# Real-Time Free Throw Feedback Device
# Team 22
# Final Report

---


## Sanjay Kalidindi (kalidnd2)
## Mathew Kizhakkadathu (kizhakk2)
## Joseph Vande Vusse (vandevu2)

## TA: Hershel Rege (davereg2)


## May 2018

# Abstract

We intend to showcase our device as an attempt to aid individuals in performing fundamentally sound athletic practice without the aid of a coach, a camera, or other players to critique form. With our limited time, we chose to apply this theme to free throw shooting in basketball as it is a relatively simply motion that can be characterized well with only two sensors. These sensors were accelerometers positioned atop an individual's wrist to measure the degree to which the wrist was flicked and on the lower thigh to measure the degree to which the legs were bent. The maximum acceleration values over a five-second polling cycle, along with the corresponding label of make or miss, are passed to the microcontroller, a Bluetooth module, and finally a computer. This shot data is compiled in a text file. Then a machine learning clustering algorithm is applied to it to generate user feedback.

# Table of Contents

# 1 Introduction

Imagine you are an amateur basketball player who likes to play occasionally. You decide you want to improve your game, but you do not want to spend much money on private training sessions. You are not left with many options to receive feedback on your shots. Motions capture systems are readily available to give advice on your shots but cost upwards of $20,000. Wearable devices are available to professional athletes to improve their game but, unfortunately, are very costly also. Therefore, we have proposed a cheap, alternative solution for an amateur basketball player to receive feedback on his/her shot.

## 1.1 Objective

We came up with a wearable device under $100 to give feedback on a user's free throw. The wearable device consists of two accelerometers connected to a microcontroller. Then, a bluetooth module takes the data extracted from the the accelerometers and transfers it to a computer where we would run a support vector machine algorithm to analyze the acceleration values taken during many shots. After the shot is taken, the user can look at their computer and receive feedback on how to improve their next free throw. Essentially, we have enabled the user to receive feedback on their shot quickly without a coach or trainer being present based on previous successful shots.

## 1.2 Background

Numerous technologies currently exist on the market which sports enthusiasts use to improve their shots. The problem with these products is that they are targeted towards professionals which makes it very difficult, for example, a juvenile athlete trying to improve his/her game to make the high school team. One of the existing technologies on the market is the Wilson X. The Wilson X is a basketball that is designed with a chip that relays statistics back to a user when the user shoots [1]. The problem with this device is that is does not provide feedback which our product does. We believe that outputting feedback to a user is much more valuable than simply relaying statistics back to a user.

The Noah Basketball Shooting system is another analytical tool for basketball players to try and improve their game. Regardless of whether or not it works, it costs $6,000 [3]. This is not a realistic price tag for the majority of amateurs.

Many of our competitors who already have products on the market try to improve a user's shot by analyzing the shooter's arm angle and arc of the ball's path. We believe that the way we are analyzing a user with accelerations, we can provide unique and meaningful feedback to our user.

## 1.3 High-Level Requirements List

- Sensor circuits and microcontroller must be fastened to the user to be considered a wearable device
- System must be affordable for any amateur (≤ $100)
- Entire data transfer from analog sensor data to computer script performing high-level functions and display results must occur in ≤ 5 s to live up to "real-time" name

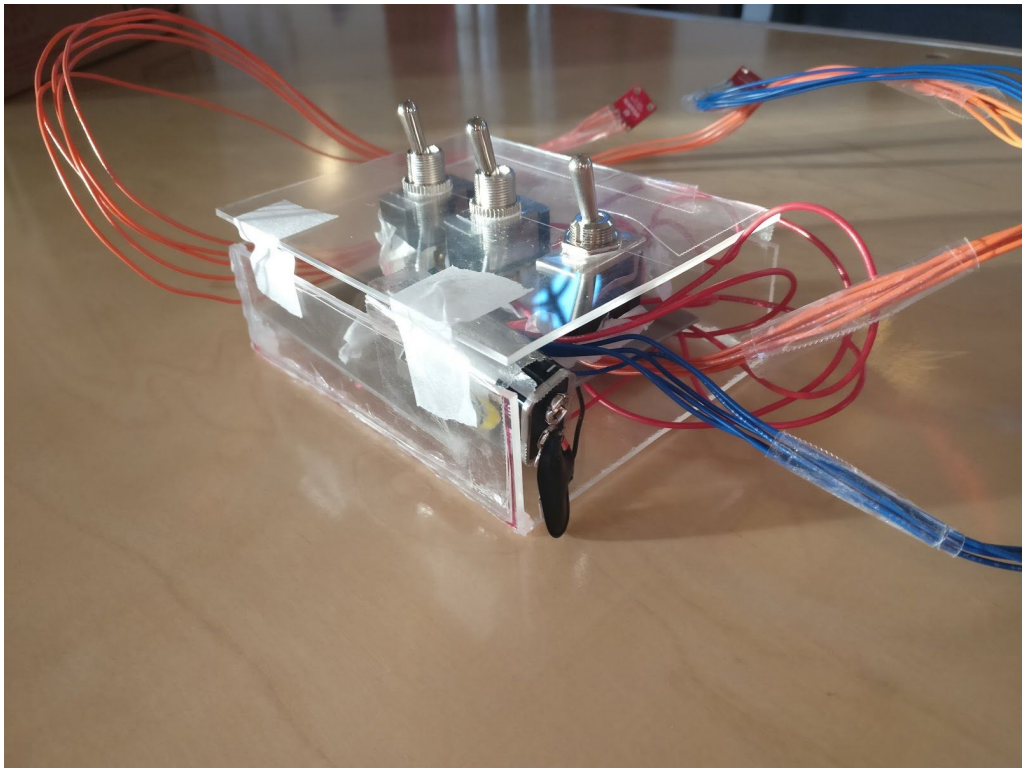


*Figure 1.3.1: Physical prototype*

# 2 Design

## 2.1 Block Diagram

The block diagram below gives an overview of the high-level details of our device. As we can see on the diagram, our project was split up into three main modules: the power module, the control unit module, and the sensing module. The power module consists of all the necessary components needed to power our entire project. The control unit consists of the microcontroller and illustrates the transfer of data to the Bluetooth where it is then relayed then to the computer. The sensing module illustrates how the sensor is powered with a voltage divider which steps down the voltage to 3.3V (a requirement for the accelerometers to function.

This block diagram for the most part is similar to the one we originally designed in our design document with some minor adjustments. The control unit and sensing module remained the exact same as before; however, some minor adjustments were made to the power module. Originally, we planned on using watch batteries to generate 6V of power and then use a buck converter to step down the voltage. Insead, we ended up using a 9V alkaline battery and a 5V regulator to supply power for the ATmega and the Bluetooth module [12]. The reason for this change was for the sake of simplicity and also to reduce the space it would take up on our PCB. Using two coin batteries in series would require the usage of two coin holders as well which would increase the complexity of the design, increasing the chances of issues arising [6]. As a result, a 9V alkaline battery was used. For similar reasons described above, we decided to switch out the buck converter for a 5V regulator. In addition to these changes, we also implemented a voltage divider in order to step down the voltage from approximately 4.6V to 3.3V which is what the accelerometers needed to function.

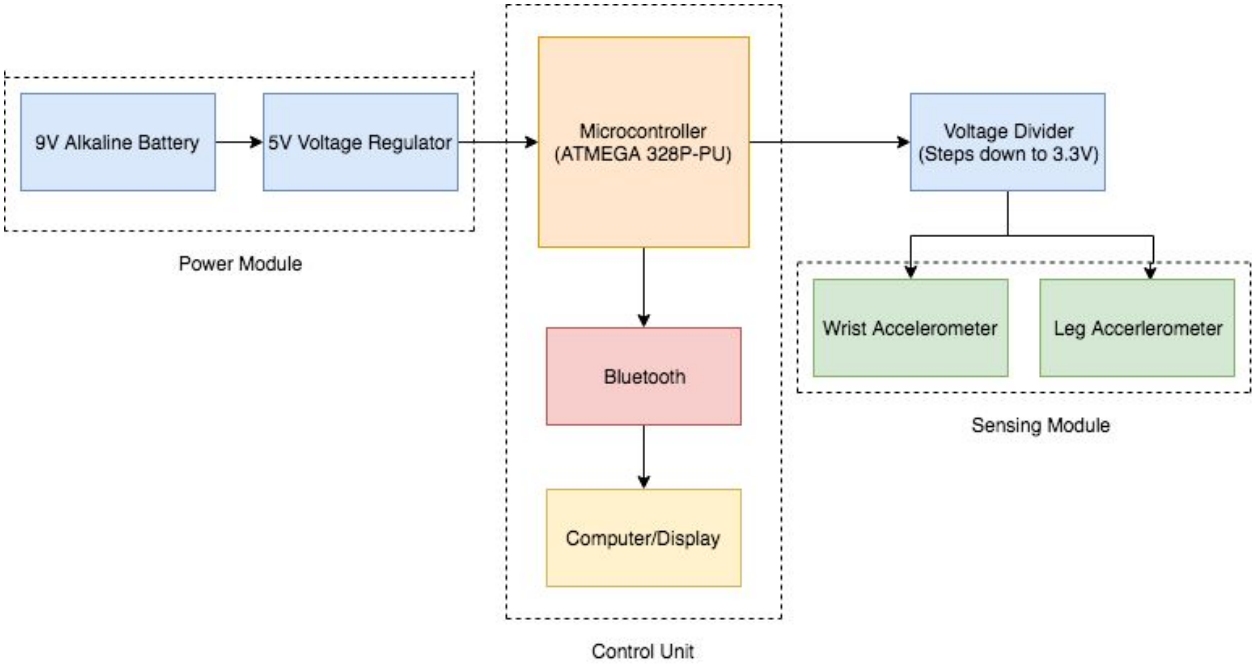



*Figure 2.1.1: Device block diagram*

## 2.2 Physical Diagram

In our original design, we had accelerometers placed on the wrist and calf areas. We assumed placing a sensor on the wrist would be the most accurate. The accelerometer on the calf would tell us how quickly the user was squatting with their lowering body. Then, we would be able to tell the user for their next shot if they needed to use more or less legs. The only change we implemented for the physical diagram is that, rather than placing an accelerometer on the individual's calf or shin, we placed it on his/her thigh instead. The reason is when gathering data we found that there was a greater range of motion on an individual's thigh compared to the calf/shin area. As a result, we were getting more variation in data when using the thigh.Therefore, we changed our final design accordingly. The final physical diagram for the project was also very similar to what we had envisioned in the design document. We decided to go forward and collect all our data with the accelerometer attached to the individual's thigh. The microcontroller and second accelerometer we kept attached to the wrist and waist as we envisioned previously.

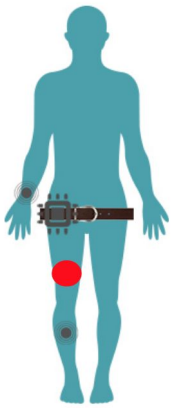


*Figure 2.2.1: Sensor/MCU locations on user*

## 2.3 Power Module

### 2.3.1 Battery

As mentioned in the block diagram above, we originally planned on using two 3V coin batteries in order to power our entire device. However, to simplify our design and conserve space, we decided to switch to a 9V alkaline battery. Conserving space on the PCB is important as we want to produce a product that is compact and easy for the user to attach to his/her body. Therefore, even changes that conserve something as small as several inches in total space can be significant.

### 2.3.2 DC/DC Buck Converter (Voltage Regulator/Divider)

A DC/DC buck converter is a converter which simply takes a voltage from a source and steps it down to a smaller value while at the same time increasing the current. This characteristic of the buck made it perfect for our design. However, similar to what we stated in section 2.3.1, we decided to change our design from a DC/DC buck converter to a voltage regulator and voltage divider to conserve as much space as possible. Implementing our design with a small voltage regulator and a simple voltage divider would save considerable space compared to implementing two DC/DC buck converters [12].

### 2.3.3 Bluetooth Module

In the design document of our project, we mentioned we would need 3.1V to power our Bluetooth module. On the datasheet, the minimum voltage required to power the device was 3.1V. However, this changed when we started designing our circuit. Rather than powering it with 3.1V, we decided to power it with 5V. The reason why we decided to power the Bluetooth module with 5V was when we first started to test our Bluetooth module with an Arduino, we fed it with the 5V pin on the Arduino. Therefore, we decided to keep consistent with the previous build and continue to use 5V as an input when we put together our PCB.

### 2.3.4 Sensing Module

Similar to the Bluetooth module, we provided a different amount of voltage to power the accelerometers for our project compared to what we initially mentioned in our design document. In the datasheet for the accelerometers, the necessary voltage to power the sensors were between 1.8V and 3.6V. Hence, we planned to use 1.8V to power the sensors. However, similar to the Bluetooth module, we tested our accelerometers with an Arduino. The Arduino powered the sensors with the 3.3V pin. To keep consistent, we decided to continue to use 3.3V to power the sensors.

## 2.4 Control Unit

### 2.4.1 Microcontroller

After some deliberation, we ended up using the ATmega328P-PU instead of our original ATmega328P-AN. The decision revolved primarily around requiring a through hole part as opposed to a surface mount one for our PCB. The ATmega328P-PU is the exact chip used on the Arduino Uno [9]. We assumed this would be a natural transition as our initial tests with the Arduino Uno were successful.

Once we gained some experience with Arduino programming, we were confident in our decision to abandon the requirement of storing 24 KB of data in flash memory. We instead chose to keep track of the maximum accelerations during the polling cycle and pass that data to the Bluetooth module directly. We were able to avoid this requirement with no loss of functionality.

Despite an initial misunderstanding to what data the accelerometers were displaying to us, we continued to use the maximum acceleration values as our data from a given poll. The data ended up being more of a measure of the amount of flexion in an individual's wrist or leg versus the acceleration of that body part. The degree of flexion is not unrelated to acceleration or force of a given body part. Hence, the data we gathered was just as useful as we originally intended.

### 2.4.2 Analog-to-Digital Converter

Our ATmega's built-in ADC was not something we ended up having to tweak to any substantial degree. Properly configuring the hardware and software around it was all that was necessary for its functionality. The hardware setup involved properly establishing a 16 MHz external clock signal via a crystal oscillator and protecting both its corresponding pins and the AVCC pin with capacitors. The software setup required beginning the Arduino code at the proper band rate corresponding to 16 MHz. After setup, the Arduino analogRead() function was all that was necessary to convert the accelerometers' analog signals to discrete integer values.

### 2.4.3 Bluetooth Module

The HC-06 Bluetooth module properly received integer data via the UART communication peripheral and transmitted it to the computer's receiver. It did so at a range exceeding its 5 ft requirement, but it did not always sustain the connection [10]. Approximately every fifteen free throw attempts, the module would become disconnected from the computer. It was still paired and would connect again with little effort, but it would be a nuisance if an individual was alone in the gym. It is something we would like to diagnose and correct if given more time.

### 2.4.4 Computer

The computer is responsible for performing all of the high-level functionality associated with generating user feedback. This software, in addition to the microcontroller programming, will be addressed in a later section.

The saved integer values are from the sensor on the user's lower thigh corresponding to the bend of his/her leg and one on the top of the wrist corresponding to the forward bend of that joint. A Windows application called PuTTy was necessary for displaying the data from the Bluetooth module in a terminal window. Enabling session logging and specifying an output text file in the aforementioned application creates a permanent

storage location with program-readable data. The machine learning algorithm that provides user feedback operates on said data and is explained later.

## 2.5 Sensing Module

### 2.5.1 ADXL335 Functionality

We originally stated we would verify the functionality of the accelerometers through a continuous bit stream. We figured a better way to prove functionality was not to look at the analog-to-digital converted values, but to look at voltage of the accelerometers. We placed voltage probes on the traces connected to the accelerometers and witnessed an increase in voltage while moving the accelerometer around [5]. When we moved the sensors faster, we noticed a larger increase in voltage.

### 2.5.2 Calibrate ADXL335

In our original design, we stated we would take the bit stream of accelerometer values and average them on each axis against gravity. The problem with this method is each time the accelerometers were turned on and off, they would start at much different base values. Therefore, we needed to implement a better a calibration technique. We quickly discovered that Arduino's IDE provides a sketch which calibrates our sensor for us. Hence,  we used a sketch in Arduino IDE where we only had to tilt each axis against gravity.

## 2.6 Software

### 2.6.1 Arduino MCU Programming

The complete code that was uploaded onto our microcontroller is available in Appendix A. The overall intention of the code is to pass integer values to the Bluetooth module. Based on which switch is flipped, several different things can be transmitted. When the make or miss switch is flipped, a 1 or 0 will be transmitted related to whether the shot was made or missed, respectively. A delay is placed specifically here both to debounce the switches and to not rush the user in flipping the switch back and forth. When the polling switch is flipped, a five-second poll is started during which the analogRead() function allows the code to interpret the accelerations in the x, y, and z directions of both accelerometers as discrete, integer values. If a greater value is read for any of the six values in successive reads, the saved maximum value is updated accordingly.

Regardless of which data is available, the itoa() function is utilized to convert the integer(s) to a character array which is readable in PuTTy's terminal and logs. A custom serial communication monitor must be created by including the SoftwareSerial library. In doing so and writing to this custom monitor, data is specifically passed to the data pins of the Bluetooth module much like it displayed in the Arduino's serial monitor. The Bluetooth module can then pass on the character arrays masquerading as integer values as intended.

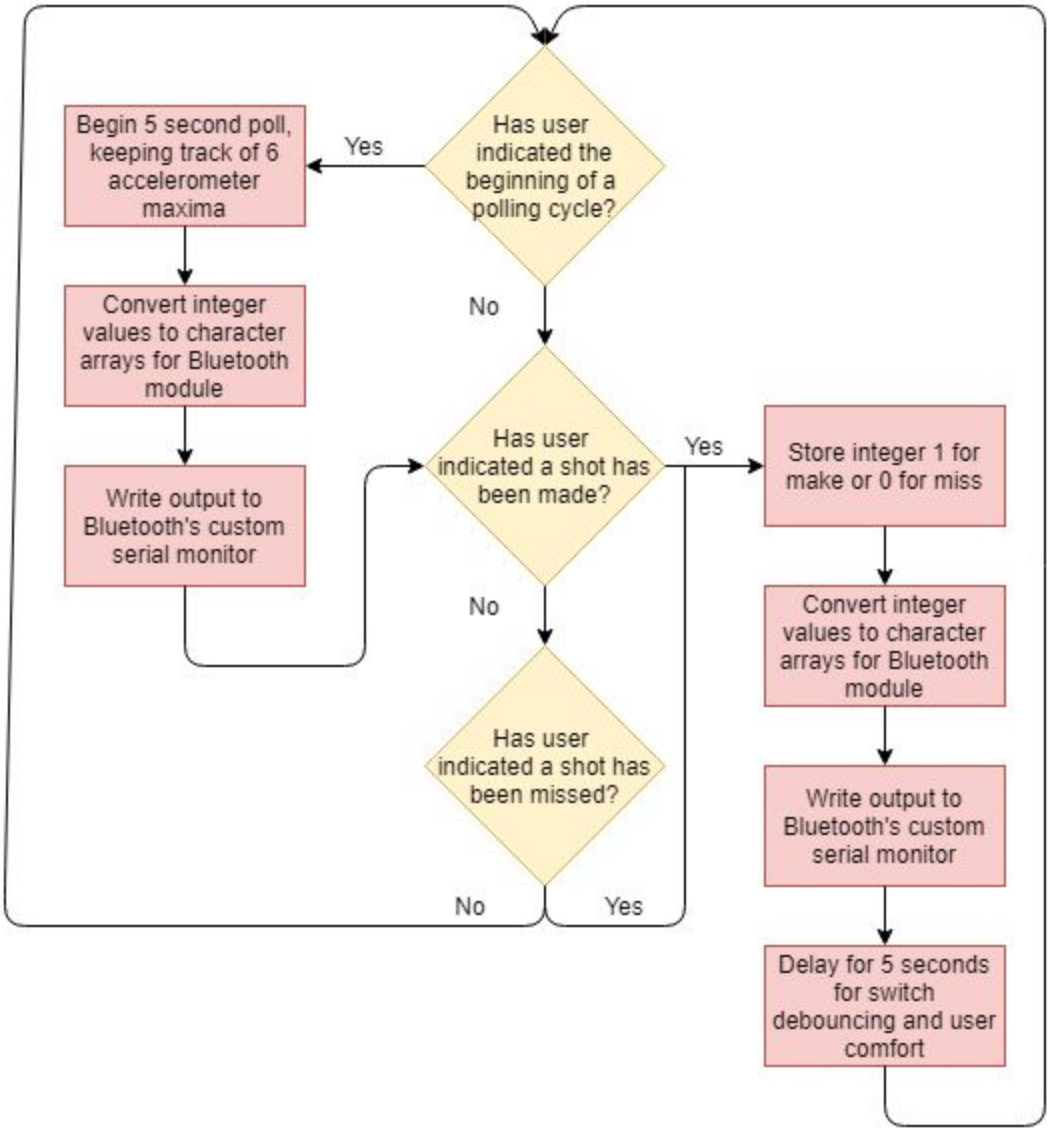


*Figure 2.6.1.1: Arduino microcontroller code flowchart*

### 2.6.2 Python Machine Learning Feedback

The complete code that acts in real-time upon the PuTTy log file to generate visual and command-line user feedback is available in Appendix B. The script is designed to run continuously while a user is taking free throws, and the PuTTy logging text file is being populated. If at any point there is a discrepancy between the number of data points and the number of labels, the script will cease to run. The script reads said file every five seconds checking for updates. When data from a poll is available, the entire log file is read, and wrist data, leg data, and labels are separated and loaded into respective arrays. At this point, a check is performed to determine if the most recent label has been inputted. If not, the script prints a message to the command line for a user to do so.

When the corresponding label is in place, the support vector machine is created as our machine learning clustering model of choice [11]. Points are plotted with their corresponding labels, and the background is shaded in like colors to the plotted points according to classification region. A green star and yellow star are plotted representing the most recent shot and a theoretical best possible shot, respectively. A white line is drawn between them evidencing how a user's most recent shot strayed from the optimal one in terms of wrist and leg acceleration. These values are also printed to the command line. Finally, the classification accuracy is printed, and the script returns awaiting the next polling data.

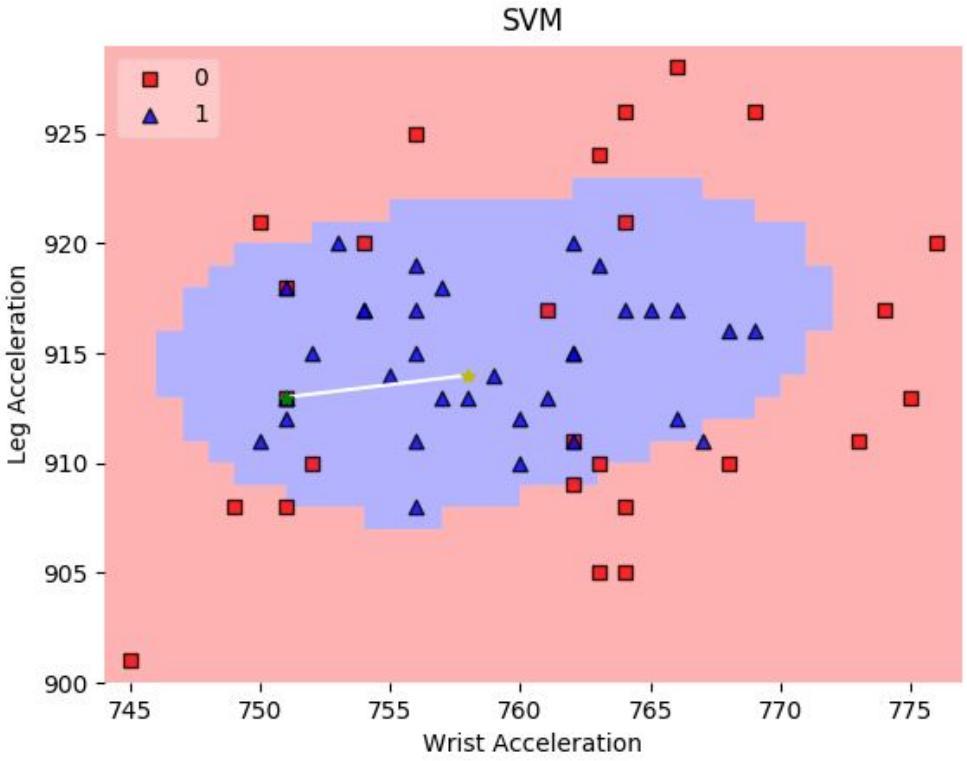



*Figure 2.6.2.1: Final support vector machine graphical output*

## 2.7 Printed Circuit Board

### 2.7.1 Switches

In our original proposal and design, we intended to use buttons in our wearable device to track made shots, missed shots, and initiate our polling cycle. While building our prototype, we realized buttons would not work in our design. We attached buttons to the pcb, but our PCB was not working correctly. We were getting raw acceleration values from both sensors, but they were taking data at random times. We would press the button then at random times data would come out with integer zeros and ones. Essentially, we were not in control of the start of the polling cycle or the make and miss buttons. In our breadboard, we would manually input a high signal (5V) and a low signal (GND) to the microcontroller. The breadboard version of circuit worked perfectly. The only new variable in the PCB was the buttons. Therefore, we knew the buttons were causing problems. By using a voltmeter, we saw the voltage go up to 5V when the button was pressed.

Once the button was released, the voltage wouldn't be at GND. It was actually feeding a floating value to the microcontroller which was incorrect. We realized we were only giving the microcontroller a high signal without a low signal. Hence, we could not create the correct polling cycle. We solved the problem immediately by using a three-pronged switch. One prong was given 5V, and the other prong was connected to GND. The last prong would be connected to whatever digital pin corresponded to a switch and needed to be toggled. Finally, when the switch moved from low to high and back to low, we initiated the correct polling cycle for the accelerometers to take data.

# 3 Cost & Schedule

## 3.1 Cost Analysis

### 3.1.1 Parts Costs

The project does not have a fixed development cost. In other words, there are no overhead costs. This means that all the costs for this project are dependent on the actual product and its design. Any parts that are currently owned by the members are considered as having no cost.

| Part | Vendor | Quantity | Cost (Bulk) |
|---|---|---|---|
| Duracell MN1604 9V Alkaline Battery With Connector | Batteryjunction.com | 1 | $1.30 |
| Voltage Regulator (LM7805) | Digikey.com | 1 | $1.32 |
| ATmega328P Microcontroller (ATMEGA328P-AN) | Microchip.com | 4 | $8.44 |
| Switches (611-7101-021) | Mouser.com | 3 | $16.41 |
| HC-06 Bluetooth Module | Amazon.com | 1 | $8.99 |

| | | | |
|---|---|---|---|
| ADXL335 Analog Accelerometer | SparkFun.com | 2 | $30.00 |
| Acrylic (2’ x 2’) | HomeDepot | 1 | $12.00 |
| Waist Strap | Amazon.com | 1 | $11.00 |
| Wrist Strap | Amazon.com | 1 | $10.00 |
| PCB (2-layer, 50x50 mm, shipping included) | PCBWay.com | 5 | $26.00 |
| **Total** | | | **$125.46** |

### 3.1.2 Labor Costs

ECS reports that for the 2015-2016 academic year the average graduating salary for a B.S. in Electrical Engineering was $68,392 [8]. Assuming a 40-hour work week and an individual working about 50 weeks each year, the equation below yields an hourly salary of $34.20.

$$\$68{,}392 / 50 \text{ weeks per year} / 40 \text{ hours per week} = \$34.20$$

*Equation 3.1.2.1: Hourly wage calculation for EE graduate*

Our three group members anticipate working on this project approximately 15 hours per week. We will consider a conservative 12 weeks of work: every week from the approval of the project until the end of the semester, excluding spring break.

$$15 \text{ hours/week} * 12 \text{ weeks} * \$32.40/\text{hour} * 3 \text{ group members} = \$17{,}496$$

*Equation 3.1.2.2: Total labor cost calculation*

### 3.1.3 Grand Total

Our project cost a total of $17,621.46 including parts and labor.

$17,496 + $125.46 = $17,621.46

*Equation 3.1.2.3: Grand total project cost*

# 4 Conclusion

## 4.1 Accomplishments

At the end of our project, our group was very proud of what we had accomplished this semester. We came into the course with a clear goal in mind, and we strongly believe that we put in a tremendous amount of effort to successfully complete our goal. As stated in the introduction, we are all very passionate about sports, and, as a result, we know how much work it takes to improve one's game. We truly feel that by the end of this course, we were able to properly implement a device that an individual can use to significantly improve his/her game when practicing alone. Most importantly to us, we were able to do this by keeping the cost of the total product under $100 which to us is very important because we understand as a student it is not feasible at times to spend a great deal of money. When testing this device out on our own game, we did see that our form as well as the number of shots being made both improved, and we all hope that other users will be able to have the same success with our product one day as well.

Regarding the device itself, we accomplished each stated goal. All hardware and software functioned such that data was transmitted properly from the accelerometers to the microcontroller to the Bluetooth module to the computer where a support vector machine was constructed. Beyond that, the data clustered fairly well with an 82% classification accuracy. Therefore, we would like to consider the device a success from the standpoints of technical completion and usefulness to a user.

## 4.2 Uncertainties

After completing this project, there were a couple of uncertainties that remained. For example, the largest uncertainty was the fact that the Bluetooth module would be unresponsive with the Bluetooth connection on the computer. The reason for this is still unknown. Originally, we believed that this was occuring because the Bluetooth device was greater than 5 ft away from the computer which would definitely cause them to disconnect per the datasheet [10]. However, this was debunked as the computer and the Bluetooth module were the same distance away from one another. The Bluetooth

was always on the same position on the user's waist and the computer was always on the same position on the floor. With these two located in the exact same place, we would sometimes be able to get well over twenty readings before the two suddenly disconnected. Sometimes we would not even be able to get ten readings without them disconnecting. Since we eliminated the distance factor, there could be a plethora of things that actually caused this to occur. This could be from just a glitchy device to problems with the computer we were using.

Another uncertainty that still remains was a code crash that occurred during our demo. Unfortunately, this particular flaw occurred during our final demonstration. After some investigation, it turned out to be an error in not branching back to the beginning of the feedback() function in the Python code after asking a user to input the most recent label. It was actually quite an easy fix.

## 4.3 Future Work/Alternatives

If we choose to pursue this project further in the future, we would choose to implement a variety of new features. Such improvements include wireless accelerometers, a sturdier and more comfortable enclosing for the MCU, expanding the device and code to expand to shots of varying distances, and also to implement a wearable display to eliminate the computer.

The reason why we would like to implement wireless accelerometers is for the convenience of our users. Though we did not run into any significant problems with the wires, eliminating the wires would also eliminate the chance of the wires getting tangled or even worse tearing. Therefore, we believe that this would also extend the life of this product. Another change which we would like to implement is a sturdier enclosure for the MCU. As it appears in section 1.3, the enclosure for the MCU could be argued as slightly bulky, especially when it is attached to the side of a user's waist. Reducing the size of the enclosure could make it significantly easier to use, and it would prevent the possibility of it getting in the way of a user's hand when he/she is shooting. The next change we would like to implement would be to expand the device to be able to provide feedback to shots of varying distances. This not only make our product more valuable and marketable to consumers, but it would also allow the product to have a much more significant and wider impact on an individual's game which was the main of this product. Finally, the last change we would like to implement in the feature is a wearable screen, such as a watch, that a user can wear on his arm when using the product. This wearable device would essentially serve the same purpose as the laptop as it would

relay to the user what they can do in order to improve their shot. The reason for this new feature would be to eliminate the use of a laptop when practicing. Taking a shot and then walking over to a laptop to see the feedback is not ideal for a user, and this would eliminate that problem.

## 4.4 Ethical and Safety Considerations

Safety and ethics are an issue of importance to our group, and they are something we made sure to take into consideration for every portion of our project. As a result we made sure that everything we did for our project aligned with the IEEE Code of Ethics.

As a group we understand the importance of safety, and that there were numerous things that could have gone wrong that could have caused a safety hazard to a user. These safety hazards could have either stem from human errors to things that may not entirely be in our control. Such uncontrollable incidents could be an injury on the court such as hurting an ankle or wrist when shooting the basketball. Human errors that could cause safety hazards include misuse of laboratory equipment such as not turning off soldering irons after using them. Hardware issues such as overheating of the battery and other components could lead to safety issues for a user.

As a group, we also made it a priority to handle any safety issues that arose professionally. This would have included pulling a fire alarm and notifying the proper authorities if something such as a fire were to occur while we were working on our product in the lab. Fortunately, we were never in a position where we had to.

As stated above, our group worked hard to abide by all the IEEE Code of Ethics. One example of this includes IEEE Code of Ethics #3. This particular one states “to be honest and realistic in stating claims or estimates based on available data” [4]. We strongly believe that we did a good job of abiding by this as everything we presented was actual data we gathered, and this is important as if we decided to ever commercialize our product for consumers. Then, it would be highly unethical to present data which could have been tainted or not accurately represent what our product is capable of.

Using the IEEE Code of Ethics, we believe that we implemented a safe and ethical product while taking into account numerous different design choices to try and mitigate any issues that could have potentially arisen.

# 5 Citations


[1] Struzik, A., Pietraszewski, B. and Zawadzki, J. (2018). *Biomechanical Analysis of the Jump Shot in Basketball*. [online] Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4234772/ [Accessed 7 Feb. 2018].

[2] Wilson.com. (2018). *Wilson LABS - Wilson X Connected Basketball*. [online] Available at: http://www.wilson.com/en-us/explore/labs/basketball/wilson-x [Accessed 8 Feb. 2018].

[3] Volleyball, P. (2018). *Noah Basketball*. [online] Available at: http://www.noahbasketball.com/ [Accessed 20 Feb. 2018].

[4] Ieee.org, "IEEE IEEE Code of Ethics", 2016. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 6- Feb- 2018].

[5] SparkFun.com (2018). *ADXL335 Data Sheet*. [online] Available at: https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf [Accessed 13 Feb. 2018].

[6] Duracell.com. (2018). *2032 Lithium Coin Battery* [online] Available at: https://www.duracell.com/en-us/product/2032-lithium-coin-button-battery/ [Accessed 20 Feb. 2018].

[7] Mouser.com (2018). *Compact 20mm Battery Holder* [online] Available at: https://www.mouser.com/ds/2/215/066-745915.pdf [Accessed 20 Feb. 2018].

[8] Engineering Career Services (2018). *Salary Information 2015-2016* [online] Available at: https://engineering.illinois.edu/documents/Salary.Info.Sheet.pdf [Accessed 22 Feb. 2018].

[9] Microchip.com. (2018). *ATmega328P - 8-bit AVR Microcontrollers*. [online] Available at: http://www.microchip.com/wwwproducts/en/ATmega328p [Accessed 23 Feb. 2018].

[10] Olimex.com. (2018). *HC-06 - Olimex* [online] Available at: https://www.olimex.com/Products/Components/RF/BLUETOOTH...HC-06/.../hc06.pdf [Accessed 23 Feb. 2018].

[11] Scikit-learn.org. (2018). *1.4. Support Vector Machines — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/svm.html [Accessed 12 April 2018].

[12] Sparkfun.com. (2018). [online] Available at: https://www.sparkfun.com/datasheets/Components/LM7805.pdf [Accessed 3 May 2018].

# Appendices

## Appendix A: Requirements and Verification Tables

Power Module

| Requirement | Verification |
|---|---|
| Capable of powering microcontroller and two accelerometers and bluetooth module | After testing whether the Duracell batteries can successfully power the microcontroller, we will connect each accelerometer one at a time to verify the power module's capability to successfully handle all the other modules of the project. Same procedure will be done for the bluetooth module. |
| Successful voltage step down from ~3 V to 1.8 V | To make sure that the voltage steps down from ~3 V to 1.8 ± 2%, we will use a digital multimeter to test the voltage at the output. |
| Successful current increase from 0.165 mA to 0.2 mA | To make sure that the current is 0.2 mA at the output, we will use a digital multimeter to test |

*Table A.1: Requirements and verification for power module*

Microcontroller

| Requirement | Verification |
|---|---|
| Capable of receiving 1,500 voltage values from ADC via I2C in 1 s | Send known stream of sensor data from ADC to MCU via I2C. Read transmitted data on Arduino Uno using its interface provided timer to check latency and verify correctness. |
| Capable of transmitting data via UART in 1 s | Connect the Bluetooth module via UART to the Arduino Uno, and send discrete, floating-point values that must be read |

| | |
|---|---|
| | correctly by the computer after being wirelessly transmitted. Do so with Uno's timer and the computer's timer to verify latency on both ends. |
| Store 24 KB of data in flash memory | 32 KB of flash memory is sufficient. Equation 2.4.1.1 below describes the derivation of the 24 KB necessity. Using the Arduino Uno, we will read the values to ensure correctness of the 1,500 values in flash memory. |

*Table A.2: Requirements and verification for microcontroller in control unit*

Analog-to-Digital Converter

| Requirement | Verification |
|---|---|
| Analog sensor data stream must be sampled into discrete values in 1 s total | Use the Arduino Uno's interface and timer to determine if 1,500 discrete voltage values have been passed to the MCU's flash memory from the ADC without errors and under the 1 s latency threshold. |

*Table A.3: Requirements and verification for analog-to-digital converter in control unit*

Sensing Module

| Requirement | Verification |
|---|---|
| Check ADXL335 functionality | Connect to Arduino Uno and ensure continuous voltage output through bit stream. Prove 1g is around .55 V |
| Calibrate ADXL335 for accurate measurements | Connect to Arduino Uno and adjust sensor for each axis against gravity. With a 1g voltage reading from a voltmeter, we can see how close the sensor records gravity. We can do this many times and take an average. Hence, we can verify our sensitivity by moving our sensor to |

| | |
|---|---|
| | get a reading close to .003 g and subtracting the offset in voltage we got from our gravity calculation. |
| Operate at 50 Hz | Measure the frequency of axis output with an oscilloscope. |

*Table A.4: Requirements and verification for sensing module*

## Computer and Bluetooth

| Requirement | Verification |
|---|---|
| Bluetooth 2.0 compatibility | Ensure pairing and established connection with HC-06 Bluetooth module. |
| Must receive and store floating-point values HC-06 Bluetooth module | Send known value from MCU through Bluetooth module to computer's Bluetooth receiver. Verify correct value, and save in script. |
| Ability to construct support vector machine with training data | Obtain contrived data set with known SVM, and ensure SVM from script is comparable. |
| Entire process from receiving data points from Bluetooth module to outputting feedback to user must take under 2 s | Use script timer for entire Python script, and ensure it reads ≤ 2 s regardless of training/testing phase or number of saved data points. |

| | |
|---|---|
| Capable of receiving data via UART and transmitting it to a Bluetooth receiver in 1 s combined | Connect module to Arduino Uno's UART bus, and send known, discrete, floating-point values from the MCU that must be read correctly by this module after being wirelessly transmitted to a computer. Use the timer of the Uno and computer to verify latency on both ends. |
| Must maintain Bluetooth connection distance of about 5 ft between user's hip and computer | Must maintain Bluetooth connection distance of about 5 ft between user's hip and computer |

*Table A.5: Requirements and verification for Computer and Bluetooth module*

## Appendix B: PCB Layout

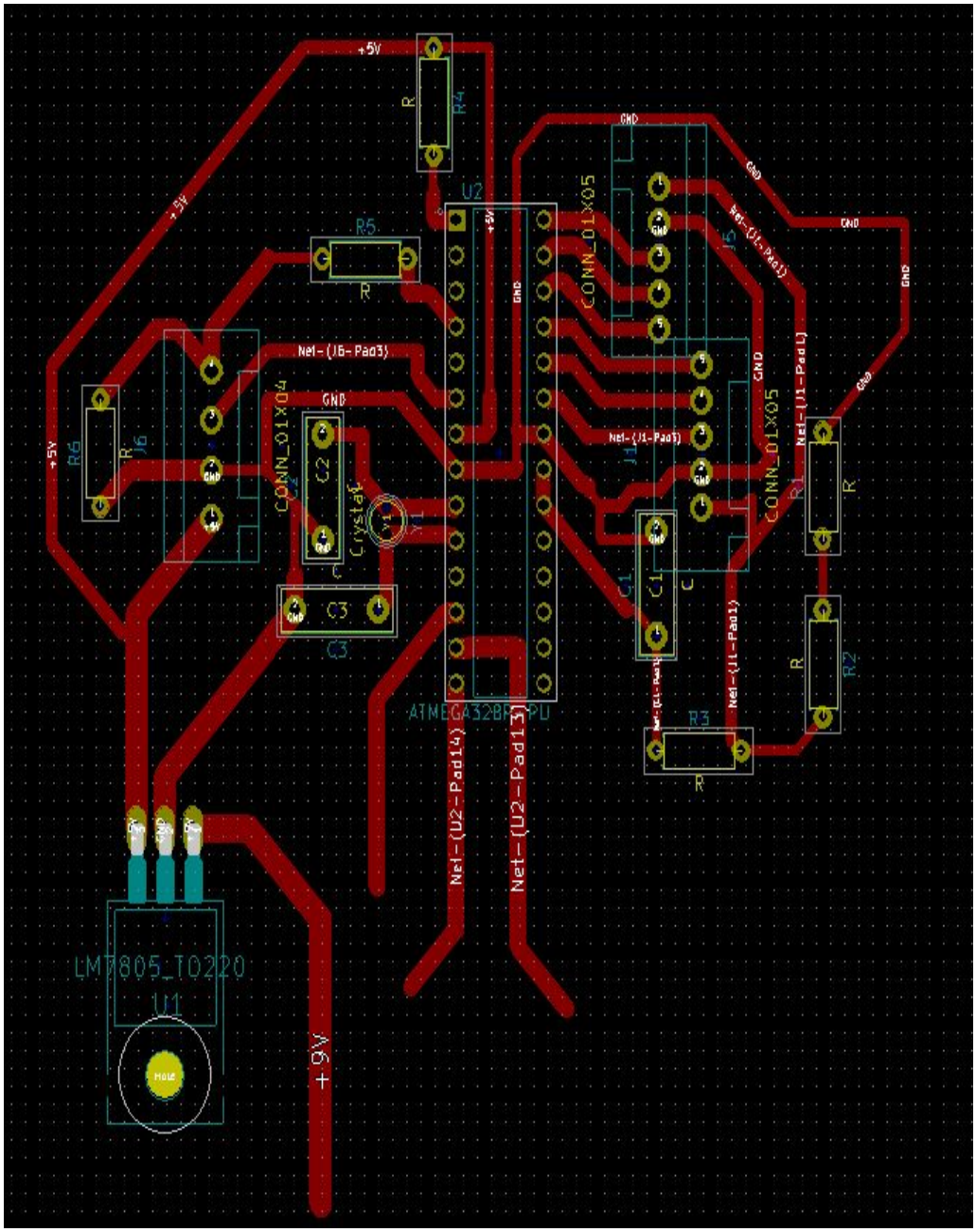



*Figure B.1: Printed circuit board for entire device*

# Appendix C: Circuit Schematic

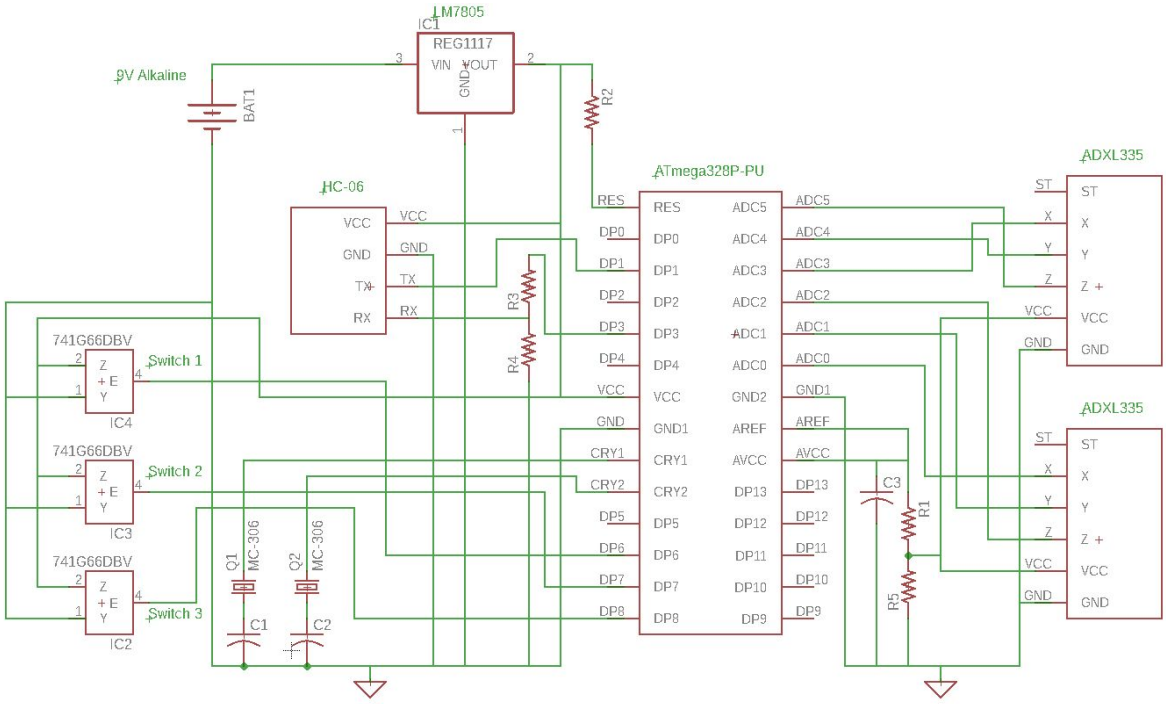
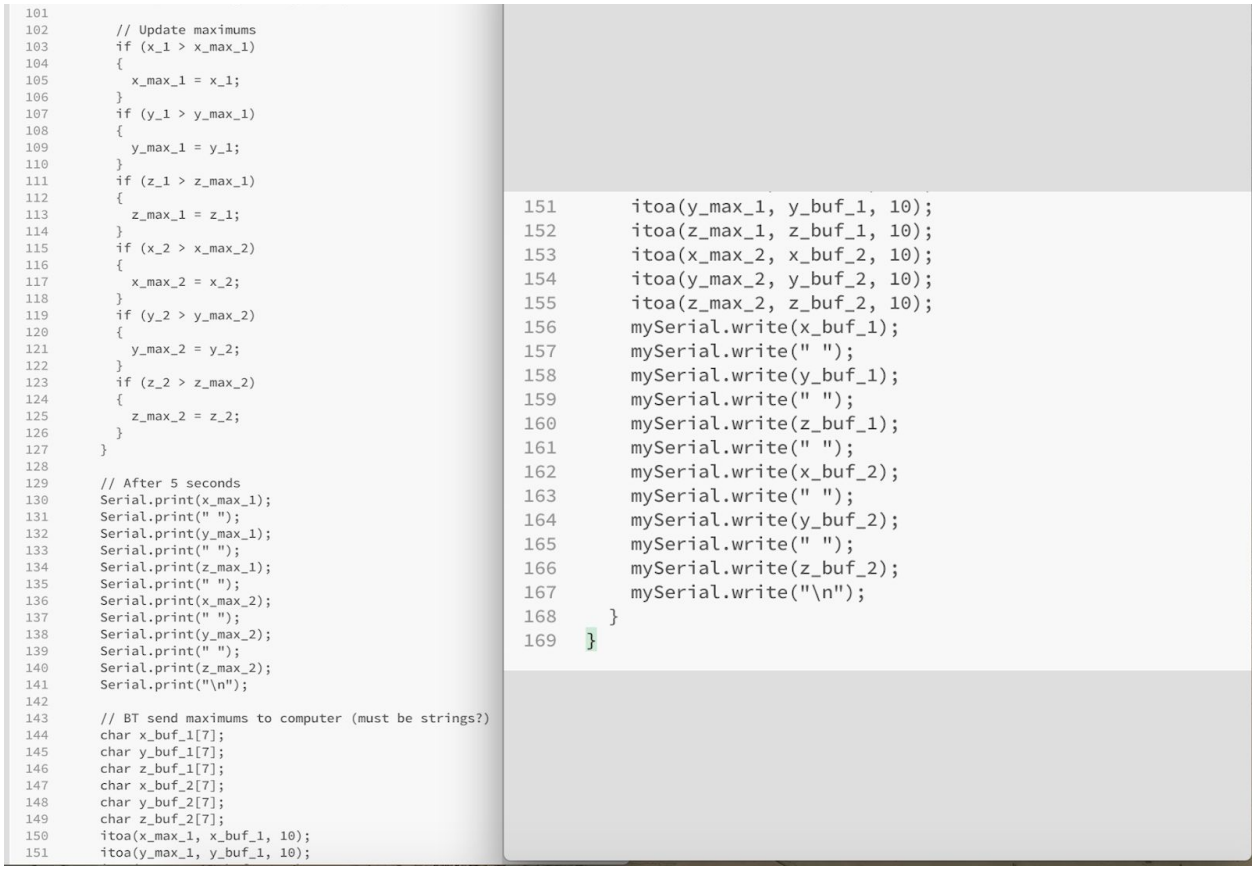



*Figure C.1: Circuit schematic for entire device*

# Appendix D: Arduino Code

```
#include <SoftwareSerial.h>
#include <stdlib.h>
SoftwareSerial mySerial(4, 2); // RX, TX

// Accelerometer x/y/z pins
const int x_pin_1 = A0;
const int y_pin_1 = A1;
const int z_pin_1 = A2;
const int x_pin_2 = A3;
const int y_pin_2 = A4;
const int z_pin_2 = A5;

// Maximums
// TODO: alter this
int x_max_1 = 0;
int y_max_1 = 0;
int z_max_1 = 0;
int x_max_2 = 0;
int y_max_2 = 0;
int z_max_2 = 0;

// Buttons
const int start_button_pin = 6;
const int make_button_pin = 7;
const int miss_button_pin = 8;
int start_button_state = 0;
int make_button_state = 0;
int miss_button_state = 0;

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);

  // The HC-06 defaults to 9600 according to the datasheet.
  mySerial.begin(9600);

  // Buttons
  pinMode(start_button_pin, INPUT);
  pinMode(make_button_pin, INPUT);
  pinMode(miss_button_pin, INPUT);
}

void loop()
{
  // put your main code here, to run repeatedly:

  // Voltage high - simulating button
  //  int button = analogRead(A5);
  //  float voltage = button * (5.0 / 1023.0);

  // Using actual buttons
  start_button_state = digitalRead(start_button_pin);
  make_button_state = digitalRead(make_button_pin);
  miss_button_state = digitalRead(miss_button_pin);

  // Write 1 for make, 0 for miss, then newline
  if (make_button_state == HIGH)
  {
    int make = 1;
    char make_buf[7];
    itoa(make, make_buf, 10);
    mySerial.write(make_buf);
    mySerial.write("\n");
    // Debounce
    delay(5000);
  }
  if (miss_button_state == HIGH)
  {
    int miss = 0;
    char miss_buf[7];
    itoa(miss, miss_buf, 10);
    mySerial.write(miss_buf);
    mySerial.write("\n");
    // Debounce
    delay(5000);
  }

  // Polling time
  unsigned long five_secs = 5 * 1000UL;
  unsigned long start = millis();
  //if (voltage > 1.5)
  if (start_button_state == HIGH)
  {
    while (millis() - start < five_secs)
    {
      // Reading
      int x_1 = analogRead(x_pin_1);
      // 1 ms delay for "stability"
      // Will that be a problem for us?
      delay(1);
      int y_1 = analogRead(y_pin_1);
      delay(1);
      int z_1 = analogRead(z_pin_1);
      delay(1);
      int x_2 = analogRead(x_pin_2);
      delay(1);
      int y_2 = analogRead(y_pin_2);
      delay(1);
      int z_2 = analogRead(z_pin_2);

```

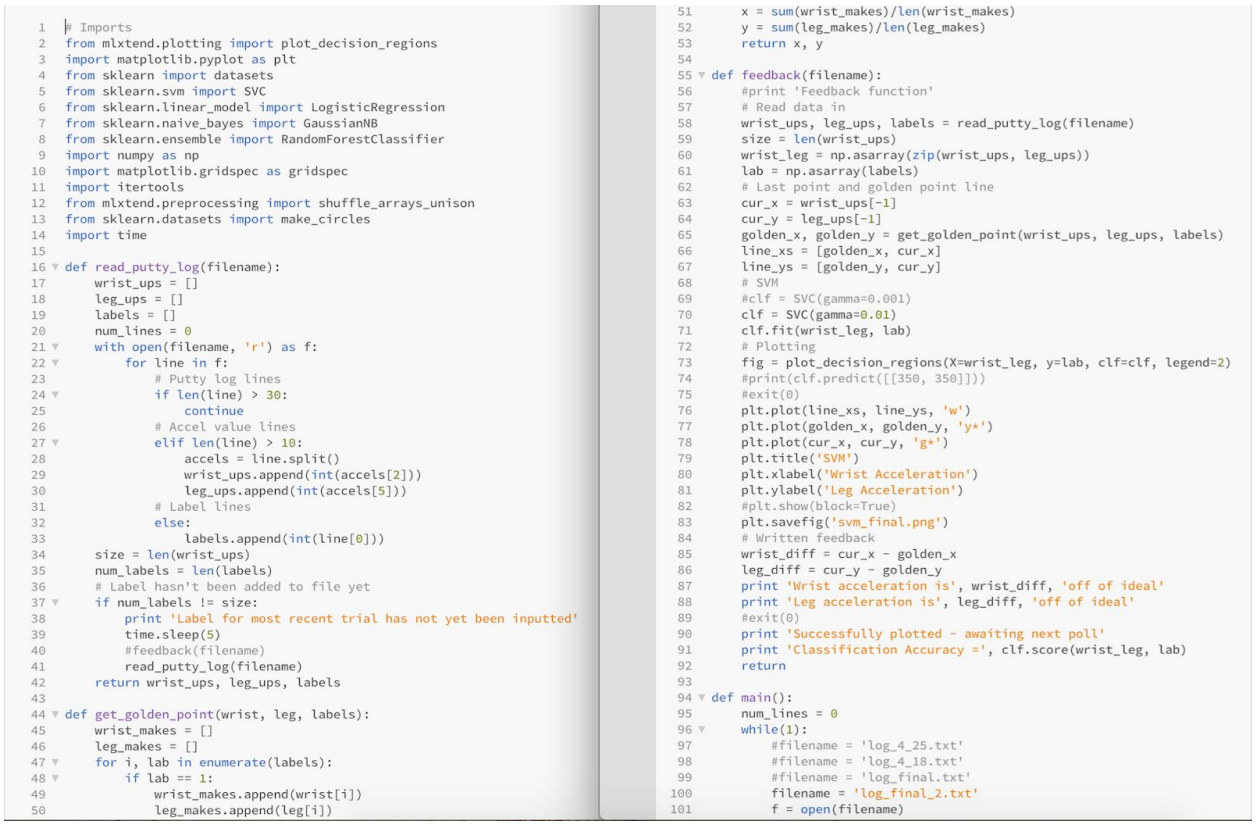


```
101
102       // Update maximums
103       if (x_1 > x_max_1)
104       {
105         x_max_1 = x_1;
106       }
107       if (y_1 > y_max_1)
108       {
109         y_max_1 = y_1;
110       }
111       if (z_1 > z_max_1)
112       {
113         z_max_1 = z_1;
114       }
115       if (x_2 > x_max_2)
116       {
117         x_max_2 = x_2;
118       }
119       if (y_2 > y_max_2)
120       {
121         y_max_2 = y_2;
122       }
123       if (z_2 > z_max_2)
124       {
125         z_max_2 = z_2;
126       }
127     }
128
129     // After 5 seconds
130     Serial.print(x_max_1);
131     Serial.print(" ");
132     Serial.print(y_max_1);
133     Serial.print(" ");
134     Serial.print(z_max_1);
135     Serial.print(" ");
136     Serial.print(x_max_2);
137     Serial.print(" ");
138     Serial.print(y_max_2);
139     Serial.print(" ");
140     Serial.print(z_max_2);
141     Serial.print("\n");
142
143     // BT send maximums to computer (must be strings?)
144     char x_buf_1[7];
145     char y_buf_1[7];
146     char z_buf_1[7];
147     char x_buf_2[7];
148     char y_buf_2[7];
149     char z_buf_2[7];
150     itoa(x_max_1, x_buf_1, 10);
151     itoa(y_max_1, y_buf_1, 10);
```

```
151     itoa(y_max_1, y_buf_1, 10);
152     itoa(z_max_1, z_buf_1, 10);
153     itoa(x_max_2, x_buf_2, 10);
154     itoa(y_max_2, y_buf_2, 10);
155     itoa(z_max_2, z_buf_2, 10);
156     mySerial.write(x_buf_1);
157     mySerial.write(" ");
158     mySerial.write(y_buf_1);
159     mySerial.write(" ");
160     mySerial.write(z_buf_1);
161     mySerial.write(" ");
162     mySerial.write(x_buf_2);
163     mySerial.write(" ");
164     mySerial.write(y_buf_2);
165     mySerial.write(" ");
166     mySerial.write(z_buf_2);
167     mySerial.write("\n");
168   }
169 }
```

*Figure D.1: Arduino IDE code uploaded to ATmega328P-PU*

# Appendix E: Python Code



```
# Imports
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import matplotlib.gridspec as gridspec
import itertools
from mlxtend.preprocessing import shuffle_arrays_unison
from sklearn.datasets import make_circles
import time

def read_putty_log(filename):
    wrist_ups = []
    leg_ups = []
    labels = []
    num_lines = 0
    with open(filename, 'r') as f:
        for line in f:
            # Putty log lines
            if len(line) > 30:
                continue
            # Accel value lines
            elif len(line) > 10:
                accels = line.split()
                wrist_ups.append(int(accels[2]))
                leg_ups.append(int(accels[5]))
            # Label lines
            else:
                labels.append(int(line[0]))
    size = len(wrist_ups)
    num_labels = len(labels)
    # Label hasn't been added to file yet
    if num_labels != size:
        print 'Label for most recent trial has not yet been inputted'
        time.sleep(5)
        #feedback(filename)
        read_putty_log(filename)
    return wrist_ups, leg_ups, labels

def get_golden_point(wrist, leg, labels):
    wrist_makes = []
    leg_makes = []
    for i, lab in enumerate(labels):
        if lab == 1:
            wrist_makes.append(wrist[i])
            leg_makes.append(leg[i])
    x = sum(wrist_makes)/len(wrist_makes)
    y = sum(leg_makes)/len(leg_makes)
    return x, y

def feedback(filename):
    #print 'Feedback function'
    # Read data in
    wrist_ups, leg_ups, labels = read_putty_log(filename)
    size = len(wrist_ups)
    wrist_leg = np.asarray(zip(wrist_ups, leg_ups))
    lab = np.asarray(labels)
    # Last point and golden point line
    cur_x = wrist_ups[-1]
    cur_y = leg_ups[-1]
    golden_x, golden_y = get_golden_point(wrist_ups, leg_ups, labels)
    line_xs = [golden_x, cur_x]
    line_ys = [golden_y, cur_y]
    # SVM
    #clf = SVC(gamma=0.001)
    clf = SVC(gamma=0.01)
    clf.fit(wrist_leg, lab)
    # Plotting
    fig = plot_decision_regions(X=wrist_leg, y=lab, clf=clf, legend=2)
    #print(clf.predict([[350, 350]]))
    #exit(0)
    plt.plot(line_xs, line_ys, 'w')
    plt.plot(golden_x, golden_y, 'y*')
    plt.plot(cur_x, cur_y, 'g*')
    plt.title('SVM')
    plt.xlabel('Wrist Acceleration')
    plt.ylabel('Leg Acceleration')
    #plt.show(block=True)
    plt.savefig('svm_final.png')
    # Written feedback
    wrist_diff = cur_x - golden_x
    leg_diff = cur_y - golden_y
    print 'Wrist acceleration is', wrist_diff, 'off of ideal'
    print 'Leg acceleration is', leg_diff, 'off of ideal'
    #exit(0)
    print 'Successfully plotted - awaiting next poll'
    print 'Classification Accuracy =', clf.score(wrist_leg, lab)
    return

def main():
    num_lines = 0
    while(1):
        #filename = 'log_4_25.txt'
        #filename = 'log_4_18.txt'
        #filename = 'log_final.txt'
        filename = 'log_final_2.txt'
        f = open(filename)
        f = open(filename)
        new_num_lines = sum(1 for line in open(filename))
        if new_num_lines > num_lines and num_lines == 0:
            num_lines = new_num_lines
        elif new_num_lines > num_lines:
            num_lines = new_num_lines
            #print f.readlines()[-1]
            new_accels = f.readlines()[-1]
            if len(new_accels) < 10:
                f = open(filename)
                new_accels = f.readlines()[-2]
            wrist_x = new_accels[0:3]
            wrist_y = new_accels[4:7]
            wrist_z = new_accels[8:11]
            leg_x = new_accels[12:15]
            leg_y = new_accels[16:19]
            leg_z = new_accels[20:23]
            print 'New relevant values are: wrist accel =', wrist_z, 'leg accel =', leg_z
            feedback(filename)
        print 'No new data - taking a nap'
        time.sleep(5)

main()
```

*Figure E.1: Python script providing user feedback*