLayout) findViewById(R
            .id.drawer_layout);
        if (drawer.isDrawerOpen(GravityCompat.START)) {
            drawer.closeDrawer(GravityCompat.START);
        } else {
            super.onBackPressed();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action
            bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action
            bar will
        // automatically handle clicks on the Home/Up
            button, so long
        // as you specify a parent activity in
```

```
        AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        Intent intent = new Intent(this,
            SettingsActivity.class);
        startActivityForResult(intent, 0);
        return true;
    }

    return super.onOptionsItemSelected(item);
}

@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(MenuItem item)
    {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_measure) {
        Intent intent = new Intent(this,
            MeasureActivity.class);
        startActivityForResult(intent, 0);
    } else if (id == R.id.nav_control) {
        Intent intent = new Intent(this,
            ControlActivity.class);
        startActivityForResult(intent, 0);
    } else if (id == R.id.nav_settings) {
        Intent intent = new Intent(this,
            SettingsActivity.class);
        startActivityForResult(intent, 0);
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R
        .id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}

public void fabAddProfile(View v){
    Intent intent = new Intent(this, MeasureActivity.
        class);
```

```
            startActivityForResult(intent, 0);
        }

    }
```

APPENDIX C

MeasureActivity.java

```
package com.bryansalisbury.btmeasure;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.preference.Preference;
import android.preference.PreferenceManager;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.Toast;

import com.bryansalisbury.btmeasure.bluno.Bluno;
import com.bryansalisbury.btmeasure.models.Sample;
import com.bryansalisbury.btmeasure.models.TestSequence;
import com.orm.SugarRecord;

import java.util.ArrayList;

public class MeasureActivity extends AppCompatActivity {
    private Bluno bluno;

    private static final String TAG = "MeasureActivity";

    public static final String ACTION_MESSAGE_AVAILABLE =
        "com.bryansalisbury.message.AVAILABLE";
    public static final String EXTRA_VALUE = "com.
        bryansalisbury.message.EXTRA_VALUE";
```

```
private enum RemoteState {NULL, TEST, SENDBUF, MEASURE,
    CONTROL, TOGGLE_LED, ECHO}
private RemoteState mState = RemoteState.NULL;

private enum ButtonState {START, STOP}
private ButtonState mStartButton = ButtonState.START;
private TestSequence mTestSequence;

private ArrayList<Sample> mSampleBuffer = new ArrayList
   <>();

// Arduino Control Variables
// TODO move to bluno.java as this is platform specific
    restriction
private static final int sampleRateMax = 4000;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create bluno now to have bt service ready when
       needed.
    bluno = new Bluno(this);

    setContentView(R.layout.activity_measure);
    final Button buttonBegin = (Button) findViewById(R.
       id.buttonBegin);

    Spinner spinner = (Spinner) findViewById(R.id.
       spinner);
    // Create an ArrayAdapter using the string array
       and a default spinner layout
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.
       createFromResource(this,
            R.array.interrupt_count_array, android.R.
               layout.simple_spinner_item);
    // Specify the layout to use when the list of
       choices appears
    adapter.setDropDownViewResource(android.R.layout.
       simple_spinner_dropdown_item);
    // Apply the adapter to the spinner
    spinner.setAdapter(adapter);

    buttonBegin.setOnClickListener(new View.
```

```
    OnClickListener() {
     @Override
     public void onClick(View v) {
         buttonBeginClick();
     }
});

final EditText lblA0 = (EditText) findViewById(R.id
    .etA0);
final EditText lblA1 = (EditText) findViewById(R.id
    .etA1);
final EditText lblA2 = (EditText) findViewById(R.id
    .etA2);
final EditText lblA3 = (EditText) findViewById(R.id
    .etA3);
final EditText lblA4 = (EditText) findViewById(R.id
    .etA4);
final EditText lblA5 = (EditText) findViewById(R.id
    .etA5);

Switch switchA0 = (Switch) findViewById(R.id.
    switchA0);
switchA0.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener(){
     public void onCheckedChanged(CompoundButton
         buttonView, boolean isChecked){
          if(isChecked){
              lblA0.setEnabled(true);
              lblA0.requestFocus();
          }else{
              lblA0.clearFocus();
              lblA0.setEnabled(false);
          }
     }
});

Switch switchA1 = (Switch) findViewById(R.id.
    switchA1);
switchA1.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener(){
     public void onCheckedChanged(CompoundButton
         buttonView, boolean isChecked){
          if(isChecked){
              lblA1.setEnabled(true);
```

```
            lblA1.requestFocus();
        }else{
            lblA1.clearFocus();
            lblA1.setEnabled(false);
        }
    }
});

Switch switchA2 = (Switch) findViewById(R.id.
    switchA2);
switchA2.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener(){
     public void onCheckedChanged(CompoundButton
        buttonView, boolean isChecked){
        if(isChecked){
            lblA2.setEnabled(true);
            lblA2.requestFocus();
        }else{
            lblA2.clearFocus();
            lblA2.setEnabled(false);
        }
    }
});

Switch switchA3 = (Switch) findViewById(R.id.
    switchA3);
switchA3.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener(){
     public void onCheckedChanged(CompoundButton
        buttonView, boolean isChecked){
        if(isChecked){
            lblA3.setEnabled(true);
            lblA3.requestFocus();
        }else{
            lblA3.clearFocus();
            lblA3.setEnabled(false);
        }
    }
});

Switch switchA4 = (Switch) findViewById(R.id.
    switchA4);
switchA4.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener(){
```

```
            public void onCheckedChanged(CompoundButton
                buttonView, boolean isChecked){
                if(isChecked){
                    lblA4.setEnabled(true);
                    lblA4.requestFocus();
                }else{
                    lblA4.clearFocus();
                    lblA4.setEnabled(false);
                }
            }
        });

        Switch switchA5 = (Switch) findViewById(R.id.
            switchA5);
        switchA5.setOnCheckedChangeListener(new
            CompoundButton.OnCheckedChangeListener(){
            public void onCheckedChanged(CompoundButton
                buttonView, boolean isChecked){
                if(isChecked){
                    lblA5.setEnabled(true);
                    lblA5.requestFocus();
                }else{
                    lblA5.clearFocus();
                    lblA5.setEnabled(false);
                }
            }
        });
    }

    private int getPrecount(){
        Spinner spinner = (Spinner) findViewById(R.id.
            spinner);
        return getResources().getIntArray(R.array.
            interrupt_values_array)[spinner.
            getSelectedItemPosition()] ;
    }

    private void buttonBeginStart(){
        // UI Elements
        final EditText lblA0 = (EditText) findViewById(R.id
            .etA0);
        final EditText lblA1 = (EditText) findViewById(R.id
            .etA1);
        final EditText lblA2 = (EditText) findViewById(R.id
```

```
    .etA2);
final EditText lblA3 = (EditText) findViewById(R.id
    .etA3);
final EditText lblA4 = (EditText) findViewById(R.id
    .etA4);
final EditText lblA5 = (EditText) findViewById(R.id
    .etA5);
final EditText lblTestName = (EditText)
    findViewById(R.id.etTestName);

Switch switchA0 = (Switch) findViewById(R.id.
    switchA0);
Switch switchA1 = (Switch) findViewById(R.id.
    switchA1);
Switch switchA2 = (Switch) findViewById(R.id.
    switchA2);
Switch switchA3 = (Switch) findViewById(R.id.
    switchA3);
Switch switchA4 = (Switch) findViewById(R.id.
    switchA4);
Switch switchA5 = (Switch) findViewById(R.id.
    switchA5);
Button buttonBegin = (Button) findViewById(R.id.
    buttonBegin);

// Create database test sequence
mTestSequence = new TestSequence(lblTestName.
    getText().toString());
mTestSequence.compressed = true;

// Sample delay calculated from sampleRate(hz)
    value expected to be in microseconds
mTestSequence.overflowCount = getPrecount();

// measureMask tells the Ardiuno which values to
    read and send over serial.
if(switchA0.isChecked()) {
     mTestSequence.measureMask = 1;
     mTestSequence.labelA0 = lblA0.getText().
        toString();
}

if(switchA1.isChecked()) {
     mTestSequence.measureMask += (1 << 1);
```

```
        mTestSequence.labelA1 = lblA1.getText().
            toString();
    }

    if(switchA2.isChecked()) {
        mTestSequence.measureMask += (1 << 2);
        mTestSequence.labelA2 = lblA2.getText().
            toString();
    }

    if(switchA3.isChecked()) {
        mTestSequence.measureMask += (1 << 3);
        mTestSequence.labelA3 = lblA3.getText().
            toString();
    }

    if(switchA4.isChecked()) {
        mTestSequence.measureMask += (1 << 4);
        mTestSequence.labelA4 = lblA4.getText().
            toString();
    }

    if(switchA5.isChecked()) {
        mTestSequence.measureMask += (1 << 5);
        mTestSequence.labelA5 = lblA5.getText().
            toString();
    }

    // Fail on no selection indicated
    if (mTestSequence.measureMask <= 0) {
        Toast toast = Toast.makeText(
                getApplicationContext(),
                "At least one (1) input must be
                    selected!",
                Toast.LENGTH_SHORT);
        toast.show();
        return;
    }

    if (Integer.bitCount(mTestSequence.measureMask) >
        1){
        Toast toast = Toast.makeText(
                getApplicationContext(),
                "Only one (1) input supported at this
```

```
                    time.",
                Toast.LENGTH_SHORT);
        toast.show();
        return;
    }

    Log.i("Bluno_cmdString", mTestSequence.
        getConfigureString());
    mTestSequence.save();

    // TODO improve handling of test start

    buttonBegin.setText("Stop");
    mStartButton = ButtonState.STOP;

    SharedPreferences prefs = PreferenceManager.
        getDefaultSharedPreferences(this);
    final String mDevice = prefs.getString("device_mac"
        , "");

    if(!bluno.connectedTo(mDevice)){
        new Thread(new Runnable() {
            @Override
            public void run() {
                bluno.connect(mDevice);
                bluno.send(mTestSequence.
                    getConfigureString());
                mTestSequence.startTime = System.
                    nanoTime();
                mTestSequence.save();
            }
        }).start();
    }else{
        bluno.send(mTestSequence.getConfigureString());
        mTestSequence.startTime = System.nanoTime();
        mTestSequence.save();
    }
}

private void buttonBeginStop(){
    Button buttonBegin = (Button) findViewById(R.id.
        buttonBegin);
    bluno.send("A");
    buttonBegin.setText("Start");
```

```
        Log.i(TAG, "Samples collected = " + mSampleBuffer.
            size());
        if(mSampleBuffer.size() > 0) {
            SugarRecord.saveInTx(mSampleBuffer);
            mSampleBuffer.clear();
        }else{
            mTestSequence.delete();
        }
        mStartButton = ButtonState.START;
    }

    private void buttonBeginClick(){
        if(mStartButton.equals(ButtonState.START)){
            buttonBeginStart();
        }else{
            buttonBeginStop();
        }
    }

    private RemoteState StateLookup(int code){
        switch (code){
            case 0:
                return RemoteState.NULL;
            case 1:
                return RemoteState.TEST;
            case 2:
                return RemoteState.SENDBUF;
            case 3:
                return RemoteState.MEASURE;
            case 4:
                return RemoteState.CONTROL;
            case 5:
                return RemoteState.TOGGLE_LED;
            case 6:
                return RemoteState.ECHO;
            default:
                return RemoteState.NULL;
        }
    }

    // TODO return sample object from ValueUnpacker to
        handle multi input record mode
    private int ValueUnpacker(byte[] data){
        if(data.length == 2){
```

```
            return ((data[0] & 0x00FF) | ((data[1] & 0x00FF
                ) << 2));
        }
        return -1;
    }

    private void MessageHandler(byte[] data, String message
        ) {
        ProgressBar mProgress = (ProgressBar) findViewById(
            R.id.progressBar);

        if(message.startsWith("STATE")){
            String[] parts = message.split("=");
            int state = Integer.parseInt(parts[parts.length
                - 1]);
            if(mState != StateLookup(state)){
                if(mState.equals(RemoteState.SENDBUF) ||
                    mState.equals(RemoteState.MEASURE)){
                    if(RemoteState.NULL.equals(StateLookup(
                        state))){
                        buttonBeginStop();
                        mProgress.setProgress(0);
                    }
                }
            }
            mState = StateLookup(state);
            Log.i(TAG, mState.name());
            return; // Return on state change. done
                processing this message

        }else if(message.startsWith("ERROR")){
            Log.e(TAG, message);
            return; // not data to save from this message

        }else if(message.startsWith("INFO")){
            Log.i(TAG, message);
            String[] parts = message.substring(5).split("="
                );
            if(parts.length == 2){
                if(parts[0].equals("START")){
                    mTestSequence.startTime = Integer.
                        parseInt(parts[1]);
                    mTestSequence.save();
                }else if(parts[0].equals("STOP")){
```

```
                    mTestSequence.finishTime = Integer.
                        parseInt(parts[1]);
                    mTestSequence.save();
                }

            }
            return; // diagnostic message
        }

        if(mState.equals(RemoteState.MEASURE) || mState.
            equals(RemoteState.SENDBUF)){
            if(mTestSequence.compressed){
                if(data.length == 2){
                    Sample mSample = new Sample("A0",
                        ValueUnpacker(data), mTestSequence);
                    Log.v(TAG, Integer.toString(mSample.
                        value));
                    mSampleBuffer.add(mSample);
                    bluno.send("K"); // send the ACK
                        command
                }
            }else{
                Sample mSample = new Sample("A0", Integer.
                    parseInt(message), mTestSequence);
                Log.v(TAG, message);
                mSampleBuffer.add(mSample);
            }
        }
        mProgress.setProgress(mSampleBuffer.size());
    }

    private final BroadcastReceiver mBroadcastReceiver =
        new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent
            intent) {
            if(ACTION_MESSAGE_AVAILABLE.equals(intent.
                getAction())){
                byte[] data = intent.getByteArrayExtra(
                    EXTRA_VALUE);
                if(data != null && data.length > 0) {
                    MessageHandler(data, new String(data));
                }
            }
```

```
        }
    };

    private static IntentFilter makeIntentFilter() {
        final IntentFilter intentFilter = new IntentFilter
            ();
        intentFilter.addAction(ACTION_MESSAGE_AVAILABLE);
        return intentFilter;
    }

    @Override
    protected void onPause(){
        super.onPause();
        bluno.pause();
        getApplicationContext().unregisterReceiver(
            mBroadcastReceiver);
    }

    @Override
    protected void onResume(){
        super.onResume();
        bluno.resume();
        getApplicationContext().registerReceiver(
            mBroadcastReceiver, makeIntentFilter());
    }

}
```

APPENDIX D

ControlActivity.java

```
package com.bryansalisbury.btmeasure;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.ProgressBar;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.ToggleButton;

import com.bryansalisbury.btmeasure.bluno.Bluno;
import com.bryansalisbury.btmeasure.models.Sample;
import com.bryansalisbury.btmeasure.models.TestSequence;
import com.orm.SugarRecord;

import java.util.ArrayList;
import java.util.Locale;

public class ControlActivity extends AppCompatActivity {
    private Bluno bluno;
    private double kp=0, ki=0, kd=0;
    private int desiredPosition, outputPin, inputPin;
    private int maxOut = 255;
    private int minOut = 0;
    private TestSequence mTestSequence;
    private ArrayList<String> outputBuffer = new ArrayList
        <>();
```

```
public static final String ACTION_MESSAGE_AVAILABLE = "
   com.bryansalisbury.message.AVAILABLE";
public static final String EXTRA_VALUE = "com.
   bryansalisbury.message.EXTRA_VALUE";
private static final String TAG = "ControlActivity";

String mDevice;

private enum RemoteState {NULL, SENDBUF, MEASURE,
   CONTROL}
private RemoteState mState = RemoteState.NULL;

private ArrayList<Sample> mSampleBuffer = new ArrayList
   <>();

private SharedPreferences prefs;
private ProgressBar mProgress;
private TextView tvSensorValue;
private boolean feedback = true;
private boolean measure = true;


private RemoteState StateLookup(int code){
    switch (code){
        case 0:
            return RemoteState.NULL;
        case 2:
            return RemoteState.SENDBUF;
        case 3:
            return RemoteState.MEASURE;
        case 4:
            return RemoteState.CONTROL;
        default:
            return RemoteState.NULL;
    }
}
private static IntentFilter makeIntentFilter() {
    final IntentFilter intentFilter = new IntentFilter
       ();
    intentFilter.addAction(ACTION_MESSAGE_AVAILABLE);
    return intentFilter;
}

private int ValueUnpacker(byte[] data){
```

```
        if(data.length == 2){
            return ((data[0] & 0x00FF) | ((data[1] & 0x00FF
                ) << 2));
        }
        return -1;
    }

    private void saveSamples(){
        bluno.send("A");
        if(mSampleBuffer.size() > 0) {
            mTestSequence.save();
            SugarRecord.saveInTx(mSampleBuffer);
            Snackbar snackbar = Snackbar
                    .make(findViewById(android.R.id.content
                        ), "Test complete", Snackbar.
                        LENGTH_LONG)
                    .setAction("DISMISS", new View.
                        OnClickListener() {
                        @Override
                        public void onClick(View view) {

                        }
                    });
            snackbar.show();
            mSampleBuffer.clear();
        }else{
            mTestSequence.delete();
        }
    }

    private void MessageHandler(byte[] data, String message
        ) {
        if(message.startsWith("STATE")){
            String[] parts = message.split("=");
            int state = Integer.parseInt(parts[parts.length
                - 1]);
            if(mState != StateLookup(state)) {
                if(mState.equals(RemoteState.SENDBUF) ||
                    mState.equals(RemoteState.CONTROL)){
                    if(RemoteState.NULL.equals(StateLookup(
                        state))){
                        saveSamples();
                        mProgress.setProgress(0);
                    }
```

```
            }
        }
        mState = StateLookup(state);

    }else if(message.startsWith("ERROR")){
        Log.e(TAG, message);

    }else if(message.startsWith("INFO")){
        Log.i(TAG, message);
        String[] parts = message.substring(5).split("="
            );

        if(parts.length == 2){
            //confirm setting values here
            /*
            if(parts[0].equals("START")){
                mTestSequence.startTime = Integer.
                    parseInt(parts[1]);
                mTestSequence.save();
            }else if(parts[0].equals("STOP")){
                mTestSequence.finishTime = Integer.
                    parseInt(parts[1]);
                mTestSequence.save();
            }*/

        }

    }else if(mState.equals(RemoteState.SENDBUF)){
        if(data.length == 2){
            Sample mSample = new Sample("A0",
                ValueUnpacker(data), mTestSequence);
            Log.v(TAG, Integer.toString(mSample.value))
                ;
            mSampleBuffer.add(mSample);
            bluno.send("K"); // send the ACK command
            mProgress.setProgress(mSampleBuffer.size())
                ;
        }
    }else{
        tvSensorValue.setVisibility(View.VISIBLE);
        try{
            tvSensorValue.setText(String.format(Locale.
                getDefault(), "%1$.2fv", (5.00/1023.0)*(
                Integer.valueOf(message))));
```

```
            }catch (Exception ex){

            }
        }

    }

    private final BroadcastReceiver mBroadcastReceiver =
       new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent
           intent) {
            if(ACTION_MESSAGE_AVAILABLE.equals(intent.
               getAction())){
                final byte[] data = intent.
                   getByteArrayExtra(EXTRA_VALUE);
                if(data != null && data.length > 0) {
                    MessageHandler(data, new String(data));
                }
            }
        }
    };

    private void sendConfig(){
        bluno.send("C");

        if(feedback) {
            bluno.send(String.format(Locale.getDefault(), "
               PA%1$.1f\nPB%2$.1f\nPC%3$.1f\n", kp, ki, kd)
               );
            bluno.send("PF1\n");
        }else{
            bluno.send("PF0\n");
        }

        if(measure){
            bluno.send("PM1\n");
        }else{
            bluno.send("PM0\n");
        }
        bluno.send("PD" + desiredPosition + "\n");
        bluno.send("PH" + maxOut + "\nPL" + minOut + "\n");
        bluno.send("PO" + outputPin + "\nPI" + inputPin + "
           \n");
```

```
        bluno.send("PE" + mTestSequence.overflowCount + "\n
            ");
        bluno.send("PS\n"); // start flag
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_control);

        //instantiate bluno
        bluno = new Bluno(this);

        final SeekBar seekKp = (SeekBar) findViewById(R.id.
            seekKp);
        final SeekBar seekKi = (SeekBar) findViewById(R.id.
            seekKi);
        final SeekBar seekKd = (SeekBar) findViewById(R.id.
            seekKd);
        mProgress = (ProgressBar) findViewById(R.id.
            progressBar);

        final SeekBar seekTarget = (SeekBar) findViewById(R
            .id.seekTarget);
        final SeekBar seekMaxOutput = (SeekBar)
            findViewById(R.id.seekMaxOutput);

        final TextView tvKp = (TextView) findViewById(R.id.
            tvKpValue);
        final TextView tvKi = (TextView) findViewById(R.id.
            tvKiValue);
        final TextView tvKd = (TextView) findViewById(R.id.
            tvKdValue);
        final TextView tvTarget = (TextView) findViewById(R
            .id.tvTargetValue);
        final TextView tvMaxOut = (TextView) findViewById(R
            .id.tvMaxOutVal);

        Button btnPos1 = (Button) findViewById(R.id.btnPos1
            );
        Button btnNeg1 = (Button) findViewById(R.id.btnNeg1
            );
        Button btnPos10 = (Button) findViewById(R.id.
```

```
    btnPos10);
Button btnNeg10 = (Button) findViewById(R.id.
    btnNeg10);
Button btnSend = (Button) findViewById(R.id.btnSend
    );

ToggleButton btnFeedback = (ToggleButton)
    findViewById(R.id.toggleFeedback);
ToggleButton btnTestmode = (ToggleButton)
    findViewById(R.id.toggleTestmode);
tvSensorValue = (TextView) findViewById(R.id.
    textSensor);


tvKp.setText(Integer.toString(seekKp.getProgress())
    );
tvKi.setText(Integer.toString(seekKi.getProgress())
    );
tvKd.setText(Integer.toString(seekKd.getProgress())
    );
tvTarget.setText(Integer.toString(seekTarget.
    getProgress()));
tvMaxOut.setText(Integer.toString(seekMaxOutput.
    getProgress()));

seekKp.setOnSeekBarChangeListener(new SeekBar.
    OnSeekBarChangeListener() {
     @Override
     public void onProgressChanged(SeekBar seekBar,
         int i, boolean b) {
          kp = i / 10.0;
          tvKp.setText(String.format(Locale.
              getDefault(), "%1$.1f", kp));
     }

     @Override
     public void onStartTrackingTouch(SeekBar
         seekBar) {

     }

     @Override
     public void onStopTrackingTouch(SeekBar seekBar
         ) {
```

```
    }
});

seekKi.setOnSeekBarChangeListener(new SeekBar.
    OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar,
        int i, boolean b) {
        ki = i / 10.0;
        tvKi.setText(String.format(Locale.
            getDefault(), "%1$.1f", ki));
    }

    @Override
    public void onStartTrackingTouch(SeekBar
        seekBar) {

    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar
        ) {
    }
});

seekKd.setOnSeekBarChangeListener(new SeekBar.
    OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar,
        int i, boolean b) {
        kd = i / 10.0;
        tvKd.setText(String.format(Locale.
            getDefault(), "%1$.1f", kd));
    }

    @Override
    public void onStartTrackingTouch(SeekBar
        seekBar) {

    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar
        ) {
```

```
    }
});

seekTarget.setOnSeekBarChangeListener(new SeekBar.
    OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar,
        int i, boolean b) {
        desiredPosition = i;
        tvTarget.setText(String.format(Locale.
            getDefault(), "%1$.2fv", (5.00/1023.0)*i
            ));
    }

    @Override
    public void onStartTrackingTouch(SeekBar
        seekBar) {

    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar
        ) {

    }
});

seekMaxOutput.setOnSeekBarChangeListener(new
    SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar,
        int i, boolean b) {
        maxOut = i;
        tvMaxOut.setText(String.format(Locale.
            getDefault(), "%1$d", i));
    }

    @Override
    public void onStartTrackingTouch(SeekBar
        seekBar) {

    }
```

```
        @Override
        public void onStopTrackingTouch(SeekBar seekBar
            ) {

        }
    });

    btnNeg1.setOnClickListener(new View.OnClickListener
        () {
        @Override
        public void onClick(View view) {
            seekTarget.setProgress(seekTarget.
                getProgress() - 1);
        }
    });

    btnPos1.setOnClickListener(new View.OnClickListener
        () {
        @Override
        public void onClick(View view) {
            seekTarget.setProgress(seekTarget.
                getProgress() + 1);
        }
    });

    btnNeg10.setOnClickListener(new View.
        OnClickListener() {
        @Override
        public void onClick(View view) {
            seekTarget.setProgress(seekTarget.
                getProgress() - 10);
        }
    });

    btnPos10.setOnClickListener(new View.
        OnClickListener() {
        @Override
        public void onClick(View view) {
            seekTarget.setProgress(seekTarget.
                getProgress() + 10);
        }
    });

    btnSend.setOnClickListener(new View.OnClickListener
```

```
    () {
     @Override
     public void onClick(View view) {
          if(mSampleBuffer.size() > 0){
               mSampleBuffer.clear();
               mProgress.setProgress(0);
               bluno.send("A");
               return;
          }

          mTestSequence = new TestSequence("Control_
             Mode_Run");
          prefs = PreferenceManager.
             getDefaultSharedPreferences(
             getApplicationContext());

          inputPin = Integer.parseInt(prefs.getString
             ("control_input", "0"));
          outputPin = Integer.parseInt(prefs.
             getString("control_output", "9"));
          mTestSequence.overflowCount = Integer.
             parseInt(prefs.getString("
             control_precount", "65285"));
          mDevice = prefs.getString("device_mac", "")
             ;

          if(bluno.connectedTo(mDevice)){
               sendConfig();
          }else {
               new Thread(new Runnable() {
                    @Override
                    public void run() {
                         bluno.connect(mDevice);

                         sendConfig();

                    }
               }).start();
          }
     }
});

btnFeedback.setOnCheckedChangeListener(new
   CompoundButton.OnCheckedChangeListener(){
```

```
            @Override
            public void onCheckedChanged(CompoundButton
                buttonView, boolean isChecked){
                if(!isChecked){
                    seekKp.setProgress(0);
                    seekKd.setProgress(0);
                    seekKi.setProgress(0);
                    seekKp.setEnabled(false);
                    seekKd.setEnabled(false);
                    seekKi.setEnabled(false);
                    feedback = false;
                }else{
                    seekKp.setEnabled(true);
                    seekKd.setEnabled(true);
                    seekKi.setEnabled(true);
                    feedback = true;
                }
            }
        });

        btnTestmode.setOnCheckedChangeListener(new
            CompoundButton.OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(CompoundButton
                buttonView, boolean isChecked){
                measure = isChecked;
            }
        });
    }

    @Override
    protected void onPause(){
        bluno.pause();
        getApplicationContext().unregisterReceiver(
            mBroadcastReceiver);
        super.onPause();
    }

    @Override
    protected void onResume(){
        super.onResume();
        bluno.resume();
        getApplicationContext().registerReceiver(
            mBroadcastReceiver, makeIntentFilter());
```

```
        }
    }
```

APPENDIX E

ResultsActivity.java

```
package com.bryansalisbury.btmeasure;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.support.design.widget.Snackbar;
import android.support.v4.view.MenuItemCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.support.v7.widget.ShareActionProvider;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import com.bryansalisbury.btmeasure.models.Sample;
import com.bryansalisbury.btmeasure.models.TestSequence;
import com.github.mikephil.charting.charts.ScatterChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.data.Entry;
import com.github.mikephil.charting.data.ScatterData;
import com.github.mikephil.charting.data.ScatterDataSet;
import com.orm.SugarRecord;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

import static android.support.v4.content.FileProvider.
    getUriForFile;

public class ResultsActivity extends AppCompatActivity {
```

```
private Long mTestSequenceID;
private ShareActionProvider mShareActionProvider;

private List<Sample> mSamples = null;
private TestSequence mTestSequence = null;
private List<Entry> entries = new ArrayList<>();

private Runnable sampleLoader = new Runnable() {
    @Override
    public void run() {
        mTestSequence = TestSequence.findById(
           TestSequence.class, mTestSequenceID);
        mSamples = Sample.findWithQuery(Sample.class,
                "select * from Sample where
                   test_sequence = ?",
                mTestSequenceID.toString());

        if(mSamples.size() != 700 && mSamples.size() !=
            750){
            Snackbar snackbar = Snackbar
                    .make(findViewById(android.R.id.
                       content), String.format(Locale.
                       getDefault(), "Unexpected count
                       (%1$d)", mSamples.size()),
                       Snackbar.LENGTH_INDEFINITE)
                    .setAction("DISMISS", new View.
                       OnClickListener() {
                        @Override
                        public void onClick(View view)
                           {

                        }
                    })
                    .setActionTextColor(getResources().
                       getColor(android.R.color.
                       holo_red_light ));
            snackbar.show();
        }

        int i = 0;
        for (Sample theSample : mSamples) {
            // turn your data into Entry objects
            entries.add(new Entry((float)
               getTimeForSample(i, mTestSequence.
```

```
                getTImeDelta()), (float) theSample.
                getVolts()));
        i++;
    }

    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            ScatterChart chart = (ScatterChart)
                findViewById(R.id.chart);
            if(mShareActionProvider != null) {
                mShareActionProvider.setShareIntent
                    (createShareIntent());
            }
            ScatterDataSet dataSet = new
                ScatterDataSet(entries, "Analog␣
                Input");
            ScatterData scatterData = new
                ScatterData(dataSet);
            chart.getAxisLeft().setAxisMaxValue((
                float)5.00);
            chart.getAxisLeft().setAxisMinValue((
                float)0.00);
            dataSet.setScatterShape(ScatterChart.
                ScatterShape.CIRCLE);
            dataSet.setScatterShapeHoleRadius(0f);
            dataSet.setScatterShapeSize(8f);

            Legend l = chart.getLegend();
            l.setWordWrapEnabled(true);
            l.setVerticalAlignment(Legend.
                LegendVerticalAlignment.BOTTOM);
            l.setHorizontalAlignment(Legend.
                LegendHorizontalAlignment.CENTER);
            l.setOrientation(Legend.
                LegendOrientation.HORIZONTAL);
            l.setDrawInside(false);

            chart.setData(scatterData);
            chart.invalidate(); // refresh

            setTitle(mTestSequence.testName);
        }
    });
```

```
    }
};


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_results);

    Intent intent = getIntent();
    mTestSequenceID = (long) intent.getIntExtra("
       TEST_SEQUENCE_ID", -1);
    if (mTestSequenceID != -1) {
        new Thread(sampleLoader).start();
    }else{
        this.onBackPressed();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action
       bar if it is present.
    getMenuInflater().inflate(R.menu.results, menu);
    MenuItem item = menu.findItem(R.id.menu_item_share)
       ;
    mShareActionProvider = (ShareActionProvider)
       MenuItemCompat.getActionProvider(item);
    Intent shareIntent = new Intent(Intent.ACTION_SEND)
       ;
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action
       bar will
    // automatically handle clicks on the Home/Up
       button, so long
    // as you specify a parent activity in
       AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
```

```
        if (id == R.id.menu_item_delete) {
            AlertDialog.Builder builder = new AlertDialog.
                Builder(this);
            builder.setMessage("Permanently delete this
                dataset?")
                    .setPositiveButton("Delete", new
                        DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface
                            dialog, int id) {
                            SugarRecord.deleteInTx(mSamples
                                );
                            SugarRecord.delete(
                                mTestSequence);
                            onBackPressed();
                        }})
                    .setNegativeButton("Cancel", new
                        DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface
                            dialogInterface, int i) {

                        }
                    });
            builder.show();
            return true;
        }else if(id == R.id.menu_item_details){
            AlertDialog.Builder builder = new AlertDialog.
                Builder(this);
            builder.setMessage("Number of samples: " +
                mSamples.size())
                    .setPositiveButton("Close", new
                        DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface
                            dialog, int id) {
                            // FIRE ZE MISSILES!
                        }})
                    .setTitle("Details");
            builder.show();
        }

        return super.onOptionsItemSelected(item);
    }

    private double getTimeForSample(int i, double delta){
```

```
        return (i * delta);
    }

    private Uri writeSamplesToFile() {
        File testPath = new File(getApplicationContext().
            getFilesDir(), "tests");
        testPath.mkdirs();
        File newFile = new File(testPath, mTestSequence.
            timestamp +".csv");

        try {
            FileOutputStream f = new FileOutputStream(
                newFile);
            PrintWriter pw = new PrintWriter(f);
            pw.println("Start time:, " + mTestSequence.
                startTime);
            pw.println("End time:, " + mTestSequence.
                finishTime);
            long duration = (mTestSequence.finishTime -
                mTestSequence.startTime);
            pw.println("Duration:, " + duration);
            pw.println("Sample count:, " + mSamples.size())
                ;
            pw.println("Sample rate:, " + (1.00 /
                mTestSequence.getTImeDelta()));
            pw.println("");
            pw.println("num, time, value, volts");
            Integer i = 0;

            for (Sample sample : mSamples) {
                pw.println(i + "," + getTimeForSample(i,
                    mTestSequence.getTImeDelta()) + "," +
                    sample.value + "," + sample.getVolts());
                i = i + 1;
            }

            pw.flush();
            pw.close();
            f.close();

            return getUriForFile(getApplicationContext(), "
                com.bryansalisbury.fileprovider", newFile);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
```

```
            //TAG, "File not found");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    private Intent createShareIntent(){
        Uri fileUri = writeSamplesToFile();
        Intent shareIntent = new Intent(Intent.ACTION_SEND)
            ;
        shareIntent.setType("text/csv");
        shareIntent.putExtra(Intent.EXTRA_STREAM, fileUri);
        shareIntent.setFlags(Intent.
            FLAG_GRANT_READ_URI_PERMISSION);
        return shareIntent;
    }
}
```

APPENDIX F

VB Serial Test Client

```
 Imports System
Imports System.IO.Ports

Public Class Form1

    Private Sub btnConnect_Click(sender As Object, e As
       EventArgs) Handles btnConnect.Click
        serialPort.PortName = txtPort.Text
        serialPort.BaudRate = CInt(txtBaud.Text)
        serialPort.Open()

        If serialPort.IsOpen Then
            btnConnect.Text = "Connected!"
            btnConnect.Enabled = False
        End If
    End Sub

    Private Sub btnTest_Click(sender As Object, e As
       EventArgs) Handles btnTest.Click
        Dim timeFinish As ULong
        Dim samples As New List(Of Integer)
        Dim samplesTime As New List(Of ULong)
        Dim sampleBuf As New List(Of Byte)
        Dim results As String = ""

        If serialPort.IsOpen Then
            lblStatus.Visible = True
            Application.DoEvents()

            ' Empty the buffer so we can properly measure
               time
            serialPort.DiscardInBuffer()

            ' timeFinish = Ticks (100 nanoseconds) +
               numDuration(100 nanoseconds) (in ticks)
            ' 10000 ticks per millisecond or 10 million
               ticks per second
            Dim value As ULong = Math.Round(numDuration.
               Value * (10 * 10 ^ 6))
```

```
timeFinish = DateTime.UtcNow.Ticks + value


' Fill a buffer with data
' This could be done in realtime, but for
    testing purposes this is sufficient
While DateTime.UtcNow.Ticks < timeFinish
     'If (serialPort.BytesToRead > 0) Then
     sampleBuf.Add(serialPort.ReadByte())
     'End If
End While

For i As Integer = 0 To sampleBuf.Count - 1
     Try
          If sampleBuf(i) = 10 Or sampleBuf(i +
             1) = 10 Then
               Continue For
          End If

          Dim LSB As Integer = sampleBuf(i)
          Dim MSB As Integer = sampleBuf(i + 1)
          samples.Add((MSB << 2) + (LSB))
          samplesTime.Add(DateTime.UtcNow.Ticks)

     Catch ex As Exception
          'Might see an exception if i+1 is not
             set
     End Try
Next

results += "Test Date:      " & DateTime.Now.
    ToLocalTime & vbNewLine
results += "Test Duration: " & numDuration.
    Value.ToString & " sec" & vbNewLine
results += "Total Samples: " & samples.Count.
    ToString & vbNewLine
results += "Sample Freq:    " & Math.Round(
    samples.Count / numDuration.Value) & " Hz" &
     vbNewLine
results += "Last sample:    " & samples(samples.
    Count - 1).ToString

txtResults.Text = results
lblStatus.Visible = False
```

```
            results = ""
            For i As Integer = 0 To samples.Count - 1
                results += samplesTime(i).ToString & "," &
                    samples(i).ToString & vbNewLine
            Next
            Dim Filename As String = Application.
                StartupPath & "\results.txt"
            IO.File.WriteAllText(Filename, results)
        End If
    End Sub

    Private Sub btnWrite_Click(sender As Object, e As
        EventArgs) Handles btnWrite.Click
        If serialPort.IsOpen Then
            serialPort.Write(txtOutput.Text)
            txtOutput.Text = ""
        Else
            MsgBox("Serial connection must be open first!")
        End If
    End Sub

    Private Sub txtOutput_KeyDown(sender As Object, e As
        KeyEventArgs) Handles txtOutput.KeyDown
        If e.KeyCode = Keys.Enter Then
            If serialPort.IsOpen Then
                serialPort.Write(txtOutput.Text)
                txtOutput.Text = ""
            Else
                MsgBox("Serial connection must be open
                    first!")
            End If
        End If
    End Sub
End Class
```

BIBLIOGRAPHY

Arduino. "Hardware & related initiatives." Accessed: Dec 2, 2016. [Online]. Available: http://playground.arduino.cc/Main/SimilarBoards#goShie

Bajarin, B. Time Inc. "The pc has been decentralized mobile is the new center." Accessed: December 10, 2016. [Online]. Available: http://time.com/4040/the-pc-has-been-decentralized-mobile-is-the-new-center/

Bitar, H. "Ardudroid: A simple 2-way bluetooth-based android controller for arduino." Accessed: Feb 10, 2015. [Online]. Available: http://www.techbitar.com/ardudroid-simple-bluetooth-control-for-arduino-and-android.html

DFRobot. "Bluno - an arduino bluetooth 4.0 (ble) board." Accessed: Dec 2, 2016. [Online]. Available: https://www.dfrobot.com/index.php?route=product/product&product_id=1044

Jouaneh, M. K., *Fundamentals of mechatronics*. Stamford, CT: Cengage Learning, 2013.

Jouaneh, M. K. and Palm III, W. J., "Control systems take-home experiments," *IEEE CONTROL SYSTEMS MAGAZINE*, vol. 31, pp. 44–53, Aug. 2013.

Kaufmann, B. and Buechley, L., "Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing," in *12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10)*, New York, NY, USA, 2010.

Measurement Computing Corporation. "Wireless multifunction daq device with android and windows support." Accessed: Dec 2, 2016. [Online]. Available: http://www.mccdaq.com/wireless-data-acquisition/BTH-1208LS.aspx

Practical Arduino. "Oscilloscope / logic analyzer." Accessed: April 15, 2015. [Online]. Available: http://www.practicalarduino.com/projects/scope-logic-analyzer

Richardson, G. "Sensorgraph." Accessed: April 10, 2015. [Online]. Available: http://bygriz.com/sensor-graph/#more-1344