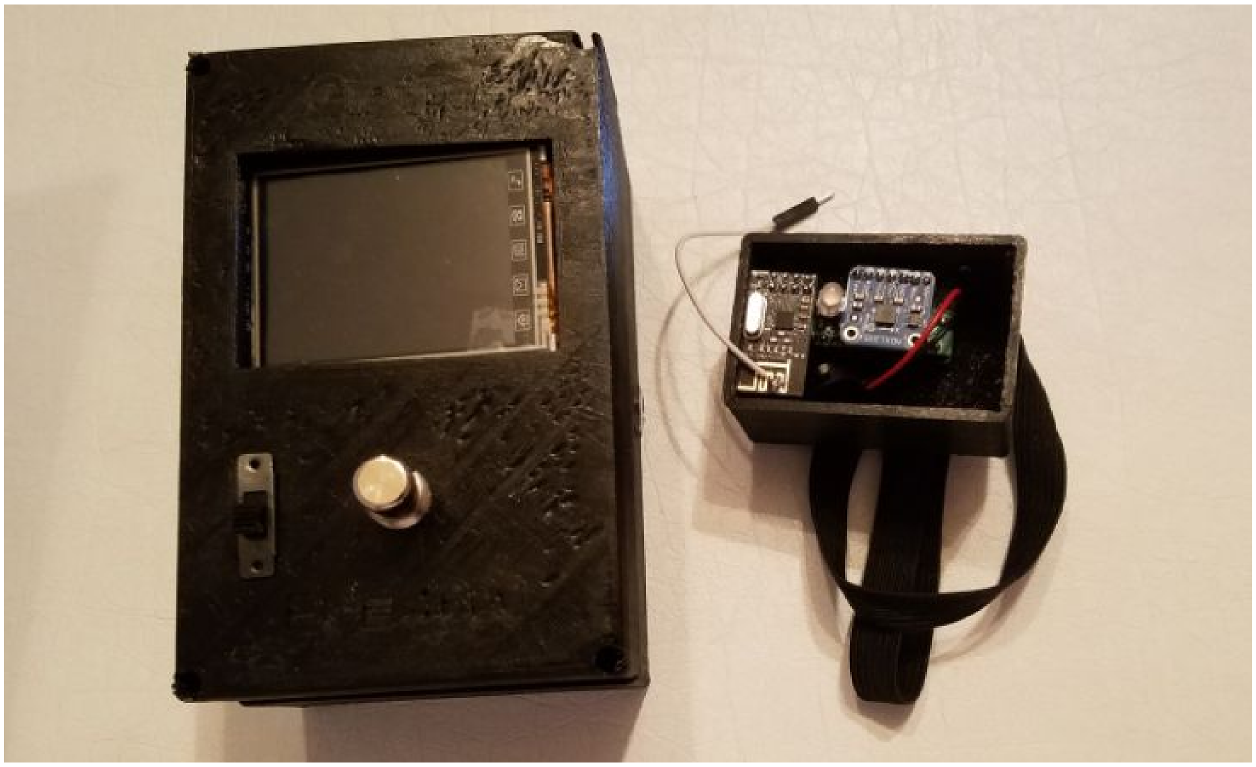


Wireless Wah Pedal

ECE:4600 Capstone Project

Winter 2017 Instructor: Dr. Syed Mahmud

Group 4:Michael A. Fornoff Jeffrey D. Berg Nicholas D. Patten

Table of Contents

Abstract 3

Executive Summary 4

Introduction 5

Background Materials 5

Description of Work 7

- Transmitter Design 7
- Receiver Design 11
- Filter Operation 18

Operating Procedure 20

Analysis, Results, Discussion 20

Conclusion 22

References 23

Appendix 24

- A: Alternate Designs 24
- B: Regulatory Constraints 24
- C: Parts List/Cost Analysis 25
- E: Problems Encountered/Solved 26
- F: Distribution of Work 27
- G: Program Listing 28
- H: Component Data Sheets 36

Table of Figures

Figure 1: Patent, Wah Filter Board, 1971 6
Figure 2: Transmitter Schematic (Omitting Breakout Board) 7
Figure 3: Transmitter Breakout Board and Pin Mapping 8
Figure 4: Transmitter Program Flowchart 10
Figure 5: 3d Model for Transmitter Housing 11
Figure 6: Completed Transmitter 11
Figure 7: Receiver Schematic 12
Figure 8: Receiver Program Flowchart 13
Figure 9: GUI Menu Icons 14
Figure 10: X-Y Plotting Area 14
Figure 11: GUI Options Screen 15
Figure 12: Screen Display for Changing Foot Position 16
Figure 13: 3d Model for Receiver Housing 17
Figure 14: Completed Receiver 17
Figure 15: Wah Filter Operation 18
Figure 16: Wah Filter Response vs. Potentiometer Resistance 19
Figure 17: Wah Filter Schematic Diagram 19
Figure 18: Packet Error Rate as a Function of Distance 21
Figure 19: Starting Locations for Trials A,B, and C 21

Abstract

Guitar players use a number of analog-filtered devices to add a wide range of effects to the sound of their instruments. Among these are distortion, chorus, reverb, delay, and wah. The pedals are often attached to a board and connected in series, one into the next, to complete an electrical circuit between the guitar and amplifier/speakers. However, the use of these devices often leaves the musician anchored to one spot on stage to manually manipulate the attached pedals to obtain a specific sound. This is a problem for any user who wishes to move around while playing. There currently exists no viable solution to this problem for musicians, who are forced to return to their assigned position on the stage to properly use their equipment.

In this paper we present a simple and effective means for a musician to manipulate the filters of a guitar-effects board without having to be physically located near its pedals. Using some programmable microcontrollers, simple communications equipment, and a physical interface to the filter of the pedal, the user can realize a simple and effective, wireless effects pedal that can be operated from anywhere on stage. This paper describes in detail the design of such a device, and the implications it has for stage musicians.

Executive Summary

In the world of music, the guitar is certainly among the most popular instruments. Since the 1940s and 1950s, guitar players have used stand-alone effects units, or guitar pedals, to modify the sound of their instruments. The pedals use analog filters to achieve many different effects: to add a chorus, reverberation, favor certain frequencies over others, or completely distort the signal altogether in “hard” rock songs. However, a guitar player must return to the position of his pedals when he wishes to make a modification to the setup of the pedals, which can happen many times for each song.

Our team has designed a proof-of-concept prototype for professional musicians, which enables them to move about the stage and manipulate the effect of a wah pedal, wirelessly, from any location nearby. We used a gyroscopic sensor to get the user’s foot position, two programmable microcontrollers, wireless modules to transmit the information, and a physical interface to the wah pedal. The resulting prototype is very effective, with minimal signal-to-action latency, and offers a guitar player the freedom to move unrestricted without having to worry about returning to position for a particular cue during a song.

This product does not currently exist in the market, and nearly every musician we have consulted has expressed a strong interest in such a device. It features steady and reliable gyroscope input, fast data transmission that should provide coverage for an entire stage, good sound, and a user interaction experience with many adjustable features.

Our team has already encountered and solved many of the construction/compatibility issues inherent in the device technology. More research and development is needed to change the construction of the device to make is smaller and less expensive. However, we estimate the total cost of such a product to be in the range of $50 per transmitter/receiver pair, or $25 without the touchscreen. We estimate that such a package could easily sell for more than $150 at retail, especially with the benefit of being the first to market.

Introduction

Any successful concert guitar player knows that the music is only half of the show. Modern rock musicians cannot simply sit around and play a good album version of their music if they wish to be known for putting on a good show. Guitar players often move the entire length of the stage to interact with the crowd, but must return to their effects setup each time they wish to make modifications to their effects pedals.

Guitar effects pedals are linked together, one after the other using standard guitar cable, with the output of each pedal being attached to the input of the next pedal until the circuit is completed at the amplifier/speakers. The pedals are typically physically attached to a board using hardware fasteners, and positioned on stage where the guitar player must return each time an adjustment is needed to the effects setup. Often, several changes to the effects of the guitar loop must be made during the course of a single song. For example, one section may involve a “clean” sound, with all pedals being inactive. Later, distortion or overdrive may be added to make the sound “fuzzy” or “aggressive.” For sections in which the guitar is featured as a solo, the musician often adds chorus, reverberation, or wah to give the instrument some emphasis, or to make it sound “prettier” on its own.

What follows is a description of a design for a wireless effects pedal that offers the same functionality as a standard wah pedal - but with the flexibility to allow the user to move freely around the stage during normal use.

Background Materials

Wah pedals themselves currently exist in the market and have been around since the 1960s, but were not popularized until the early 1970s. One of the first patents to been seen for a wah pedal was from a group of three men: B.J Plunkett, L.L Kushner, and W.T Jessup in 1971. The approval came a handful of years after the initial submission. The figure on the next page shows the patent drawing for the device and shows a basic overview of the filter schematic. The patent shows a rudimentary block diagram, schematic, and the frequency response for the entire pedal.

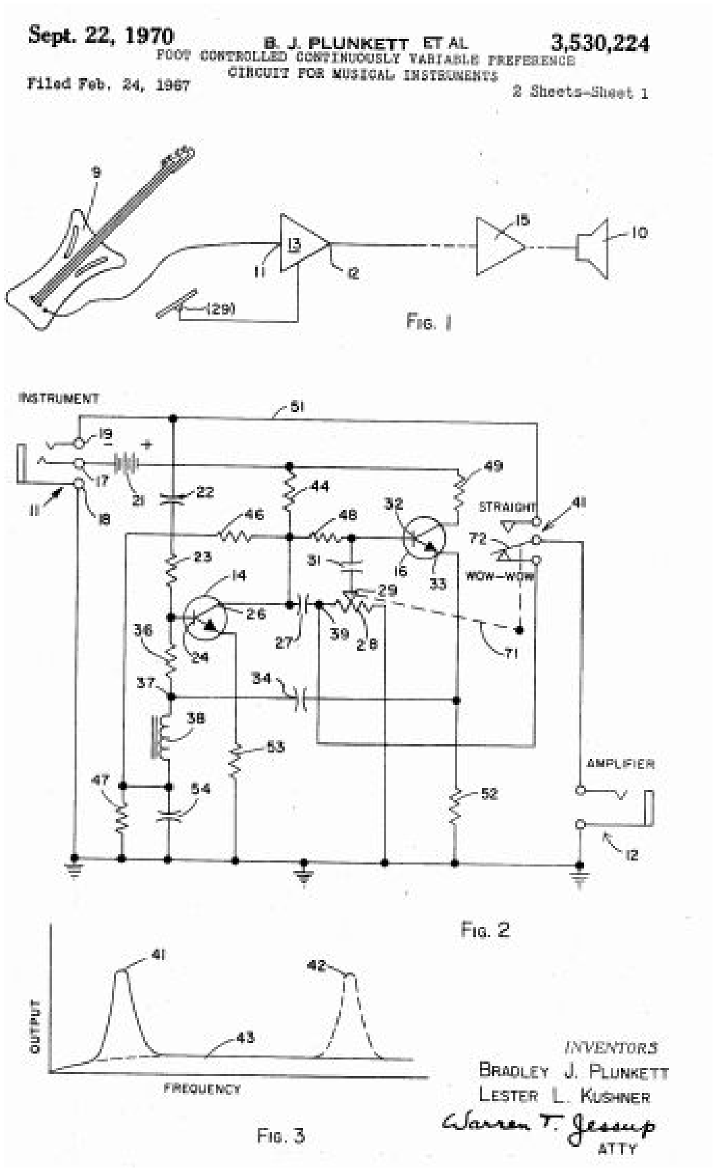


Figure 1: Patent, Wah Filter Board, 1971

Description of Work

Transmitter Design

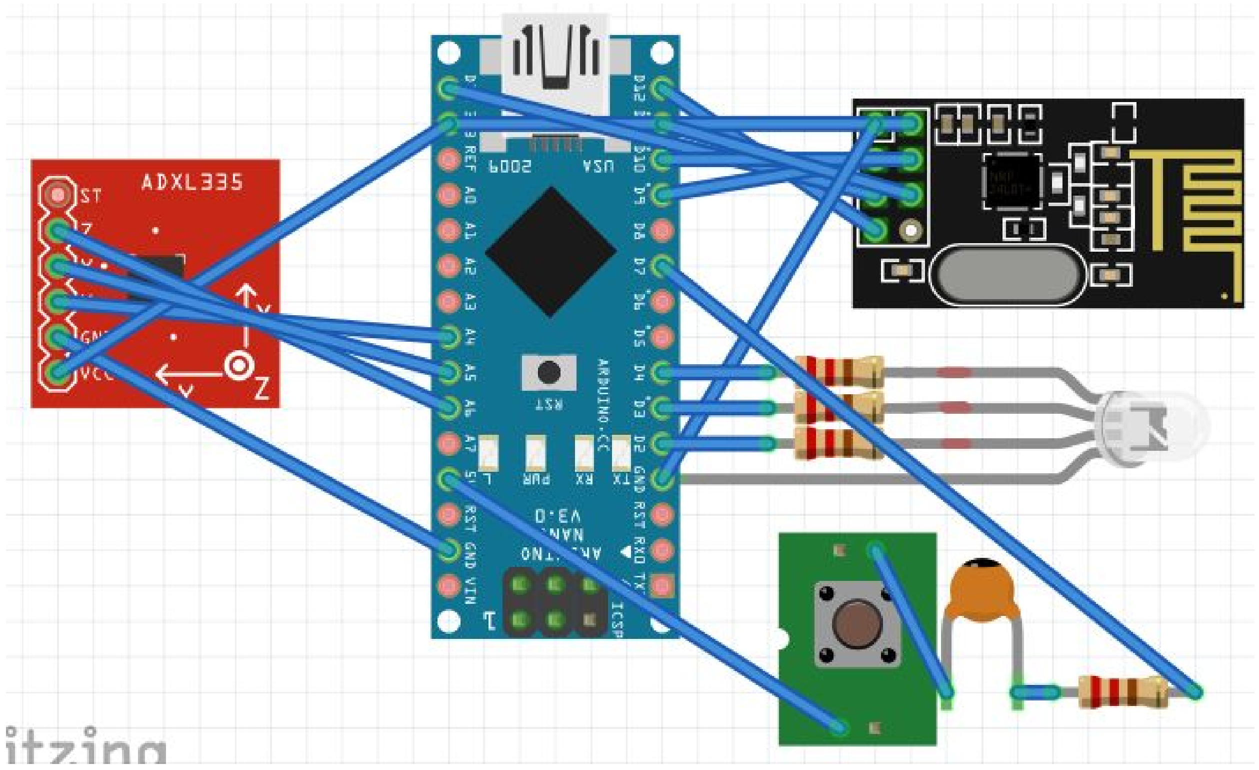
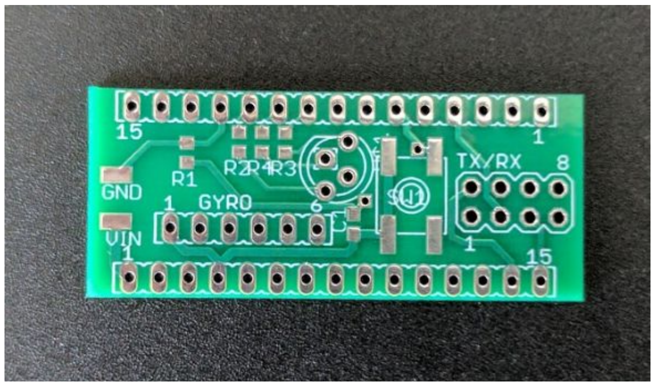


Figure 2: Transmitter Schematic (Omitting Breakout Board)

Hardware:

For the transmitter, we needed to keep the hardware as tight and secured as possible. Seeing as we were dealing with a very sensitive gyroscope, we needed it to be as secure to the board as possible so any vibrations from the user will not contaminate the data to some extreme amount. We also wanted our transmitter to be as noise free as possible. So we couldn't have it hanging off to the side, or loosely fitting inside the housing. This means that for the transmitter, we would need a suitable surface to replace the breadboard prototype and offer solid support for the devices. This is where we decided to draft a Printed Circuit Board (PCB) for the project. PCBs are used in every commercial electronics product on the market for holding components and other devices that need to interface with a circuit. The figures below show the PCB and the schematic that we used that were designed using EagleCAD, a free and open-source PCB fabrication software tool.



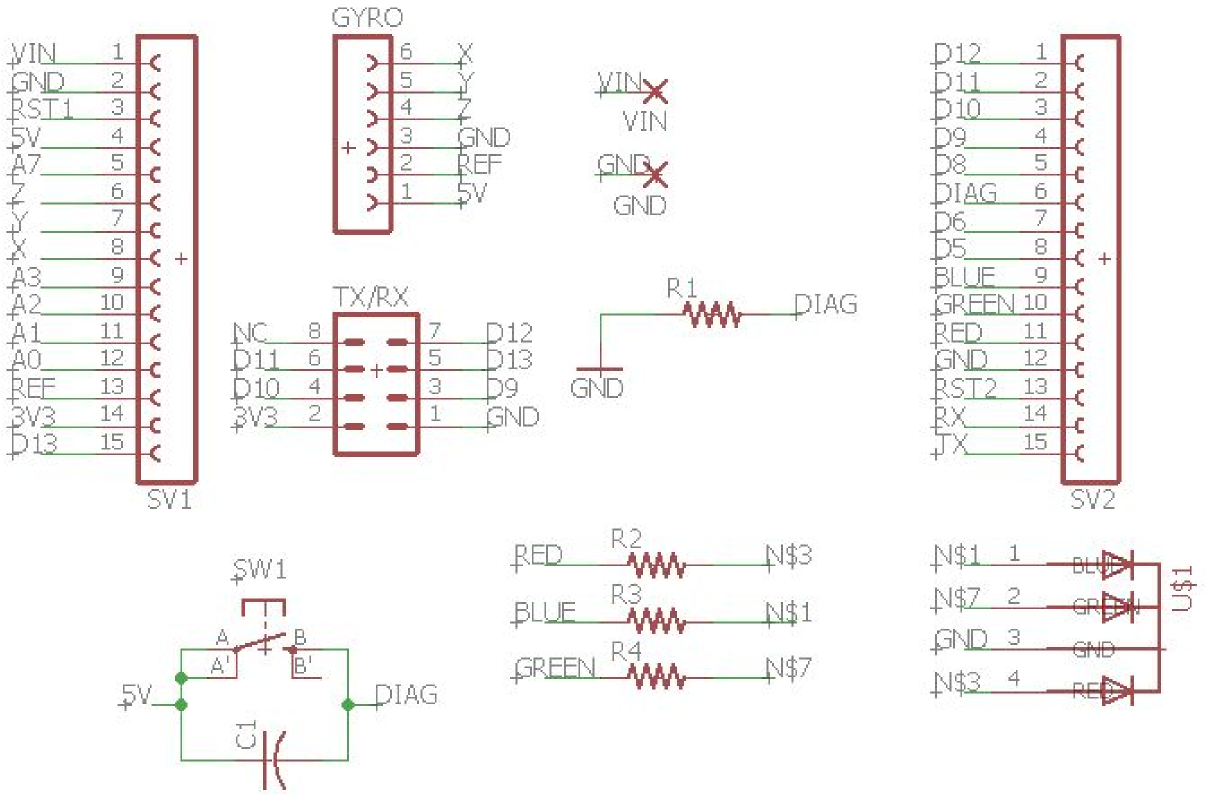


Figure 3: Transmitter Breakout Board and Pin Mapping

The schematic above shows what was used on the completed PCB for the project. The idea was that the PCB would be designed to be directly soldered upside down relative to the Arduino Nano. This was so we could have our Arduino inside the housing while the gyro and transmitter would be poking out the top of the housing to avoid colliding pieces. This was mainly accomplished by mounting female pin headers to the board for the gyro and transmitter breakout boards to be seated properly. We also added an RGB LED so that the user would be able to receive visual cues from the controller as to what mode or condition the transmitter is in. Green was normal status, red blinking was calibration, and blue was diagnostic mode. There's also a momentary pushbutton switch that was mounted with a hardware debounce. This is because with the arduino mounted in the housing upside down we lose the ability to interface with the reset button. We're able to reset the Arduino using outside switches by pushing the button and using software to dictate if we can look at that new switch as a reset switch or not. We could also use the same logic to switch between operation modes. The battery for the entire transmitter system was directly soldered onto two pads. In the future we'll be adding an SPST switch to conserve battery life.

Software:

The transmitter was the easier of the two devices to code. It begins by including libraries for the NRF24 and general serial communication. It defines certain variables using pre-compiler macros so that those values can be defined by reference and easily changed at a later time. It also pre-initializes global variables as necessary so that the module operates as intended during the first iteration of the loop.

The setup( ) function initializes class objects for the radio module and serial lines, and calls a function to configure the gyroscope. The gyroscope is an analog device that generates a signal in the 0-3.3v range. Even at a zero-degree incline, it generates a non-zero signal. Thus, when the device is first powered on, it illuminates the LED a red color to alert the user that calibration is taking place. It takes 15 measurements, averages them, and records the result as a down-position value. The LED then lights yellow so that the user can lift his foot into the highest position. After two seconds, the LED turns red and the device calibrates the up position using the same 15-measurement procedure. Once the up position is calibrated, all future readings are mapped from the 'low-high' values to 0-180 values consistent with servo operation on the receiver side.

The main loop uses functions to read a new gyroscope value with every iteration. The map( ) and constrain( ) functions convert the gyroscope reading to a value between 0 and 180. It also maintains the value of the reading from the previous iteration of the main loop, and only updates the gyro reading variable if the difference between the old and new values is greater than three. This prevents the transmitter from sending values that are rapidly fluctuating due to sensor noise. The transmitter includes a resistor and capacitor at the gyro pin for the purpose of debouncing the signal, but some software debouncing is also desirable. If the difference is less than three, then the transmitter will simply send the old value.

The only remaining operation of the transmitter is to send diagnostic messages to the user via a serial console in the event that the diagnostic pin 7 is set to high. This pin is activated through the use of an external button. Serial communication has a heavy impact on the performance of the transmitter, so the diagnostic messages are turned off by default.

The following flowchart describes the program flow of the transmitter:

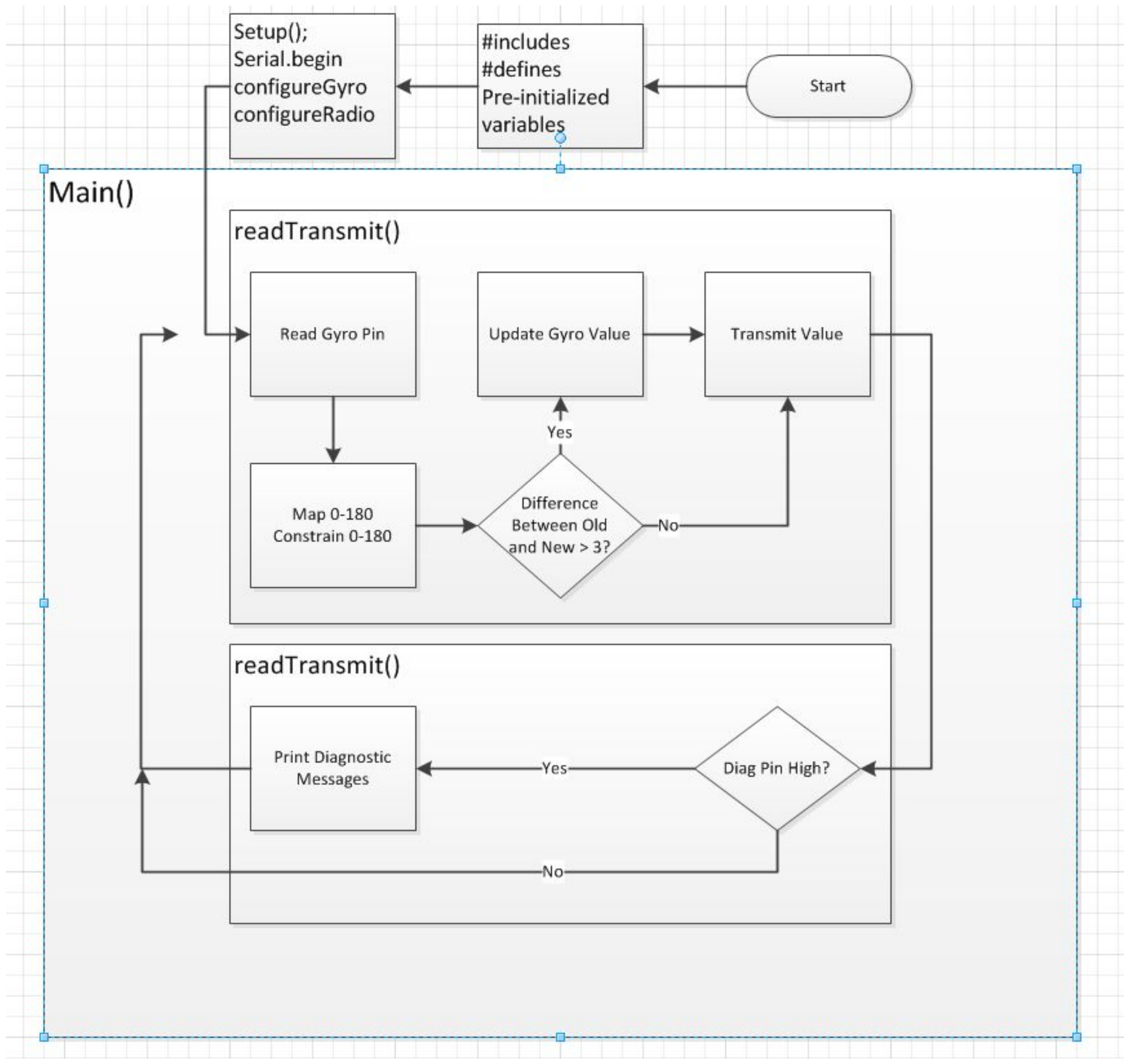


Figure 4: Transmitter Program Flowchart

The housing for the transmitter was modeled using 123D Design and printed using a Robo R1+. Estimating the size of the housing was not difficult because the custom PCB provided us with a very accurate idea of the total space that would be required for the assembly. Additionally, 9v batteries are a standard size used in musical peripherals. We only needed to print one housing for the transmitter because our measurements were accurate. The following images depict the model of the 3d housing and completed transmitter prototype:

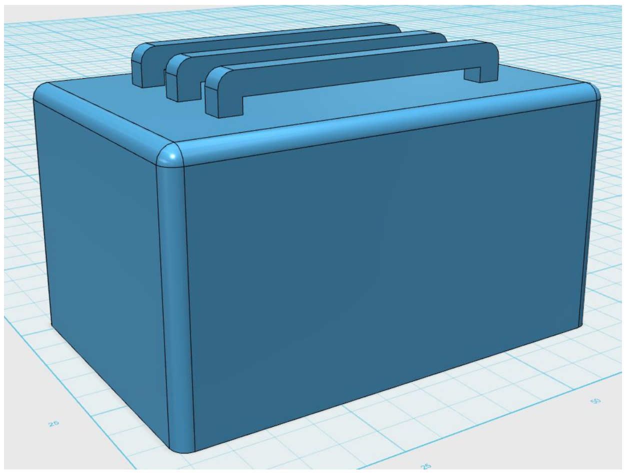


Figure 5: 3d Model for Transmitter Housing

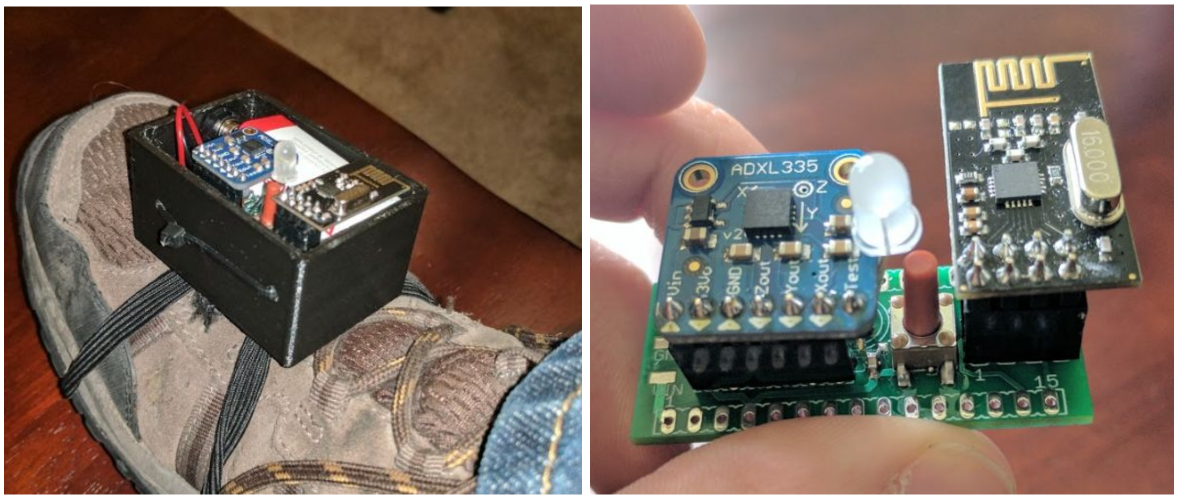


Figure 6: Completed Transmitter

Receiver Design

Hardware:

The receiver design was more complex than the transmitter both in hardware and software design. The devices attached to the 9v battery required a great deal of current to operate properly, so special consideration was given to power management as described in the Problems Encountered/Solved Section, Appendix E.

Ultimately, the Arduino Uno powered the screen and transmitter, while a separate 5v regulation circuit powered the servo. The various connections to the Arduino and regulator circuit can be seen in the following image:

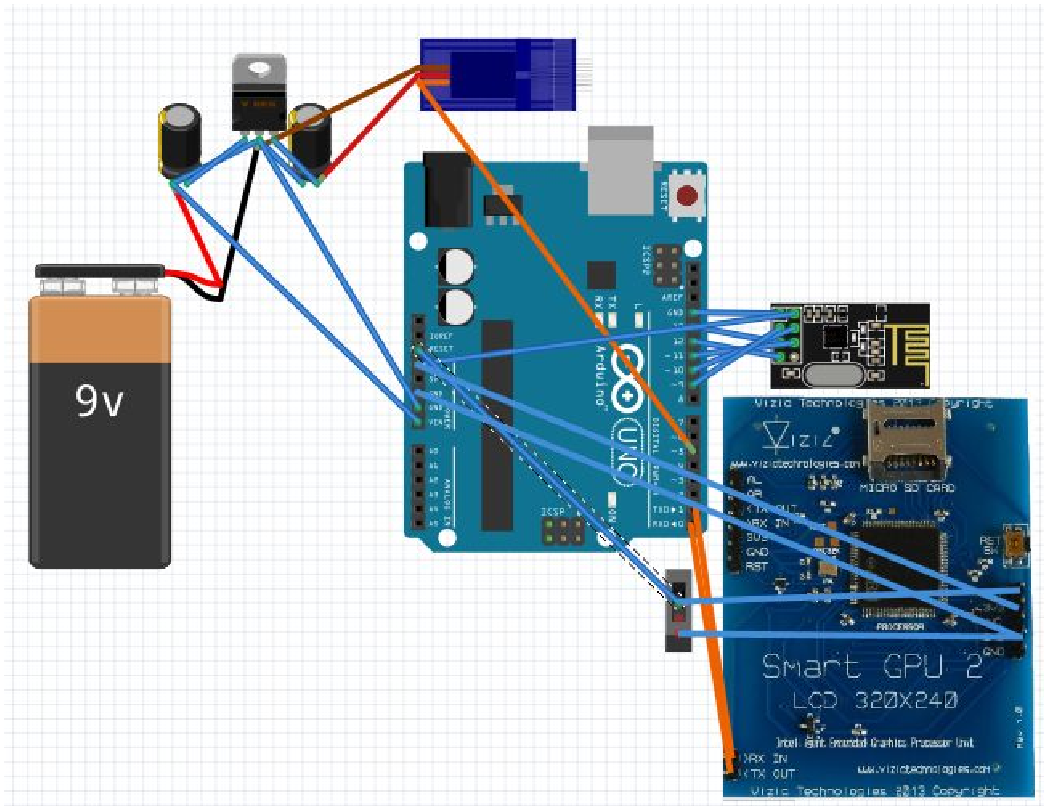


Figure 7: Receiver Schematic

Software:

The receiver involved a more-complicated software design simply because it incorporates an LCD screen. The SmartGPU2 TFT screen includes a dedicated graphics processor which handles screen drawing with direction from the arduino via the TX/RX serial lines. Each iteration of the main loop includes several serial messages for the GPU, so the receiver software needed to be tested under actual operating conditions to be sure that the device would operate as intended. This communication and power draw is taxing on the arduino, so the default touchscreen menu was designed to operate with as few serial messages as possible.

The main loop checks for radio transmissions from the transmitter and adjusts the servo position if a new value is available. If there is no message, which is somewhat rare, the receiver increases counts for dropped messages and adjusts diagnostic measurements so that they can be displayed on the screen.

The receiver then looks for touches on the TFT screen and gets the coordinates of user interaction. Based on the coordinates of the screen, the receiver determines which menu is active, and then redraws screen elements if a new menu is selected. The options menu includes settings that can be changed to affect the operation of the device. The locations for these settings all have x-coordinates less than 270, so the software will adjust an option in the event that a screen touch is within that region. The options themselves are discussed next in the GUI section. The following flowchart depicts the normal operation of the receiver unit:

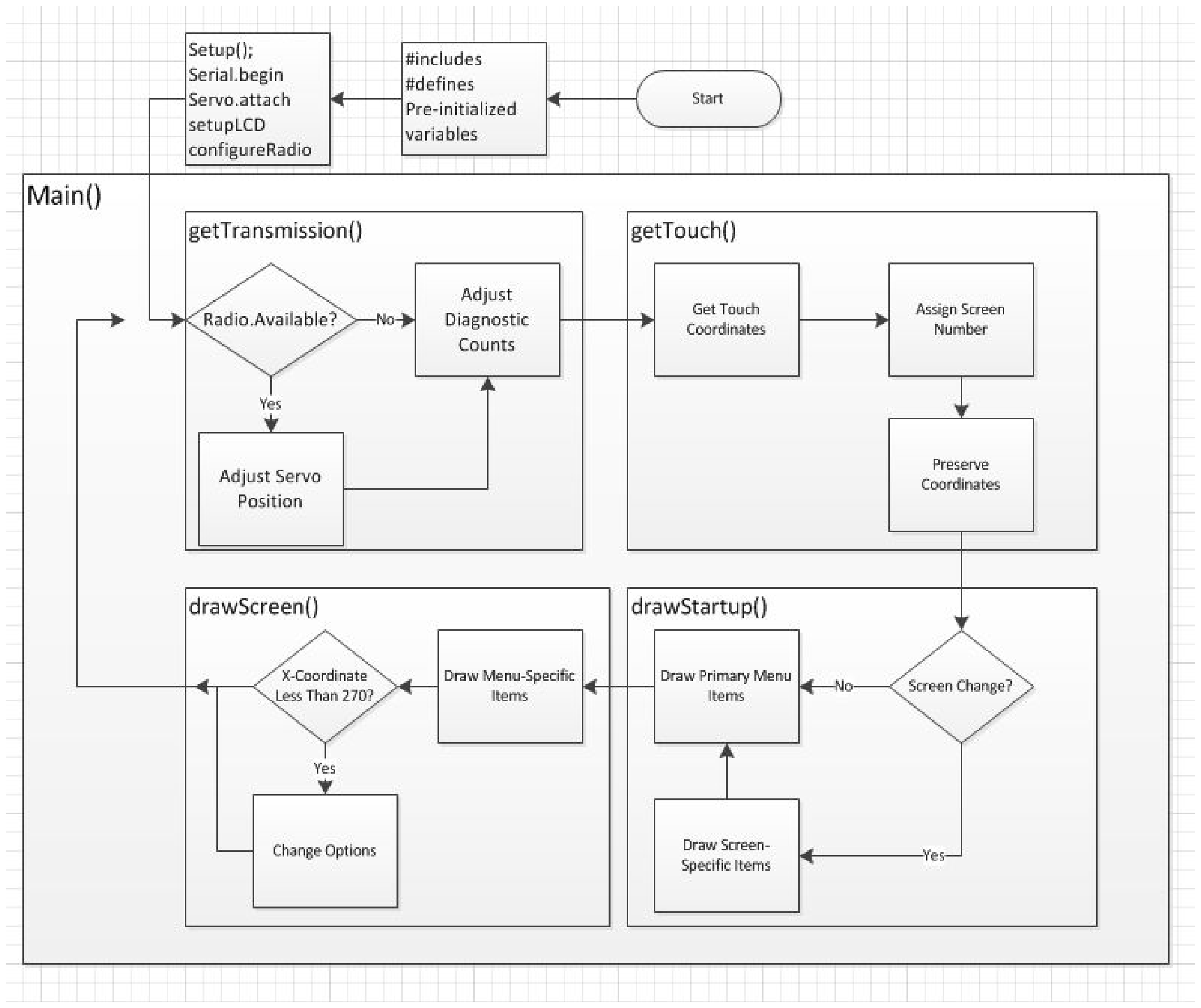


Figure 8: Receiver Program Flowchart

The graphical user interface was designed to be simple, but informative and visually appealing. It is divided into two main areas: the main interface and the menu interface. The menu interface is always the same and is divided into four icons:

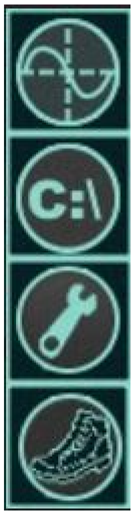


Figure 9: GUI Menu Icons

The top icon for the graphing window. This window plots the last 135 values (270 pixels / 2) received from the transmitter unit. This view is helpful to diagnose problems with the gyroscope. If the gyro is not correctly calibrated, or it is generating a lot of noise, the problem will be clearly visible in the graphing window.

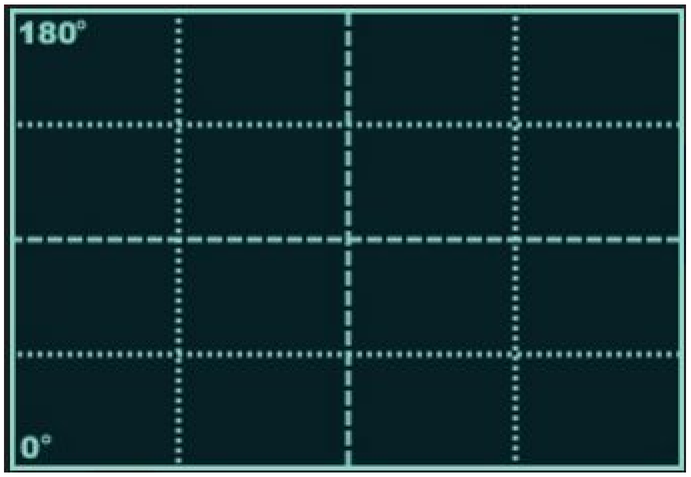


Figure 10: X-Y Plotting Area

The next icon is for the console, which prints several diagnostic messages when active. It prints the last value received from the transmitter, the number of consecutive dropped packets, the packet error rate, and a diagnostic trouble code (if any). The number of consecutive dropped packets, or drop count, resets to zero when a new message is received. This metric gives the user a good idea when something is wrong with the transmitter/receiver when it begins to count to a high number. The packet error rate, or PER, is an average of the number of successful transmissions received. It is calculated using an array of boolean values for the last 200 expected transmissions, in which a 1 is recorded on successful transmission or 0 if unsuccessful. With each transmission, the software adds the newest value to a cumulative sum and subtracts the oldest received value, and then replaces the oldest value in the array with the new value. This method is much faster than using a loop to count all values in the array during each main( ) iteration, which added a noticeable lag to the device response. The transmission rate is roughly 50 transmissions per second, so this array fills and must begin writing from the beginning every four seconds. This metric was very helpful in determining the effective range of the device, as described in the Results/Analysis section.

The next icon is for options. The four options available in this menu are hi-low swap, brightness, dead zone, and activation time:

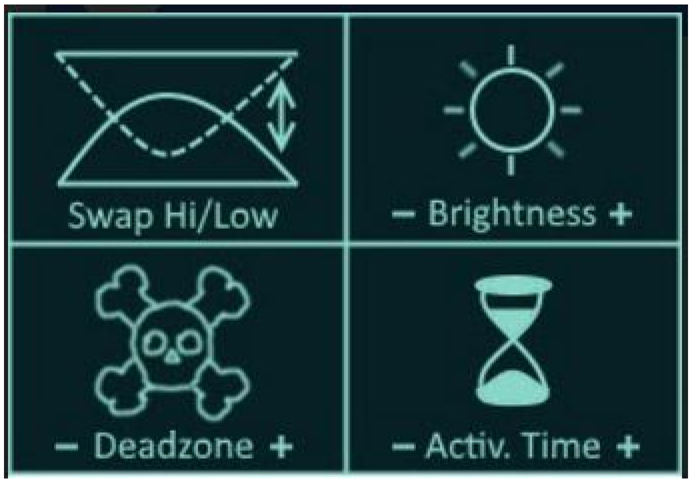


Figure 11: GUI Options Screen

The hi-low swap feature is available to provide guitar players the operation to which they are accustomed. Most pedals operate with the low position being most active (wah effect at 100%) and the high position being inactive. However, the nature of the gyro is to produce the opposite result. The high-low swap simply reverses this by using the following formula to adjust the output servo value: output = abs(input - 180). The brightness option simply increases or decreases the brightness of the screen. The dead zone option increases or decreases the threshold inclination that must be reached before the servo activates. The user does not want the device to activate every time the foot is moved, so the receiver reduces any received value below the threshold (default 20 degrees) to zero. This prevents unstable operation resulting from slight movements. Finally, the activation-time option enables the user to walk normally or keep time via foot tapping without activating the device. The device keeps a running average of the foot angle by accessing the array used to store the last 135 screen values. The average is calculated using a method similar to that of the PER - by maintaining a cumulative sum and replacing the oldest value with the latest received value at each iteration through the loop. The 135 values represent roughly 2.5 seconds of data. The device remains in an inactive state until the running average reaches at least 70% of the maximum gyro output value. The device then manipulates the servo as directed by the last received value, and remains in an active state until the running average drops below 10% (which happens quickly due to dead-zone reduction).

The last icon activates the normal operation window. This window was designed to use the least amount of communication between the Arduino and the touchscreen. This window simply displays an image of a foot at various levels of inclination as the user raises the gyro. The receiver is designed to update this window only when a threshold level has been crossed and a new image should be displayed. There are three levels: flat to 40 degrees, 40 degrees to 100 degrees, and greater than 100 degrees.

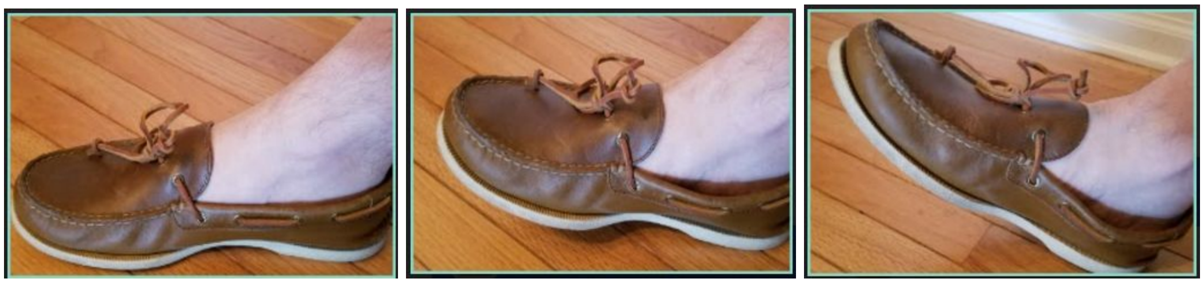


Figure 12: Screen Display for Changing Foot Position
(From left to right; down, intermediate, up)

Modeling a housing for the receiver proved to be more difficult than for the transmitter. Without a custom PCB, which was impractical before power regulation was required, wire management was a big problem. The housing includes a mount for the servo and potentiometer which can be seen in the center of the image below:

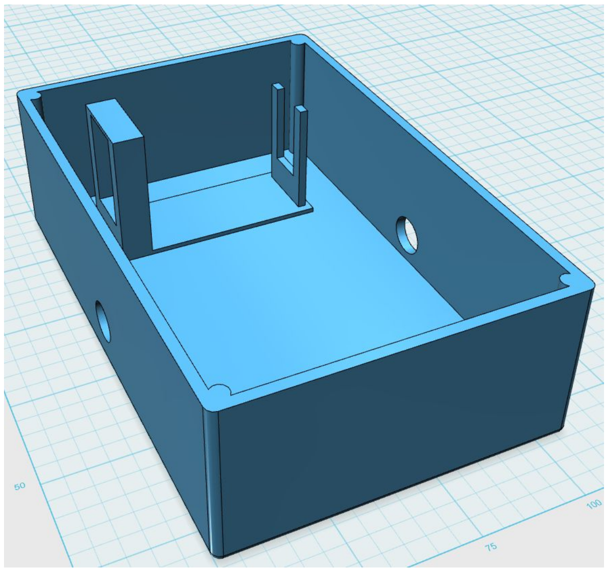


Figure 13: 3d Model for Receiver Housing

However, fitting the Arduino, the touchscreen, a 9v battery, two switches, and the wah board required creative placement in the lower space of the housing. Additionally, the wires moving between components required some creative placement. Often, the wires were pulled out when attempting to attach the lid of the box to the rest of the assembly. The lid was printed a total of three times, being taller each time to accommodate an increasing number of components and wires. The completed receiver can be seen in the image below:

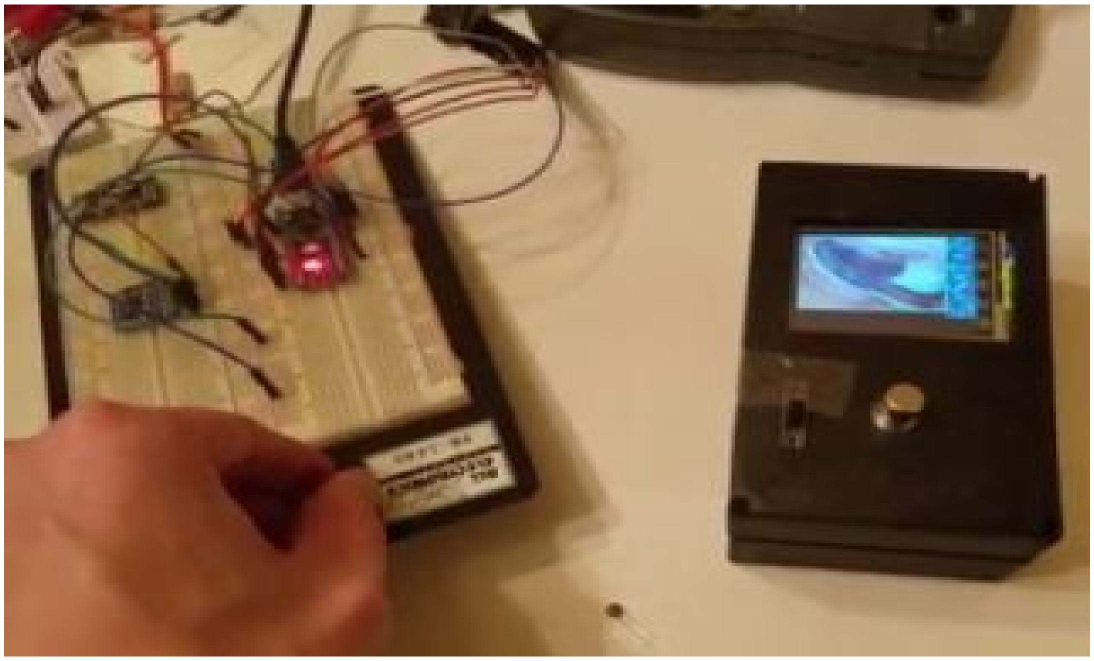


Figure 14: Completed Receiver

Filter Operation

A wah pedal operates as a variable band-pass filter. The following process diagram describes the relationship between the servo position of our receiver and the response of the filter:

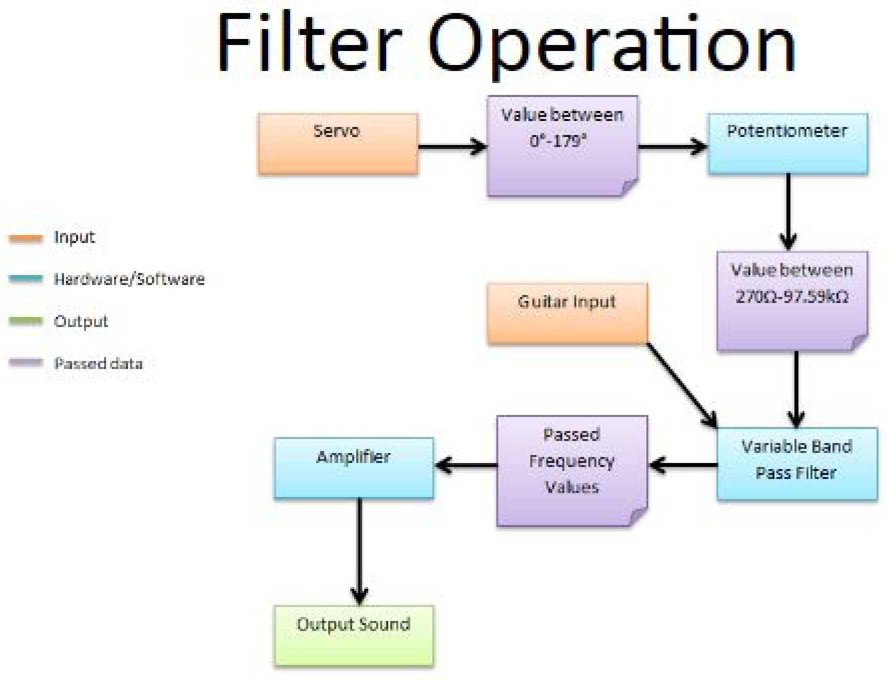


Figure 15: Wah Filter Operation

When the receiver receives a new value, the servo turns the potentiometer, which increases or decreases logarithmically between 270 and 98,000 ohms. This resistance, when tied into the wah circuit board, translates the bell-shaped frequency response of the filter along the x-axis. Thus, when we adjust the value of the potentiometer we are able to translate the frequency response of the filter along the x-axis, as shown in figure X below. This is how the classic “wah-wah” sound is achieved.

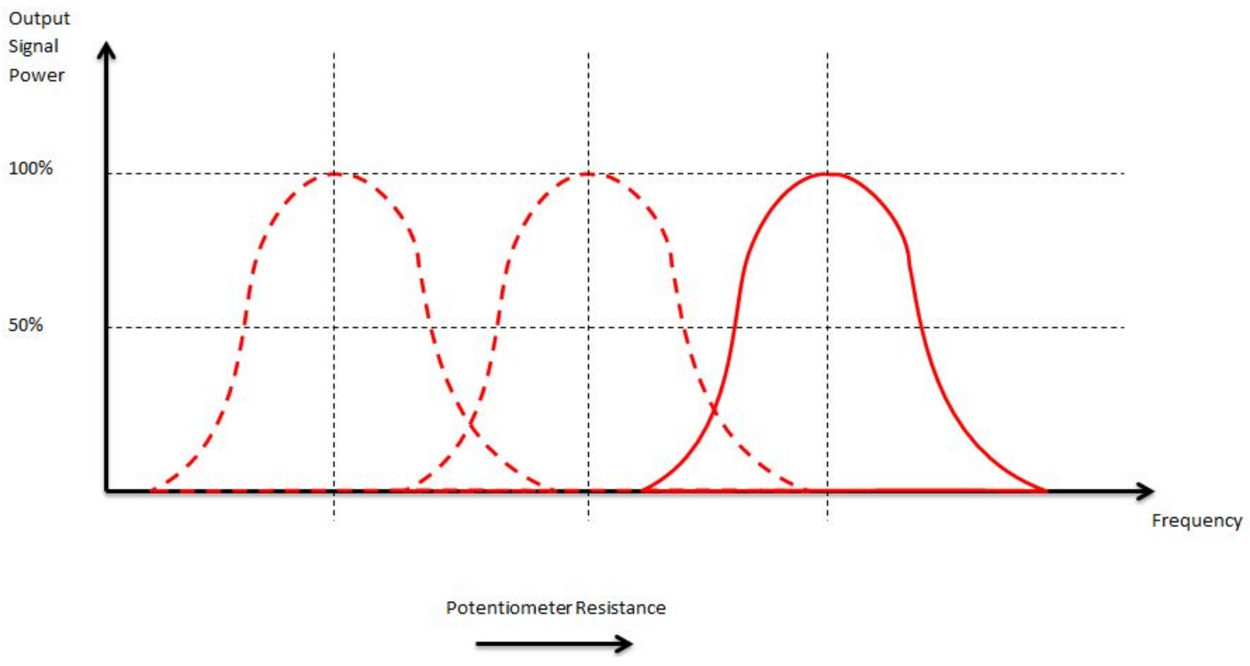


Figure 16: Wah Filter Response vs. Potentiometer Resistance

The design of the wah filter has not changed much since the 1960s and 1970s, so many filters are no longer covered by existing patents, and are now considered public domain. The following is a classic implementation of the Cry Baby effects pedal, built by Dunlop in the mid 1990s. Dunlop incorporated the original ‘60s Vox filter design, and used the Cry Baby name because Vox failed to renew its trademark. Because the design is public domain, there are many off-brand boards available for purchase on the internet. Ours cost $20.

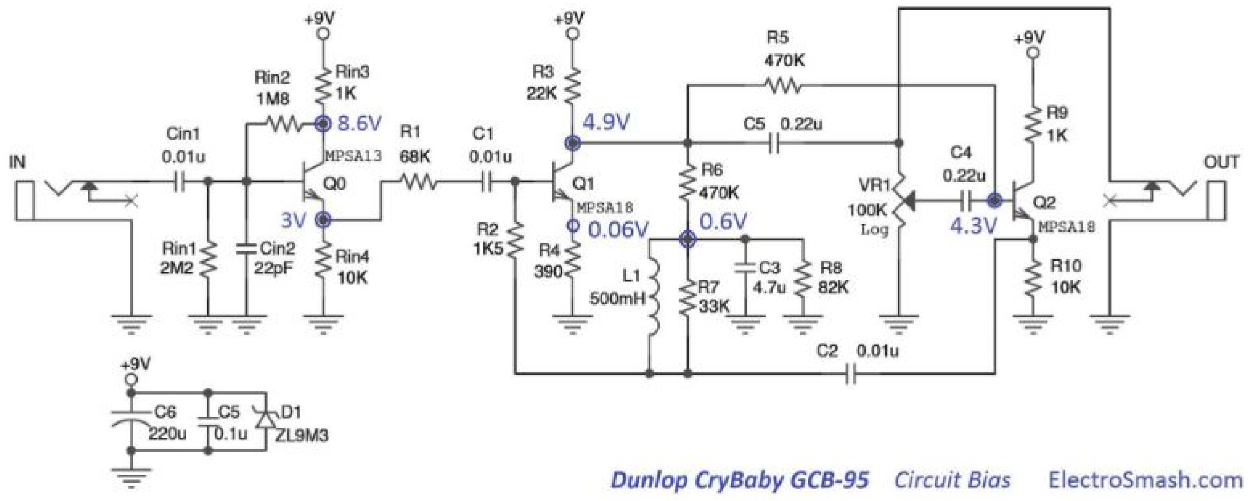


Figure 17: Wah Filter Schematic Diagram

Operating Procedure

The transmitter must first be attached to the user's foot. The 3d-printed housing includes two through-hole extensions for attaching elastic bands. The bands are sewn in loops so that the user can slide them over shoes. The shoe goes through the larger loop first, then the smaller one. When the transmitter is powered on, the gyroscope operates as a function of the user's foot position. It must be calibrated first, however. When the device is powered on, the LED turns yellow for two seconds, then turns red when calibrating the down position. It then turns yellow so the user can tip the foot up. After another two seconds the LED turns red while the device calibrates the up position. When calibration is complete, the LED turns green indicating that it is ready for normal use.

The user is now free to move about, and the device will remain inactive until the user raises the foot for 2.5 seconds, at which time the device will activate and begin moving the servo normally. While in active mode, the transmitter will report all changes in foot angle, so the user should remain stationary during this time. When the user places the foot in a flat position for longer than 2.5 seconds, the device will revert to an inactive state, and the user can again walk freely.

The touch screen of the receiver can be operated at any time, regardless of the state of the transmitter. The screen operates with the touch of any object with sufficient pressure. The point of contact requires more pressure per unit area than most other touchscreen devices, so it is typically better to use a fingernail rather than the tip of the finger. The four menu icons at the right activate the UI elements. The screens are very intuitive and operate with the functionality described in the the GUI interface portion of the design section of this document.

Analysis, Results, Discussion

The overall transmitter/receiver setup works exactly as we first envisioned. The response time is low, around 150 ms, which is more than acceptable for this particular pedal. Additionally, the transmitter's effective range covers the length of any professional concert stage. A simple test was performed by increasing the distance between a transmitter and the receiver unit and recording the average Packet Error Rate at each 50-foot increment. The test was performed at three starting locations; A,B, and C; and measured to a final distance of 300 feet. The following chart describes the Packet Error Rate as a function of distance:

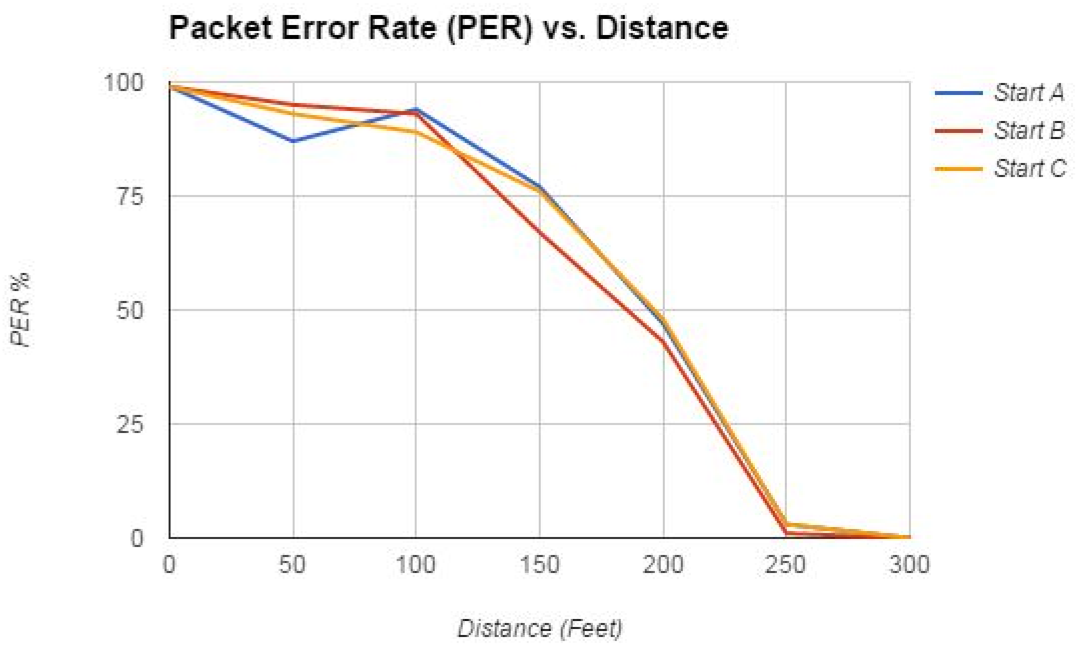


Figure 18: Packet Error Rate as a Function of Distance

The various starting locations for the test (approximate) are included on the following map:

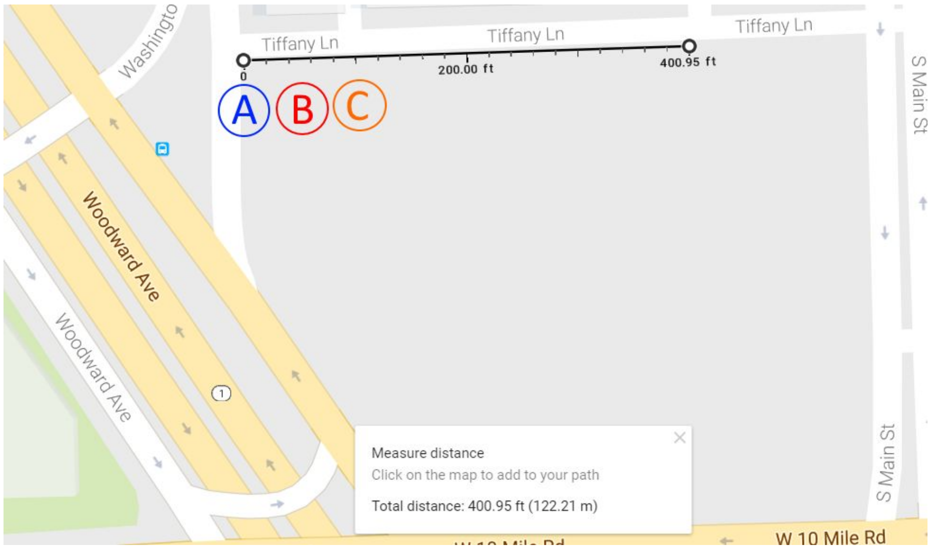


Figure 19: Starting Locations for Trials A,B, and C

At a packet error rate of 60%, the receiver unit begins to operate in a 'jittery' fashion. Thus, we have determined that the user should remain within 150 feet of the receiver to maintain normal operation.

The touchscreen does encounter problems during normal operation. After approximately 20 minutes of use, it is required to be reset along with the Arduino controller. However, we have noticed a sharp increase of the stability of the unit since it was removed from its breadboard environment and placed into the final prototype. Its unstable performance was perhaps the cause of weakly attached wires. The screen is also somewhat old, and the glue is beginning to separate the screen from the controller board, which could have some negative impact on its operation. A custom-programmed and assembled screen would likely eliminate these problems.

The rest of the transmitter and receiver components work as expected with no problems experienced so far. Further Research and development is needed, as described in the future improvements section of the appendix, but the early prototype is solid and very satisfying to use.

Conclusion

The final product we have created was both a substantial learning experience and an informative exercise in consumer product prototyping. This included things like the Design for Manufacturing (DFM) process, learning how to use 3D modeling software, and learning how to create custom PCB solutions. The latter is the most important skill for us to use as developing electronics engineers since all commercial circuit work will be put onto PCBs when finalized. Our goal at the outset was to create a custom consumer product for which there exists a need in the current market. The receiver/transmitter combination works well - in some ways better than expected - and we are very proud of what we have developed as a team. With the changes/improvements suggested in the alternate designs section, appendix A, we believe that a viable device could be brought to market, and that a sufficient demand exists.

References

Our references included the following sites:

- https://www.adafruit.com/ was for dealing with the transceivers and the gyroscope.
- http://patft.uspto.gov/ was used to collect patent information regarding the wah pedal filter itself.
- http://www.electrosmash.com/crybaby-gcb-95 gave us a high level understanding of the filter we were using for our project.

Appendix

A: Alternate Designs

The easiest improvement to our design is to replace the servo/potentiometer with a digital, logarithmic potentiometer. The time required to turn the potentiometer and servo adds mechanical latency to the operation of the device. A digital potentiometer does not require any time to turn, so the overall system would be much faster if it incorporated one. However, the digital potentiometer must have a sufficiently high resolution of outputs, or the wah effect may have a choppy output. The servo/mechanical potentiometer combination is a truly analog system, so it has an infinite resolution.

The x-axis is not the only axis tied to the input pins of the arduino. The y and z axes can also be used to calibrate the sensor, which means that the transmitter can be attached to the side of the foot. Adding a software option to implement this feature would be fairly easy, and could be preferable. We have received input from several sources that the transmitter looks bulky or strange on the top of the foot, and that it may be less obvious on the side of the foot or positioned at the ankle.

Security is needed to prevent unauthorized persons from interfering with the normal operation of the equipment during a stage show. In its current state the receiver will act on any transmission on its assigned channel. While choosing a particular channel may prevent most interference, any skilled programmer could develop software to locate the channel and disrupt its operation. End-to-end encryption between the transmitter/receiver is needed to fully protect the device.

Finally, the system should not be used to operate a single effects pedal. Ideally, several pedals should be attached to a common node used to select which devices are active at a particular time. The pedals do not need to send any data to the receiver, so a single line could be easily used to select any combination of 16 device effects to activate at any time. These configurations could be preprogrammed by the musician ahead of time and accessed via an attachment on the guitar itself. In essence, our design should enhance the operation of all pedals, rather than just a wah pedal.

B: Regulatory Constraints

The Federal Communications Commission (FCC) has limited the 2.4 GHz spectrum to the “amatuer spectrum” for things like RC cars, home automation products, and even Wi-fi. Since our transceivers operate on this spectrum, this is the only regulation to which we must adhere.

C: Parts List/Cost Analysis

The project required the following materials:

Transmitter	
Arduino Nano	$5
9v Battery Snap	$0.30
RGB LED	$0.40
NRF22L01 Transceiver	$5
ADXL335 Gyroscope	$15
Custom PCB	$0.10 (In bulk, multiple per board)
Molded Housing	~$.50

Total: $21

Receiver	
Arduino Uno	$5
9v Battery Snap	$0.30
Pedal Pushbutton	$2
Two-Position Switch	$0.30
NRF22L01 Transceiver	$5
SmartGPU2 Touchscreen	$100
4Gb SD Card	$2.50
SG90 Servo	$1
5v Regulator	$0.50
2x 100 Microfarad Capacitor	$0.10
Crybaby Wah Board	$20
Molded Housing	~$0.50

Total: $137

The cost of the transmitter can be reduced slightly by developing a dedicated board for a microcontroller and gyroscope, which would cut the cost of those components by at least 50%. This step is necessary to reduce the size of the unit and make it less conspicuous on the foot. The transmitter should remain separate from the rest of the construction and be attached to the frame of the unit. Buying components in bulk, we estimate that the unit could be constructed for as little as $10 per unit.

The receiver requires far more research and development, as the touchscreen would need to be developed to lower its cost. The code for the dedicated graphics processor would likely be custom, to fit a cheap ARM microprocessor selected for the job. However, the custom display/controller would be able to handle all tracing and eliminate much of the wiring, so the final product could be much smaller than the prototype. After the unit is custom developed, and components purchased in bulk, we estimate the true cost of a production model to be less than $35.

E: Problems Encountered/Solved

We encountered many problems while completing the project. The transmitters we initially purchased required software that conflicted with the internal Arduino timer. This created serious compatibility issues when combined with the servo, whose PWM functions required the use of the same timer. The primary issue with the old transmitters was that they did not include an internal register or dedicated controller for signal transmission. The arduino was required to use its own timing and transmission protocol to accomplish the task. The new transmitters are synchronous peripheral interface (SPI) devices, which keep pace with the arduino clock. Thus, when the arduino needs to transmit a message, it simply sends the message to the transmitter via serial and the transmitter handles the rest.

Additionally, we needed to replace the receiver Arduino Nano with an Uno for increased processing power. At a later stage we decided to add the touch screen for added user functionality, but the burden of communicating with the screen was too much for the Nano to handle.

The touch screen created its own problems, as well. Because it uses the Tx/Rx pins for serial communication between the controller and the screen, the screen needed to be disconnected each time the Uno was reflashed, because it interfered with the updating process. Drawing the screen also proved to be more complicated than expected. The screen cannot simply be drawn - the designer must consider what existed on the screen beforehand. For example, if an image is drawn, or if several lines of text are displayed, drawing new text in that area will overwrite what existed before, so the text could easily be superimposed on top of the old image. The old image must be cleared by drawing a black box over the existing material, or by drawing the intended back image, before drawing new objects. The serial communication between the screen and Uno is taxing on the processor, so the fewer commands the better. This required a great amount of pre planning and resource management. The screen also required the most power of all devices used in the project. An Uno can power several transmitters and servos by itself, but the addition of the screen created too much of a burden. The Uno was used to power the screen and the transceiver, because it provided a reliable, regulated supply. The servo, however, requires a large amount of power when moving and very little while idle, and its drain frequently reset the Uno during normal use. So, a 5 volt regulator and two 100 microfarad capacitors were used to supply power to the servo in parallel with the Uno. This eliminated the power issue and enabled the system to be run on a single 9 volt battery.

F: Distribution of Work

Work was distributed among group members on an as-available basis. Often, due to exams or projects, at least one member of the group was unavailable to assist in a particular task. However, the workload was split as evenly as possible.

Mike Fornoff, as a control systems engineer, has the greatest amount of general hardware knowledge. Mike created the housings for the transmitter and receiver, managed the wires of both devices, and created the PCB for the transmitter unit. Jeff Berg had the greatest amount of programming experience, so he helped to make the code for the transmitter and receiver as efficient as possible. All members coded at least some of the project software, but that code often needed to be trimmed down or limited to less than half of its processor usage. Jeff handled those tasks. Nick Patten helped to create the first working transmitter/receiver prototypes, he compiled necessary information from each component's datasheet, and ultimately solved the power usage problems of the receiver.

It is not often that all members of a group carry their weight, but this particular group did not have any weak points.

G: Program Listing

Transmitter Code:

```
// Include Necessary Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#define CE   9
#define CSN 10
#define GYROPIN A4
#define DEBUGPIN 7
#define NUMREADS 15

// Define Global Variables and Library Objects
RF24 radio(CE, CSN);
int newGyro;
int gyro;
int average, high, low;
byte dCount=0;
bool RDinit = false;

void setup()
{
  Serial.begin(9600); // Begin serial communication
  pinMode(DEBUGPIN, INPUT); // Define debug pin
  calibrate(); // Calibrate gyro
  low = 510; // Set low and high gyro values for quick testing
  high = 555;
  radio.begin(); // Start Radio comm
  radio.openWritingPipe(0xF0F0F0F0E1LL); // Define transmitter receive channel
  radio.openReadingPipe(1,0xF0F0F0F0D2LL); // Define transmitter send channel
  radio.startListening(); //

}

void loop()
{
  readTransmit();
  RDinit = true;
  //dChat();
  if(debug()){
    Serial.print("Transmitting: ");
    Serial.println(gyro);
  }

}
////////////////////////////////////////////////////////////
void readTransmit(){
  newGyro = analogRead(GYROPIN);  // read gyro pin
  //Serial.print(newGyro); Serial.print(" "); // uncommented for debugging
  newGyro = map(newGyro,low,high,0,180); // map the 0-1023 analog value to servo
value
  newGyro = constrain(newGyro,0,179); // Constrain to eliminate values outside range
  //Serial.println(newGyro);
  if(abs(gyro-newGyro) >= 3){gyro=newGyro;}; // only send new value if difference >= 3
  radio.write(&gyro, sizeof(gyro)); // send gyro value
};
////////////////////////////////////////////////////////////
bool debug(){
  return digitalRead(DEBUGPIN); // check to see if debug is on
};
////////////////////////////////////////////////////////////
void calibrate(){
  Serial.println("Put Flat"); // instruct user to put flat
  low = getAverage()+5; // get average of 15 readings and add 5 (experimentally
determined)
  Serial.println(low);
  Serial.println("Raise Up");  //  instruct to raise
  high = getAverage()-5; // do 15 readings high and subtract 5
  Serial.println(high);
};
////////////////////////////////////////////////////////////
int getAverage(){
  for(int i=0; i<3; i++){Serial.print(i); Serial.println("...."); delay(1000);};
  Serial.println("Calibrating"); // let user know not to move
  average = 0;
  for(int i=0; i<NUMREADS; i++){ // get 15 readings over 1.5 seconds
    delay(100);
    average += analogRead(GYROPIN);
  };
  average = round(average/(NUMREADS)); // determine average
  return average;
};
////////////////////////////////////////////////////////////
```

```
int dChat(){ // initiate diagnostic chat with receiver
  bool respond = false;
  int transmit = 0;
  byte received[1];
  char text[10];
  if(RDinit == true){
      while(!respond){
        radio.write(&transmit, sizeof(transmit));
        transmit++;
        if(transmit == 3){
          transmit = 0;
          radio.startListening();
          unsigned long count = millis();
          while(millis() - count < 200){
            if ( radio.available() )
            {
              bool done = false;
              while (!done)
              {
                done = radio.read( received, sizeof(received) );
                //if(received[0] == 200){respond = true;};
              };
             Serial.println(received[0]);
            };
          };
          radio.stopListening();
        };
      };
  };
}
```

Receiver Code:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Servo.h>
#include <SMARTGPU2.h>     //include the SMARTGPU2 library!
#include <math.h>
#define CE_PIN   9
#define CSN_PIN 10
#define SERVPIN 5
#define DEBUGPIN 7
```

```
#define ICONSIZE 50
#define WAVE 1
#define CONSOLE 2
#define SETTINGS 3
#define FOOT 4
#define RSIZE 269
#define TEXTSPACE 30
#define DOWN 1
#define HALF 2
#define UP 3

// Define array of GUI elements
char images[12][30]={"grid","wave","console","settings","foot"}; //array containing the
names of the different called images
byte rData[RSIZE];
byte lastIndex;
byte oldScreen;
int dCount=0;
byte lastReceived = 0;
bool RDinit = false;
int numSent=0;
byte footPos;
unsigned long totalDropped = 0;
unsigned long totalReceived = 0;
Servo servo;
SMARTGPU2 lcd;
POINT point;

RF24 radio(CE_PIN, CSN_PIN);

byte received[1];
int servPos;
int dropCount = 0;
byte screen = FOOT;
byte genericCounter = 0;

void setup()
{
  Serial.begin(9600); // begin serial,
  servo.attach(SERVPIN); // set servo at servo pin
  pinMode(DEBUGPIN, INPUT); // define debug pin as input
  setupLCD(); // begin lcd comm
  radio.begin(); // begin radio comm
  radio.openWritingPipe(0xF0F0F0F0D2LL); // define receiver writing channel
  radio.openReadingPipe(1,0xF0F0F0F0E1LL); // define receiver reading channel
  radio.startListening(); // start receiving transmissions
  lcd.setTextSize(FONT2); // set default size and text for LCD
  lcd.setTextColour(WHITE);
  delay(1000); // Delay 1 second for stability
}

void loop()
{
  getTransmission(); // look for transmission and do something with it
  getTouch(); // look for touches on screen

  drawStartup(); // draw common gui elements
  if(screen != oldScreen){screenChange();}; // if screen has changed, do transition
drawing
  drawScreen(); // draw iteration elements
}

/////////////////////////////////////////////////////////
void getTransmission(){
  if ( radio.available() ) // if new transmission available
  {
    bool done = false; // receive via SPI until transmitter sends done signal
    while (!done)
    {
      done = radio.read( received, sizeof(received) ); // chug spi out of bus
      if(received[0] != 0){servPos = received[0];}; // do not update for zero values (trans
error)
      if(received[0] < 22){servPos = 0;}; // set servo to zero if received value < 22 (default)

      if(received[0]-lastReceived == 1) {dCount++;} // keep track of diagnostic chat
sequence
      else {dCount = 0;};
      lastReceived = received[0]; // keep track of last received value
      if (dCount >= 2) {
        dCount = 0;
        dChat(); // initiate diag chat if 2 diag sequence received
      };
    }
    dropCount = 0; // reset drop count on successful trans
    totalReceived++;  // keep trag of diag variables
    servo.write(servPos);  // update servo
```

```
      rData[lastIndex] = abs(received[0]-180); // maintain array of received values
      lastIndex++; // update array index
      if(lastIndex == RSIZE){lastIndex = 0;}; // reset array if larger than max size
    }
    else{dropCount += 1; totalDropped++;} // increase consecutive drop count and total drop
count

  };
  ////////////////////////////////////////////////////////
  bool debug(){
    return digitalRead(DEBUGPIN); // check if diag on
  };
  ////////////////////////////////////////////////////////
  void setupLCD(){
    lcd.init();  // Initialize LCD, begin communication
    lcd.start();
    lcd.baudChange(BAUD6);
    lcd.SDFopenDir("UserGUI"); // set default subDir on SD card
    lcd.bright(50); // set default brightness to 50%
    lcd.setTextBackFill(FILLED); // set default draw operation to backfilled
  };
  /////////////////////////////////////////////////////////
  void drawStartup(){ // Draw standard LCD Menu Items
    lcd.imageBMPSD(270,0,images[1]);
    lcd.imageBMPSD(270,50,images[2]);
    lcd.imageBMPSD(270,100,images[3]);
    lcd.imageBMPSD(270,150,images[4]);
  };
  ////////////////////////////////////////////////////////
  void getTouch(){
    int i;
    byte j;
    byte submenu = 0;
    if(lcd.touchScreen(&point)){
      if (point.x>270){ // If x coordinate of touch > 270, determine menu item touched
          screen = 0;
          for(i=0; i<4; i++){
            if(point.y > i*ICONSIZE) {screen++;};
          };
        };
      };
  }
  ////////////////////////////////////////////////////////
```

```
void drawScreen(){
  if (screen == WAVE){
    if(lastIndex == 0){lcd.imageBMPSD(0,0,images[0]);}; // redraw grid if starting at zero
    lcd.drawCircle(lastIndex*1,abs(servPos-180),2,WHITE,FILL); // draw new grid point
  };
  if (screen != FOOT){lcd.string(0, 205, 259, 239, "Last Received: ", 0);};
  if (screen == CONSOLE){ // if console, draw diag messages
    genericCounter++;
    if(genericCounter >= 40){lcd.drawRectangle(160,0,200,184,BLACK,FILL);}
    float blah = 0.214;
    int PER = (float)(totalDropped)/(totalDropped+totalReceived)*100;
    lcd.printNumber(160,5,servPos);             // Last Received
    lcd.printNumber(160,TEXTSPACE+5,dropCount);   // Dropped
    lcd.printNumber(160,2*TEXTSPACE+5,PER); // Error %
    lcd.printNumber(160,3*TEXTSPACE+5,servPos); // DTC
  };
  if (screen == FOOT){// draw foot position based on servo position
     static byte oldFoot;
     if(servPos < 180){footPos = UP;};
     if(servPos < 60){footPos = HALF;};
     if(servPos < 30){footPos = DOWN;};
     if(oldFoot != footPos){ // only draw if crossing threshold
        oldFoot = footPos;
        switch (footPos){
          case DOWN:
            lcd.imageBMPSD(0,0,"FootDown");
            break;
          case HALF:
            lcd.imageBMPSD(0,0,"FootHalf");
            break;
          case UP:
            lcd.imageBMPSD(0,0,"FootUp");
            break;
        };
     };
  };
  if (screen == SETTINGS){// determine which option pushed if settings active
    byte selection = 0;// default selection = 0
    if(point.x <= 269){
      selection++;
      if(point.x >= 135){selection++;};
      if(point.y >= 93){selection += 2;};
    };
    lcd.imageBMPSD(0,0,"Options");
    lcd.printNumber(160,205,selection); // print option number selected (remove later)
  }
};
/////////////////////////////////////////////
void screenChange(){ // if screen has changed
  oldScreen = screen;
  if (screen == WAVE){
    lcd.imageBMPSD(0,0,images[0]); // if new screen is wave, draw grid
    //for(int i=0; i<RSIZE;i++){lcd.drawCircle(rData[i]*2,i,2,WHITE,FILL);};
  }
  else{lcd.drawRectangle(0,0,269,184,BLACK,FILL);};

  if(screen == FOOT){
    lcd.drawRectangle(0,186,269,239,BLACK,FILL);
    lcd.imageBMPSD(0,0,"FootDown");
  };
  if(screen == SETTINGS){ // place holder for settings menu
    lcd.string(5, 5, 160, TEXTSPACE, "Last Received: ", 0);
    lcd.string(5, TEXTSPACE+6, 250, 2*TEXTSPACE, "Dropped: ", 0);
    lcd.string(5, 2*TEXTSPACE+6, 250, 3*TEXTSPACE, "Error %: ", 0);
    lcd.string(5, 3*TEXTSPACE+6, 250, 4*TEXTSPACE, "Diag Code: ", 0);
    lcd.string(5, 4*TEXTSPACE+6, 250, 5*TEXTSPACE, "Text 5: ", 0);
  };
  if(screen == CONSOLE){
    lcd.string(5, 5, 160, TEXTSPACE, "Last Received: ", 0);
    lcd.string(5, TEXTSPACE+6, 250, 2*TEXTSPACE, "Dropped: ", 0);
    lcd.string(5, 2*TEXTSPACE+6, 250, 3*TEXTSPACE, "Error %: ", 0);
    lcd.string(5, 3*TEXTSPACE+6, 250, 4*TEXTSPACE, "Diag Code: ", 0);
    lcd.string(5, 4*TEXTSPACE+6, 250, 5*TEXTSPACE, "Text 5: ", 0);
  };
};
/////////////////////////////////////////////
void dChat(){ // initiate diagnostic chat
    int okay = 200;
    lcd.printNumber(160,205,numSent);
    radio.stopListening();
    radio.write(&okay, sizeof(okay));
    radio.startListening();
}
```

H: Component Data Sheets

Attached on following pages, or in separate folder for soft copy.