User Manual

Engineering in Context, *Companion Cane*

ENGR 1624 Professor Fitz-Gerald

December 17th, 2021

The University of Virginia, School of Applied Science and Engineering

Table of Contents

Executive Summary

Rationale:

As we age, our ability to maneuver and travel over long distances may be inhibited. In the event of this, many require the use of canes and other devices to traverse in an effective manner. However, should someone requiring the use of one of these devices fall, it becomes increasingly difficult to rise again as age increases. Our Companion Cane supports and helps alleviate this issue by notifying the proper contacts in the event of a failure to get up again. There are 37.3 million falls that occur each year that require medical attention and an estimated 684,000 falls are fatal on a global scale. Persons of age 65< are more likely to die due to fall-related injuries than younger adults (*2007 Trends in Trauma and Emergency Medicine)*. The purpose of the companion cane is to support the elderly who need assistance walking or those who are in recovery from underlying injuries, but ultimately, the main purpose is to try to reduce the number of fatalities due to fall-related injuries.

Objectives:

- Detect Falls
- Measure Heart Rate (bpm)
- Measure Acceleration of the Cane
- Measure how much weight/force is being put on the cane
- The ability to send data to a device/exportable data

Description:

The cane is designed to achieve all of the following objectives.

Using a heartbeat sensor that will be integrated into the handle of the cane and a circuit board, the cane will be able to detect and monitor the heart rate of the individual using the cane. To measure the acceleration of the cane, the cane will have an accelerometer implemented in the shaft of the cane which will be able to determine how fast the user is moving with the cane. The cane will also be equipped with a strain gauge at the bottom near the foot of the cane which will allow us to record how much weight/force is being put on the cane. Using the data from the

strain gauge and the accelerometer, the cane will be able to use this information to detect a fall. Lastly the data should be exportable via bluetooth for the user and medical professionals.

Conclusions:

The outcome of this project is the *Companion Cane*. The prototype includes a 3D printed exterior with a built-in heart-rate monitor, accelerometer, and strain gauge. Using these sensors, the *Companion Cane* software detects falls and in the event that the user does not get back up and apply force within 15 seconds of the fall, emergency contacts are messaged via texts and doctors are alerted via email and texts if registered as contacts of the user. The outcome is a prototype but in terms of the class it is a final product with all features working to the best of their ability.

Project Mission Statement

Lives can be saved if emergency services knew exactly what happened and how it happened. There is already some technology like this in our day of age, but it could always be improved and we can implement this technology in everyday objects to create new ideas to help EMS and the caller. Our goal as upcoming engineers is to create a device that can help EMS or other emergency services determine how the victim is doing or help resolve certain situations.

From 2007 to 2016, there was a 30% increase in the United States fall death rates. In 2018, 28% of U.S. adults age 65 and older had reported falling that year of which 25.3% are in Virginia. In 2018 it was also recorded that there are 36 million falls along with 950,000 hospitalizations, and 32,000 deaths due to falls in the U.S. (“Older Adult Falls 2021”). The target audience for canes usually are unable to recover from a catastrophic fall on their own and oftentimes may be severely injured due to that fact. The Companion Cane plans to more rapidly alert authorities in the event of a fall, reducing the injuries sustained from falls.

Objectives

The objectives for the *Companion Cane* and this project align with the teams formed within the class to build the cane. First, the cane must be durable in structure and material. Inside the cane, the electrical circuit must connect all the sensors to the main unit. That circuit must be able to send data to the program in order to run fall detection algorithms. The program will send text messages to emergency contacts in the event of a fall and communicate data with physicians. All of these objectives must be tested thoroughly in order to provide a friendly user experience.

Design

The design of the cane amalgamated from all the different teams' visions of how their part of the cane would work. For the outline, the cane is meant to help patients in the event of a fall by alerting emergency contacts that a fall has occurred. Additionally, it will provide an immense amount of data for doctors to assist in diagnosis and recovery. However, in practice the cane is much more complex and a summary of how each team contributed to the design and functionality of the cane is detailed below.

Legal Team

The primary focus of the Engineering Legal, Health, Marketing and Safety team is to identify the liabilities associated with the development of a walking cane and the potential market for such a device. If an actual product is produced instead of a prototype, it would require extensive testing in order to determine what the actual specifications are. Similarly, tests on battery life, longevity of the circuits, accuracy of the sensors, overall function, and other mechanical and electrical components must be tested to ensure that the Companion Cane functions properly. Perhaps the most important part of the Companion Cane, the transmitter that emails collected data to medical professionals, is a major source of liability. This is because sensitive medical data is being sent through email, which is a service that malignant hackers can intercept. Since this is a prototype, email serves as a proof-of-concept and if the Companion Cane were to be developed, a more secure means of transmission must be developed. It will also be important to alert consumers to the extent to which their data is collected and how the emergency contact system works.

Marketing for the Companion Cane is another portion of the detailed design that must be addressed in order for the product to be considered successful. The Companion Cane has a long way to go before it would be released to the public, but part of our job is to make sure that a target market is kept in mind during the construction of the cane and that the product has a need in the market.

Our main target market consists of the elderly and the physical therapy industry, including private practices and hospitals. Industrial marketing will be conducted via direct contact with large healthcare organizations and business, pitching our cane and its abilities, and through ad publications in medical journals. Direct consumer marketing will be through infomercials and product pushing. Additionally, upon completion of the design final pricing will be determined, but keeping the target market in mind includes product pricing and affordability, so our team will also closely monitor finances of the cane's construction and research potential production options, to estimate product manufacturing costs.

Electrical Team

To accomplish the objectives of this project, the electrical team plans to incorporate the essential sensors and electrical components for the Companion Cane that make it capable of accomplishing its primary purpose of alerting appropriate authorities if the user requires critical assistance.

The final schematic works the same way that block diagram worked except that the emergency button was removed per request of the Programming team, believing that it was no longer necessary. The battery connects to the B+ and B- terminals of the TP4056 so that it is charged and protected. The power from the output of the TP4056 connects to the input side of the buck-boost converter (XL 6009). There is a switch in line with this power to switch the device on and off. The reason that the power switch is not between the battery and the TP4056 is because if the device was left off for too long, the battery could over discharge because the TP4056 can no longer measure its voltage. There is a power indicator LED that is connected in series with a 2k resistor. The reason that the resistor value is relatively high is because it does not need to be at full brightness to indicate that the device is on. The XL 6009 will be fine-tuned to 5v and then will be connected to all of the sensors and to the EXP32 to provide power. It was initially thought that the heart rate sensor's analog output would need to be digitized to be able to be used as an interrupt for the microcontroller to process. Despite this, a digitizer was not needed and the output of the heart rate sensor connected straight to GPIO pin 34 of the ESP32. The programming team sampled the heart rate at a specified rate instead of using an interrupt because they had a decent idea of how long each pulse would be, no interrupt was necessary.

The ADXL345 accelerometers are connected to the designated I2C pins of the ESP32. These pins are GPIO22 for SCL and GPIO21 for SDA. Both SCL and SDA of the two accelerometers are attached to these pins because multiple I2C devices can be used with the same pins; this was clarified by the datasheet for the ADXL345. The datasheet also said that CL should be pulled to 5v and SDO should be pulled to ground when communicating with I2C. These requirements were incorporated in the schematic. Because I2C requires pull-up resistors on both of the pins, a 10k pull up resistor on each pin.

The load cell connects to the HX711 with their corresponding colors: white, green, red, black, and yellow. A new load cell was chosen that has a limit of 200 kg: the TAS501. The

HX711 connects to the ESP32 for serial communication. The DAT pin connects to GPIO4 and the CLK pin connects to GPIO2. Both of these GPIO pins can be used as inputs and outputs so communication will be possible. Pull up resistors are not required here because it is not the I2C communication standard.

All of the dimensions of the parts were given to the BME/MEC team and the schematic was given to the programming team to incorporate the necessary pins into their code. The 24 AWG wire selected has a diameter of 1.4 mm and this information was given to the mechanical team along with the amount of wires going to each of the components. The wire will be stripped, twisted, tinned, and then soldered to the top sides of the boards after they have been mounted or epoxied on the body of the cane.

Pictured below is the final circuit schematic.

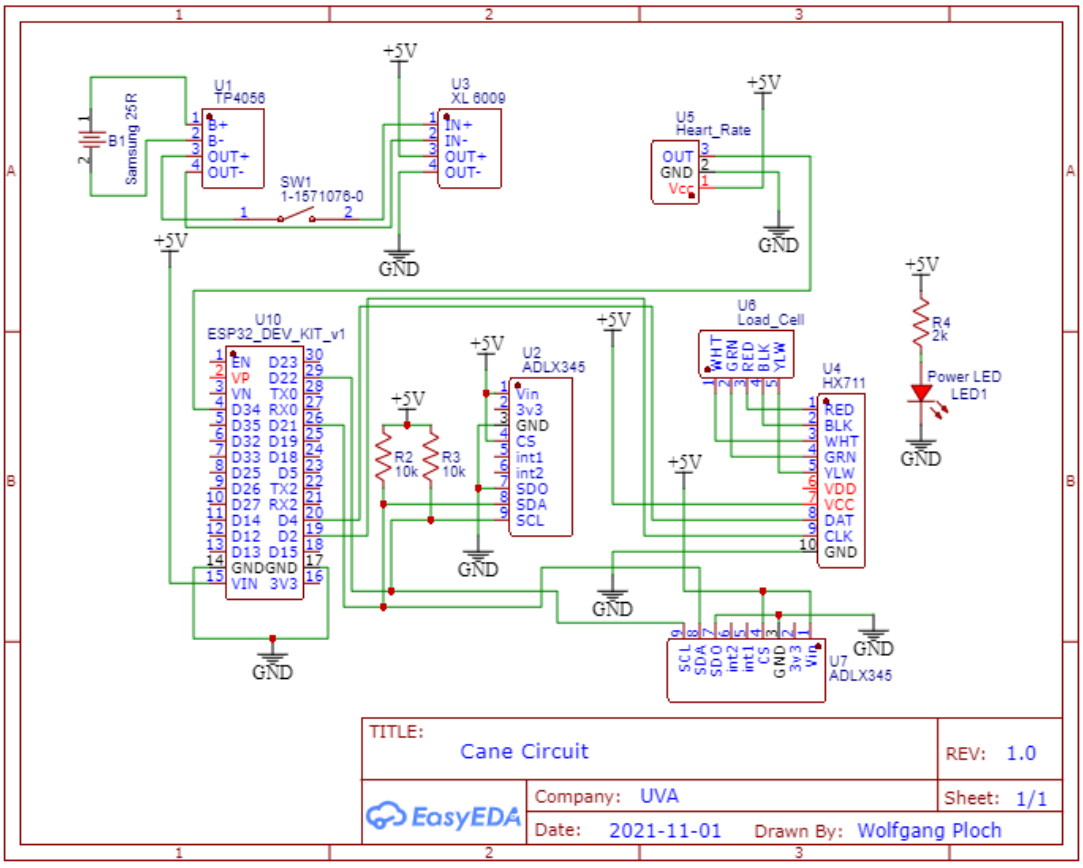


Fig. 1. Final Schematic of the Circuit

Program Team

The Programming team worked closely with the Electrical, BioMech, Drafting, and Testing teams in the preliminary design and fabrication phases. In the Preliminary design phase, we heavily discussed with the electrical team on which sensors they would utilize along with whether the electrical team would use a Raspberry pi or an Arduino, ultimately choosing the Arduino. In later phases of the project, the Programming team worked with Electrical, BioMech, and Drafting teams in fabrication, producing the cane and implementing the code into the product.

The Programming team was tasked with compiling code that would run in the Arduino itself and outside of the Arduino in order to establish wireless connection capabilities for the cane. The Arduino code had to be able to read from its various sensors which were the pulse sensor, load cell amplifier, and accelerometer and enable the Arduino to communicate with bluetooth. We decided to analyze the data externally while sending the data at a regular rate via bluetooth. The data analysis consisted of fall detection and scheduled graphs of health information for a physician to view.

To accomplish the bluetooth connection, we utilized the Arduino IDE's "BluetoothSerial" library which would begin a bluetooth serial communication that external devices could connect to. For the python code to communicate with the Arduino, the computer had to register the device as a bluetooth device and check the port in which the device is connected to in the settings. Depending on which port the Cane is connected to, the code needs to be adjusted to communicate with the Arduino. Then, using the "pyserial" library, we were able to connect the python code to the bluetooth serial port. Sensor data is then sent to the computer by having the Arduino regularly print data to the bluetooth serial connection at a regular rate which we set to one-twentieth of a second and the python code connects to the bluetooth serial and reads the printed lines and adds them to a csv file. This saves the data for fall detection and for later use in graphing. (See Appendix K: Arduino Code and Appendix L: main.py) Also we decided to go with one-twentieth of a second because it was the best rate at which data can be read while also giving enough readings for the fall detection to work as intended.

The python code communicates the health data and emergencies by first taking input from the user using a text service. (See Appendix L: app.py) This code for the prototype is run

on the user's computer, however, for a finished product the code would be run on a server maintained by our company and information registered will be saved to a database. The text service initially requires the user to text the provided number an authentication code to activate their number for the service. For simplicity and showcasing the device, we have set the code to "1234" and after sending the authentication code, the text service will then prompt the user to provide several inputs for their information. These inputs are saved as text files in the same directory as the python files. Some of those inputs are then used in the code itself such as the emergency and physician contact information to communicate emergencies and health data. There are also checks on what information the user provides in order to ensure that the user inputs are correctly logged and can be utilized for other sections of the code. This code requires the "twilio" and "flask" libraries which enable communication with the computer's port 5000. When activating the code, a command window called "ngrok" must be downloaded to open port 5000 for messages to be sent to the port connection through webhooks. With the "ngrok" window available, to open port 5000, the command "ngrok http 5000" must be inputted and then a link will be listed on the window and must be copied into the twilio number's webhook on the twilio website.

The next section of code is the portion that emails health information to the user's physician registered with the text service. (See Appendix M: emailsender.py) This code is imported into the main line of code and consists of three functions. One sends daily data to a physician to monitor the health through the heart rate graphs and another sends weekly emails of the heart rate graphs. The last email function is used when an emergency occurs graphing all sensor data for physicians to get a better understanding of the cause of the emergency. This code requires the "collections", "matplotlib", "datetime", and "pandas" libraries.

Then, following the emails, the text messaging code is also utilized in emergencies to notify emergency contacts. (See Appendix N: sms.py). This code in its prototype form utilizes the Programming team's twilio account to send text messages from the twilio number. When an emergency occurs, the code checks the registered information and uses that to determine what number to send the text messages to and depending on the emergency the message will change. This code, like the text service, requires the "twilio" python library.

The final segment of code is the fall detection which is part of the 'main.py' code that includes the bluetooth connection and imports other segments of the code including the email,

text messaging, and graphing. This code threads two while loops so that they run simultaneously. The second while loop continues the communication with the bluetooth serial port as described earlier. The first while loop consists of the fall detection code and the code is split into three parts. The first segment checks the past thirty data points and checks for intersections within the data points if graphed on a continuous graph. If the intersection exists within a certain range, the fall is initially detected and the failsafe is initiated. The code begins the next segment which determines if the cane was in use prior to the fall. This checks the last 50 data points for a certain force threshold to determine if the cane was in use. If it is determined that the cane was in use, then the third check is initiated, and if it is determined that the cane was not in use, then the detected fall is canceled and it returns to the first segment of code. If it detected that the cane was in use, then it would check for data points fifteen seconds after the fall was detected. This checks for force that meets a certain threshold concluding that the cane was picked up and in use again and that the individual is fine. If the first segment fails then it checks for the orientation and stillness of the cane, if it detects that either of these is true, then the fall is confirmed and the registered emergency contact and physician contact are texted and the registered physician email is sent the graphed data alongside the raw csv data.

To program our code, we utilized most of our time researching python libraries and online tutorials to get a sense for what we had to accomplish. One of our members had experience with data science and was able to create the graphing program for the project. Aside from that code, we diligently researched and referenced online resources for the text messaging, emails, and bluetooth communication. The fall detection was the most challenging part to code and ultimately required trial and error and analyzing how changes in the data related to certain actions. However, with the assistance of the Testing team, the Programming team was able to analyze the data and better understand the conditions required for a fall.

Drafting Team

Since multiple subgroups were working towards the final assembly of the Companion Cane, proper communication between the teams was a crucial aspect that benefited the success of this project. In particular, the drafting team worked closely with other specialized groups by collecting rough sketches from them. Teams that needed completed drawings, submitted design ideas and requested drawings to be made by the drafting team. Once requests were sent in, the drafting team turned these drafts into clear technical drawings for all parts of the cane. To better facilitate and advance the multilinear progress of this collaboration, the drafting team made sure the drawings were completed as soon as possible. The main goal of the drafting team was to create clear and simple CAD drawings to help the other teams better represent their designs. Accordingly, it was the drafting team's job to unify the designs of all other teams into a cohesive schematic unit that achieved the overall goals of the project. Furthermore, the drafting team was also responsible for educating other design teams on locating resources of free drawing software for college students, the usage of drawing software and conventional mechanical drawing standards.

Many different CAD software were available to use, and each had different abilities that could have been used to carry out whatever tasks needed to be done. The three main programs that were under consideration were AutoCAD, Inventor, and Fusion360. These programs were all designed for the purpose of drawing and designing mechanical parts. The drafting team chose these three, specifically, because they had previous knowledge and experience using them which saved time in the drafting process. AutoCAD, while having a 3-dimensional drawing setting, was primarily used for two-dimensional designs requested by the other teams. Inventor and Fusion360 were used to create the 3-D renderings of the parts, as well as assemblies and all final drawings. The drafting team decided to use both Inventor and Fusion360 because two of their members had Apple computers which could only run Fusion360. One of their members, however, was well versed in Inventor, and as each program was used to open the other's drawings, they had decided to use both Inventor and Fusion360.

Criteria (scale of 1-10)	AutoCAD	REVIT	Inventor	Fusion 360	Onshape	Rhino	SketchUp
System Requirements	4	4	4	5	8	3	5
Price (1 or 0)	1	1	1	1	1	0	0
Team Knowledge	7	1	7	7	8	1	0
Applicability	8	2	10	10	4	10	5
Versatility	5	3	9	7	2	4	3
TOTAL	25	11	31	30	23	18	13

Fig. 2. Computer Assisted Design Program Selection

Biomech Team

The Biomech Team was responsible for finalizing the structure of the cane which was in tune with the biomechanics of the target consumers and the overall functionality of the cane. We needed to create a design that is not only aesthetically pleasing but also able to implement medical technology and analyze data to improve the human condition. Biomech worked closely with the drafting and electrical team.

During the detailed design stage Biomech had to complete, optimize, and revise a prototype of the companion cane. This required us to specify and finalize materials, dimensions, and components in communication with the design and electrical group. We also needed to finalize an assembly plan. As per the preliminary design plan, a heart rate sensor, a load cell, and accelerometer was to be included in the cane's design, as well as all the necessary electrical components to power the circuit and communicate with a computer. However, elements of the design changed, such as the materials that will be used to build the cane and the fabrication process going into the cane, as well as several design choices that affected the implementation of electrical components. For example, the cane handle, which was designed with an ergonomic style, had to be somewhat simplified for electrical component purposes. The cane was ultimately built of several sections, each 3D printed for ease of design and production, and the electrical components were epoxied into sections of the cane that are designed to hold those components (See Appendices A-J). The cane also features a rubber base screwed into the load cell to support itself.

Testing Team

The testing team had devised three major tests that the cane will need to pass in order for it to be deemed a success. The first major test conducted was a drop test. The purpose of this test was to determine the durability of the cane, with the desired results being that the cane, and its components, were able to withstand a fall from at least 6 feet at varying positions: vertical, horizontal, and sideways. While it is by no means the definitive max height, it is assumed that in most falls the cane will not travel higher than the height of the user. Two details that were considered: first, that the 6 feet will be reached in increments of two feet, starting at two feet, and second, that falls will, ideally, occur over a hard surface, to replicate the most damaging conditions to the cane.

The Durability Drop Test was meant to expose many structural flaws evident in the cane. Perhaps certain connectors between pieces are not sufficient in remaining connected, or electrical components not fastened to the cane rigidly enough, or even the material being too fragile at certain points. All this is to say, the Drop test was meant to serve as a structural test and does not look at the electrical or programming components of the cane. Unfortunately section 4 was broken in the 4 feet height drop test and our testing process had to be revised with falls tested onto a soft surface instead with the understanding that this cane was a prototype and the materials were not ideal.

When the drop test was satisfied, the cane could move on to the second test, the Positive Feedback Test. The aim of this test is to create the conditions meant to satisfy a positive feedback from the code and incite the emergency message response. Knowing the cane simply reads for certain trends in acceleration and force, this test did not require a highly structured or controlled apparatus, contrary to our initial plan. Thus the cane simply needed to be accelerated while a force was applied on the head of the cane. To achieve this, the testing team performed a slightly controlled acceleration of the cane, tipping it over from a stationary upright position, while applying force just over the head of the cane. The application of force is meant to simulate a spike in force as the cane begins to accelerate towards the ground. While this was occurring, the sensors would be monitored for feedback, as well as our dummy phone number, which would be messaged instead of EMS services. To pass this test, the cane needed to return accurate readings on its accelerometer and force gauge, while simultaneously activating a positive feedback. It is worth noting that, as the cane developed, the Programming team's method for detecting a fall

evolved. Thus, it was the job of the testing team to work closely with the programming team to ensure this test was able to satisfy all requirements.

The third and final major test conducted was the Failsafe and False Positive Test. The purpose of this test was to determine whether or not the failsafe mechanism worked and how often the cane returned false positives. To do this, we began by following the same procedure in the Positive Feedback Test. However, once positive feedback had been established, we would attempt to activate the failsafe, which involves using the force gauge prior to make sure the cane was in use and if the cane had force applied after the fall. For the cane to pass this test, the failsafe simply needed to be activated and no message be sent to our dummy phone number. In this process, the testing team looked for potential areas of false positive readings, where the cane might detect a fall when there was not. These tests would be mostly informal, and involve the practical usage of the cane. We looked for examples in everyday usage where a false positive could be detected. Does walking too fast activate positive feedback? Does going up or down stairs activate positive feedback? If the cane is left alone and falls over, will there be positive feedback? These tests were critical in ensuring even the smallest of technicalities was fulfilled by the programming team's code, and required the rethinking of our failsafe mechanism. It should be known, these tests were not meant to eradicate any chances of a false positive appearing, but testing whether the failsafe mechanism was effective in addressing these false positives. While it is ideal for no false positives to occur, if the failsafe mechanism was truly effective, it shouldn’t matter how many false positives occur. In the actual testing process, the failsafe code was revised continuously before performing a measured testing process on the final iteration of code. This test was the true finale, ensuring all parts worked as intended in full harmony.

Final Design Analysis

The Drafting team drafted various parts of the cane in Inventor and Fusion360 and supplied these files to the 3D printers located in Jesser along with 3D printers utilized by other students in contact with Dr. Fitz. The parts were then sanded down to fit snugly. Fabrication was a collaboration of the Drafting team and the Biomedical/Mechanical team.

The Electrical team wired the various components based on their final schematic, placed them inside the cane and epoxied them and the parts of the cane together. However, the battery could not be installed, so the Electrical team created a usb cable attached to the ESP-32 that could be used to power the device.

The Programming team, after the cane was constructed, uploaded their Arduino code to the cane and supplied the code to be run on an external computer that would communicate and record data as well as detect falls.

The Testing team conducted their drop test first which resulted in a need for a redesign of the fourth section of the cane, in which the part attached to the load cell had its thickness increased. After the redesign, they assisted Programming with testing the fall detection algorithm and compiled the results. Finally, when Programming noticed the heart rate sensor was not accurate, the Testing team compared it to a Smart Watch to determine that the sensor purchased did not work as intended. Thus, the heart rate data in its current state with the current sensor should not be used in a rigorous medical setting.

	Actual	
Detection	Fall	No Fall
Fall	16	0
No Fall	0	16

Fig. 3. Results from the Fall Detection Testing

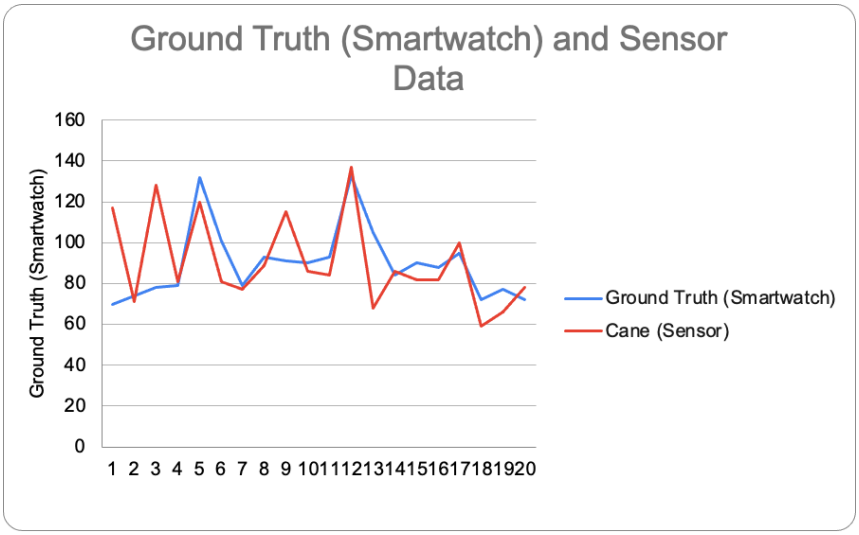


Fig. 4. Results from the Heart Rate Sensor Test

Final design drawings (revised)

Final design: see Appendix A-J: CAD Drawing Assembly Design and individual part drawings

Revised Base section 4:

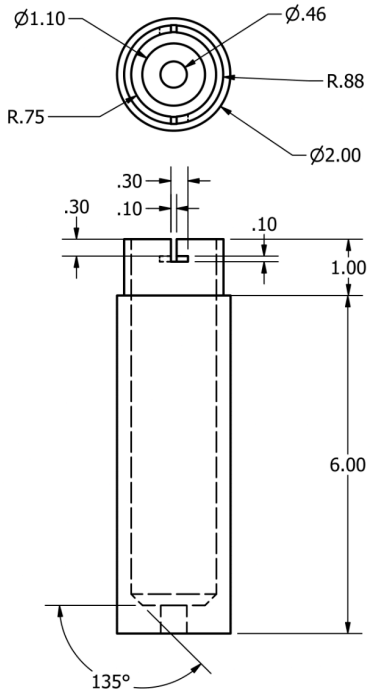


Fig. 5. section four

Manufacturing Approach

In order to have full control over the process of manufacturing, the Biomedical/Mechanical team decided to 3D print the pieces of the cane. However, some parts had to be store bought such as the cane tip and bolt. The electrical parts were all ordered online individually then connected and soldered together. This way we had a lot of control over the manufacturing process in the event that something would have to be redesigned. The programming was all done in a Python IDE and the Arduino IDE.

Fabrication

The designs made by the Drafting team were mostly able to be printed prior to Thanksgiving break with some parts printed during the break. The parts were printed by both student printers and the printers provided to us in Jesser 235. This process incurred some printing failures, however, the parts were able to be finished by the end of break.

The Programming team was also able to test sections of their code prior to Thanksgiving break. These sections were the email sender, text messaging code, and user input text service which all worked with no problems.

After Thanksgiving break, the circuit was wired together by the Electrical team enabling the Programming team to test the Arduino and bluetooth communication code.

The Programming team, once the circuit was wired, promptly began testing the code and referenced the schematic of the circuit to call for correct pin numbers depending on the sensor. The Arduino code worked for the most part, other than parts of the schematic being outdated and the need to reference the ESP32 unit itself. The bluetooth communication code ran and worked with slight adjustments to print only necessary data because of miscellaneous characters printed alongside the crucial data. In this testing process, the load cell was also calibrated using 2.5 lb and 5 lb weights from the Aquatic Fitness Center.

With the Arduino code working, the Electrical team rewired the circuit inside the cane's printed sections and epoxied them together, creating a single unit. With the cane itself being completed, the Programming team with the Testing team began testing other sections of code including the fall detection and graphing.

The fall detection code initially consisted of two variations of code. One detected the orientation of the cane using the accelerometers inside the cane. The other detected for a degree of chaos in the accelerometer data that would determine movement of the cane and detect a fall. However, these both proved to be insufficient when the Testing team began testing the Cane's fall detection alongside the Programming team.

The fall detection code was then refined to combine aspects of both variations and include other checks that would ensure that no false positives were detected and that all genuine falls were accounted for. The code was refined by revising and adding several conditionals. The code was revised by changing the amount of data points that were checked and by checking for

the force present on the cane prior to the fall to ensure it was in use during the fall. It was also refined by checking for force after the fall for fifteen seconds to check if the user has picked up the cane and is not in a critical state. The final check determines if the cane was still or if its orientation met a certain threshold. The fall would only be confirmed if it was able to clear all the conditionals.

Experiences

No design process is without failures and the *Companion Cane* was no different. During the 3D printing process, our slicing software created inefficient auto supports, which created the mess seen in Figure 6. Additionally, the back handle end cap had issues printing because the dimensions did not convert correctly. The hole was not big enough to fit the USB-C port. However, because the booster converter did not work, the USB charging could not work. This is why the handle does not have end caps because it needs the wires hanging out for power. If the cane were to be manufactured, the final product would have end caps and a USB charging port. The issue with the boost converter resulted in having to use a different battery. This required a redesign for the battery holder because it needed to be able to fit the new battery, which had different dimensions. Overall, the 3D printing process took a long time, so that caused set backs in our timeline for fabrication.

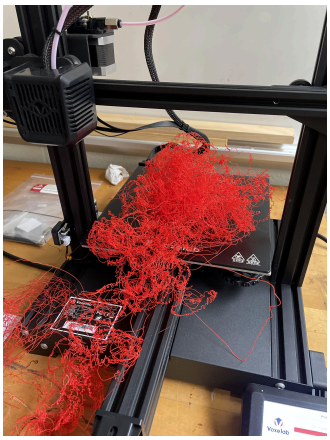


Fig. 6. Result of Section 4 having inefficient auto supports

While these issues did not require a total redesign, they still created some setbacks in the fabrication process. First, the sections were not given enough tolerance. To fix this, all of the sections were sanded down. Secondly, the pegs that are part of the locking mechanism were not big enough to be helpful in making sure the sections stayed together. The pegs could easily break off. However, we fixed this issue with some epoxy to make sure the cane was strong and sturdy. Thirdly, the bolts purchased initially for the load cell were the wrong size, so the correct size had to be found from the store.

During the testing phase, yet another redesign was required. When the cane was dropped

horizontally from 4 feet in the air onto a hard surface, section four broke as seen in Figure 7. While there were some clear errors in the test conducted, section four was still redesigned to have a thicker base as shown in Figure 8.

Fig. 7. Section four broken after drop test

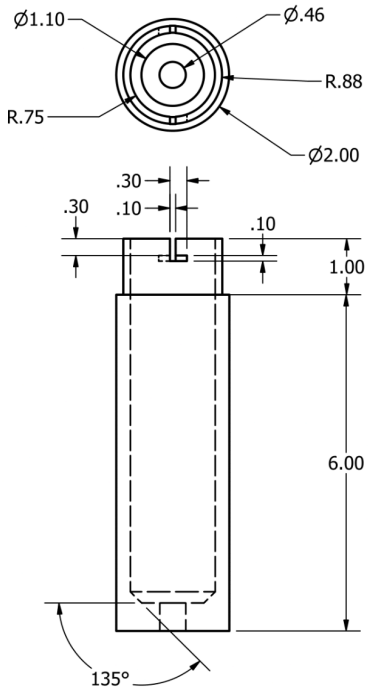


Fig. 8. CAD drawing of the redesigned section four

Cost

Rubber Cane Tips (2.5” Diameter)	**$6.38**
Rubber Cane Tips (2.5” Diameter Black)	**$7.99**
Two 1.75mm 3D Printer Red Filament	**$24.99**
Two 1.75mm 3D Printer Blue Filament	**$18.99**
Pulse Sensor	**$24.99**
Lithium Battery Charging Board	**$6.89**
Three ADXL345 Digital Accelerometer	**$20.97**
XL6009 Boost Converter	**$5.89**
ESP32 Arduino	**$19.88**
HX711 Load Cell Interpreter	**$8.98**
25R Samsung Battery	**$7.77**
TP4056 Charger	**$5.99**
Load Cell	**$59.95**
KCD3 Switches	**$0.54**
Twilio Account	**$20.00**
Mxuteuk Snap-in Round	**$12.09**
Total	**$296.27**

Cost of Production

Similar canes on the market currently that utilize sensors and gauges to determine the habits of the user can go for up to $1000 (Barrett, 2020). The *Companion Cane* is not at the same level of quality as a market-ready item, but it is a viable prototype that was successful in testing and could theoretically be sold in its current state. This would be less than ideal as there are still many improvements that can be made to the design to increase the effectiveness of the cane. However, these improvements would increase the cost of the *Companion Cane*, so a viable balance must be found. The unit cost of production of the *Companion Cane* in its current state would most likely be around $200 dollars by buying products such as filament and the electronic components in bulk. This unit cost could very easily increase if the final design includes an aluminum body or more accurate electronics.

Production

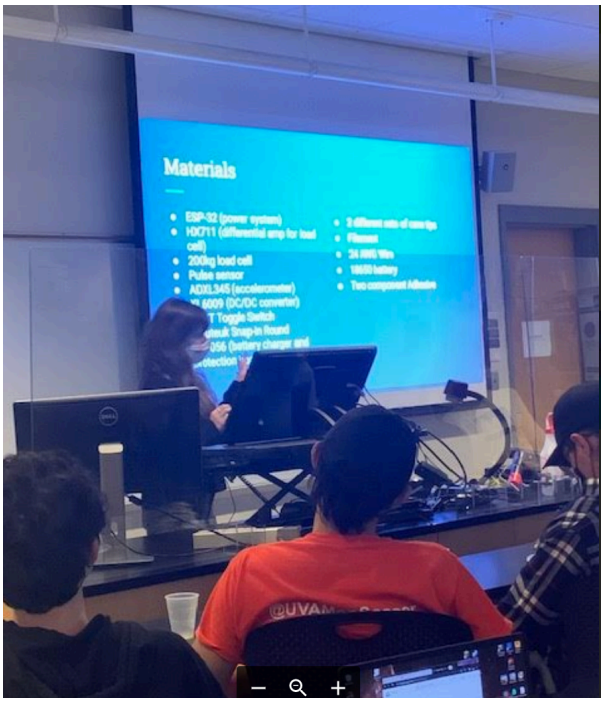
To make the *Companion Cane* on a large scale, the overall design would have to be changed to fit the needs of a mass production scale. For instance, 3D printing might not be the best material long term. Originally, the Biomedical/Mechanical team suggested making the cane out of PVC or Aluminum. However, the prototype was made from 3D printing because this material gave us the most independence and control over the process. Long term, plastic might not be the most durable material. Materials would have to be rigorously tested to see which ones support the most weight and which ones are the most sturdy before production.

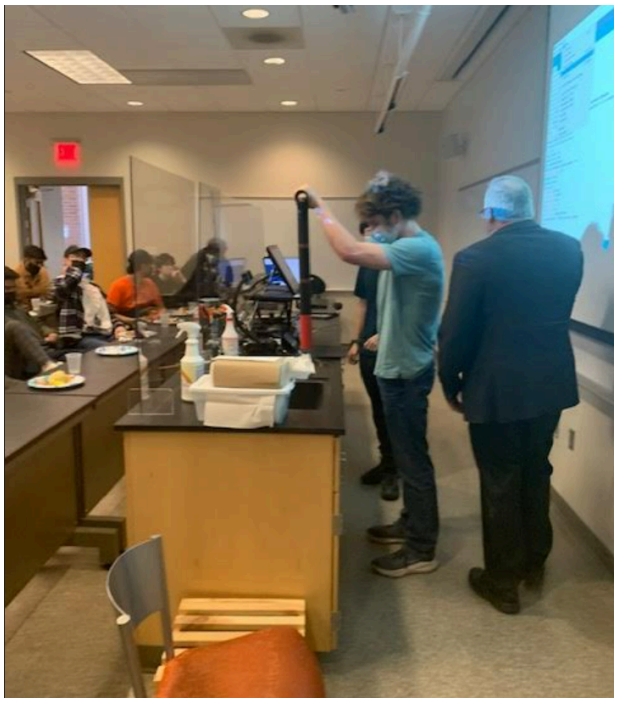
Also, we had many issues with electrical components not working as intended. Quality control would be very important in the production of the *Companion Cane*. Again, this would require a more rigorous testing process than what we conducted on the prototype.

Prototype Demonstration and Presentation

Available at this link:

https://youtu.be/Beia0n57x1g





Installation, Use, and Maintenance

The *Companion Cane*, if it does not already have the Arduino code uploaded to it, requires the code to be uploaded through the ESP32's port by separating the top portion of the cane from the body revealing the port. The code can be downloaded from our github(https://github.com/BAltenburger/CompanionCane-) and will require that the ESP32 library is installed on the Arduino IDE along with the code libraries, Adafruit ADXL345, Adafruit Unified Sensor, HX711, and PulseSensor Playground.

The text service setup requires a twilio account, the 'twilio' library installed on the preferred python IDE, and the ngrok console. To run the code, first open the ngrok console and type "ngrok http 5000" and copy the first link listed and paste it to the messaging webhook of the number registered on twilio (twilio console>#Phone Numbers>manage>active numbers>(your number)). After opening the port 5000 connection on the computer and connecting it to the twilio number, The code can be ran and the twilio number can be texted the authentication code which is set to '1234' and will begin the text response chain to receive user information that is used in other sections of the code. The user inputs can also be manually created by making text files for each input listed in the comments of the 'app.py' code and placing them inside the folder that has the python programs.

With the user inputs set, the Cane must be powered on and the computer can connect to the cane through bluetooth by connecting to the device named ESP32. With the bluetooth connection established, the serial port for the bluetooth connection must be determined and can be seen in 'more bluetooth options' on windows computers. Then, under 'COM ports' identify the COM number associated with the outgoing connection with the ESP32. In the 'main.py' code, adjust the port number in line 147 of the code to match what was identified earlier.

```
147        arduino=serial.Serial('COM6',115200)
```

The bluetooth is now set up and the cane can be powered on and the 'main.py' code can be run after the cane has been powered.

Also, after making a twilio account and upgrading the account, adjust the 'account_sid' and 'auth_token' to match those that are on the twilio console page.

For use, make sure to turn off the cane, when it is not in use and to restart the 'main.py' code on the computer when turning it back on because the code will print an error message

because of the bluetooth disconnection. When placing the cane down briefly, make sure to not put any force on the cane for at least 5 seconds to ensure that the cane does not give any false positives and mistakenly contacts emergency contacts.

For maintenance, the raw data from the sensors can be viewed by using the Arduino IDE and its serial monitor. This can be used to ensure that all sensors are working properly by monitoring the live output.

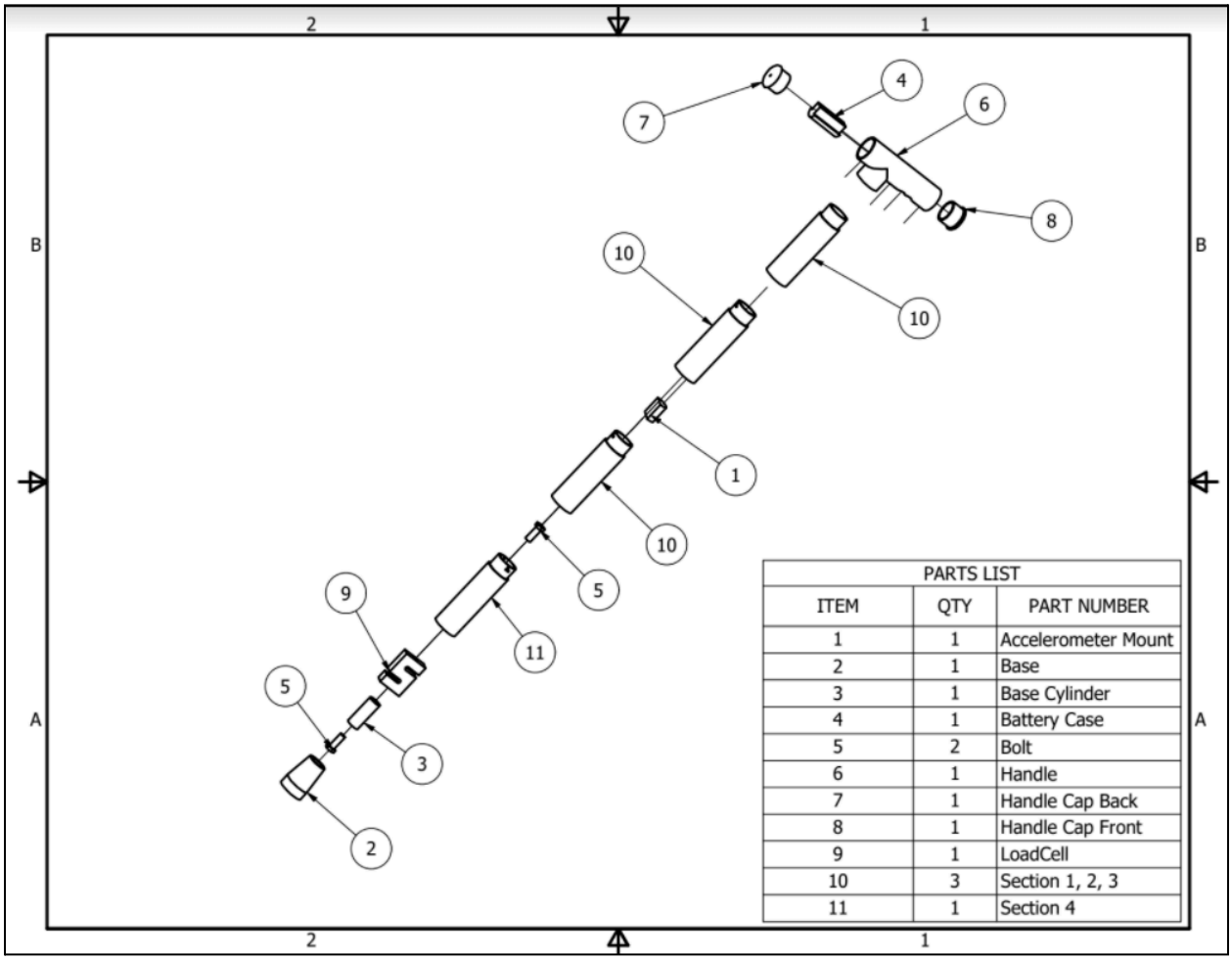
Bibliography

Barrett, P. (2020, September 12). *Force walking cane - intelligently monitors gait pattern: Biofeedback*. nCounters Engineering Biofeedback Rehabilitation Products. Retrieved December 17, 2021, from https://ncountersonline.com/force-cane

2007 Trends in Trauma and Emergency Medicine. Virginia Department of Health, https://www.vdh.virginia.gov/content/uploads/sites/23/2016/06/2007_Trends.pdf.

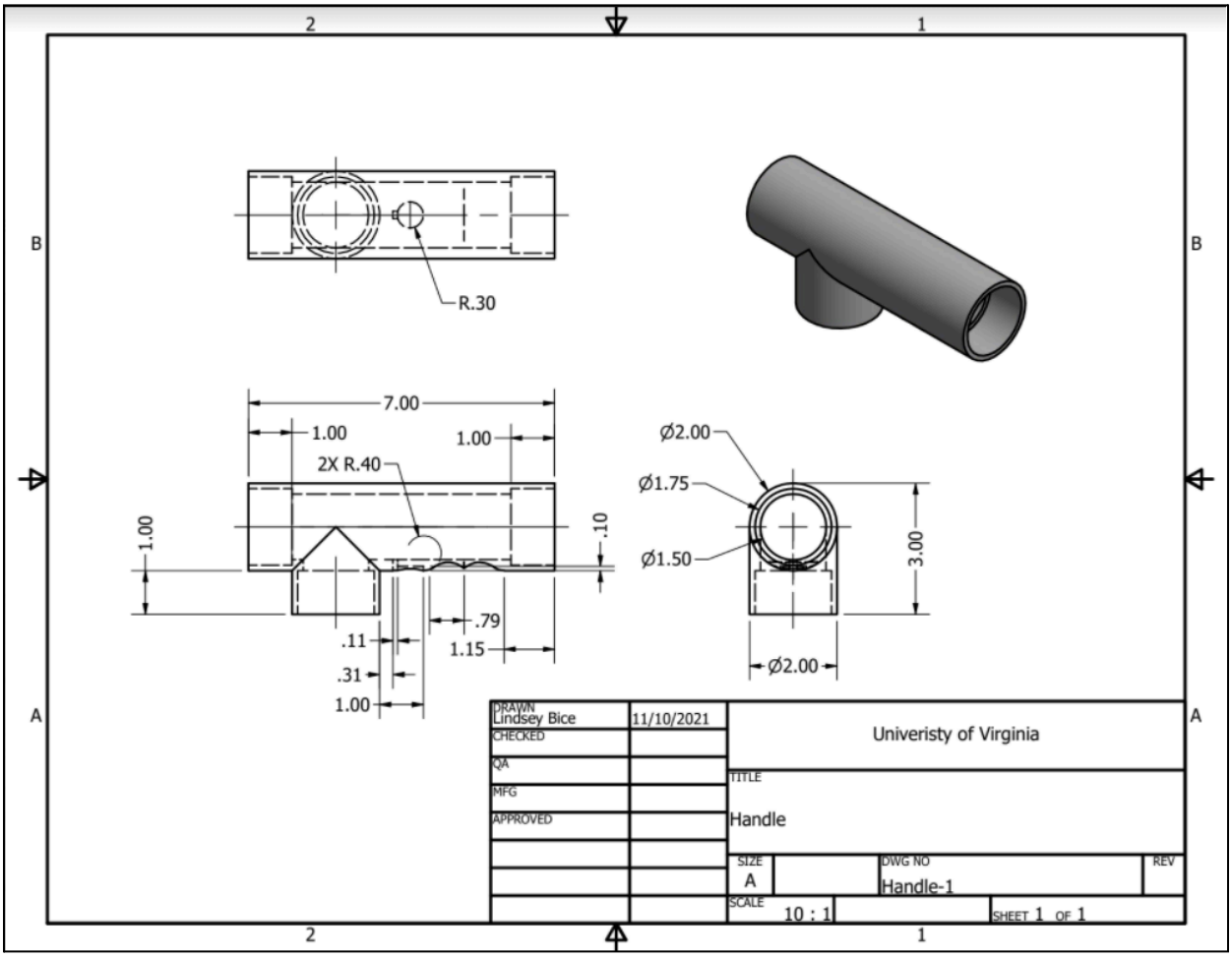
"Older Adult Falls." *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 14 July 2021, https://www.cdc.gov/falls/index.html.

Appendix A: Assembly Design

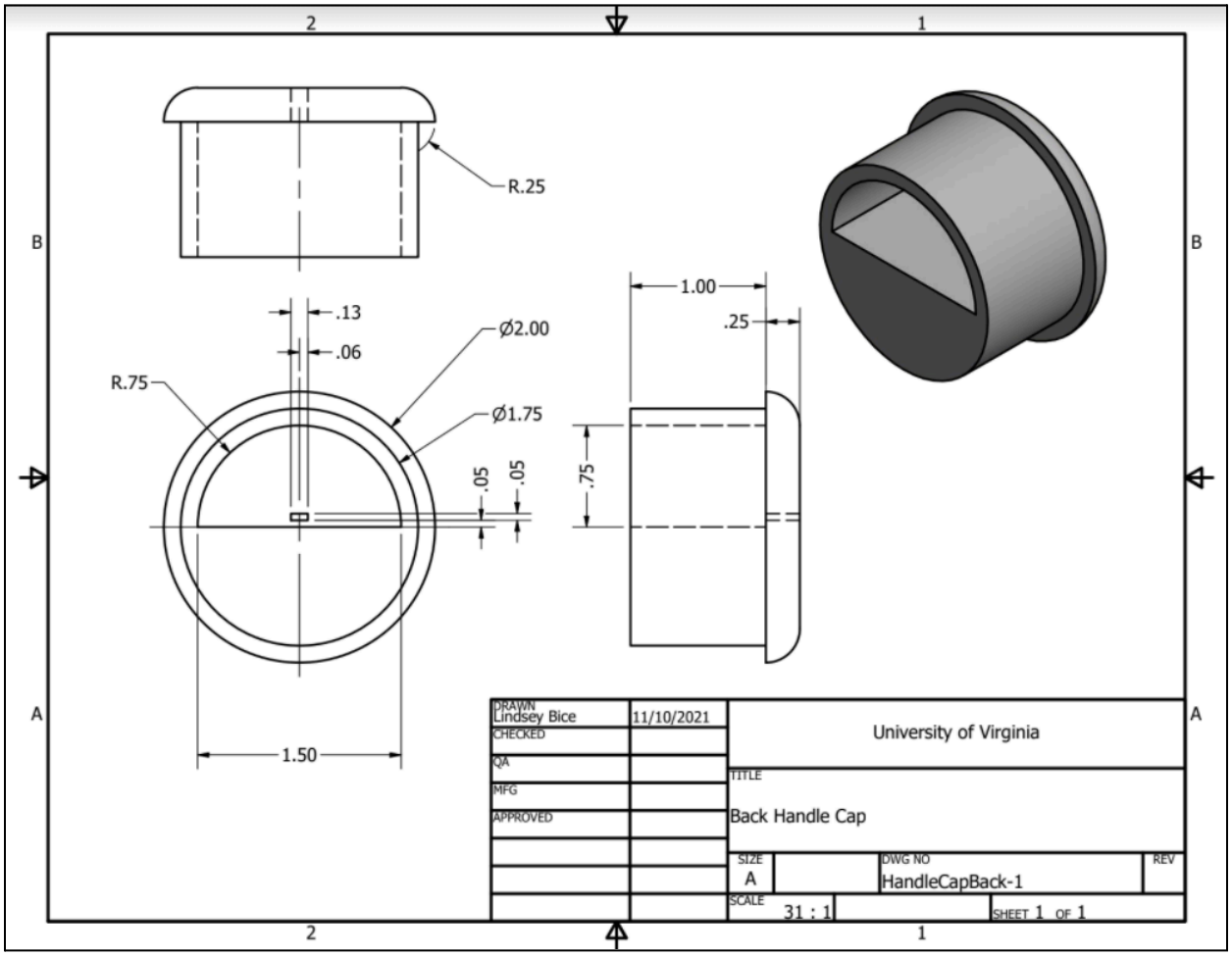


PARTS LIST		
ITEM	QTY	PART NUMBER
1	1	Accelerometer Mount
2	1	Base
3	1	Base Cylinder
4	1	Battery Case
5	2	Bolt
6	1	Handle
7	1	Handle Cap Back
8	1	Handle Cap Front
9	1	LoadCell
10	3	Section 1, 2, 3
11	1	Section 4

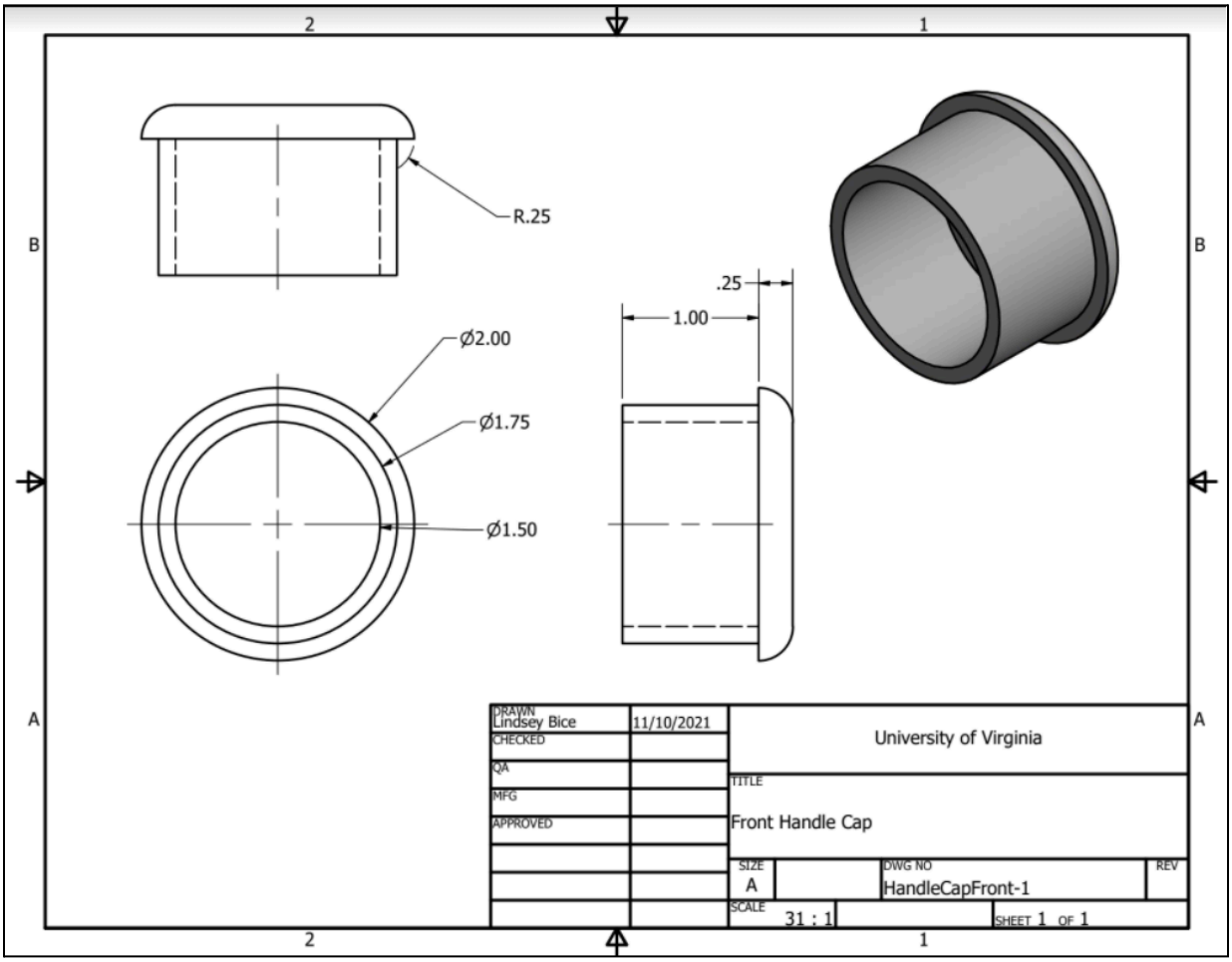
Appendix B: Handle



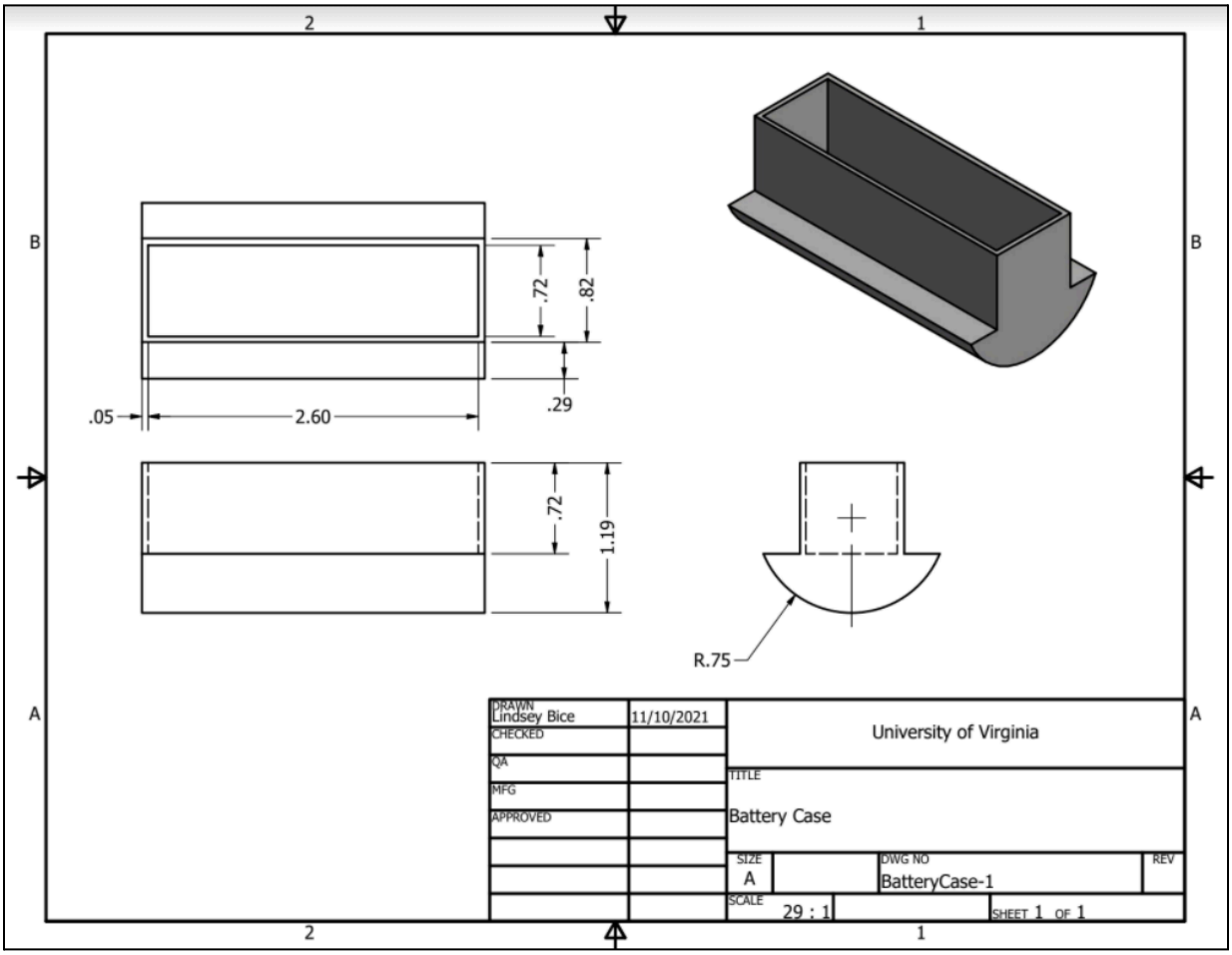
Appendix C: End Cap 1



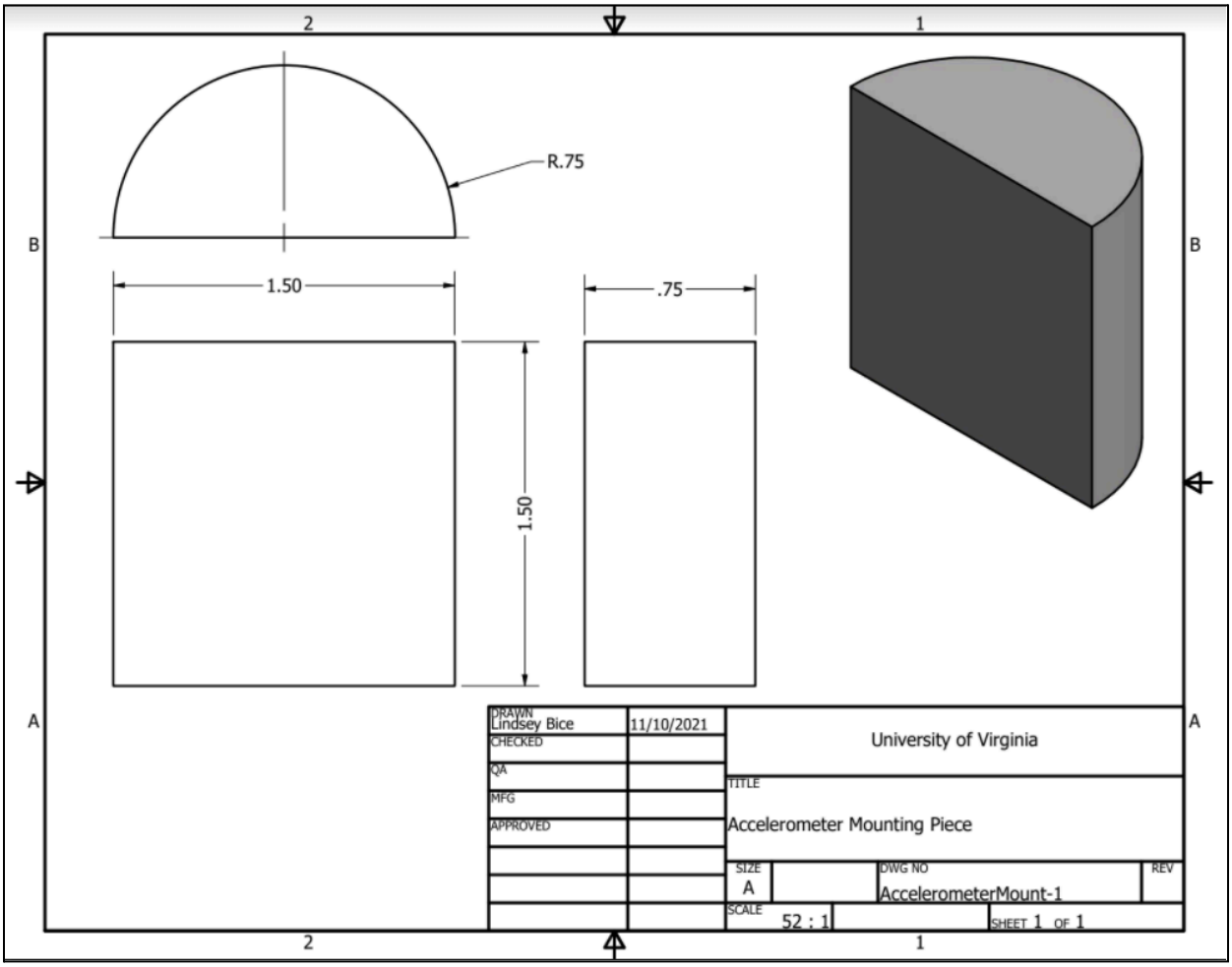
Appendix D: End Cap 2



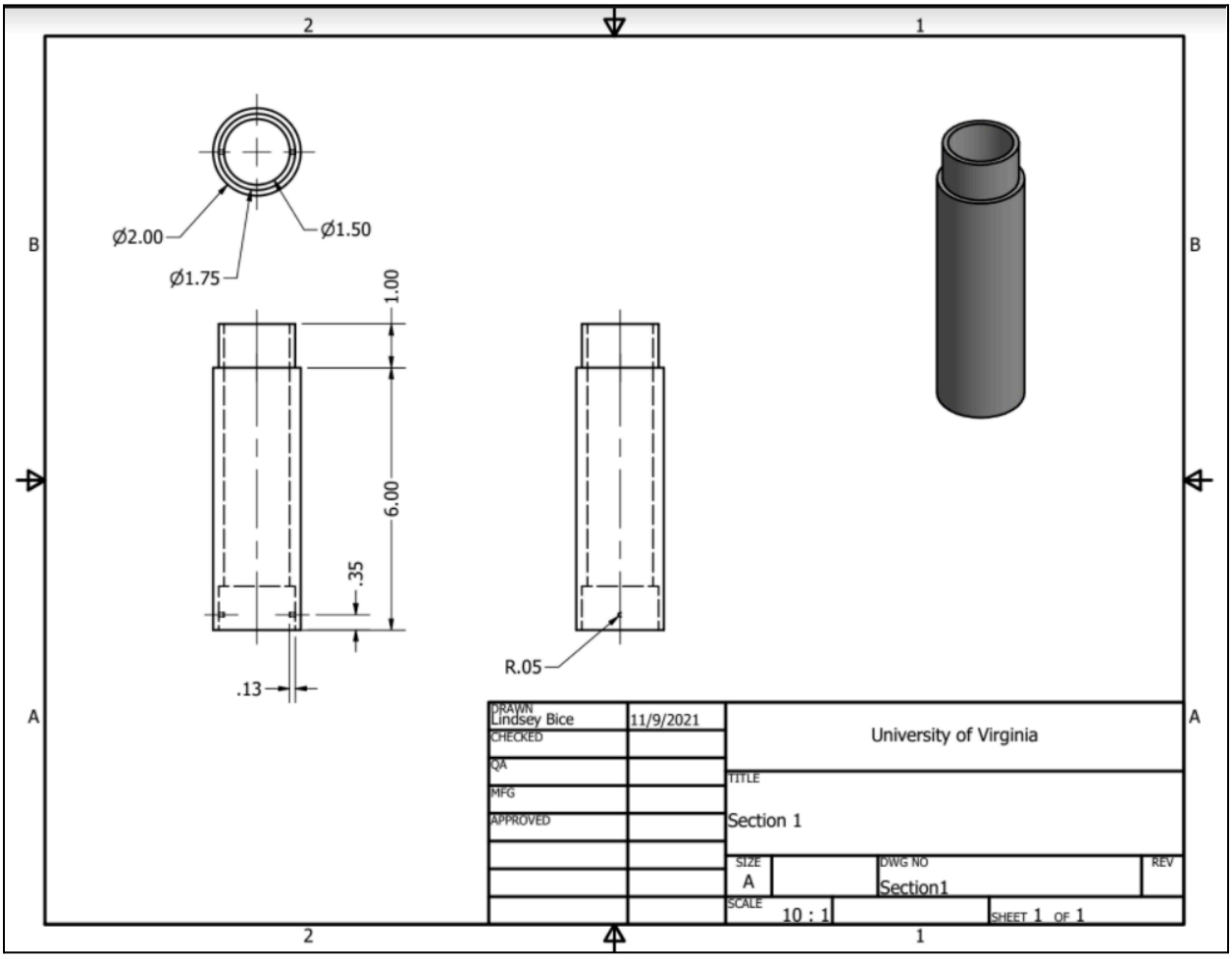
Appendix E: Battery Holder



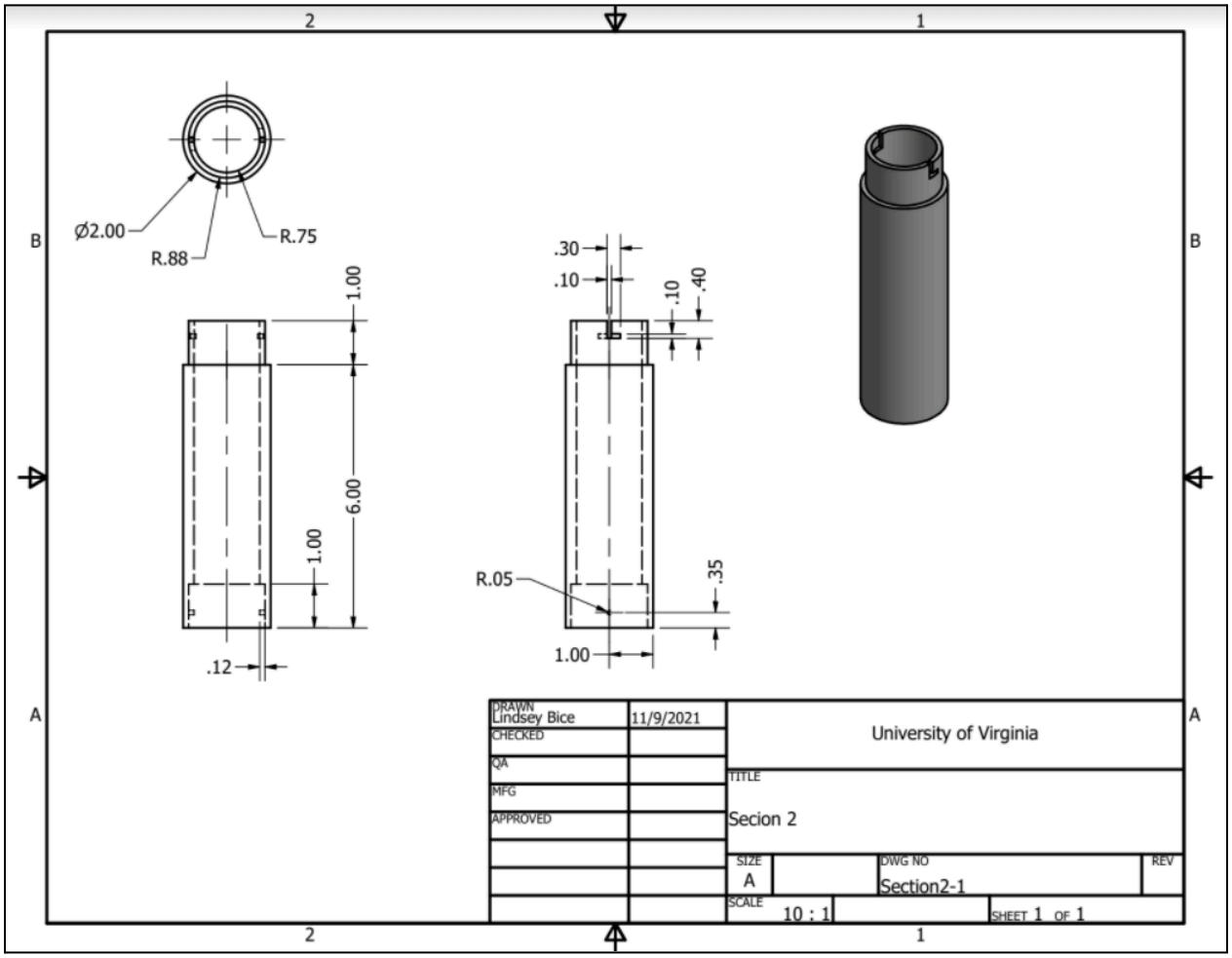
Appendix F: Attachment Piece



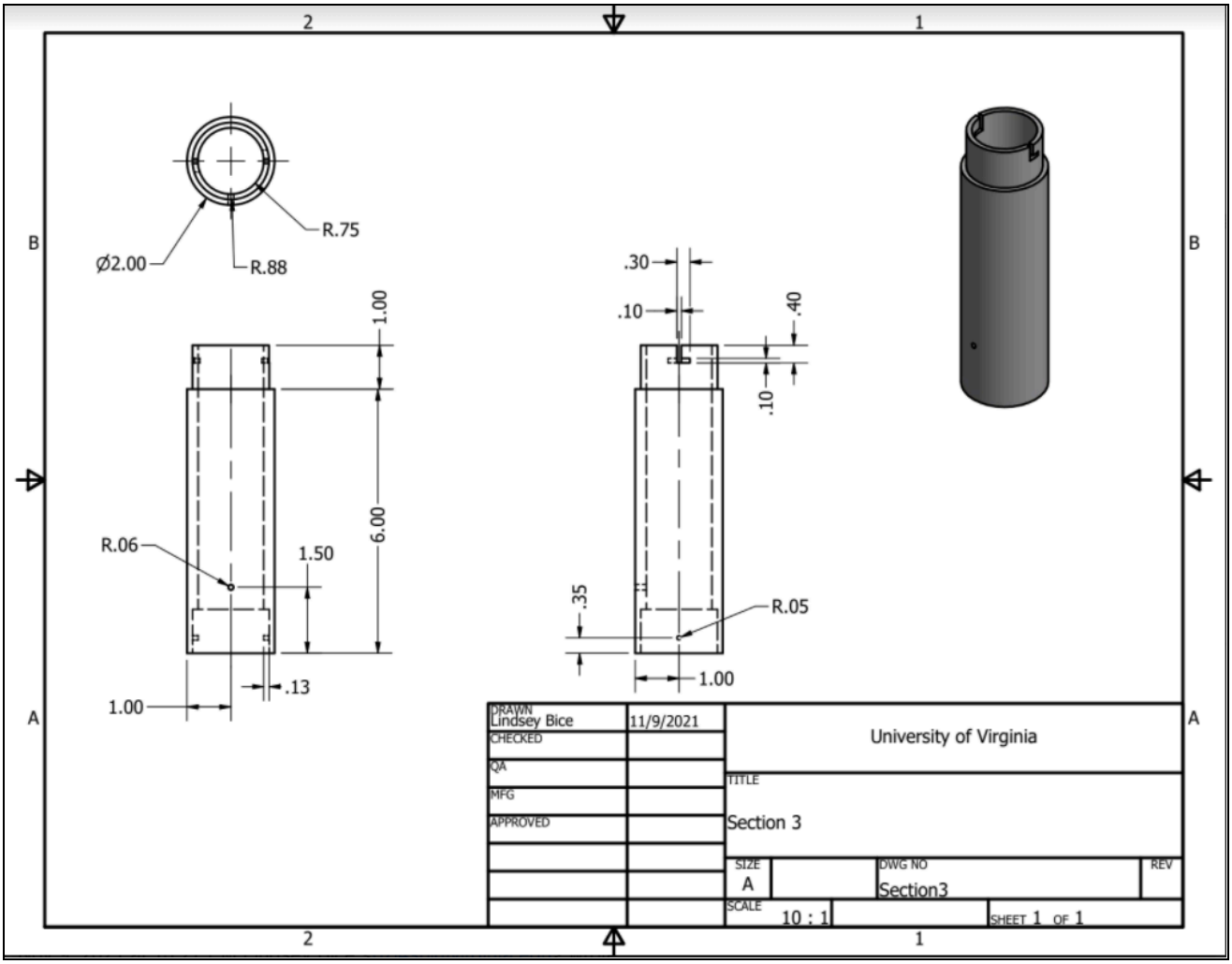
Appendix G: Base Section 1



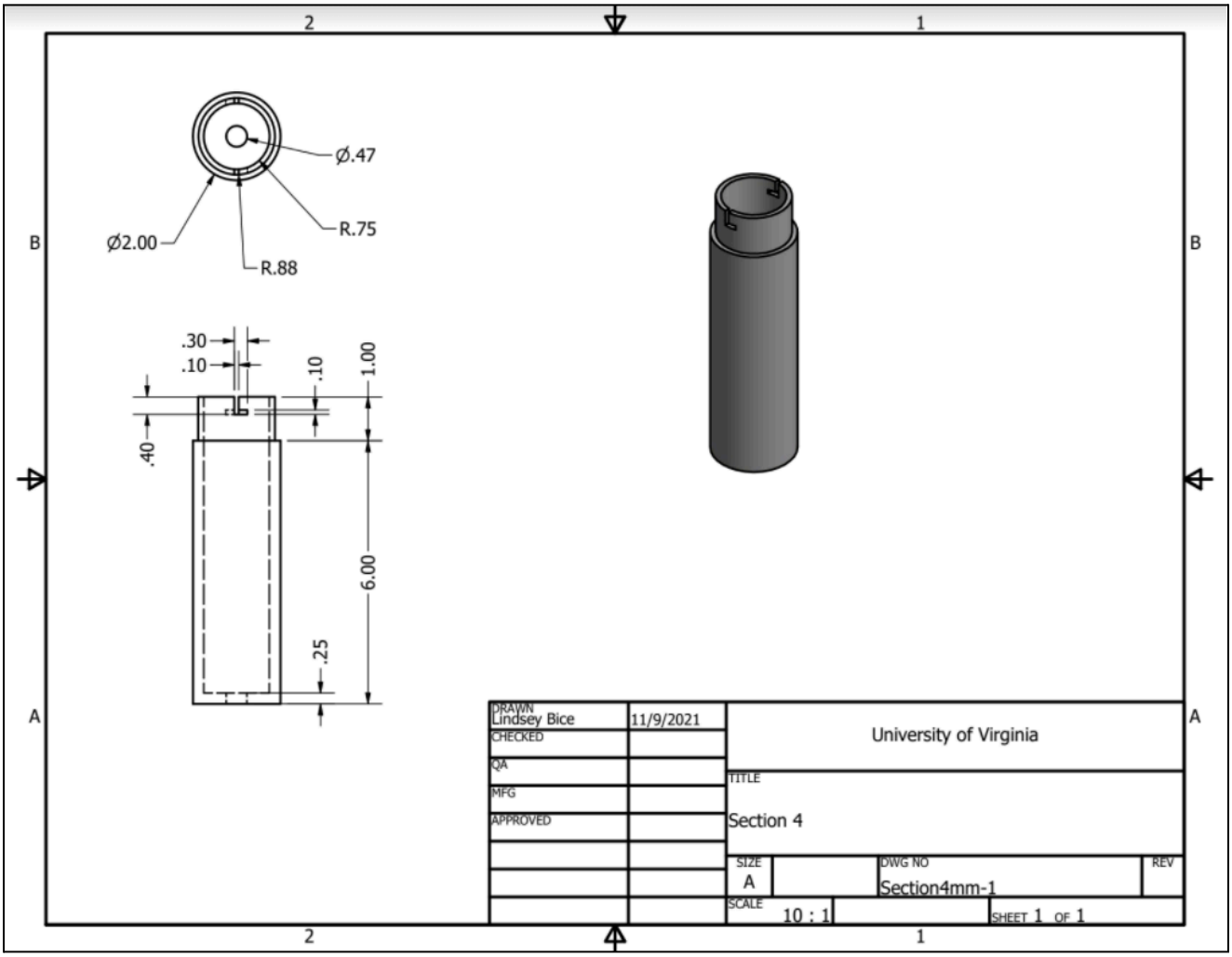
Appendix H: Base Section 2



Appendix I: Base Section 3



Appendix J: Base Section 4



Appendix K: Arduino Code

Compcane.ino

```
#define USE_ARDUINO_INTERRUPTS false
#include <PulseSensorPlayground.h>
#include "BluetoothSerial.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include "HX711.h"
#define calibration_factor -10000.0
#define DOUT 4
#define CLK 2
#if !defined(CONFIG_BT_ENABLED) || ! defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run 'make menuconfig' to and enable it
#endif

BluetoothSerial SerialBT;
Adafruit_ADXL345_Unified a1(0x53);
HX711 lc;
PulseSensorPlayground pS;

const int pin = 34;
const int th = 550;
byte samplesUntilReport;
const byte SAMPLES_PER_SERIAL_SAMPLE = 10;
unsigned long cm;
unsigned long sm;

void setup(){
  Serial.begin(115200);
  SerialBT.begin("ESP32");

  pS.analogInput(pin);
  pS.setSerial(Serial);
  pS.setThreshold(th);
  pS.setOutputType(SERIAL_PLOTTER);
  samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE;

  pS.begin();

  a1.begin();
  a1.setRange(ADXL345_RANGE_4_G);

  lc.begin(DOUT, CLK);
  lc.set_scale(calibration_factor);
```

```
  lc.tare();

  sm = millis();
}
void loop(){
  cm = millis();

  sensors_event_t event;
  int bpm = pS.getBeatsPerMinute();

  if(Serial.available()){
    SerialBT.write(Serial.read());
  }
  if(SerialBT.available()){
    Serial.write(SerialBT.read());
  }

  if(pS.sawNewSample()){
    if(--samplesUntilReport == (byte) 0){
      samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE;
      pS.outputSample();
      if(pS.sawStartOfBeat()){
        pS.outputBeat();
      }
    }
    if(cm - sm>= 50){
      SerialBT.print(bpm);
      SerialBT.print(", ");
      a1.getEvent(&event);
      SerialBT.print(event.acceleration.x);
      SerialBT.print(", ");
      SerialBT.print(event.acceleration.y);
      SerialBT.print(", ");
      SerialBT.print(event.acceleration.z);
      SerialBT.print(", ");
      SerialBT.println(lc.get_units(), 1);
      sm = cm;
    }
  }
}
```

Appendix L: app.py

```
from flask import Flask, request
from twilio.twiml.messaging_response import Message, MessagingResponse
from os.path import exists

app = Flask(__name__)
def clear(user_number):
    open(user_number + "service_order.txt", "w").write("")
    open("user_age.txt", "w").write("")
    open("user_number.txt", "w").write("")
    open("user_gender.txt", "w").write("")
    open("user_weight.txt", "w").write("")
    open("emergency_contacts1.txt", "w").write("")
    open("physician_contact.txt", "w").write("")
    open("physician_email.txt", "w").write("")
    open("user_preference.txt", "w").write("")

@app.route('/sms', methods=['POST'])
def sms():
    def user_data(filetype, user):
        file = open(filetype, "w")
        file.write(user)

    def read_data(filetype):
        return open(filetype, "r").read()

    number = request.form['From']
    message_body = request.form['Body']
    resp = MessagingResponse()
    message = str(message_body)
    if message_body == "1234" and (not exists(number + "service_order.txt") or
read_data(number + "service_order.txt") == "0"):
        user_data("user_number.txt", str(number))
        user_data(number + "service_order.txt", "1")
        resp.message('Welcome to our Companion Cane Service\nWhat is your age? (message back
with "age:(your age)"')
    elif not exists("user_number.txt"):
        resp.message('Your number is not registered with our service')
    elif read_data("user_number.txt") == str(number) and message[0:4] == "age:" and
read_data(number + "service_order.txt") == "1":
        if not message[4:len(message)].isdigit():
            resp.message('Make sure to enter an integer for your age.')
        elif int(message[4:len(message)]) < 10 or int(message[4:len(message)]) > 130:
            resp.message('Make sure to enter an age is between 10 and 120')
        else:
            user_data("user_age.txt", message[4:len(message)])
```

```
            resp.message('What is your gender? (message back with "gender:(gender)"')
            open(number + "service_order.txt", "w").write("2")
    elif read_data("user_number.txt") == str(number) and message[0:7] == "gender:" and
read_data(number + "service_order.txt") == "2":
        user_data("user_gender.txt", message[7:len(message)])
        resp.message('What is your weight? (message back with "weight:(weight in lbs)"')
        open(number + "service_order.txt", "w").write("3")
    elif read_data("user_number.txt") == str(number) and message[0:7] == "weight:" and
read_data(number + "service_order.txt") == "3":
        try:
            float(message[7:len(message)])
            if float(message[7:len(message)]) > 40 and float(message[7:len(message)]) < 500:
                user_data("user_weight.txt", message[7:len(message)])
                resp.message('What is the number of your emergency contact? (message back with
"EC:(#))"')
                open(number + "service_order.txt", "w").write("4")
            else:
                resp.message('Make sure to enter your proper weight between 40 and 500 lbs.')
        except ValueError:
            resp.message('Make sure to enter a proper float value for your weight.')
    elif read_data("user_number.txt") == str(number) and message[0:3] == "EC:" and
read_data(number + "service_order.txt") == "4":
        try:
            int(message[3:len(message)])
            if len(message[3:len(message)]) == 11 or len(message[3:len(message)]) == 10:
                user_data("emergency_contacts1.txt", message[3:len(message)])
                resp.message('What is the number of your physician? (message back with "PC:(#)"')
                open(number + "service_order.txt", "w").write("5")
            else:
                resp.message('Make sure to enter a proper 10 or 11 digit phone number.')
        except ValueError:
            resp.message('Make sure to enter only numbers for the phone number along with the
"EC:".')
    elif read_data("user_number.txt") == str(number) and message[0:3] == "PC:" and
read_data(number + "service_order.txt") == "5":
        try:
            int(message[3:len(message)])
            if len(message[3:len(message)]) == 11 or len(message[3:len(message)]) == 10:
                user_data("physician_contact.txt", message[3:len(message)])
                resp.message('What is the email of your physician? (message back with
"email:(email)"')
                open(number + "service_order.txt", "w").write("6")
            else:
                resp.message('Make sure to enter a proper 10 or 11 digit phone number.')
        except ValueError:
```

```
            resp.message('Make sure to enter only numbers for the phone number along with the
"PC:".')
    elif read_data("user_number.txt") == str(number) and message[0:6] == "email:" and
read_data(number + "service_order.txt") == "6":
        if message.find("@") == -1:
            resp.message('Make sure to properly format the email with an @ symbol.')
        else:
            user_data("physician_email.txt", message[6:len(message)])
            resp.message('Thank you, your information has been registered.\nIf you would like to
prevent emergency calls, message back with "STOP". If you would like to reset your information
message back with "reset"')
            open("service_order.txt", "w").write("7")
    elif read_data("user_number.txt") == str(number) and message[0:4] == "STOP" and
read_data(number + "service_order.txt") == "7":
        user_data("user_preference.txt", str(message))
        resp.message('Your emergency contact preferences have been updated')
        open("service_order.txt", "w").write("8")
    elif read_data("user_number.txt") == str(number) and message == "reset":
        clear(number)
        resp.message('Your information has been reset')
        open(number + "service_order.txt", "w").write("1")
    elif read_data("user_number.txt") == str(number):
        resp.message('make sure you formatted your message correctly or that you provide
information in the correct order. your message:' + message)
    return str(resp)


if __name__ == '__main__':
    app.run()
```

Appendix M: emailsender.py

```
import email, smtplib, ssl
from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

def emaildaily():
    file = "daily_time_heart_rate.png"
    subject = "health data"
    body = "daily heart rate"
    sender_email = "eptprogramming2021@gmail.com"
    receiver_email = open("physician_email.txt", 'r').read()
    password = "programmingEPT2021"

    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message["Bcc"] = receiver_email
    message.attach(MIMEText(body, "plain"))
    with open(file, "rb") as attachment:
        part = MIMEBase("application", "octet-stream")
        part.set_payload(attachment.read())
    encoders.encode_base64(part)
    part.add_header(
        "Content-Disposition",
        f"attachment; filename= {file}",
        )
    message.attach(part)
    text = message.as_string()

    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, text)


def emailweek():
    file = "weekly_heart_rate.png"
    subject = "Health data"
    body = "weekly heart rate"
    sender_email = "eptprogramming2021@gmail.com"
    receiver_email = open("physician_email.txt", 'r').read()
    password = "programmingEPT2021"
```

```
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message["Bcc"] = receiver_email
    message.attach(MIMEText(body, "plain"))
    with open(file, "rb") as attachment:
        part = MIMEBase("application", "octet-stream")
        part.set_payload(attachment.read())
    encoders.encode_base64(part)
    part.add_header(
        "Content-Disposition",
        f"attachment; filename= {file}",
        )
    message.attach(part)
    text = message.as_string()

    context = ssl.create_default_context()
    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, text)


def emailtool(message):
    files = ["data.csv", "all_time_heart_rate.png", “daily_time_heart_rate.png”,
“weekly_heart_rate.png”, "ranges_heart_rate.png",
            "three_line_plot_accelerometer.png", "three_dimension_plot_accelerometer.png",
"force_plot.png", "data.csv"]
    subject = "Health data"
    body = message
    sender_email = "eptprogramming2021@gmail.com"
    receiver_email = open("physician_email.txt", 'r').read()
    password = "programmingEPT2021"

    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = subject
    message["Bcc"] = receiver_email
    message.attach(MIMEText(body, "plain"))
    for file in files:
        with open(file, "rb") as attachment:
            part = MIMEBase("application", "octet-stream")
            part.set_payload(attachment.read())
        encoders.encode_base64(part)
        part.add_header(
```

```
        "Content-Disposition",
        f"attachment; filename= {file}",
        )
    message.attach(part)
    text = message.as_string()

context = ssl.create_default_context()
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, text)
```

Appendix N: sms.py

```
from twilio.rest import Client

def sms(message):
    account_sid = 'ACc311cdf474fb9bad277dfb2bf1872ed0'
    auth_token = '436c27e335f9febebe8860848e53a366'
    client = Client(account_sid, auth_token)

    message1 = client.messages \
                .create(
                    body=message,
                    from_='+15204127054',
                    to=str(open("emergency_contacts1.txt", "r").read())
                    )
    message2 = client.messages \
                .create(
                    body = message,
                    from_='+15204127054',
                    to=str(open("physician_contact.txt", "r").read())
                    )
    message1
    message2
```

Appendix O: graphing.py

```
import pandas as pd
from matplotlib import pyplot as plt
import datetime
from collections import Counter

"""
time    heart rate      accelerometer1x      accelerometer1y      accelerometer1z
accelerometer1z      accelrometer2x      accelerometer2y      accelerometer2z      strain
gauge

"""

def plot_today(heart_rate):
    # plots the data for today
    now = datetime.date.today()
    end_of_day = now - datetime.timedelta(hours=24)
    now = now.strftime("%d/%m/%Y %H:%M:%S")
    end_of_day = end_of_day.strftime("%d/%m/%Y %H:%M:%S")


    heart_rate = heart_rate[(heart_rate['time'] > end_of_day) & (heart_rate['time'] < now)]
    plt.plot(heart_rate["time"], heart_rate[" HR"])
    plt.xlabel("Time")
    plt.ylabel("Heart Rate")
    plt.title("Heart Rate Data over Time")
    plt.savefig("daily_time_heart_rate.png")
    plt.close()


def plot_week(heart_rate):
    # plots the data for the week
    now = datetime.date.today()
    end_of_week = now - datetime.timedelta(hours=168)
    now = now.strftime("%d/%m/%Y %H:%M:%S")
    end_of_week = end_of_week.strftime("%d/%m/%Y %H:%M:%S")

    heart_rate = heart_rate[(heart_rate['time'] > end_of_week) & (heart_rate['time'] < now)]
    plt.plot(heart_rate["time"], heart_rate[" HR"])
    plt.xlabel("Time")
    plt.ylabel("Heart Rate")
    plt.title("Heart Rate Data over Time")
    plt.savefig("weekly_heart_rate.png")
    plt.close()
```

```
def heart_rate_graphing(heart_rate, filename):

    # plots the all time heart rate data over time
    plt.plot(heart_rate["time"], heart_rate[" HR"])
    plt.xlabel("Time")
    plt.ylabel("Heart Rate")
    plt.title("Heart Rate Data over Time")
    plt.savefig("all_time_heart_rate.png")

    plot_today(heart_rate)
    plot_week(heart_rate)


    #checks to see if there are irregularities
    age = 19                # open(“user_age.txt”, ‘r’).read()
    level = []
    for i in heart_rate[" HR"]:
        if i >= (220 - age) * 0.93:
            level.append("Dangerous")
        elif ((220 - age) * 0.77) < i < ((220- age) * 0.93):
            level.append("High")
        elif  ((220- age) * .64) < i < ((220 - age) * 0.77):
            level.append("Medium")
        else:
            level.append("Normal")

    heart_rate['Level'] = level

    heart_rate.to_csv("Levels.csv")
    levels = pd.read_csv(filename)



    letter_counts = Counter(level)
    levels = pd.DataFrame.from_dict(letter_counts, orient='index')
    levels.plot(kind='bar')


    plt.savefig("ranges_heart_rate.png")
    plt.close()


def accelerometer_graphing(accelerometer, num):

    x = " accelerometer" + str(num) + "x"
    y = " accelerometer" + str(num) + "y"
```

```
    z = " accelerometer" + str(num) + "z"

    #prints 3 line graphs of the data to separate x, y, and z out
    plt.subplot(3, 1, 1)
    plt.plot(accelerometer["time"], accelerometer[x], '.-')
    plt.title('Accelerometer 1')
    plt.ylabel('X acceleration')

    plt.subplot(3, 1, 2)
    plt.plot(accelerometer["time"], accelerometer[y], '.-')
    plt.xlabel('time (s)')
    plt.ylabel('Y acceleration')
    plt.subplot(3, 1, 3)
    plt.plot(accelerometer["time"], accelerometer[y], '.-')
    plt.xlabel('time (s)')
    plt.ylabel('Z acceleration')

    plt.savefig("three_line_plot_accelerometer.png")
    plt.close()


    #3d plot
    ax = plt.axes(projection='3d')
    ax.plot3D(accelerometer[x], accelerometer[y], accelerometer[z], "green")
    plt.savefig("three_demension_plot_accelerometer.png")
    plt.close()

def force_graph(force_gauge):

    plt.plot(force_gauge["time"], force_gauge[" strain gauge"], "-")
    plt.xlabel("Time")
    plt.ylabel("Force")
    plt.title("Force Gauge Data over Time")

    plt.savefig("force_plot.png")
    plt.close()



def graphing():
   # imports the heart rate data as a pandas data frame
   filename = "data.csv"
   data = pd.read_csv(filename, parse_dates = ["time"])

   heart_rate_graphing(data, filename)
   force_graph(data)
```

```
accelerometer_graphing(data, 1)
```

Appendix P: main.py

```
import serial
from datetime import datetime
import schedule
import emailsender
from sms import sms
import time
import pandas as pd
import graphing
from os.path import exists
import threading

if not exists("data.csv"):
    file = open("data.csv","w")
    file.write("time, HR, accelerometer1x, accelerometer1y, accelerometer1z, strain gauge,
accelerometer2x, accelerometer2y, accelerometer2z\n")
    file.close()
def scheduling():
    time.sleep(60)
    schedule.every().day.at("21:59").do(graphing.graphing())
    schedule.every().day.at("22:00").do(emailsender.emaildaily)
    schedule.every().monday.do(graphing.graphing())
    schedule.every().monday.do(emailsender.emailweek)

def check():
    Cane_fall=False
    Cane_fall2=False
    sc=0
    oc=0
    fg=0
    index=0
    t=0
    while thread1.is_alive():
        compressed=pd.read_csv("data.csv",chunksize=100000,parse_dates = True)
        csv=pd.concat(compressed)
        length=len(csv.index)
        l=length-1
        if Cane_fall and not Cane_fall2:
            l3=l
            l2=index-50
            if l2<2:
                l2=2
            fc=0
            while l2<index:
                f=float(csv.iloc[l2][" strain gauge"])
                if abs(f)>3:
```

```
            fc+=1
        l2+=1
    if fc>25:
        l2+=300
        print(fc)
        Cane_fall2=True
        print("fall stage 1 cleared")
    if not Cane_fall2:
        print("fc:"+str(fc))
        Cane_fall=False
        print("fall cancelled")
if Cane_fall2 and Cane_fall and l>l3:
    while l>l3 and l>3 and t<15:
        b2=float(csv.iloc[l][" accelerometer1x"])
        b1=float(csv.iloc[l-1][" accelerometer1x"])
        c2=float(csv.iloc[l][" accelerometer1y"])
        c1=float(csv.iloc[l-1][" accelerometer1y"])
        d2=float(csv.iloc[l][" accelerometer1z"])
        d1=float(csv.iloc[l-1][" accelerometer1z"])
        f=float(csv.iloc[l][" strain gauge"])
        bm=abs(b1-b2)
        cm=abs(c1-c2)
        dm=abs(d1-d2)
        th=0.15
        if bm<th and cm<th and dm<th:
            sc+=1
        if abs(float(b2))<5.2:
            oc+=1
        if abs(f)>3:
            fg+=1
        l-=1
    time.sleep(0.5)
    t+=1
    print(t)
    print(fg)
    if fg>30:
        print("fall detection cancelled2")
        print("fg:"+str(fg))
        print("oc:"+str(oc))
        print("sc:"+str(sc))
        sc=0
        oc=0
        fg=0
        t=0
        Cane_fall=False
        Cane_fall2=False
```

```
    elif (sc>99 or oc>99) and Cane_fall and t>14:
        graphing.graphing()
        print("fall confirmed")
        sms("Fall detected")
        emailsender.emailtool("Fall data")
        print("fg:"+str(fg))
        print("oc:"+str(oc))
        print("sc:"+str(sc))
        sc=0
        oc=0
        fg=0
        t=0
        Cane_fall=False
        Cane_fall2=False
    elif t>14:
        print("fall detection cancelled3")
        print("fg:"+str(fg))
        print("oc:"+str(oc))
        print("sc:"+str(sc))
        sc=0
        oc=0
        fg=0
        t=0
        Cane_fall=False
        Cane_fall2=False
while l>3 and l>length-30 and not Cane_fall:
    b2=float(csv.iloc[l][" accelerometer1x"])
    b1=float(csv.iloc[l-1][" accelerometer1x"])
    c2=float(csv.iloc[l][" accelerometer1y"])
    c1=float(csv.iloc[l-1][" accelerometer1y"])
    d2=float(csv.iloc[l][" accelerometer1z"])
    d1=float(csv.iloc[l-1][" accelerometer1z"])
    bm=b1-b2
    cm=c1-c2
    dm=d1-d2
    if not bm-cm==0:
        r1=(b1-c1)/(cm-bm)
    else:
        r1=0
    if not cm-dm==0:
        r2=(c1-d1)/(dm-cm)
    else:
        r2=0
    r1=abs(r1)
    r2=abs(r2)
    if r1<=1 and r2<=1:
```

```
            if (r1>0 and r2>0) or (b1==c1 and b1==d1):
                print(l)
                print("fall detected")
                index=l
                Cane_fall=True
                l-=40
        l-=1
    time.sleep(0.5)

def cane():
    arduino=serial.Serial('COM6',115200)
    filename="data.csv"
    samples=2000000
    file=open(filename,"a")
    line=0
    while line<samples:
        getData=str(arduino.readline())
        now=str(datetime.now())
        data=str(getData)[2:len(getData)-5]
        # print(data)
        file=open(filename, "a")
        file.write(now+","+data+"\n")
        file.close()
        line+=1
        schedule.run_pending()
    arduino.close()

thread1=threading.Thread(target=cane)
thread2=threading.Thread(target=check)
thread3=threading.Thread(target=scheduling)
thread1.start()
time.sleep(1)
thread2.start()
thread3.start()
```