DESIGN REPORT

TEAM EXER-VIBES

Nikki Chi

Akyra Lee

Syona Mehra

Jenny Tran

ENGR 7B: Fitness Tracker

University of California, Irvine

Winter 2021

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Team Exer-vibes strived throughout the quarter to create a wearable fitness device that could accurately record the number of steps a user takes and the number of staircases they ascend or descend. In addition, an engaging app was also created allowing the user to track their progress and set a step count goal. Specific hardware used to create such a device included an accelerometer, a barometer, a vibration motor, and specific Arduino devices including the UNO, Bluno, and Beetle. Our team also had to work within a $150 budget. Altogether, our team used SOLIDWORKS to design a suitable structure hold our electrical components, wrote several algorithms in the Arduino language manipulating the data gathered from our sensors, mapped out our circuity and soldered our device in lab, and used Blynk to create a handy cell phone app. Ultimately, our team was able to overcome many struggles, learn valuable lessons, and produce a fully functioning device at the end of the quarter.

PROBLEM DEFINITION

I. Introduction

The goal of this project is to design, build, and code a working fitness tracker that is able to track both our step and stair count with 90% accuracy. In addition, the fitness tracker must also be connected to a mobile app, which will be Blynk for this specific project, in order to relay to us our step and stair count, as well as allow us to set goals and indicate that we've met them, like a modern day fitness tracker would. Utilizing teamwork, ingenuity, and trial and error, we hope to not only achieve our desired product, but also to further our technical skills in Computer Aided Drafting (CAD) and coding. Throughout the progression of this project, we also hope to further our project development and management skills through our presentations, research, and technical reports.

II. Technical Review/Background

"Fitness Trackers" can be traced back all the way to the 15th century, where Leonardo da Vinci had sketched out a device that would be able to track the foot movements of Roman troops, and the concept would gradually expand from there, from French craftsman Jean Fennel's turning dial in 1525 to Swiss hologist Abraham Louis-Perrelet's self-winding watch in 1780, and henceforth. The designs were rather primitive, focused around the winding of levers or swinging of pendulums, but they achieved the autonomy aspect of a fitness tracker as they were motion sensitive. The first modern day "fitness tracker", however, actually first became popular in Japan in the 1960s through Dr. Yoshiro Hatano, a researcher from Kyushu University, who developed the 'manpo-kei', which translates to '10,000 steps meter', in order to combat obesity. After that, the gradual expansion of fitness trackers would grow into something more rapid.

Today, the fitness tracker market value grosses over USD 30 billion, with popular brands such as Fitbit, Garmin, Whoop, and Apple Watch, just to name a few, dominating the masses. And with the rapid advancement of technology, a fitness tracker now, does more than just track steps. They can also track flights climbed,

heartbeats, calorie intake, sleep, time, goals met, as well as double at a GPS, phone notification center, and much more, all within a small device attached to the wrist.

III. Design Requirements

1. Size and Durability Requirements

    (a) Easily detachable and components must be secured to withstand for easy movement

    (b) No more than 4 inches in width (along length of arm)

2. Fitness Tracking and Mobile App Requirements

    (a) 90% accurate and consistently track the (1) step count and (2) stair count (altitude) of the user

    (b) Allow user to input goals into mobile app and track progress/achievements with indicator

    (c) Show update of data on mobile app when activity is completed

3. Other

    (a) Must be able to operate for at least 1 hour

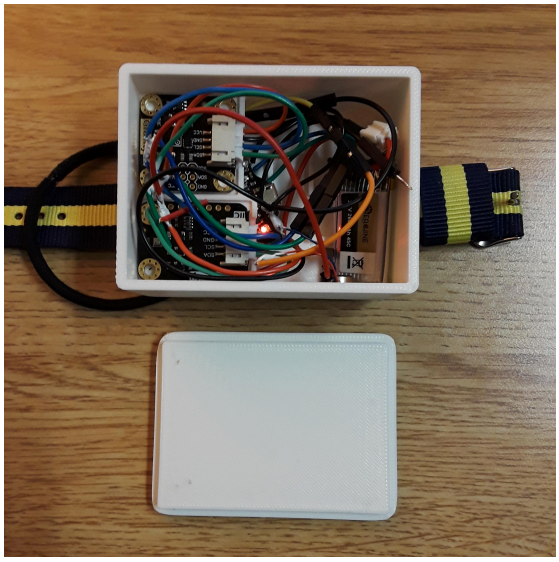
    (b) Wires must be insulted and electrical components secured

    (c) Max budget of $150

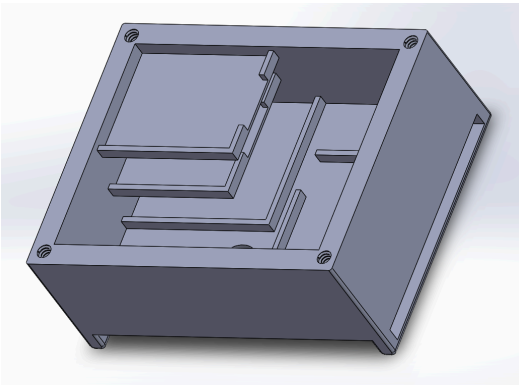
DESIGN DESCRIPTION

I. Summary of Design

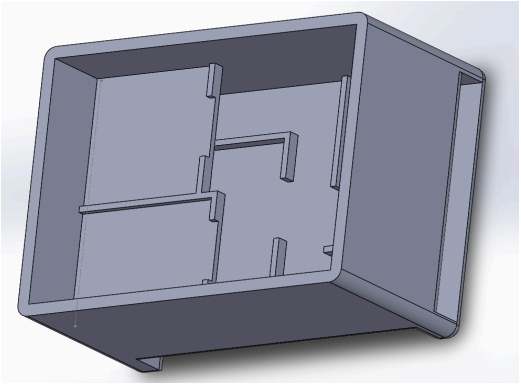
The major structural parts of the design were the wrist strap, the lid, and the case. Overall, the wrist strap and the lid were very straight forward; the wrist strap was simply a product that was already made and the lid was designed to fit on top of the case. In the end, the team connected the case and lid with a simple hair elastic in order to maintain easy access to the beetle to upload any updated code. Though reprinting was considered, this solution was effective and saved a bit on costs, with the final cost being $67.86. Our main spending went towards 3D printing, the Arduino Beetle, and the wrist strap - the three highest cost items. While there is no recorded weight of the device, the final size ended up as 2.25” x 3.00” x 1.50” which fit well within the design requirements without restricting movement of the user.



II. Fabrication Design



(Initial Design)



(Final Design)

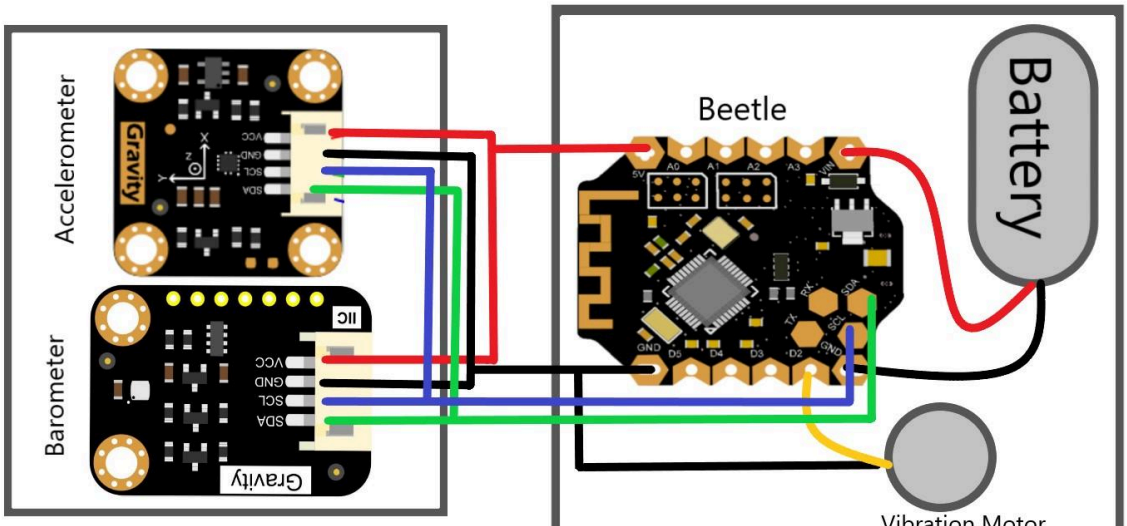
There were many design decisions that made the final product unique and effective for the device's purposes. Firstly, the design for the case implemented the idea of shelving in order for the electronic components to securely fit while creating space for the wires and optimizing the overall size of the device. In our initial design, it's clear to see the change from using three levels to using two in order to make our device much smaller overall. 3D printing was the main method of creating the device since it could account for the support needed in making the shelving within the case and give the structural integrity needed to hold everything together.

Another design choice made was in regard to the indicator chosen - the vibration motor. In conceptualizing this product, accessibility is an important part of using a fitness device. If an LED light blinked when a goal is met, the user isn't 100% likely to see or notice it. With this in mind, the vibration motor is crucial because it allows all users to notice and feel when their goal is met. In the design, it is also clear to see that there is a designated part embedded in the case for the vibration motor to truly ensure that the user could feel it. In the final design after assembly, the vibration motor's wires were unfortunately a bit longer than expected and did not fit in the spot designed for it, however, the vibration of it was powerful enough to feel when it went off and worked out in the end.

The final design choice made was the choice of not using screws to secure the lid in the design. In our initial design, four screws were considered to secure the lid. The reason this was somewhat of a design flaw is due to the fact that it greatly minimized the amount of space within the case needed for the electronic components and the wiring. Thus, the design was changed to be a simple lid that, if not snug, would be secured with double sided tape on the inside. With this, there was an issue with the 3D print design shrinking a bit for the lid, but not the case. This resulted in a loose lid, but the solution was to use a hair elastic to keep the parts all together. Though it wasn't an ideal solution, it actually proved to be extremely effective because it allowed for easy lid removal in order to connect the beetle to a computer to upload updated code while testing without giving issues.

In conclusion, many design decisions were made with practicality for the user and the creators/testers in mind. With the vibration motor, it is a good product for anyone with fitness goals in mind and does not require any hassle or question as to if they reached their goal or not. The shelving allowed for easier fabrication for the creator in that they did not have to worry much about spacing while soldering and the placement of the electronic components since the design dictated that aspect. Finally, the use of a hair elastic for adhering the lid to the case was ideal for testers to update modified code while testing, giving ease of access and a simple way to get to the beetle.

III. Circuit Design



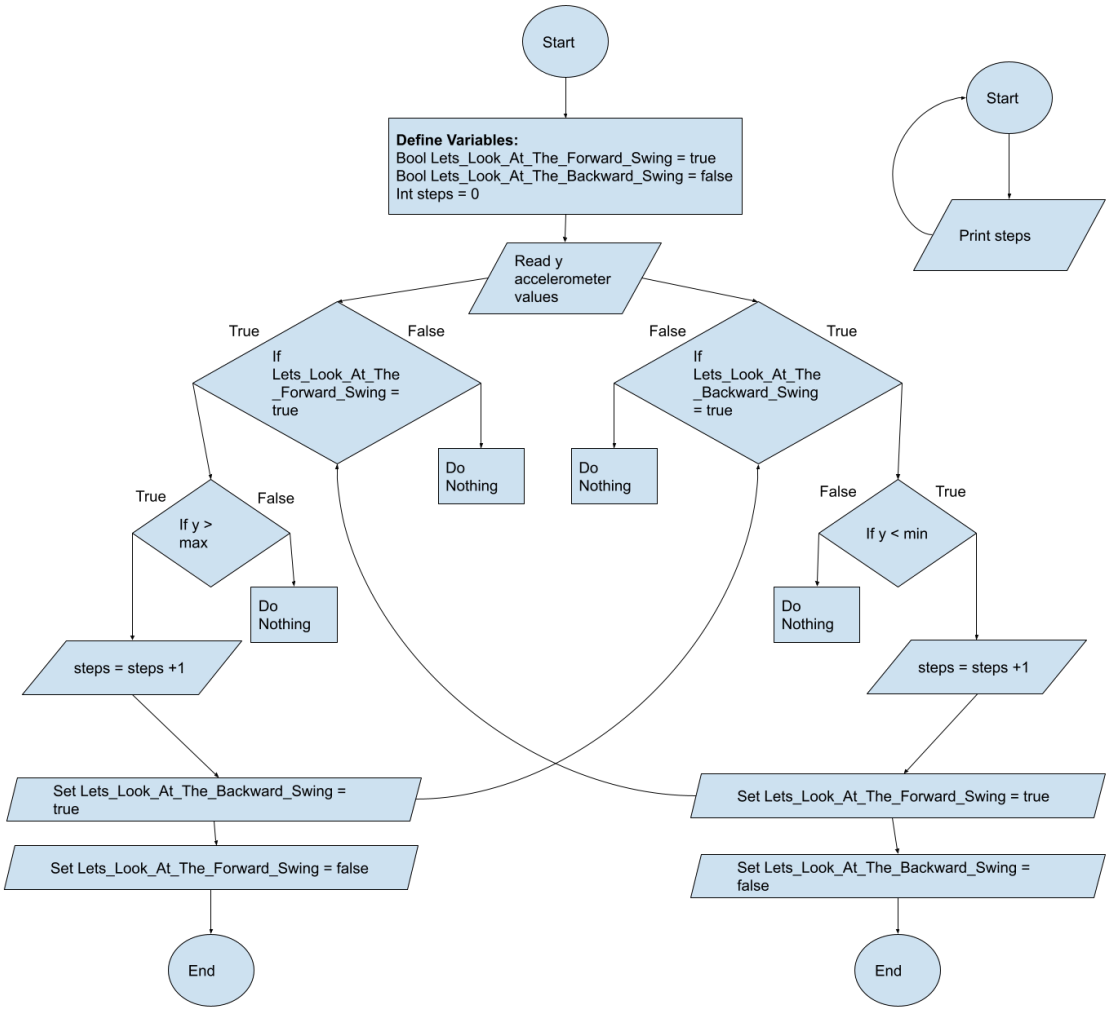
Top Floor (a shelf stacked on top of bottom floor)

Bottom Floor

Our final wiring diagram showcases four major bus line connections. The SDA bus line in green, the SCL bus line in blue, power in red, and ground in black. The SDA and SCL bus lines connect both sensors, the barometer and the accelerometer, to the beetle. These communication protocols facilitate the transfer of analog data. Red wires provide power to the sensors from the beetle. Black ground wires connect both sensors and the indicator, our vibration motor. The vibration motor is connected to digital port D2. This ensures that the vibration motor will only be turned on if port D2 is sent power by the beetle through code. Finally, a small battery acts as the beetle's power supply and is connected to the beetle through a short red and black wire.

Our initial wiring diagram had the same connections as our final version however it was not fitted to suit the SolidWorks design, the wires were longer and messier, and the accelerometer and barometer switched positions. We overall decided to recreate the wiring diagram inside our solidworks design so we had a more clear idea of how our electrical components should be soldered and to gauge the length of wire needed for each bus line connection.

IV. Algorithm Design



Pseudocode:
STEP 1: Start

STEP 2: Initialize global variable “steps” of integer data type and value 0, “Lets_Look_At_The_Forward_Swing” and “Lets_Look_At_The_Backward_Swing” of boolean data type with value “true” and “false” respectively.

STEP 3: Loop: Call the function “acceleration” which gets the data points and counts the number of steps

STEP 4: Begin the function called “void Acceleration(void)”

STEP 5: Then in the function, If the hand moves forwards, print(“hand is moving forward”) for debugging. Then if the accelerometer reading is greater than the maximum steady state value(calculated previously in the code), print("left foot took a step”) for debugging and increase the number of steps by 1(steps = steps + 1;) and set the “Lets_Look_At_The_Forward_Swing” and “Lets_Look_At_The_Backward_Swing” of boolean data type with value “false” and “true” respectively to move to STEP 5 and avoid recounting steps. End of nested “If statements”.

STEP 6: If the hand moves backward, print(“hand is moving backward”) for debugging. Then if the accelerometer reading is less than the minimum steady state value(calculated previously in the code), print("right foot took a step”) for debugging and increase the number of steps by 1(steps = steps + 1;) and set the “Lets_Look_At_The_Forward_Swing” and “Lets_Look_At_The_Backward_Swing” of boolean data type with value “true” and “false” respectively to move to STEP 4 in the next iteration and avoid recounting steps. End of nested “If statements”.

STEP 6: End of function.

Discussion

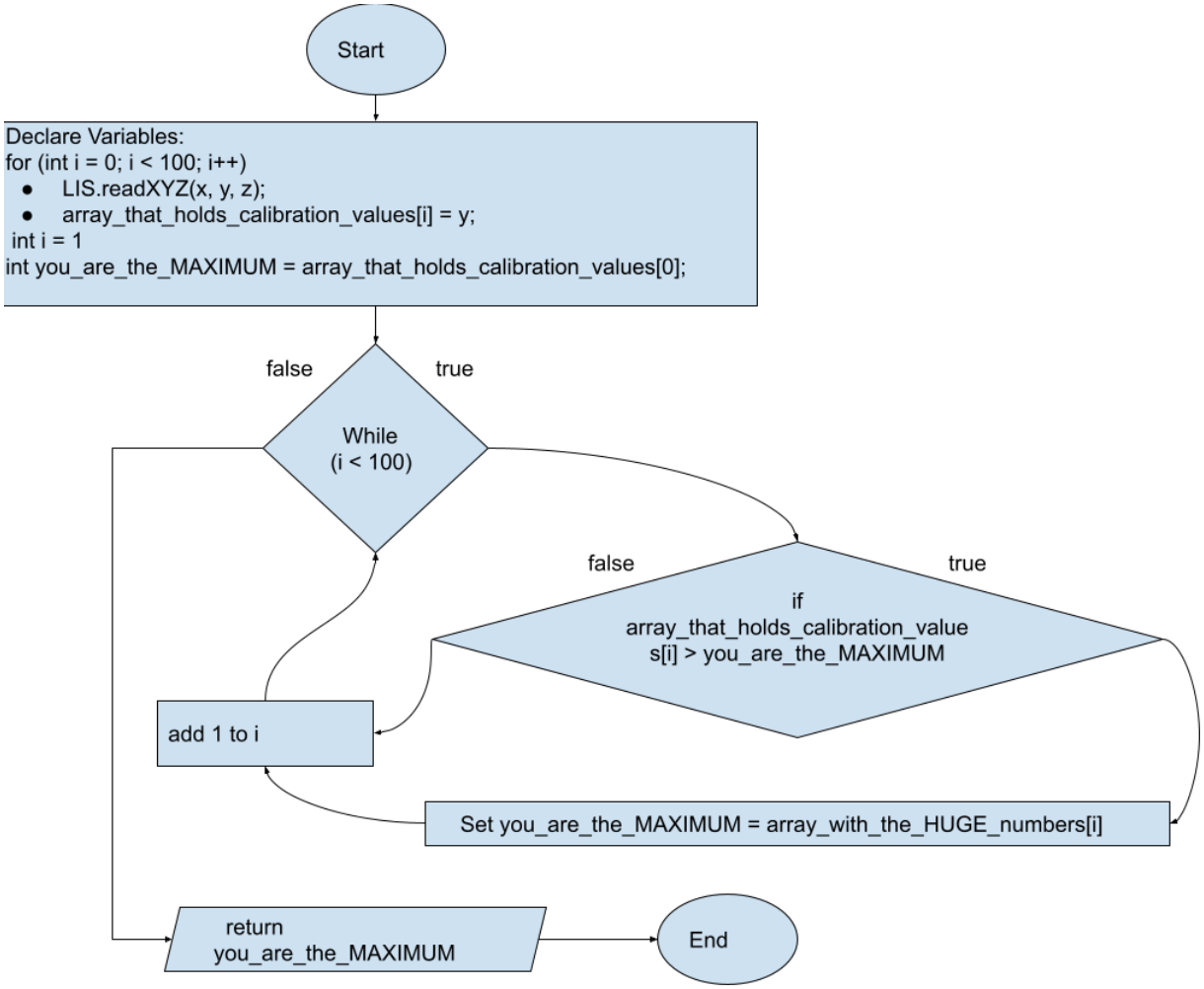
Overall, we like to call our step count algorithm the “Hoppy” Code because it “hops” from counting steps beyond the maximum steady state value to below the minimum steady state value and vice versa. Our algorithm design went through many iterations during the course of the project. In the first code we wrote we were unable to incorporate a proper step count halt. We did not know how to stop adding steps to the step count after the first step had been taken. We wrote several much more complicated forms of code, but these were unsuccessful because they relied on a much more precise accelerometer. In the end, after completing the second homework assignment we learned how to use boolean variables and had found the solution to our problem. Since bool variables can only be true or false, unlike integer variables, we were able to create halts in our code after the first step was counted. After adding this halt to our original version of code we formed the algorithm above.

Our algorithm proceeds as follows:
Two boolean variables named “Lets_Look_At_The_Forward_Swing” and “Lets_Look_At_The_Backward_Swing” would be declared and variable to record the number of steps taken named “steps”. We assumed that the user would swing the hand with the device outward first therefore we set
“Lets_Look_At_The_Forward_Swing” to true and

“Lets_Look_At_The_Backward_Swing” to false. Within void loop we created a pair of if statements to check whether “Lets_Look_At_The_Forward_Swing” or “Lets_Look_At_The_Backward_Swing” is true. If “Lets_Look_At_The_Forward_Swing” is true we consider if the user’s accelerometer y value is greater than the maximum steady state value. If so we add a step to our step count. We then set “Lets_Look_At_The_Backward_Swing” equal to true and “Lets_Look_At_The_Forward_Swing” equal to false to hop to the other if statement. Now if “Lets_Look_At_The_Backward_Swing” is true we consider if the user’s accelerometer y value is less than the minimum steady state value. If so we again add a step to our step count. Then set “Lets_Look_At_The_Forward_Swing” equal to true and “Lets_Look_At_The_Backward_Swing” equal to false to hop to the other if statement.

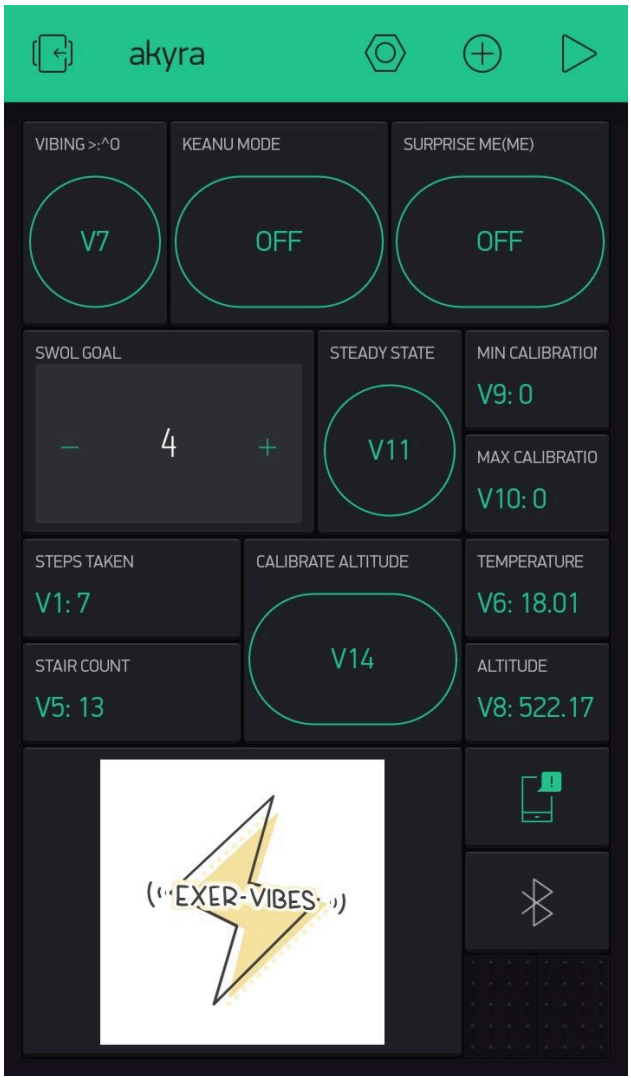
Our step count algorithm utilizes the y values of the accelerometer data. Plotting the x, y, and z values of acceleration on Adruino’s plot monitor we determined that if you hold the accelerometer flat on a desk with the wires pointing down, picking the device up above the table changes the z value, moving the device left and right changes the y value, and moving the device up and down shifts the x value. Due to the position of our accelerometer within our solidworks design we determined that the accelerometer would swing side to side, therefore moving left and right and corresponding to the y axis. To test our assumption, we attached the accelerometer to our wrist and moved it as if we were walking. We observed data in all three dimensions. We confirmed that changes in the accelerometer y values were most consistent with the swing of the user’s arm and served as the most reliable data.

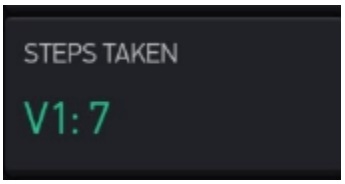
In our original version of our step could algorithm we hardcoded our maximum and minimum steady state values of 250 and 0 to our team tester. In order to make our watch wearable for anyone of any stature we needed to write code that calibrated the user’s steady state values. In this code we created an array to hold 100 accelerometer y values. We then created two functions, one function that returns the minimum value of the array and another that returns the maximum value of the array. These functions were greatly influenced by Homework Assignment #2. The maximum and minimum values determined were then made global variables and replaced the hard coded steady state values initially written in the step count algorithm. See the flowchart below, where the maximum steady state value was determined, for a summarized overview of our logic (Note: the “you_are_the_MAXIMUM” variable is equal to the “max” variable in the “Hoppy” flowchart).



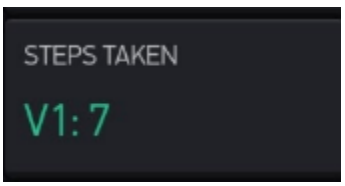
Altogether we created a button on our app to call our maximum and minimum finding functions when pressed. Incorporating this button resolved concerns over steady state values being taken while the user was not ready and allowed for new steady state values to be retrieved at any time contributing to ease of transfer between different users.

v. User Interface Design

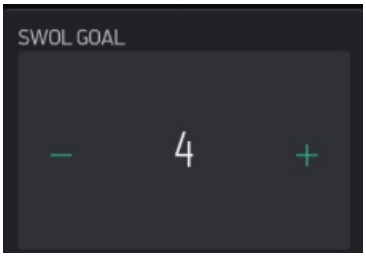




Virtual port V1 displays the number of steps taken by the user.



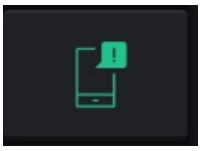
Virtual port V5 displays the number of staircases climbed by the user.



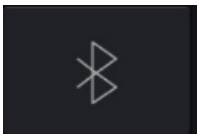
This numeric value input allows the user to set a step count goal.

Port V11 is a button that calibrates the user's steady state values when pressed. These values are showcased by port V9 and V10 where the maximum and minimum accelerometer values attained are displayed. These value displays were used to confirm that the steady state values were taken.

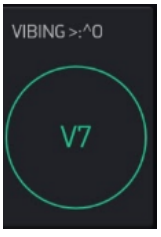
Port V14 is a button that inputs the user's current altitude into the code written for the staircase count. This button is similar to the "Steady State" button and must be pressed after the user attaches the watch to their hand. Port V6 and V8 are used to confirm that the barometer is connected and working. Port V6 displays the current temperature and port V8 displays the current altitude.



This widget enables messages to be sent to the user. Specific messages were sent after the user pressed the "Steady State" and "Calibrate Altitude" buttons to indicate the button had been successfully received. A message was also sent to congratulate a user upon reaching their step goal.

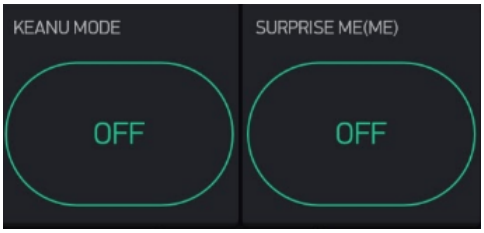


This widget enables Blynk to connect to the Beetle's bluetooth.



This button was used to verify that the vibration motor was connected and working properly. Upon pressing this button the vibration motor will be turned on and the user will be spammed with a message.

This widget is an image display used to display the team logo. It switches to display different images after specific buttons are pressed. This was created entirely for entertainment and experimental purposes.



When switched on, the “KEANU MODE” button changes the image display to a gif of Keanu Reeves perpetually growing and shrinking to help motivate the user. The “Surprise Me(me)” button flashes a new meme to the image display when pressed. Both buttons were created entirely for entertainment and experimental purposes.

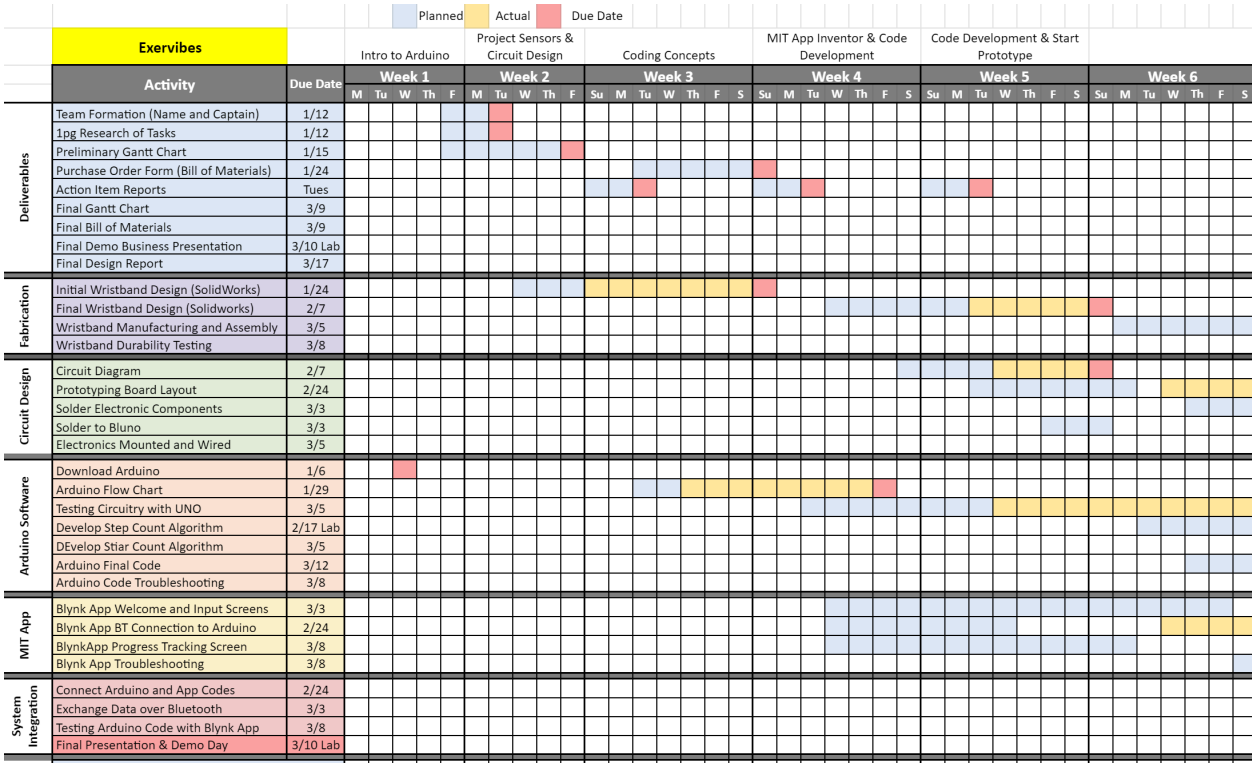
Our overall app design has evolved greatly during the course of the project. Our initial design only printed the number of steps taken by the user, enabled bluetooth, and had a button to turn on the Beetle’s LED (to verify the Beetle’s connection to the app). We then began adding more buttons and value displays to confirm that our sensors were connected and working and that proper measurements such as our maximum and minimum steady state values were being recorded. Furthermore, we started experimenting with other widgets provided by Blynk including the image display, extra buttons to change the images, and the message sending enabler to make our app more engaging.

ACTION ITEMS

I. Task Assignment

Throughout this project, the tasks assigned were based on the team individual's interests, preferences, and skills they'd already had on hand. To start, our Solidworks designs and drawings were done by Jenny Tran. She was in charge of drawing up our preliminary and final design, as well as submitting the request form for our fitness tracker design to be 3D printed. Next is our coding, which was taken care of by both our team captain, Akyra Lee, and Syona Mehra. The two worked as a team to create both our step count and stair count algorithm, coming up with new and creative solutions for each other whenever a problem within the code arose. As for our Blynk app, the downloading, initialization, and customization of it was taken care of by Akyra Lee. Assembly, with both the inclusion of getting our parts delivered as well as soldering the electrical components, was taken care of mostly by Akyra Lee and Jenny Tran, as they were either on campus, or could come to campus directly with little difficulty. Testing the fitness tracker, both in the early and late stages of this project, was done by the team as a whole, most of which was done over Zoom. Even though this was how a bulk of the work was carried out, each team member played an active role in each task, as we were cohesive in our teamwork and supported each other whenever it was needed. As for our deliverables, such as the Gantt chart and Purchase Order forms, they were taken care of by Nikki Chi. However, AIR reports, presentations, and this final design report was done by our team as a whole.

II. Gantt Chart



Planned Actual Due Date

	Exervibes		Intro to Arduino					Project Sensors & Circuit Design					Coding Concepts							MIT App Inventor & Code Development							Code Development & Start Prototype													
	Activity	Due Date	Week 1					Week 2					Week 3							Week 4							Week 5							Week 6						
			M	Tu	W	Th	F	M	Tu	W	Th	F	Su	M	Tu	W	Th	F	S	Su	M	Tu	W	Th	F	S	Su	M	Tu	W	Th	F	S	Su	M	Tu	W	Th	F	S
Deliverables	Team Formation (Name and Captain)	1/12																																				
	1pg Research of Tasks	1/12																																				
	Preliminary Gantt Chart	1/15																																				
	Purchase Order Form (Bill of Materials)	1/24																																				
	Action Item Reports	Tues																																				
	Final Gantt Chart	3/9																																				
	Final Bill of Materials	3/9																																				
	Final Demo Business Presentation	3/10 Lab																																				
	Final Design Report	3/17																																				
Fabrication	Initial Wristband Design (SolidWorks)	1/24																																				
	Final Wristband Design (Solidworks)	2/7																																				
	Wristband Manufacturing and Assembly	3/5																																				
	Wristband Durability Testing	3/8																																				
Circuit Design	Circuit Diagram	2/7																																				
	Prototyping Board Layout	2/24																																				
	Solder Electronic Components	3/3																																				
	Solder to Bluno	3/3																																				
	Electronics Mounted and Wired	3/5																																				
Arduino Software	Download Arduino	1/6																																				
	Arduino Flow Chart	1/29																																				
	Testing Circuitry with UNO	3/5																																				
	Develop Step Count Algorithm	2/17 Lab																																				
	DEvelop Stiar Count Algorithm	3/5																																				
	Arduino Final Code	3/12																																				
	Arduino Code Troubleshooting	3/8																																				
MIT App	Blynk App Welcome and Input Screens	3/3																																				
	Blynk App BT Connection to Arduino	2/24																																				
	BlynkApp Progress Tracking Screen	3/8																																				
	Blynk App Troubleshooting	3/8																																				
System Integration	Connect Arduino and App Codes	2/24																																				
	Exchange Data over Bluetooth	3/3																																				
	Testing Arduino Code with Blynk App	3/8																																				
	Final Presentation & Demo Day	3/10 Lab																																				

	Exervibes																														
	Activity	Due Date	Week 7							Week 8							Week 9							Week 10					Finals Week				
			Su	M	Tu	W	Th	F	S	Su	M	Tu	W	Th	F	S	Su	M	Tu	W	Th	F	S	M	Tu	W	Th	F	M	Tu	W	Th	F
Deliverables	Team Formation (Name and Captain)	1/12																													
	1pg Research of Tasks	1/12																													
	Preliminary Gantt Chart	1/15																													
	Purchase Order Form (Bill of Materials)	1/24																													
	Action Item Reports	Tues																													
	Final Gantt Chart	3/9																													
	Final Bill of Materials	3/9																													
	Final Demo Business Presentation	3/10 Lab																													
	Final Design Report	3/17																													
Fabrication	Initial Wristband Design (SolidWorks)	1/24																													
	Final Wristband Design (Solidworks)	2/7																													
	Wristband Manufacturing and Assembly	3/5																													
	Wristband Durability Testing	3/8																													
Circuit Design	Circuit Diagram	2/7																													
	Prototyping Board Layout	2/24																													
	Solder Electronic Components	3/3																													
	Solder to Bluno	3/3																													
	Electronics Mounted and Wired	3/5																													
Arduino Software	Download Arduino	1/6																													
	Arduino Flow Chart	1/29																													
	Testing Circuitry with UNO	3/5																													
	Develop Step Count Algorithm	2/17 Lab																													
	DEvelop Stiar Count Algorithm	3/5																													
	Arduino Final Code	3/12																													
	Arduino Code Troubleshooting	3/8																													
MIT App	Blynk App Welcome and Input Screens	3/3																													
	Blynk App BT Connection to Arduino	2/24																													
	BlynkApp Progress Tracking Screen	3/8																													
	Blynk App Troubleshooting	3/8																													
System Integration	Connect Arduino and App Codes	2/24																													
	Exchange Data over Bluetooth	3/3																													
	Testing Arduino Code with Blynk App	3/8																													
	Final Presentation & Demo Day	3/10 Lab																													

EVALUATION

I. Testing

Prototype of 3D Design

After receiving our 3D printed parts, we removed the supports and thoroughly sanded each part to remove any potentially harmful jagged edges. After attaching a strap to our 3D assembly our prototype was complete. We tested the safety of our prototype by trying the watch on and making sure that it could comfortably fit the wrist.

Furthermore, we tested the reliability of our device by evaluating the fit of our electrical components. This ensured that all components would be properly concealed and stable within our final design allowing for the most precise sensor readings to be taken. We also tested the fit of the watch's lid. The lid was a little too loose so we decided to use a strong hair elastic to fasten and protect parts from falling out of the device. In addition, we ensured that the watch was secure and did not slip on the user's wrist at any time, again to ensure the precision of our sensor readings.

Circuit Prototype

Our team used a breadboard and jumper wires to create a circuit prototype with both sensors and an indicator. We tested connection to the barometer by using sample Arduino code that prints temperature to the serial monitor. Since temperature was successfully printed to the monitor and within reason we confirmed that our barometer was properly connected to our circuit and working effectively. Furthermore, we also tested connection to our accelerometer using sample code, this time however for acceleration values. X, y, and z values for acceleration were properly displayed on the plot monitor and the serial monitor, confirming that our accelerometer was connected to the circuit and was properly functioning. We were unable to include the vibration motor in our temporary circuit because the wires were too small to insert into the breadboard. We instead trusted an LED to act as our vibration motor. We wrote a simple code that would turn our LED on in order to verify its connection. Altogether we tested the entire circuit by holding the circuit and moving it short distances to see if data could still be taken from both sensors.

Arduino Code

Throughout the course of the project we wrote several iterations of code to calculate the number of steps a user took. In order to test our code, we created a circuit that only included the accelerometer, Arduino UNO, and a long cable. The full temporary circuit was too difficult to talk around with. We also taped the accelerometer to an actual watch to ensure that the position of the sensor would be as stable as possible. Ultimately, the tester would either walk a short distance and count the number of steps they took out loud or move their arm back and forth as if they were walking to evaluate the accuracy of our code. Before the watch was soldered, the step count was printed to the serial monitor and several confirmation print statements were utilized. After the watch was soldered, the step count was showcased on Blynk. We used both methods to verify our code's accuracy.

Our team tested code written for the staircase count in a similar fashion. By the time we were ready to test our code however the soldering had already been completed therefore no temporary circuit was needed. We measured the height of Engineering Tower's staircases to be about eight feet. This value would be converted into meters within our code. For testing, we used a four foot height instead of an eight foot height. We first placed the watch on the ground, took the initial altitude using the "Calibrate Altitude" button on our app, then lifted it up to place it on a shelf a little over four feet tall. We checked to see if the staircase count, printed on the app, increased by one.

Furthermore, we tested the code written to calibrate the user's steady state values by attaching the watch to the rist and standing in a neutral position. We then pressed a button on our app that called the steady state value finding process. We used value displays and messages on our app to confirm that steady state values had been taken. We would then move our rist to a different position and recalibrate the steady state values to ensure that different values were printed.

II. Results

Altogether, our watch could count the number of steps taken and the number of staircases ascended or descended by the user with nearly 100% accuracy.

The watch remained within the size requirements of the project and could comfortably fit the user's wrist. The device could also be easily removed and attached

to the wrist. The watch's lid could easily be opened and remained secured with the help of a strong elastic. The device remained lightweight and adequately secured to the user's wrist when worn.

Nevertheless, the bluetooth that connected the watch and the app proved flimsy and unreliable. However, if kept in close proximity to the watch, a cell phone could remain efficiently connected.

After resoldering some components, specifically the beetle and the vibration motor, and sealing all connections with both liquid electrical seal and electrical tape, the strength of the soldering connections was enhanced and the watch could be handled more easily. Such efforts made the device more durable. Furthermore, seeing as soldering connections were sealed with both liquid and solid electrical tape, great care was taken to ensure electrical safety.

III. Discussion

Throughout the entirety of this project our team encountered many obstacles and we did our best to resolve each complication.

First, our team greatly struggled to create an efficient step count algorithm. We needed to create a halt within our code to prevent too many steps from being counted without using a delay. In order to do so we did our best to learn more about programming logic, the arduino language, and the capabilities of our accelerometer. In the end, completing the second homework assignment helped us figure out how to use boolean variables and make our code hop back and forth.

Another issue the team encountered involved code written to calibrate the user's steady state values. Within our code we discovered that we could not calibrate steady state values in our set up function because our device could be in any initial position. Steady state values would be taken immediately even if the watch was not on the user or before the user was ready. In order to ensure the proper values were taken, the user needed to stand in a very precise position. In order to resolve this issue, we created a button on our app that would establish the user's steady state values when pressed. This ensured the accuracy of our steady state readings and allowed new values to be taken at any time. In addition to creating this button, we also used value displays on our app to showcase the maximum and minimum steady state values collected. These methods helped us immensely to test our calibration code. Following this reasoning, we also created a button to collect the user's initial altitude, used for

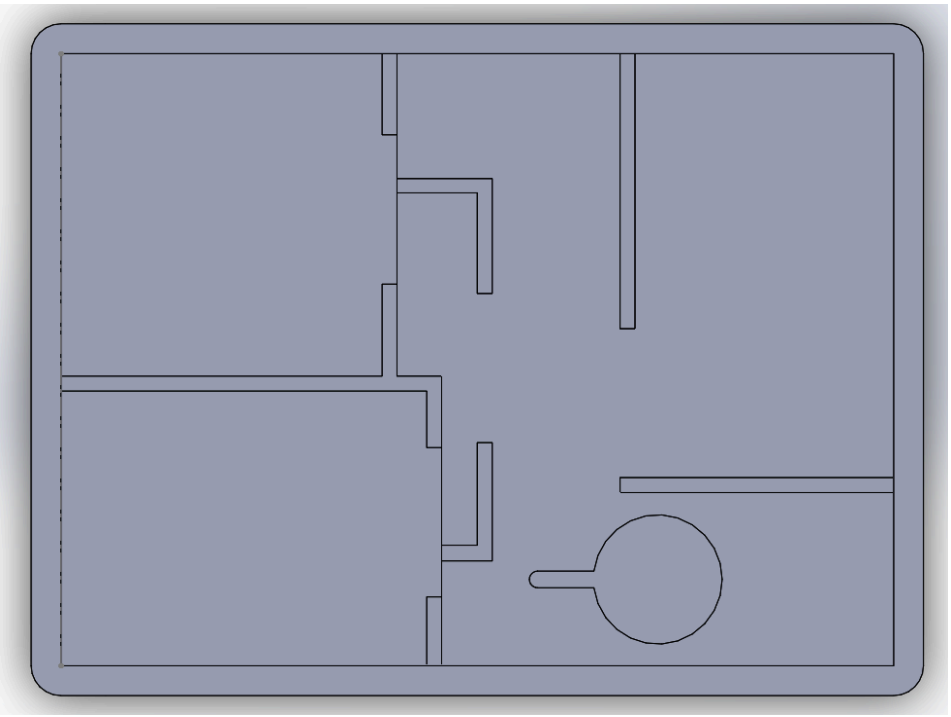
our staircase count, which greatly helped us test our code.

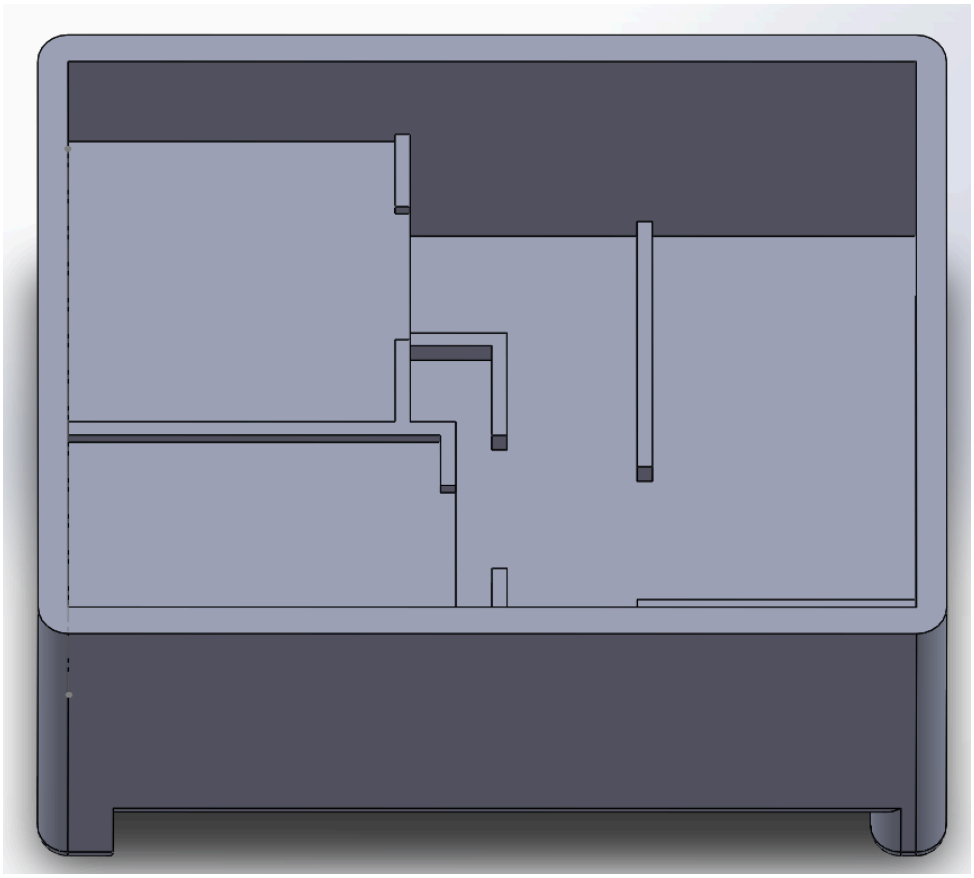
One of the most prevalent struggles our team experienced involved soldering. During our first attempt at soldering our bus lines together one of the wires on our beetle broke off and we had to get it repaired. This happened a second time and we received a new beetle with stronger soldering connections. Nevertheless, a beetle wire disconnected a third time a few hours before our demonstration and we had to have it quickly repaired. On the same day immediately after having our beetle fixed, our vibration motor's ground wire broke and we had to quickly resolder it in the lab before our demo. We also had to strengthen our other soldering connections by wrapping them in electrical tape, seeing as solely covering our connections with liquid electrical seals proved weak and risky. This method was overall successful.

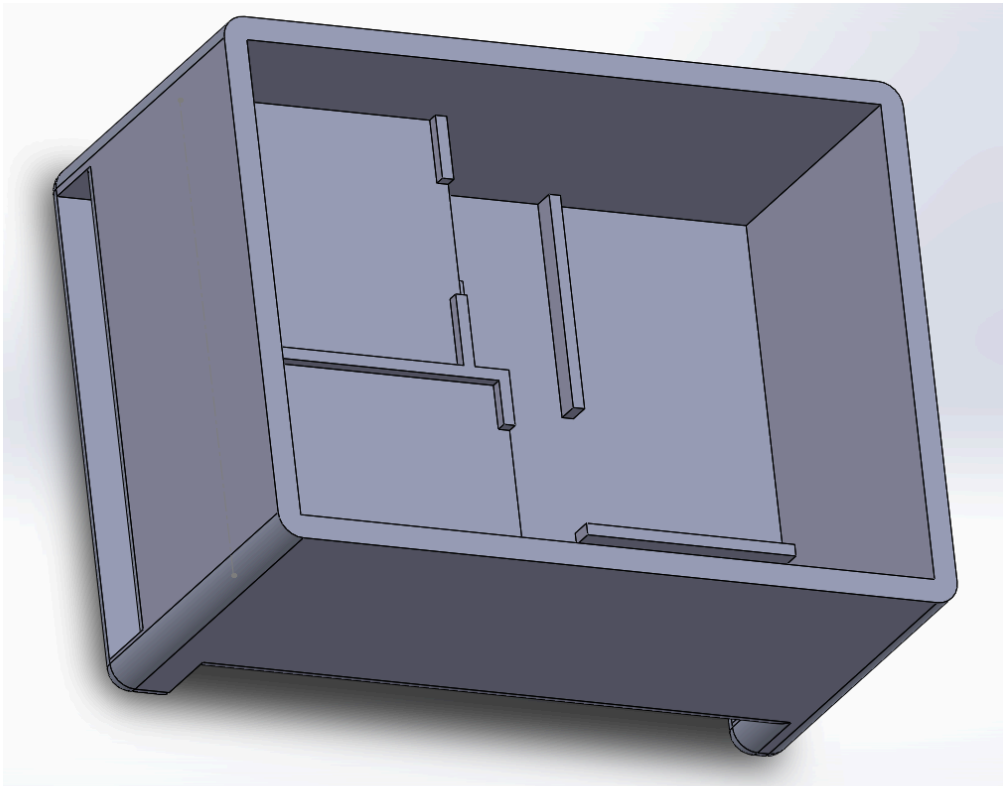
Finally, the bluetooth connection between our app and beetle often served as a large obstacle to our progress. Most often, our app and beetle would not connect to each other, or would take an unreasonable amount of time to connect. Resolving such a problem proved difficult because we could not visually see what was failing. Our team ultimately created a precise ritual that ensured ideal circumstances for bluetooth to properly connect. The cellphone's bluetooth would need to be turned off first, the code would then be uploaded to the beetle, the battery would then be attached to the beetle, the phone's bluetooth would be turned on and an attempt would be made to connect, this connection would fail, and a second try would succeed. Finally, the beetle would be disattached from the computer. Furthermore, throughout our demo we realized that the user's cellphone and beetle needed to be in close proximity in order for bluetooth to remain connected and for the device to work properly.

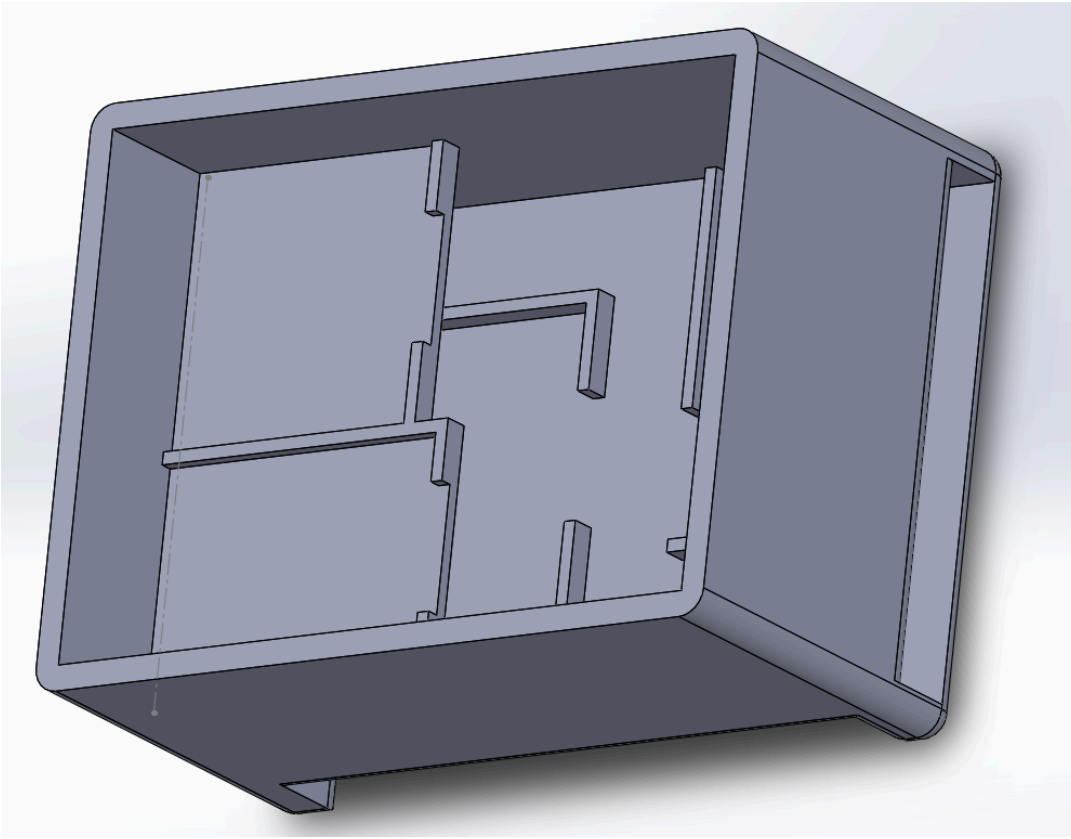
APPENDIX A: SOLIDWORKS DESIGN

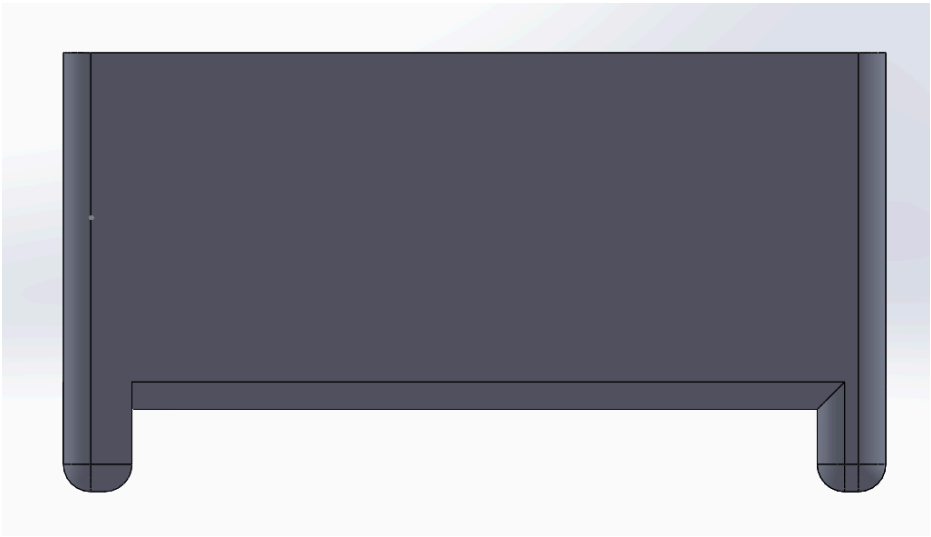
Case 3D Images:

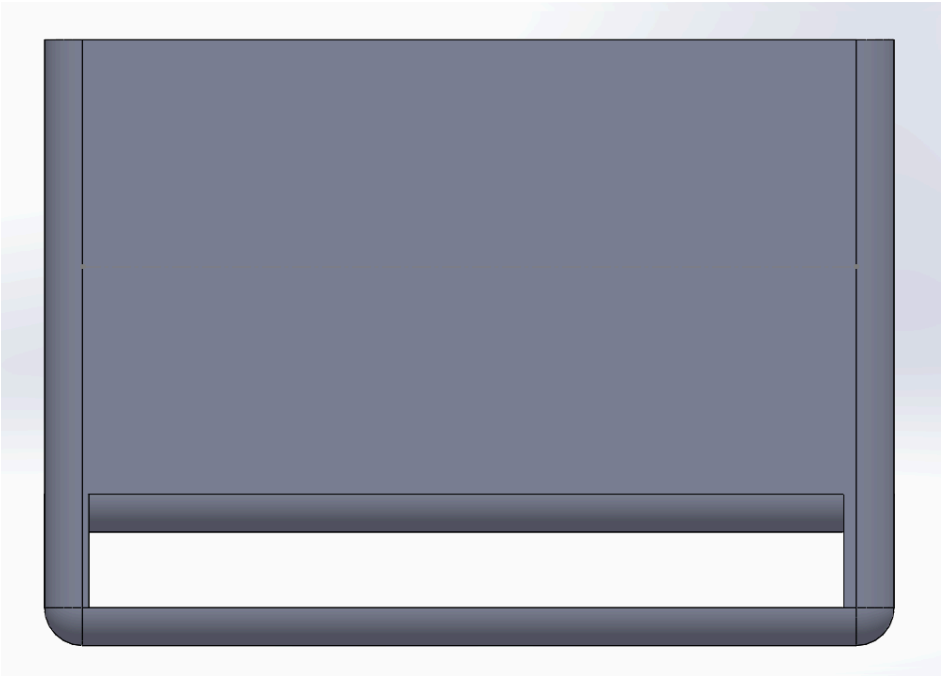




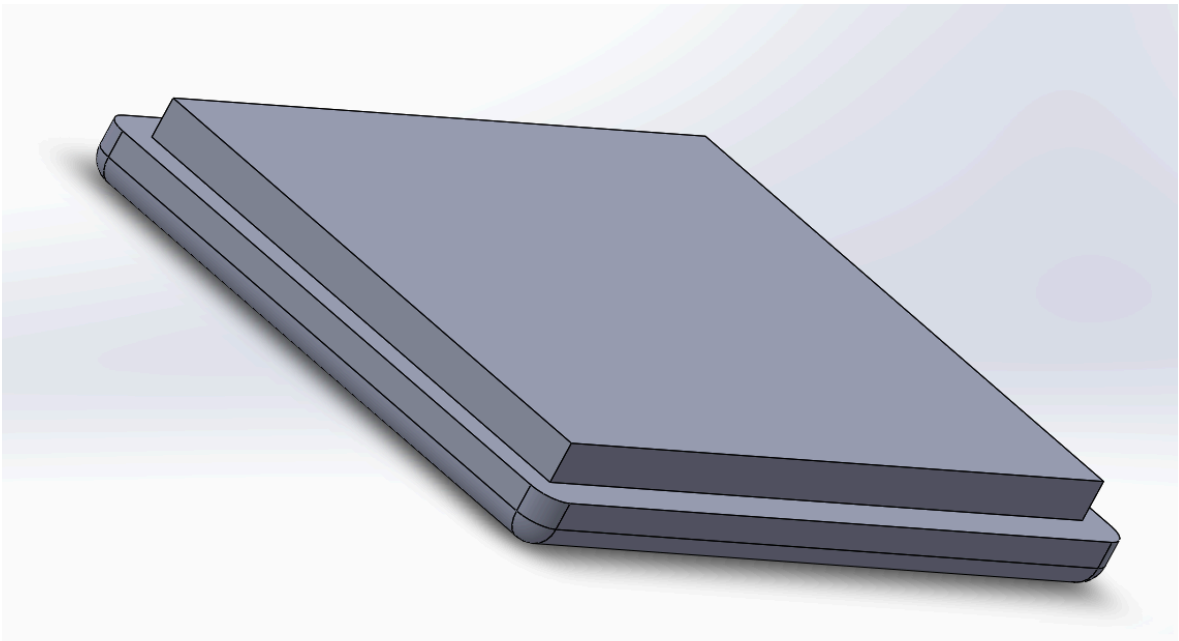






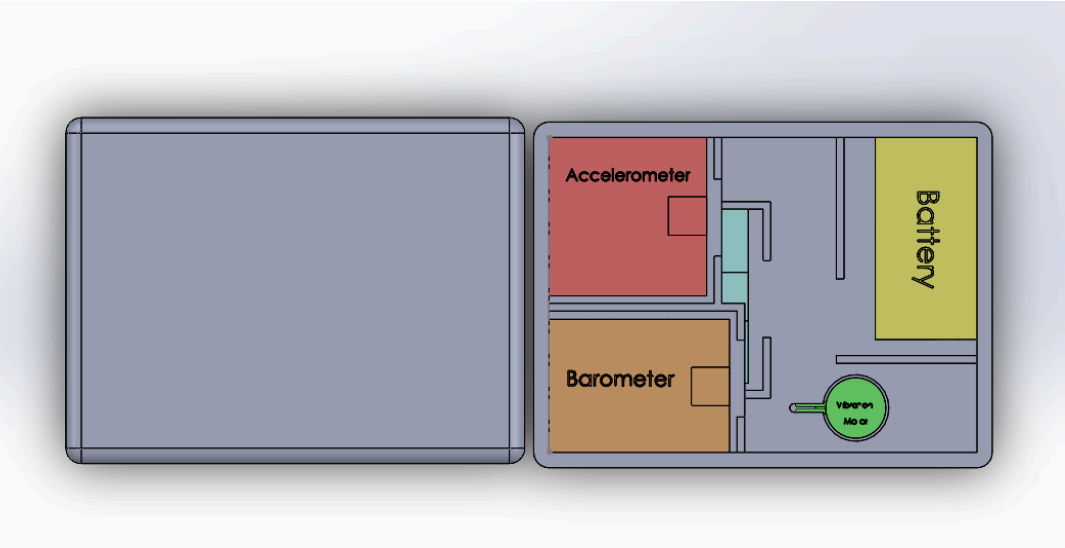


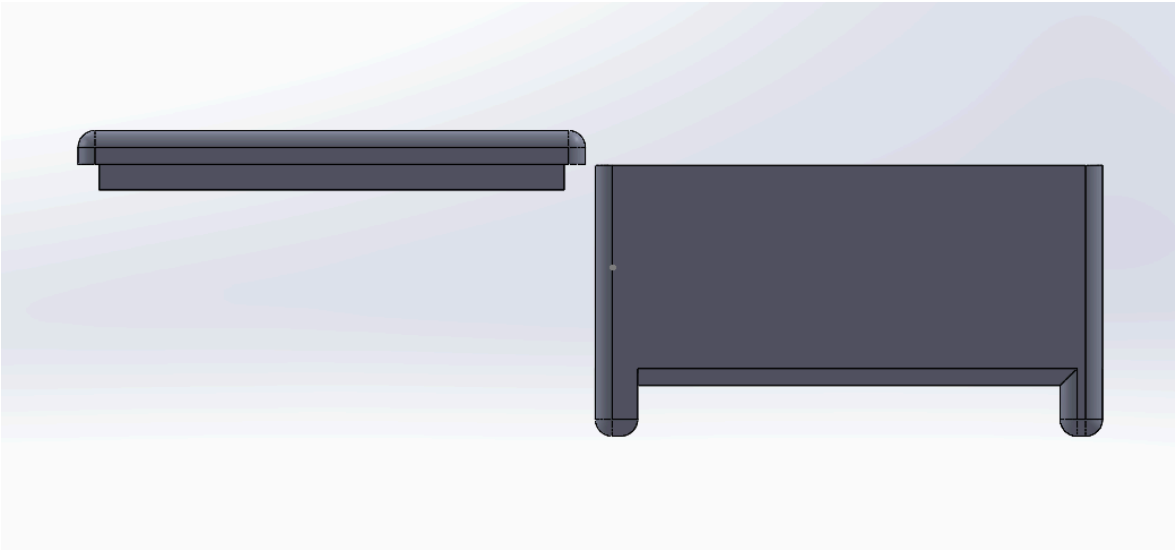
Lid 3D Images:

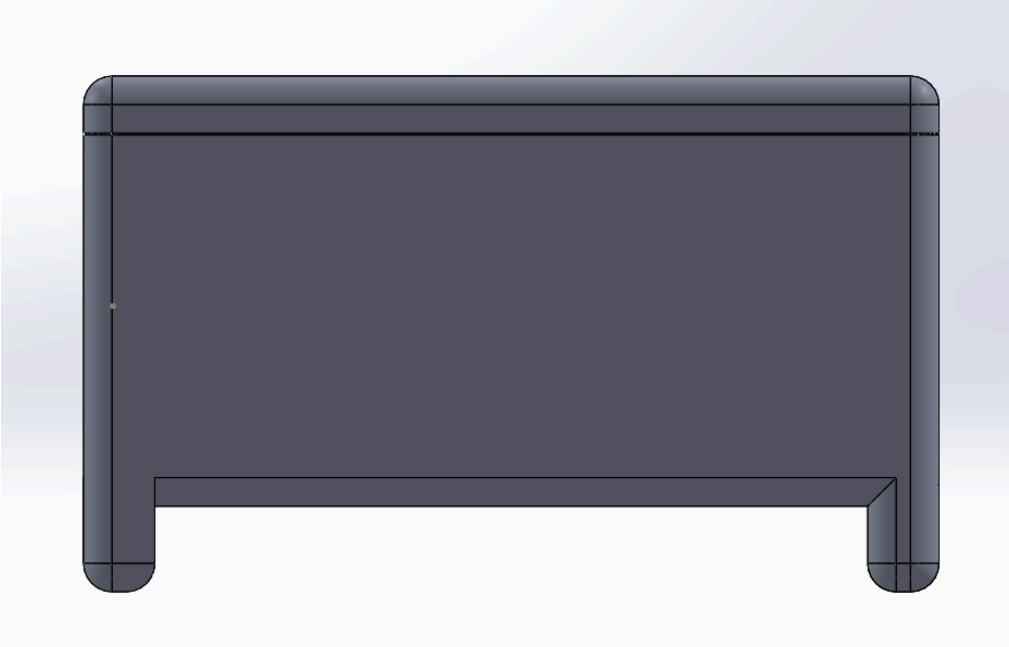


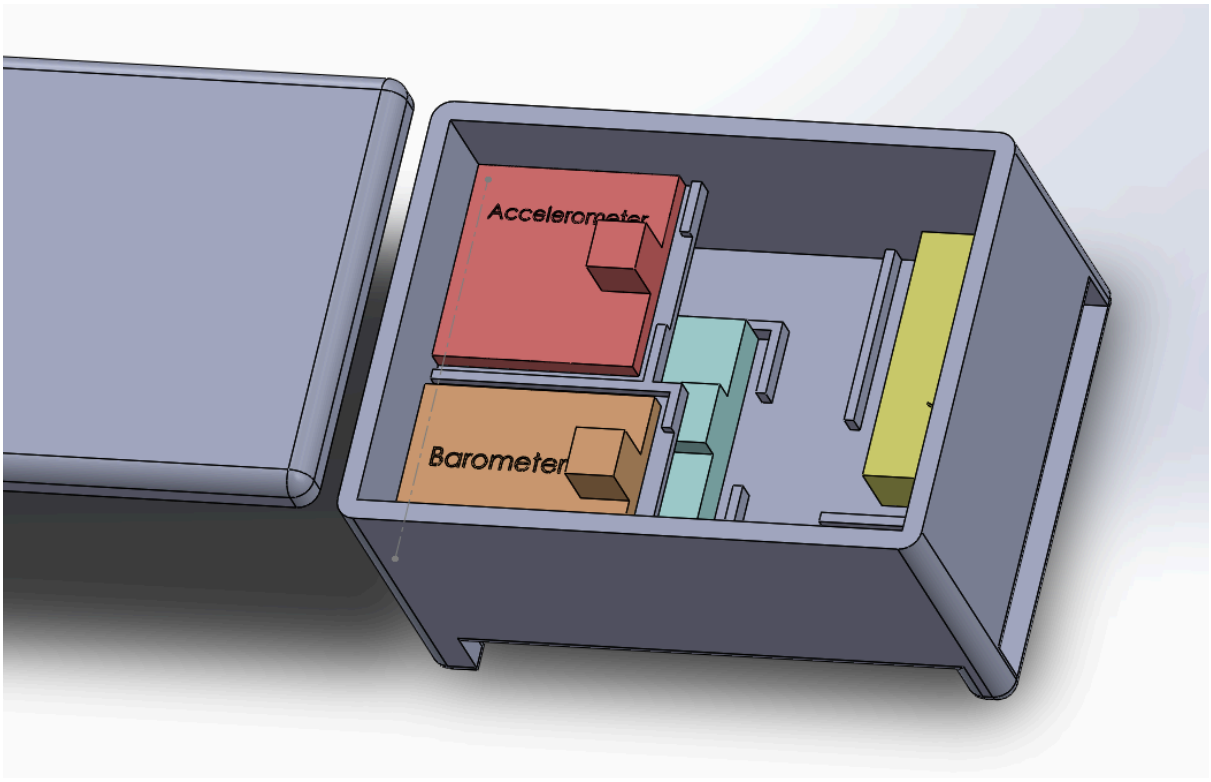
Assembly with Electrical Components and Lid:

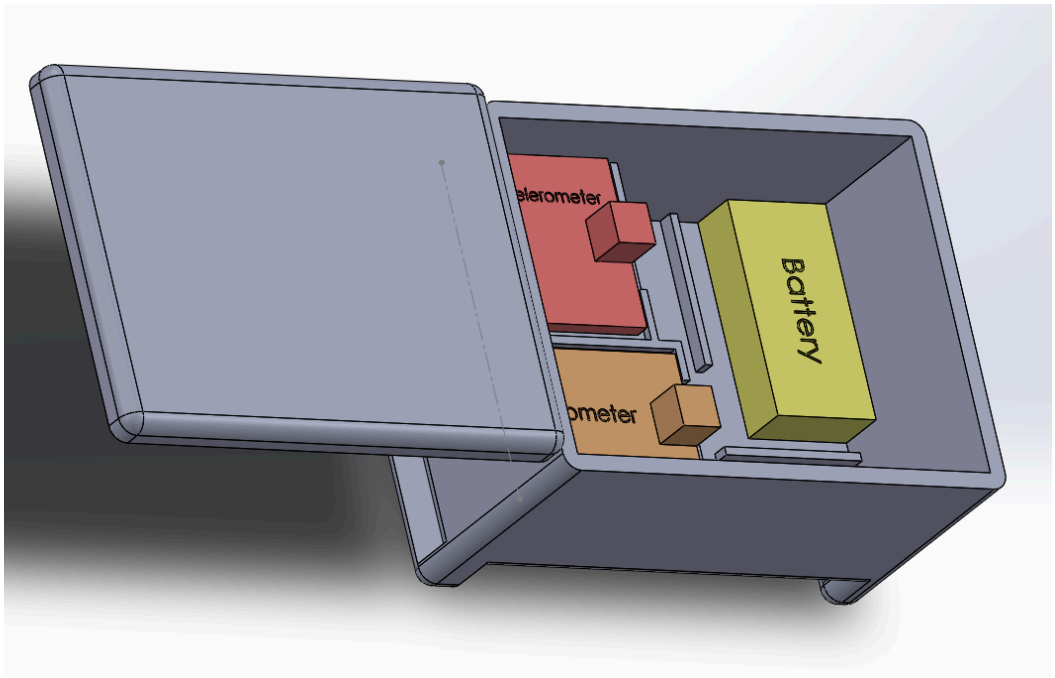
	Accelerometer
	Barometer
	Battery
	Beetle
	Vibration Motor
	3D Printed Case/Lid
	Wristband

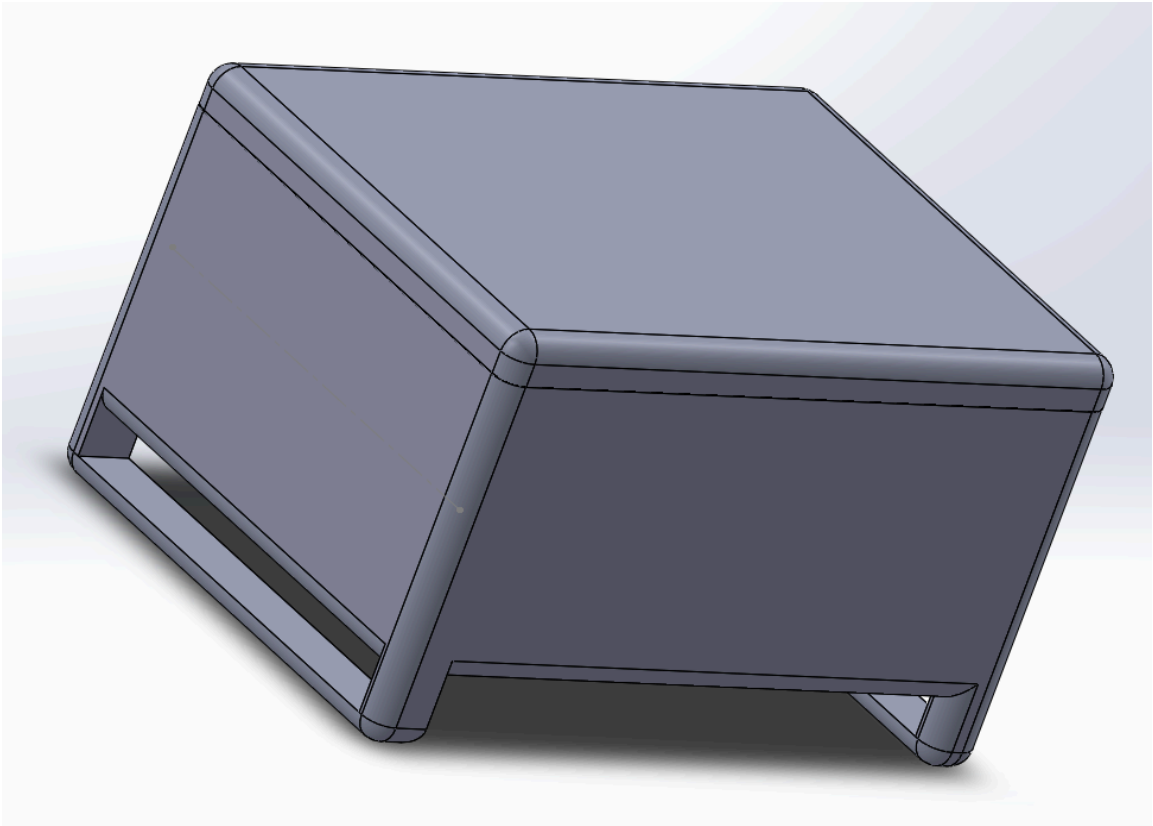




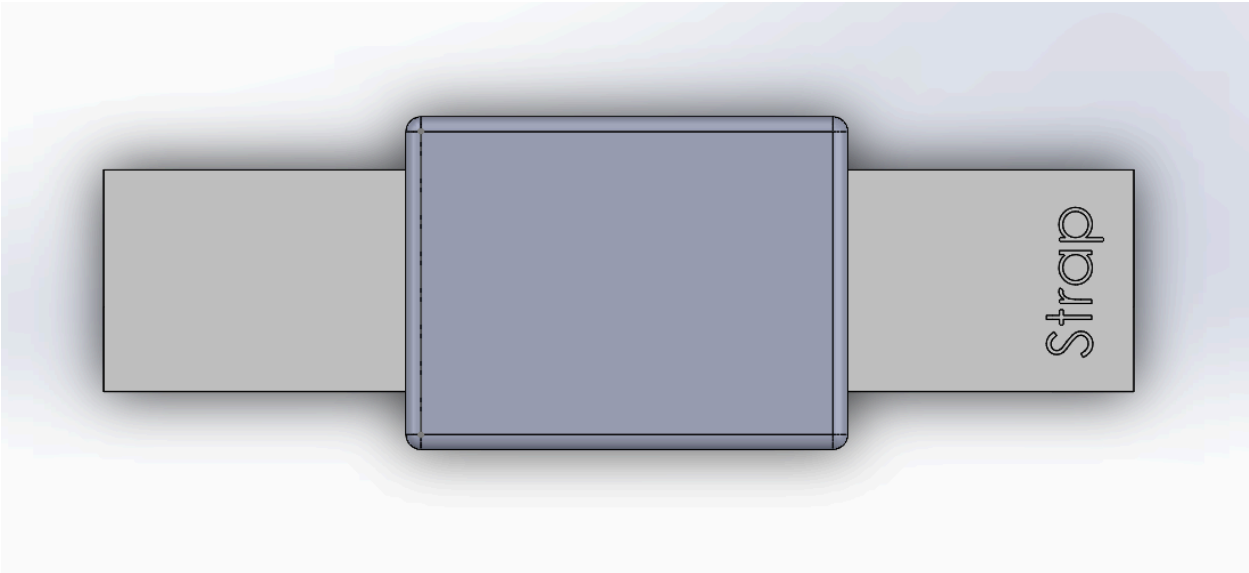


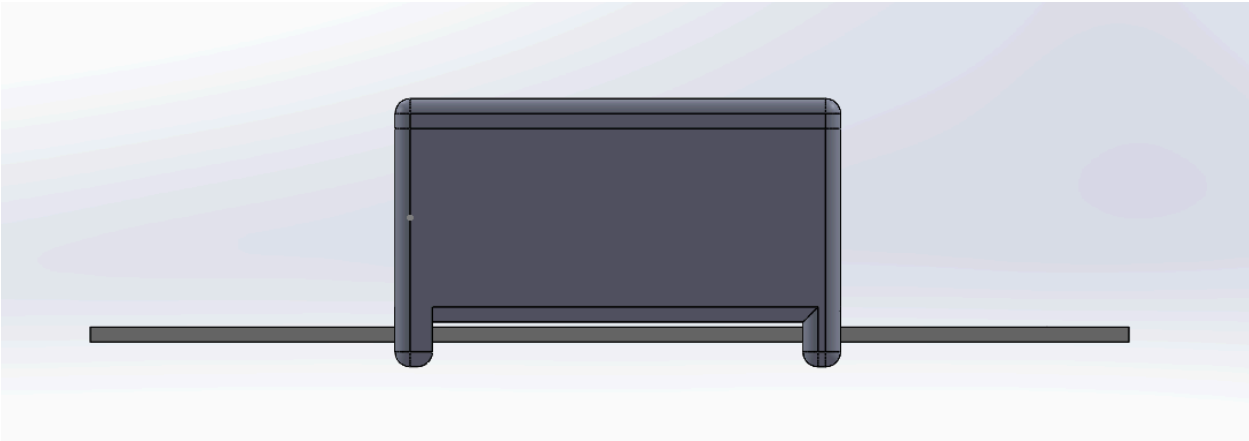

Barometer


Battery

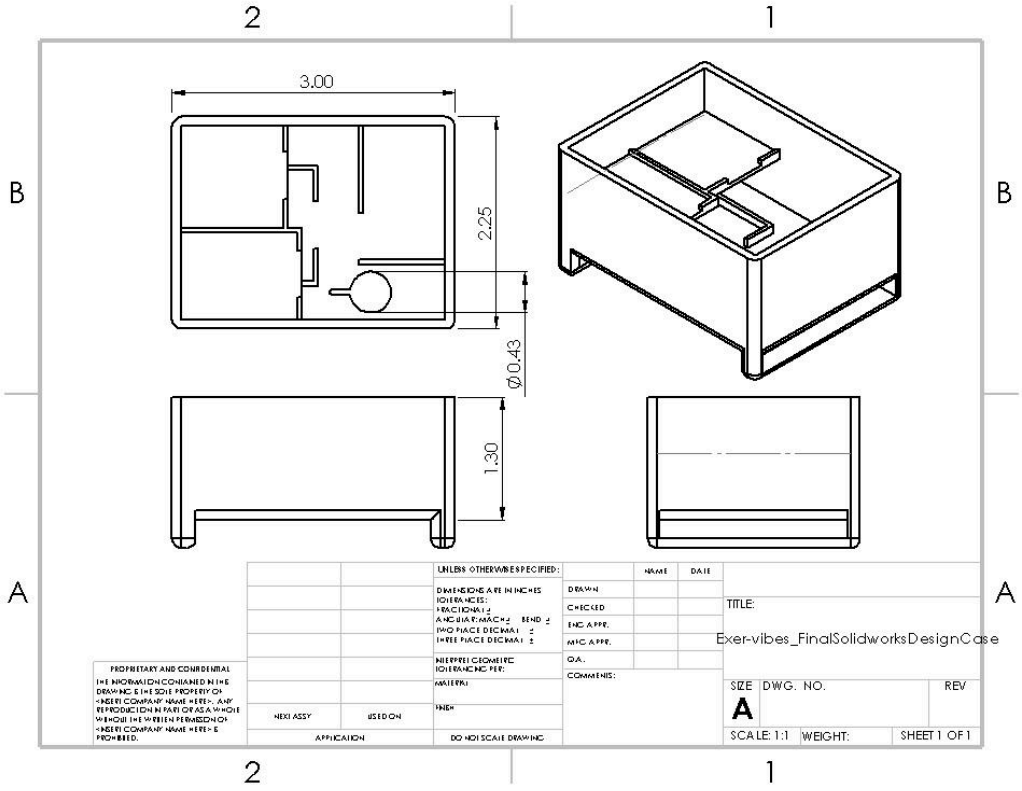


Closed Assembly with Strap included:





Solidworks Drawing of Case:



APPENDIX B: BILL OF MATERIALS

Our budget ended up being around $68, or $67.86 to be exact with the inclusion of tax, which was well under our maximum budget requirement of $150.

PURCHASE REQUEST FORM

The Henry Samueli School of Engineering

Team Name:	**Exervibes**	**Team Capt**	akyral@uci.edu
Lab Section Day/Time:	**Wednesday 3:00 PM**	**Date of Request:**	01/24/20
		Room #	
Account Name:*	ENGR7B: Fitness Tracker		

Quantity*	Unit of Measure*	Description*	Catalog #*	Unit Price*	Estimated Extended Price*	Actual Extended Price
1	1	Arduino Beetle with bluetooth	SKU:DFR0339	$14.90	$14.90	
1	1	Gravity: I2C Triple Axis Accelerometer - LIS2DH	SKU:SEN0224	$4.90	$4.90	
1	1	Gravity: i2C BMP280 Barometer	SKU:SEN0251	$6.90	$6.90	
1	6	3.7 V Battery	1205890	$3.48	$3.48	
1	1	Vibration motor	ROB-08449	$2.15	$2.15	
1	1	Navy/lemon Strap	B015NKHTQE	$12.50	$12.50	
1		3D Printing		$18.00	$18.00	
					$0.00	
					$0.00	
					$0.00	
					$0.00	
					$0.00	
				Subtotal	**$62.83**	
				***Tax rate:** **8.00%**	$5.03	
				TOTAL ORDER PRICE	**$67.86**	

APPENDIX C: ARDUINO CODE

```
/*
  Team Name: Exer-Vibes!

  Team Members:
  Akyra Lee
  Nikki Chi
  Syona Mehra
  Jenny tran

  Date of Last Edit: 3/10/21

  Code Summary:
  Our team's code was written in accordance to the project's two sensors, the accelerometer and barometer, and to comply with BLYNK.
  Our overall goals were to calculate the number of steps a user takes, allow them to set a step goal, and use a vibration motor to
  indicate when a goal is met. This overall involved creating a step count algorithm and an algorithm to calculate the user's steady
  state values. We also wished to calculate the number of flights of stairs a user climbed/descended. Both the step and stair count was
  displayed on the app. Numerous buttons and other value displays were also created on our app for confirmation, debugging, and user
  engagement purposes.

*/

//libraries required for barometer/accelerometer data collection
#include <Wire.h>
#include <DFRobot_LIS2DH12.h>
#include "DFRobot_BMP388.h"
#include "DFRobot_BMP388_I2C.h"
#include "SPI.h"
```

```
#include "math.h"
#include "bmp3_defs.h"
#include <SPI.h>
#include <Ethernet.h>
#include <BlynkSimpleSerialBLE.h>

#define BLYNK_PRINT Serial
#define CALIBRATE_Altitude

DFRobot_BMP388_I2C bmp388; //for Barometer
DFRobot_LIS2DH12 LIS; //for Accelerometer

//Personal authorization token - used to connect to bluetooth
char auth[] = "9lIT_DhFXKmk6DKfTB8p_Qq8O58-vipR";

//*************************************ALL
VARIABLES*************************************

//step count algorithm variables
int steps = 0; //keeps record of the total number of steps
bool Lets_Look_At_The_Forward_Swing = true; //assumes that the user moves the
hand wearing the device forward when they take their first step
bool Lets_Look_At_The_Backward_Swing = false; //assumes that the user's hand is not
moving backward when they take their first step

//steady state calibration variables
int max_steady_state_value; //declares a variable for the maximum steady state value
int min_steady_state_value; //declares a variable for the minimum steady state value
int array_that_holds_calibration_values[100]; //this array will hold the first 100 values
gathered from the accelerometer
int array_size = 100; //size parameter for the array

//stair count variables
float seaLevel; //used by the barometer to evaluate altitude
```

```
float current_Altitude; //declares a variable for the user's altitude
float past_Altitude; //declares a variable for a comparative altitude
int staircases = 0; //indicates that the initial number of stairs climbed is zero
float height_change = 0; //indicates that the change in altitude is initially zero

//global variables for the accelerometer values
int16_t x, y, z;

//declares that the vibration motor is located in digital port 2
int VIBE = 2;

//declares a variable for the goal
int SWOL_GOAL;

//used to declare the existence of a button the user presses on the app to calibrate their
steady state values, on is represented by 1 and off is represented by 0
int CALIBRATION_BUTTON;

//used to declare the existence of a button that turns on the vibration motor, on is
represented by 1 and off is represented by 0
int VIBE_BUTTON;

//used to declare the existence of a button that the user presses on the app to acquire
their initial altitude, on is represented by 1 and off is represented by 0
int ALTITUDE_BUTTON;

//used to declare the existence of a button that the user presses on the app to initiate
KEANU MODE (a simple gif), on is represented by 1 and off is represented by 0
int KEANU_BUTTON;

//used to declare the existence of a button that the user presses on the app to flash a
meme, on is represented by 1 and off is represented by 0
int SURPRISE_BUTTON;
```

```
//****************************************BLYNK APP
BUTTONS*************************************************************

//Step goal input will be in virtual port 3
BLYNK_WRITE(V3)
{
  SWOL_GOAL = param.asInt(); // assigns an incoming value from pin V3 to a variable
}

//button for calibrating initial altitude is in port V14
BLYNK_WRITE(V14)
{
  ALTITUDE_BUTTON = param.asInt();
}

//button for calibrating steady state values is in port V11
BLYNK_WRITE(V11)
{
  CALIBRATION_BUTTON = param.asInt();
}

//button that turns on the vibration motor for fun is in port V7
BLYNK_WRITE(V7)
{
  VIBE_BUTTON = param.asInt();
}

//button that initiates KEANU MODE is in port V13
BLYNK_WRITE(V13)
{
  KEANU_BUTTON = param.asInt();
}
```

```
//button that displays a surprise is in port V15
BLYNK_WRITE(V15)
{
  SURPRISE_BUTTON = param.asInt();
}


void setup() {
  Wire.begin(); //communicates with I2C bus line
  Serial.begin(115200); //baud rate

  pinMode(VIBE, OUTPUT); //declares the VIBRATION MOTOR to be output

  //prepares accelerometer
  while (!Serial);
  delay(100);
  while (LIS.init(LIS2DH12_RANGE_16GA) == -1) { //Equipment connection exception
or I2C address error
    Serial.println("No I2C devices found");
    delay(1000);
  }

  //prepares Blynk
  Blynk.begin(Serial, auth);

  //prepares barometer
  bmp388.set_iic_addr(BMP3_I2C_ADDR_SEC);
  while (bmp388.begin()) {
    Serial.println("Initialize error!");
    delay(1000);
  }

  //used to prepare altitude data gathered by the barometer
  delay(100);
```

```
  seaLevel = bmp388.readSeaLevel(525.0);
  Serial.print("seaLevel : ");
  Serial.print(seaLevel);
  Serial.println(" Pa");
}

void loop() {

  Blynk.run(); //starts running Blynk

  //used to acquire altitude data from barometer
#ifdef CALIBRATE_Altitude
  /* Read the calibrated altitude */
  float altitude = bmp388.readCalibratedAltitude(seaLevel);
  //Serial.print("calibrate Altitude : ");
  //Serial.print(altitude);
  //Serial.println(" m");
#else
  /* Read the altitude */
  float altitude = bmp388.readAltitude();
  //Serial.print("Altitude : ");
  //Serial.print(altitude);
  //Serial.println(" m");
#endif
  delay(100);

  //inserts 100 acceleration y values into an array
  for (int i = 0; i < 100; i++)
  {
    LIS.readXYZ(x, y, z); //reads values from accelerometer
    array_that_holds_calibration_values[i] = y; //inserts the y values of the accelerometer
data into the array
  }
```

```
  acceleration(); //calls the acceleration function

  //value displays on BLYNK
  Blynk.virtualWrite(V12, 1); //displays team logo on app
  Blynk.virtualWrite(V6, bmp388.readTemperature()); ////displays the current altitude to verify that the barometer is connected/working
  Blynk.virtualWrite(V8, altitude); //displays the current altitude to verify that the barometer is connected/working
  Blynk.virtualWrite(V10, max_steady_state_value); //displays the maximum steady state value to verify that is was collected
  Blynk.virtualWrite(V9, min_steady_state_value); //displays the minimum steady state value to verify that is was collected
  Blynk.virtualWrite(V1, steps); //displays the step count on the app

  //considers if the step goal is ever met
  if (steps == SWOL_GOAL)//if the number of steps ever equals the goal
  {
    digitalWrite(VIBE, HIGH); //turns the vibration motor on for a second
    delay(1000); //waits 1 second
    digitalWrite(VIBE, LOW); //turns the vibration motor off
    delay(500); //waits half a second
    Blynk.notify("SWOL GOAL MET YAYYYYYYYYY"); //sends the user a message that tells them they made their goal
  }

  //considers if the button that turns on the vibration motor is ever switched on
  //used to verify that the vibration motor is connected/working
  while (VIBE_BUTTON == 1) //1 = on, 0 = off
  {
    digitalWrite(VIBE, HIGH); //turns on the vibration motor
    Blynk.notify("WE'RE VIBING"); //spams the user with an obnoxious message
    Blynk.virtualWrite(V12, 2); //displays a new image, this time of the joker and peter parker dancing
  }
```

```
  //ensures that if the vibration motor button is switched off the vibration motor does not remain on
  if (VIBE_BUTTON == 0)
  {
    digitalWrite(VIBE, LOW);
  }

  //considers if the button used to calibrate the steady state values is pressed
  if (CALIBRATION_BUTTON == 1)
  {
    //calls the function that find the maximum calibration value
    int maximum_CALIBRATION_value = find_me_the_maximum_please(array_that_holds_calibration_values, 100);
    //calls the function that finds the minimum calibration value
    int minimum_CALIBRATION_value = find_me_the_minimum_please(array_that_holds_calibration_values, 100);

    //going to be inserted into the accelerometer step count algorithm
    max_steady_state_value = maximum_CALIBRATION_value;
    min_steady_state_value = minimum_CALIBRATION_value;

    Blynk.notify("CALIBRATION COMPLETE"); //sends a message that confirms to the user that their steady state values have been calibrated
  }

  //considers if the attitude button is ever pressed
  if (ALTITUDE_BUTTON == 1)
  {
    past_Altitude = altitude; //sets the past altitude equal to the first altitude reading acquired
    Blynk.notify("ALTITUDE INITIALIZED"); //sends a message that confirms to the user that their initial altitude has been calibrated
  }
```

```
 //considers if the keanu button on the app is ever switched on
 //used for fun/experimental purposes
 while (KEANU_BUTTON == 1)
 {
  //while the button is switched on a jif of the keanu meme will replace the logo image
  Blynk.virtualWrite(V12, 3); //displays tall keanu image
  delay(500); //waits half a second
  Blynk.virtualWrite(V12, 4); //displays smol keanu image
  delay(500); //waits half a second
  Blynk.notify("KEANU MODE INITIATED"); //spams user with obnoxious message
 }

 //considers if the surprise button is ever pressed
 if (SURPRISE_BUTTON == 1)
 {
  Blynk.virtualWrite(V12, 5); //flashes a surprise meme image
  Blynk.notify("YEAHHHHHHHHHHHHH"); //sends user obnoxious message
 }


 //**********************************************************STAIRCLIMBING
ALGORITHM************************************************************
 current_Altitude = altitude; //sets the user's current altitude equal to the
accelerometer value

 height_change = abs(current_Altitude - past_Altitude);
 //calculates the change in altitude and sets it equal to the height change variable
 //uses absolute value function to consider both stairs ascension and descension

 if (height_change >= 2.4 && height_change <= 4)
  //Height of ET staris is ~8 ft = 2.4 meters
  //maximum stair height is 4 meters
 {
```

```
    staircases ++; //adds one to the number of staircases climbed
    past_Altitude = current_Altitude; //resets the past altitude value
  }
  Blynk.virtualWrite(V5, staircases); //prints the number of staircases
climbed/descended by user to the app
}

//FUNCTION USED TO DETERMINE MAXIMUM STEADY STATE VALUE
int find_me_the_maximum_please(int *vals, int array_size)
{
  int i = 1;
  //we want to observe the second value in the array
  int you_are_the_MAXIMUM = array_that_holds_calibration_values[0];
  //sets the comparative value equal to the first value in the array

  while (i < 100)
    //as long as i does not exceed the maximum number of values in the array
  {
    if (array_that_holds_calibration_values[i] > you_are_the_MAXIMUM)
      //checks if the value in the array is bigger than the current maximum
    {
      you_are_the_MAXIMUM = array_that_holds_calibration_values[i];
      //if the value in the array is bigger than the current maximum this set that value as
the new maximum
    }
    i++;
    //if the value in the array is not bigger than the current maximum the next value in
the array will be checked
  }
  return you_are_the_MAXIMUM;
  //function returns the maximum value of the array
}

//FUNCTION USED TO DETERMINE MINIMUM STEADY STATE VALUE
```

```
int find_me_the_minimum_please(int *vals, int array_size)
{
  int i = 1;
  //we want to observe the second value in the array
  int you_are_the_MINIMUM = array_that_holds_calibration_values[0];
  //sets the comparative value equal to the first value in the array

  while (i < 100)
    //as long as i does not exceed the maximum number of values in the array
  {
    if (array_that_holds_calibration_values[i] < you_are_the_MINIMUM)
      //checks if the value in the array is bigger than the current maximum
    {
      you_are_the_MINIMUM = array_that_holds_calibration_values[i];
      //if the value in the array is bigger than the current maximum this set that value as
the new maximum
    }
    i++;
    //if the value in the array is not bigger than the current maximum the next value in
the array will be checked
  }
  return you_are_the_MINIMUM;
  //function returns the maximum value of the array
}

//function used the acquire data from the accelerometer
void acceleration(void)
{
  delay(100);
  LIS.readXYZ(x, y, z);
  LIS.mgScale(x, y, z);
  //Serial.print("Acceleration x: "); //print acceleration
  //Serial.print(x);
  //Serial.print(" mg \ty: ");
```

```
  //Serial.print(y);
  //Serial.print(" mg \tz: ");
  //Serial.print(z);
  //Serial.println(" mg");

  //**********************************************************STEP COUNT
ALGORITHM*****************************************************
  //Goal: calculate the number of steps taken using the accelerometer y values

  if (Lets_Look_At_The_Forward_Swing == true) //means that we want to look at the
forward hand
  {
    //Serial.println("hand is moving forward"); //Used to debug - verifies that the program
recognizes the hand is moving forward
    if (y > max_steady_state_value)
    {
      //Serial.println("left foot took a step"); //Used to debug - when right arm moves out
the left foot takes a step
      steps = steps + 1;
      Lets_Look_At_The_Backward_Swing = true; //going to start the other if statement
below
      Lets_Look_At_The_Forward_Swing = false; //halts this if statement, stops more
steps from being counted and moves on to the next if statement
    }
  }

  if (Lets_Look_At_The_Backward_Swing == true) //means that we want to look at the
backhand
  {
    //Serial.println("hand is moving backward"); //Used to debug - verifies that the
program recognizes the hand is moving backward
    if (y < min_steady_state_value)
    {
```

```
    //Serial.println("right foot took a step"); //Used to debug - when right arm moves back the right foot takes a step
    steps = steps + 1;
    Lets_Look_At_The_Forward_Swing = true; //going to start the other if statement above
    Lets_Look_At_The_Backward_Swing = false; //halts this if statement, stops more steps from being counted and hops back to the other if statement
   }
 }

}
```

APPENDIX D: REFERENCES

Douglas-Walton, Josh. *A Study of Fitness Trackers and Wearables*. 20 May 2020,www.hfe.co.uk/blog/a-study-of-fitness-trackers-and-wearables/#:~:text=Fitness%20trackers%2C%20as%20we%20know,time%20how%20to%20combat%20obesity.

Lynch, Courtney. "Stay Fit with the Best Fitness Trackers You Can Buy!" *Android Central*, Android Central, 4 Feb. 2021, www.androidcentral.com/best-fitness-trackers#:~:text=Whether%20you%20want%20the%20look,Xiaomi%2C%20to%20name%20a%20few.

"Who Invented the Pedometer? • Pedometer Reviews - Each Step You Take." *Pedometer Reviews - Each Step You Take*, 29 Apr. 2014, eachstepyoutake.com/who-invented-the-pedometer/.