It’s Everyday Code

Daria Shkel, Jeremy Chang, Noah Paige, Jessica Arias, Michelle Saldana Rivas

ENGR 7B: Fitness Tracker
University of California, Irvine
Winter 2019

Table of Contents

Executive Summary

The main goal we focused on was creating a fitness watch that was compact in design, cost effective, and accurate. We achieved this goal by creating a compact modular design that will hold all of our electrical components. By creating a compact design there will be less material being used and a lower cost. In terms of accuracy, continuously testing our arduino code and modifying it to maximize the accuracy of step count and stairs climbed.

Problem Definition

Introduction

The objective of the project is to create an affordable and practical fitness tracker. To do so, we followed a specific process of planning, designing, coding, and fabricating. Programs such as TinkerCAD, SolidWorks, MIT App Inventor were necessary for the creation of the device. Each member of the group was required to collaborate and participate in the engineering process of the project.

Technical Review / Background

A fitness tracker is a device that monitors the activity of the user, usually the steps they have taken. Fitness devices can also track how many stairs climbed, how many calories lost, how fast your heart is beating, and more. The origin of the fitness tracker can be traced back to the heart-rate monitor. When monitoring a person's heartbeat during multiple times of the day was required, the idea of the a portable tracker was created and further evolved into a modern day fitness tracker. Usually worn on the wrist or attached to a type of belt, fitness trackers measure daily activity. Fitness trackers began to grow in popularity in 2012 when Pebble launched their Kickstarter device. The success of the device created a pathway for future activity trackers and smart watches. Some of the most notable activity monitors include the Garmin Vivofit 3, Withings Activite Pop, and Fitbit Zip. The price of each tracker depends on the brand and available functions, but most vary between $40 to $100.

Design Requirements

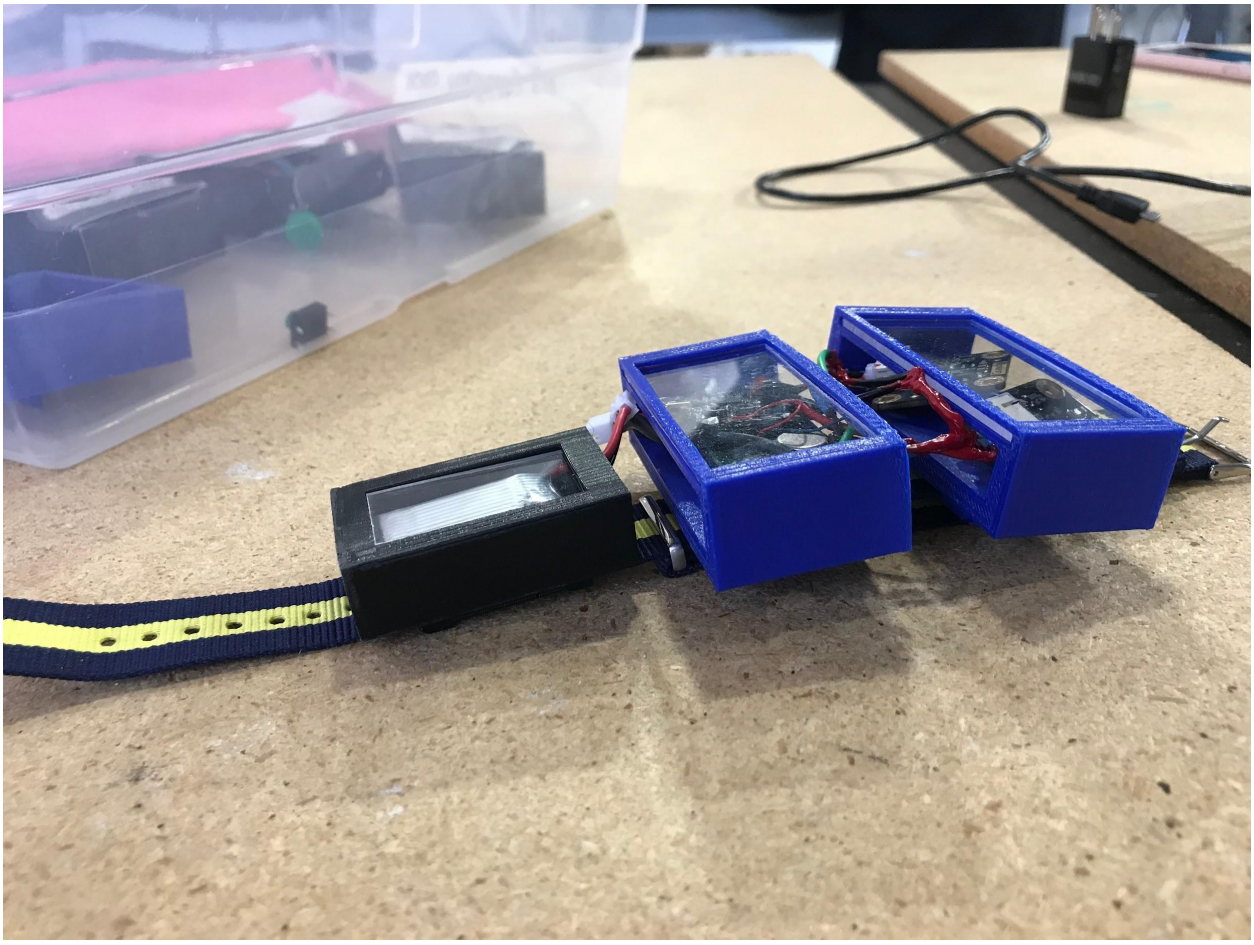
1. Size and Durability Requirements
    a. Must be able to attach and detach from any user's wrist without restricting movement
    b. No more than 4 inches in width
    c. All hardware and wires must be secured to withstand contact and movement
2. Fitness Tracking and Mobile App Requirements
    a. Consistently track the
        i. Step count
        ii. Altitude
    b. Allow user to input goals for these variables into mobile app
    c. Show update of data on mobile app when activity is completed
    d. Indicate progress or goal achieved with indicator on wristband
    e. The accuracy should be within 90% of actual step count and altitude
3. Power Requirements
    a. Must be able to operate for at least 1 hour using onboard battery source
4. Safety Requirements
    a. All wires and connectors are completely covered and/or insulated
    b. All electronic components must be secured to the wearable assembly
5. Cost Requirements
    a. The total as-built replacement cost of the fitness tracker must be at most $150
    b. The costs of all components and materials must be listed in the Bill of Materials, in which the Fair Market Value of each component is provided
6. Product Deliverables
    a. The wearable fitness tracker wristband with electronics
    b. Final Arduino Code used by wearable fitness tracker
    c. Final MIT App file
    d. Final Gantt Chart & Bill of Materials
    e. Final Presentation & Demonstration of wristband with app function
    f. Final Design Report

Design Description

Summary of Design

Each member of the group wanted to create a device that was accurate as well as compact and inexpensive. During the first week of the course, each member had an idea of what they wanted the project to look like. After

discussion, we came to the agreement that we would make the fitness tracker three components on one wristband instead of one large block. The process of making our design three separate components was much harder than we expected: however, it add a unique factor to our project. In further making our device unique, the group decided to showcase the electrical components instead of hiding them. Most fitness trackers do not display any electrical part, but our device lets the user see what is really going on inside. Our device weighing about 200 grams is 4.5 inches in length and 2 inches in width. The total cost of the device is $92.79.

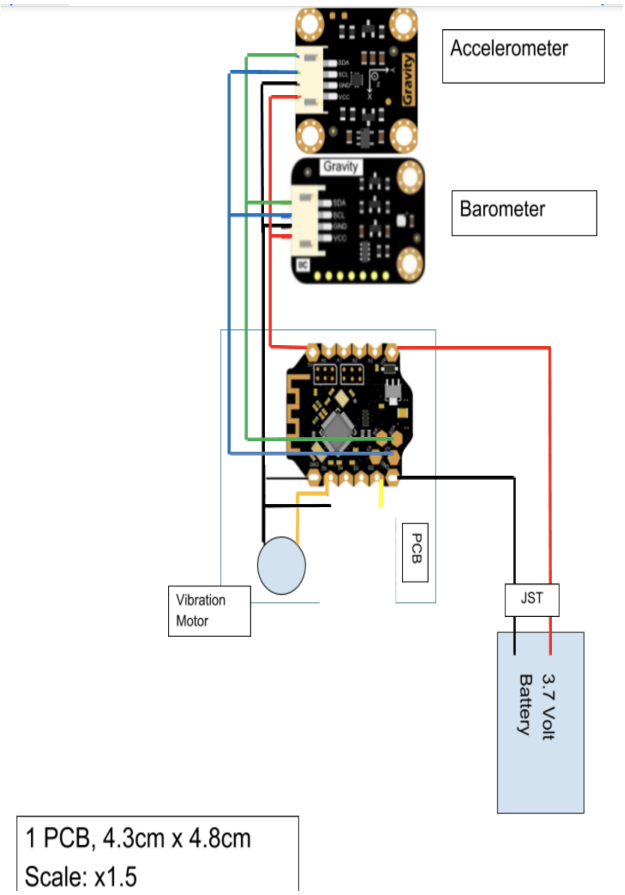


Fabrication Design

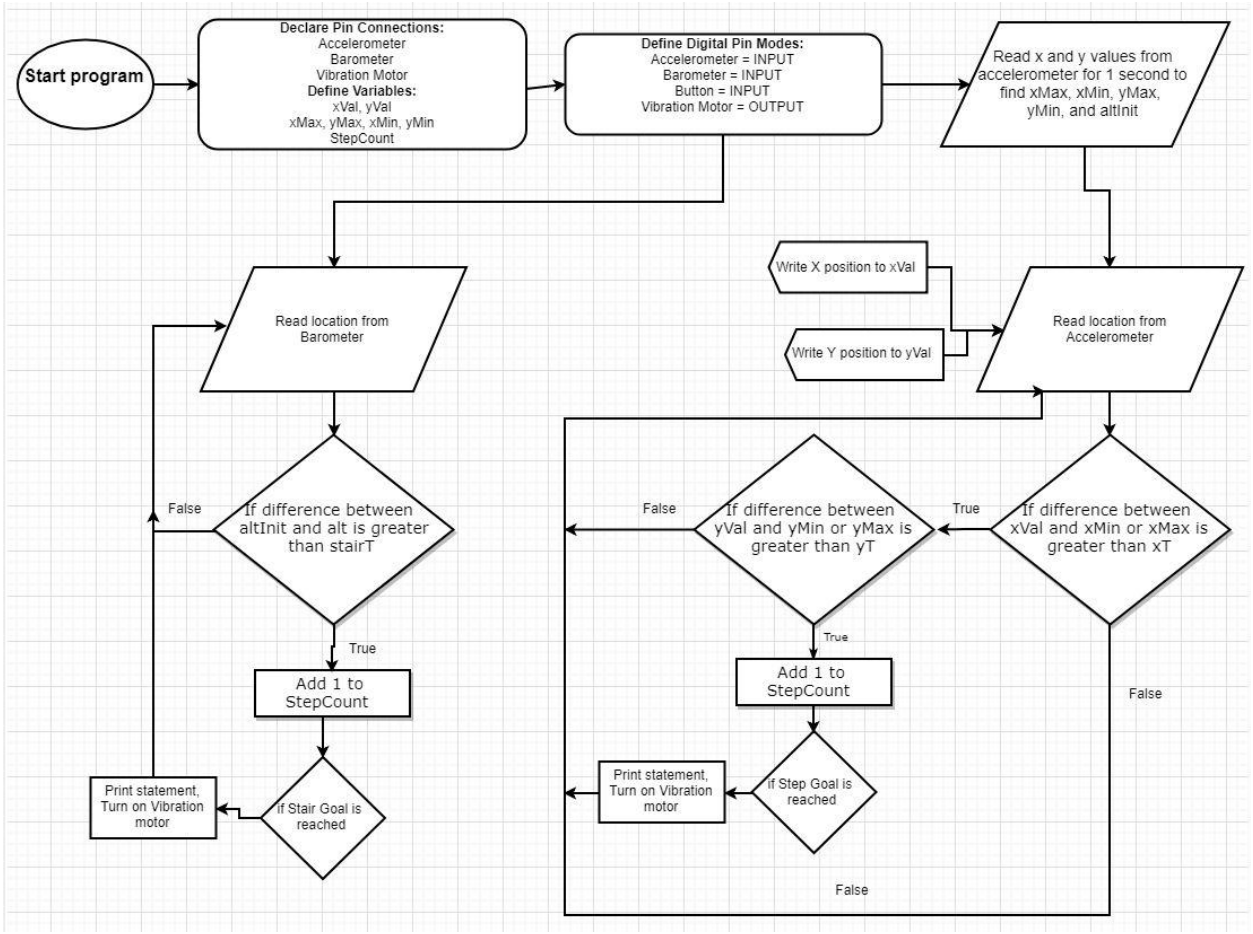
From the beginning, we aimed to reduce the thickness of our fitness watch as much as possible. We decided on a modular design as a solution to avoiding a chunky all-in-one design. In Solidworks, we created modules with different specifications to contain different components of the fitness watch. As a team, we agreed that 3D printing these parts was the most effective method in ensuring our fitness watch matched our Solidworks design measurements. After printing several iterations, we saw an opportunity to minimize even more thickness and space in our fitness watch. We shortened the lengths and widths of one of our modules to be just big enough to hold the electrical components leaving no excess space. The final print and assembly came out beautifully with the teamwork of our amazing diverse team.

Circuit Design

We choose to house the accelerometer and the barometer together on top, and the arduino and vibration motor in the middle. By centralizing the arduino, we easily connected the bus lines from the accelerometer and barometer, as well as the vibration motor and battery. We decided to take the button out of our diagram, because it would have cluttered the wiring as well as added a significant amount of time to our production. We placed the battery in an isolated box at the bottom for a more convenient connection to the back of the arduino. We also chose to implement a PCB board under the arduino. This allowed us to extend the range in which we could connect the SCL and SDA lines. The SCL and SDA OUT lines from the top plate were then soldered at the end of the lines.



Algorithm Design



Pseudocode

Initialization:

1. Import all libraries necessary for program
2. Declare sensor pin connections
3. Define Variables:
    a. int vibM = 8 //pin # of vibration motor
    b. int stepCount = 0 //counter for user's steps
    c. int stairCount = 0 //counter for flights of stairs climbed by user
    d. int stepGoal = 10 //hard goal for steps
    e. int stairGoal = 2 //hard goal for flights of stairs
    f. int xMin = 1000 //in case all values read are positive

g. int xMax = -1000 //in case all values read are negative
h. int xVal = 0 //current x coordinate (location) of arm
i. int xT = 300 //front/back swing x threshold)
j. int yT = 200 //front/back swing y threshold)
k. float stairT = 2.75
l. float altitude
m. float altInit //globalized for steadyState()
n. int loopCounter = 0 //regulate barometer readings

Setup ():

1. Declare Pin Modes for accelerometer, barometer, and vibration motor
2. Run steadyState() to attain xMax, xMin, and altInit

Loop ():

1. Get position values on accelerometer
    a. Write X and Y values into xVal and yVal respectively
    b. Check to see if xVal has gotten to or surpassed (>=) the xMax or xMin values
        i. If true, increment step count number by 1
        ii. If false, loop back from step 1
    c. If step count reaches step goal, activate vibration after 5 millisecond delay
2. Every 5 times, attain altitude values from barometer
    a. Check to see if altitude have gotten to or surpassed (>=) the stair threshold value
        i. If true, increment stair count number by 1
        ii. If false, loop back from step 2
    b. If stair count reaches stair goal, activate vibration after 5 millisecond delay

END

Our code structure is as follows:

- First, we import all our libraries and declare all our variables.
- Then we go into our setup() method where we initialize all of our pins and start our program. At the end of setup() we jump to our steadyState() method where we update the initial position from the accelerometer and initial altitude from the barometer. This is an attempt to cancel out any noise and calibrate the device to the user. At first, we only took the initial position value of the user but with the if loop we put there instead we were able to get more readings and more accurate results. This call for the steadyState() method was

originally in the loop() method, however, since we only wanted the code to run once, we moved the call for the method into setup().

- Then we flow into loop() where we call our step count method every time through the loop and our stairCount() method every 5 times through the loop. Originally we had our button code here as well but we ended up removing it due to an excess of problems that arose.
- In countingStep(), we call the acceleration() code which updates the current y position of the accelerometer. Then we compare the difference between current and initial position and if that value surpases our hardcoded threshold value we count a step. Then if our step goal is reached we produce a quick buzz from the vibration motor. Initially we had our code checking both x and y values, however we found that this did not work once we actually started walking vs just swinging our arm. Also, in our final product we were using delays to time our steps. So if given more time we would have tried to figure out a way for our prototype to count steps without needing to know the rhythm required for step counting.
- Our stairsClimbed() method works similarly to countingStep() except instead of comparing position we are comparing altitude.
- Finally, due to having had to remove the button code, we are unable to stop our program so it continues to run until we unplug it from a power source.

We structured our code this way as it seemed to be the most logical method for us. Along the way we were constantly changing and updating stepCount(), steadyState() and other components of our algorithm as problems and bugs arose. A more detailed description of some of the main problems we encountered can be found in the “Discussion” section below.

User Interface Design

In terms of our UI design,we kept it very simple and user-friendly. Our users were able to see their step count as well as the their flight of stairs climbed all in one screen. Initially we had it set up so the user can input their step goal. However, due to our time constraint we weren’t able to get that component to work. To make up for that we created a hard goal instead.

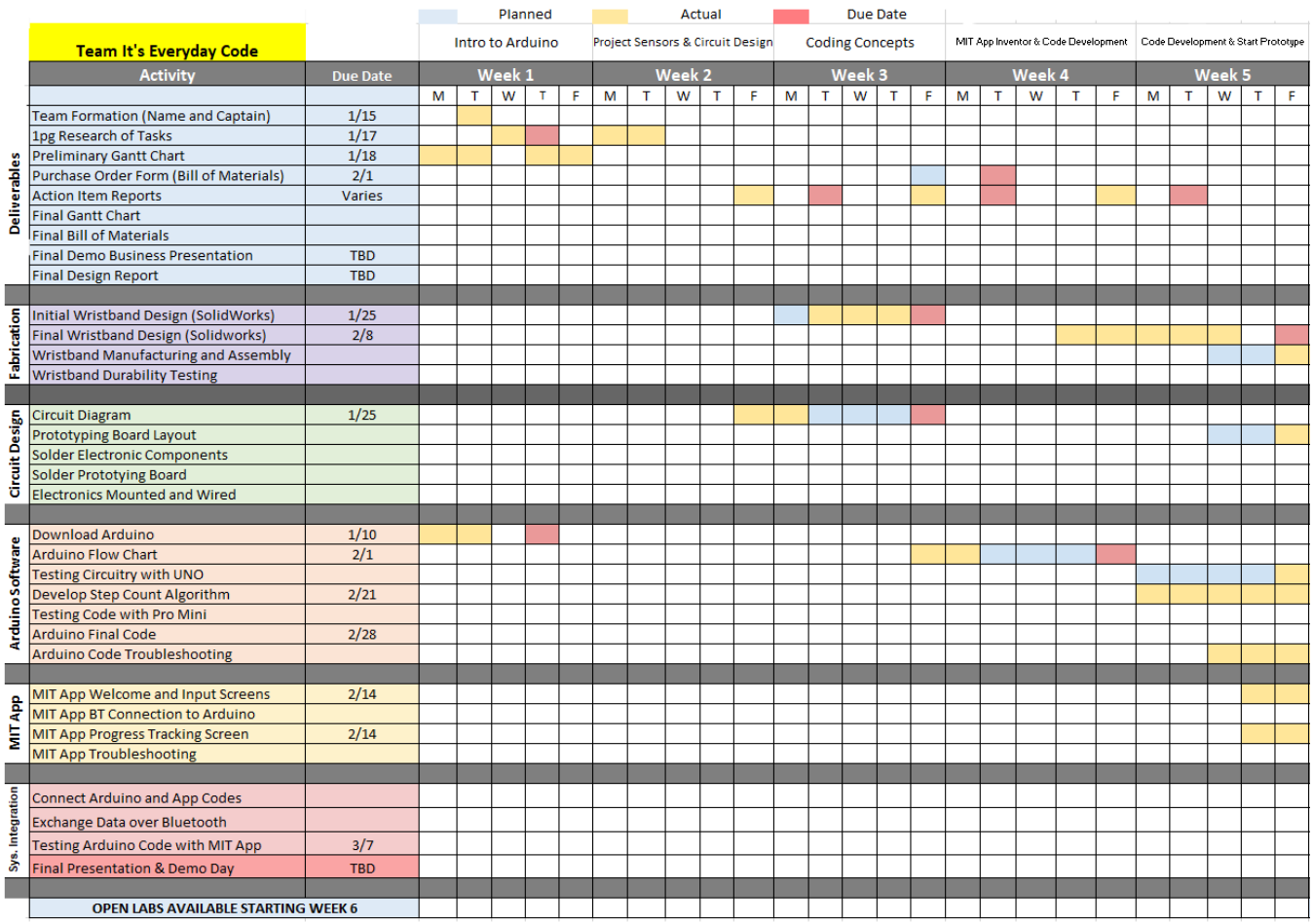
Action Items

Task Assignment

By acknowledging our strengths and weaknesses, we divvied up the tasks accordingly. The arduino code was compiled and refined by Daria and JJ. Noah designed the wiring diagram and soldered/arranged the wires together within the boxes. The 3d printing designs and manufacturing were completed by Michelle. The MIT app was designed and tested by Jessica.

Gantt Chart

Weeks 1-5:



Weeks 6-11:

	Team It's Everyday Code																														
	Activity	Due Date	Week 6					Week 7					Week 8					Week 9					Week 10					Finals Week				
			M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F
Deliverables	Team Formation (Name and Captain)	1/15																													
	1pg Research of Tasks	1/17																													
	Preliminary Gantt Chart	1/18																													
	Purchase Order Form (Bill of Materials)	2/1																													
	Action Item Reports	Varies																													
	Final Gantt Chart																														
	Final Bill of Materials																														
	Final Demo Business Presentation	TBD																													
	Final Design Report	TBD																													
Fabrication	Initial Wristband Design (SolidWorks)	1/25																													
	Final Wristband Design (Solidworks)	2/8																													
	Wristband Manufacturing and Assembly																														
	Wristband Durability Testing																														
Circuit Design	Circuit Diagram	1/25																													
	Prototyping Board Layout																														
	Solder Electronic Components																														
	Solder Prototying Board																														
	Electronics Mounted and Wired																														
Arduino Software	Download Arduino	1/10																													
	Arduino Flow Chart	2/1																													
	Testing Circuitry with UNO																														
	Develop Step Count Algorithm	2/21																													
	Testing Code with Pro Mini																														
	Arduino Final Code	2/28																													
	Arduino Code Troubleshooting																														
MIT App	MIT App Welcome and Input Screens	2/14																													
	MIT App BT Connection to Arduino																														
	MIT App Progress Tracking Screen	2/14																													
	MIT App Troubleshooting																														
Sys. Integration	Connect Arduino and App Codes																														
	Exchange Data over Bluetooth																														
	Testing Arduino Code with MIT App	3/7																													
	Final Presentation & Demo Day	TBD																													
	OPEN LABS AVAILABLE STARTING WEEK 6																														

Evaluation

Testing:

- **3D Prototype:**

  To ensure the safety of the user we made sure to insulate all the wires especially those that ran between our modules. To check if there were any loose and unsafe connections we used the voltmeter to test for connectivity. For extra protection we then planned to wrap the wires in electrical tape. However, due lack of availability of necessary supplies and time constraints we ended up wrapping the wires in regular construction tape. We also filed or reprinted our prototype if it happened to have any jagged or sharp edges. We tested the security and durability of supports by conducting stress tests on specific portions of the prototype.

- **Circuit:**

  Within the first few weeks we gathered data from the accelerometer and barometer for use in our step count and stair count codes. With the accelerometer we established thresholds found by calculating average x, y and z position of accelerometer for one step. These values were used in our Arduino step count code. For the stair count code we used the barometer to measure average change in altitude for one flight of stairs climbed. This value was held in the *stairT* or "stair threshold" variable.

  In terms of testing the circuit itself, there were many steps that we followed. First, we established our plan for general component connections with the preliminary wiring diagrams. Then we tested our wiring diagram design with a temporary circuit to see if it would work. Then once connections and bus lines proved to be effective we began the final component soldering process. Throughout the soldering process we used the voltmeter to check connectivity. This was to make sure that wires that were supposed to be connected were (mostly used for bus lines) and that wires that should not be touching weren't. If connectivity tests failed we added or removed solder, replaced the wire, or we added extra insulation at potential weak points/ points of cross connectivity. We were also continuously checking and adjusting length and bulkiness of wires and connections to fit our small and compact modular design.

- **Arduino Code/MIT App Code:**

In order to test and troubleshoot our codes we stuck with a few basic methods. For one we used comments so that we could better follow our code and make it easier to read for outside parties. Then we added print statements so that we could see what information and data the code was processing and at what point while the code was running. This gave us a better idea of the logic behind our code. Then, so we could go between the code we used for testing the circuit with the laptop and testing the circuit with the MIT code more easily, we used a boolean variable. This saved us time as it kept us from having to comment out all the print statements each time we switched from laptop to MIT app testing. Lastly to help with troubleshooting we continuously compared our complex broken code with previous renditions of working code. That way when something went wrong it was easier to backtrack in our code and figure out what was causing the error and fix it.

Results

In most areas our prototype performed up to or exceeded our predictions. In terms of weight and comfort our prototype performed better than predicted. Our design came out to be quite light, only slightly heavier than a standard fitness band. Our prototype was also comfortable on the wrist due to its adjustable nylon band, lightweight components, and distributed weight. However, due to its width across the band, it did not sit very securely on slim wrists. Perhaps if we reduced the space between modules a bit more we could have accounted for this issue. Aesthetically, due to our polycarbonate lids, clean prints and clean soldering, our model was pleasing to the eye, exceeding our expectations in that regard. In terms of durability of the device itself we did not run into any issues. Our supports from the 3D print were solid and secure and were strong enough to withstand wear and tear from standard daily functions. Unfortunately, some of our soldering connections did not withstand being crammed into the module; specifically the ones in the main module with the arduino beetle and the connection leading to the battery module.

For overall performance, our final product *did* count steps and count stairs but not as accurately as we had predicted. This was in part due to inconsistent readings from components and in part due to lingering bugs within the code.

Our final product met all the size and durability requirements set by this course. Only in terms of insulation and wires did our device not prove to be exceedingly effective due to our low cost solution of using construction tape as extra insulation.

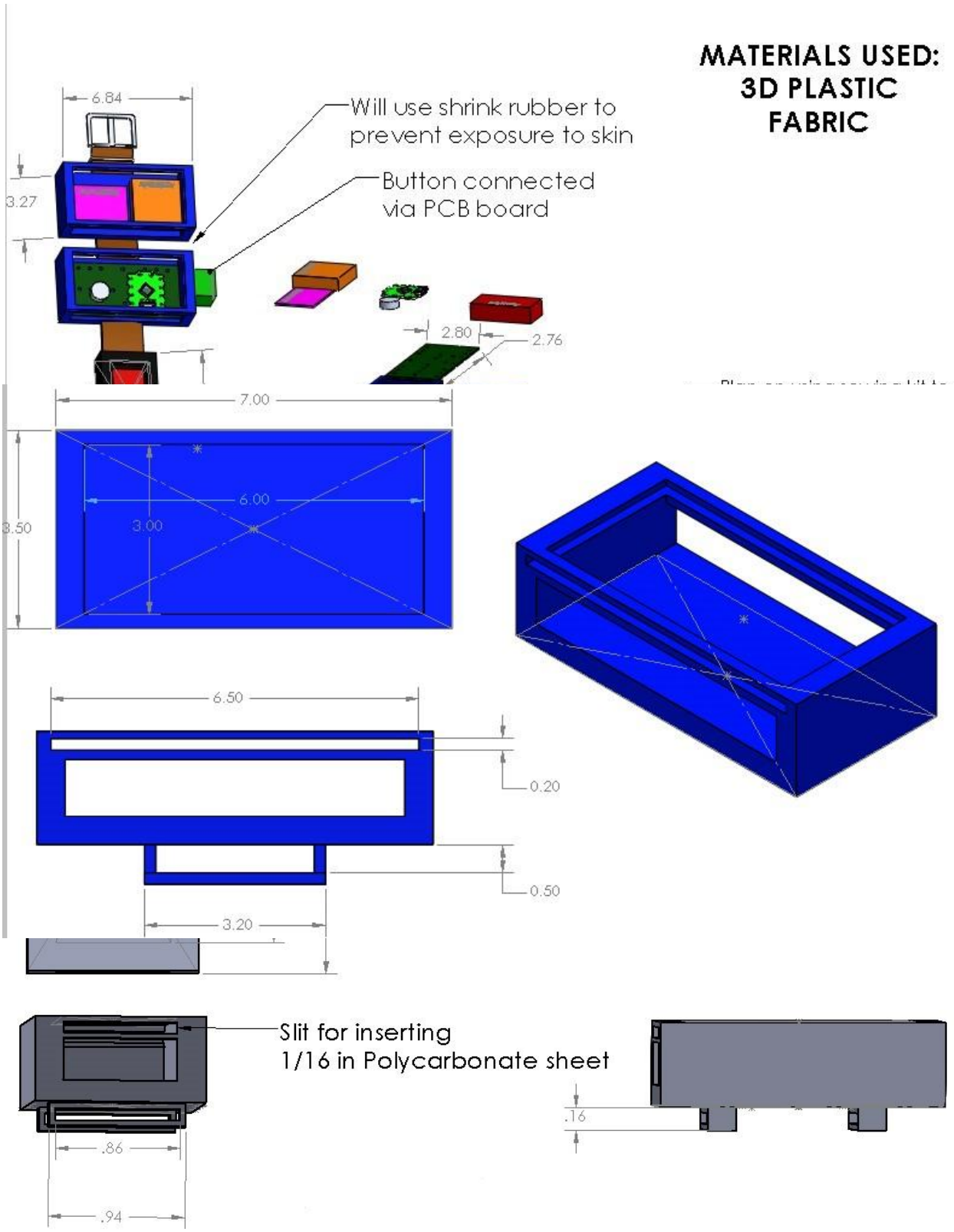
Discussion

Along the way we ran into many problems in nearly every component of the project however we found a way to fix most of them:

- **Arduino Code**:
    - *Problem*: Our code was continuously printing out symbols when we needed it to print numbers.
        - Solution: There were a few components that helped. First, we changed the bod rate to a lower bod rate which helped for the simpler renditions of our program. Then we took our excess Serial.begins which were left behind from when we combined the original accelerometer and barometer code. Lastly, we realized by experimenting with older working code that the Arduino did not respond well to concatenated print statements. So we changed all our print statements into multiple line print statements and our code began showing numbers as originally expected.
    - *Problem*: Infinite counting of steps
        - Solution: This had many factors and happened on numerous occasions. The first time we discovered this error it was because the orientation we were holding the accelerometer and our code were not coinciding. So we switched the orientation we were holding the accelerometer in. Then we noticed that Even a small movement triggered a step. This was due to an originally hard coded steady state that we then fixed to adhere to every user. We then also updated our threshold to be closer to the values we were observing. Then we started to test the code by actually walking instead of only moving our arm. Once we surpassed a certain x value, however, it would count infinitely. So we asked the accelerometer to only check the y position to see if we reached the step count. We did this by commenting out part of our nested loop.
- **MIT App**:
    - *Problem*: Bluetooth Connection. The app either wasn't connecting or we couldn't figure out which one of the blunos on the available connections list was ours.
        - Solution: we continuously closed down and reopened the app until it was connecting and recognizing the device. Then if we were having trouble figuring out which bluno, if any, on the list was ours, we would step outside where there would be less noise/ available bluno bluetooth connections. This helped us to connect our app quicker. We also figured out that a green link light on the Arduino would blink on when a device connected to it.

    - *Problem*: Printing and updating values onto the screen for step count and stair count.
        - Solution: We changed a line in our code so that the data sent from the Arduino would actually show up on the screen. Then we diminished any noise that was confusing the app. Specifically we removed the clock from the MIT app code and we commented out any lingering print statements in the Arduino code.
- **Soldering:**
    - *Problem*: Original bus lines were too long
        - Solution: The bus lines were created before the 3D print was done, therefore there was not a good way to picture the lengths of wire that we would need. However, once the 3D components were finished we had a better idea and were able to shorten our bus lines to be closer to the desired length.
- **Manufacturing:**
    - *Problem*: First print was far larger than we anticipated
        - Solution: We checked the dimensions and adjusted the solidworks so that the modules were just big enough to fit our components snuggly. We then reprinted.
    - *Problem*: First print was quite brittle and partially caved in.
        - Solution: we changed the orientation of the print so that there was no overhang and unnecessary supports were easier to remove.
- **Other:**
    - *Problem*: Our button was not working as intended.
        - Solution: first we changed our code to include flags. That worked in starting the code but unfortunately stopping the code was unpredictable. So, we tried putting in delays to try catching the code at the perfect time to stop it. This did not work. Then we found an Arduino button code online which helped fix a bit of our problem but was still unreliable in stopping the code. In the end due to both hardware and software problems, we ended up scratching the idea of the button entirely. Given more time we would have tried to figure out how to add and use the button with our prototype.
    - *Problem*: Temporary circuit was not working at all at some point.
        - Solution: we troubleshot all of the wires to check if maybe we had a broken wire or faulty components. Then when that made no difference we tried switching the breadboard at which point we discovered that we had a faulty breadboard.

Appendix A: Solidworks Design



Appendix B: Bill of Materials

PURCHASE REQUEST FORM

The Henry Samueli School of Engineering

Upload your completed Purchase Request Form to the EEE Dropbox folder by 3PM on Friday, January 22nd (Week 3)

Team Name:	**It's Everyday Code**	**Team Captain Email:**	dshkel@uci.edu
Lab Section Day/Time:	**Thursday 2-3:50**	**Date of Request:**	02/01/18
	304 **Room #**		
Account Name:*	ENGR7B: Fitness Tracker		

Quantity*	Unit of Measure*	Description*	Catalog #*	Unit Price*	Estimated Extended Price*	Actual Extended Price
1		Beetle BLE - The smallest Arduino bluetooth 4.0 (BLE)	SKU:DFR0339	$14.90	$14.90	
1		gravity: I2C Triple Axis Accelerometer - LIS2DH	SKU:SEN0224	$4.95	$4.95	
1		GravityBMP388 Barometric Pressure Sensors	SKU: SEN0251	$6.90	$6.90	
1		6 PCS 3.7V Battery Charger Set	ID:1205890	$3.95	$3.95	
1		Watch Bands - Basic Nylon Strap	ID:4331774718	$12.50	$12.50	
1		Vibration Motor	ID: 08449	$2.15	$2.15	
1		Double sided PCB Board Prototype Kit	ID: B01N3161JP	$0.33	$0.36	
1		Tactile Button switch (6mm)	ID:376	$0.13	$0.13	
4		JST Connector Kit	ID: B0731MZCGF	$0.02	$0.08	
10		3D Printing		$4.00	$40.00	
				Subtotal	**$85.92**	
				***Tax rate:** **8.00%**	$6.87	
				TOTAL ORDER PRICE	**$92.79**	

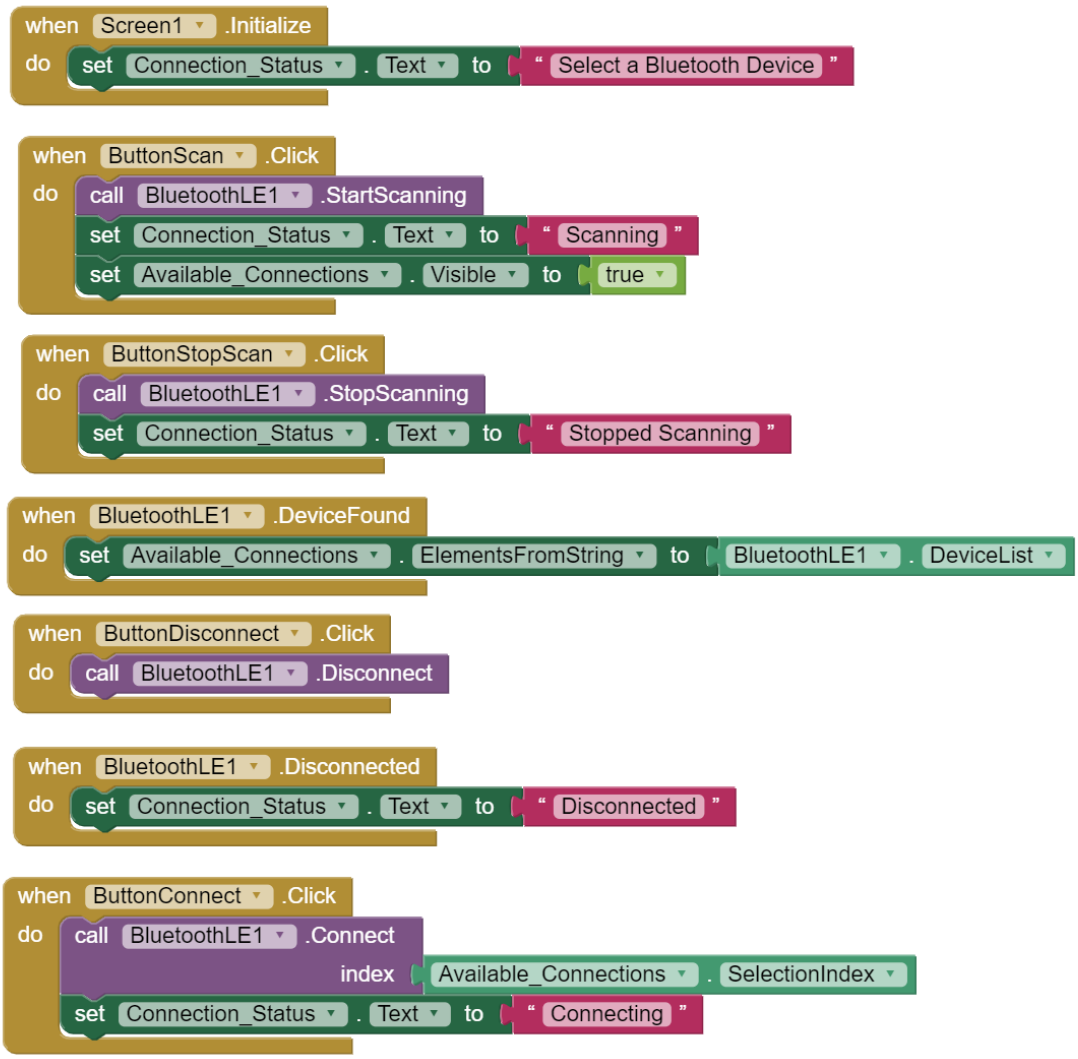
Appendix C: Arduino Code

```
 * MEMBERS: JESSICA ARIAS, JEREMY CHANG, NOAH PAIGE, MICHELLE SALDANA-RIVAS, DARIA SHKEL
 *
 * LAST DATE OF EDIT: 12:10 MARCH 15, 2019
 *
 * We implemented this code file into our ARDUINO BEETLE to run our fitness watch.
 * Our code counts the amount of steps taken and stair climbed by utilizing readings from both the barometer and accelerometer.
 * After acquiring initial x and altitude values, we run a loop that conntinously reads the user's x and altitude levels.
 * Then we dtermine if the difference between the new readings and the initial readings exceed our threshold values respectively.
 * If it exceeds the x value threshold, we count a step. If it exceeds the stairs threshold, we count a flight of stairs.
 *
*/

#include <DFRobot_LIS2DH12.h>
#include <DFRobot_BMP388.h>
#include <DFRobot_BMP388_I2C.h>
#include <SPI.h>
#include <math.h>
#include <bmp3_defs.h>
#define CALIBRATE_Altitude

// DECLARE GLOABAL VARIABLES

int vibM = 8; //pin # of vibration motor
int stepCount = 0; //counter for user's steps
int stairCount = 0; //counter for flights of stairs climbed by user
int stepGoal = 10; //hard goal for steps
int stairGoal = 2; //hard goal for flights of stairs
bool DEBUG = false;

//RELIED ON X VALUES TO DETERMINE STEP
int xMin = 1000; //in case all values read are positive
int xMax = -1000; //in case all values read are negative
int xVal = 0; //current x coordinate (location) of arm

//THRESHOLD VALUES
int xT = 300; //front/back swing x threshold)
int yT = 200; //front/back swing y threshold)
float stairT = 2.75; //step height threshold (2.75)

//VARIABLES FOR BAROMETER
float seaLevel;
float altitude;
float altInit; //globalized for steadyState()
int loopCounter = 0; //regulate barometer readings

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

void setup() { //runs code once
Wire.begin();
  pinMode(vibM, OUTPUT);
  Serial.begin(115200);
  while(!Serial);
  delay(100);
  while(LIS.init(LIS2DH12_RANGE_2GA) == -1){  //Equipment connection exception or I2C address erro
    Serial.println("No I2C devices found");
    delay(1000);
  }

  // Initialize bmp388
  while(bmp388.begin()){
    Serial.println("Initialize error!");
    delay(1000);
  }
  delay(100);
```

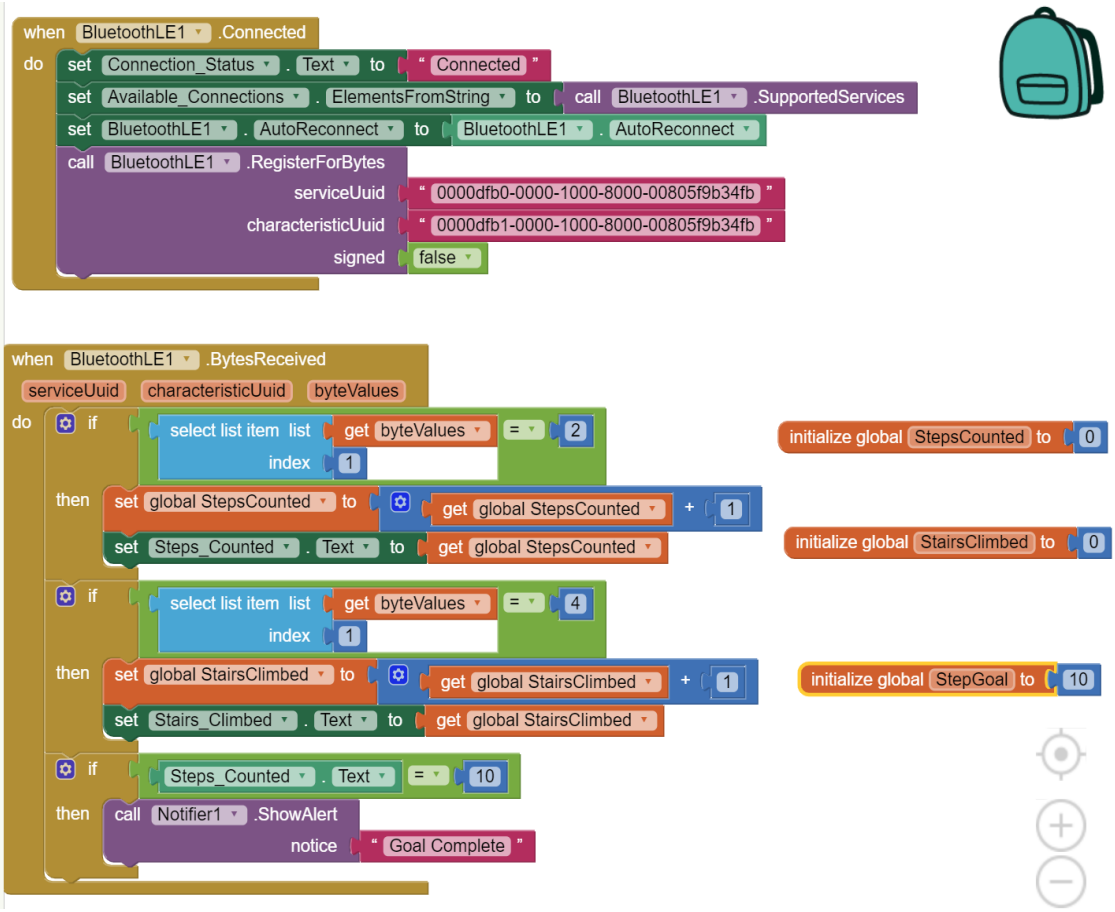
```
  steadyState(); //acquires xInit and altInit
  delay(3000);
}

void loop() {
  countingStep();
  loopCounter++;
  if (loopCounter == 5){ //regulates amount of barometer readings
    stairClimbed();
    loopCounter = 0;
}

}

// DEFINING RELEVANT METHODS
void acceleration(void) { //method updates current X position of accelerometer
  int16_t x, y, z; //locally defines x,y,z
  delay(100);
  LIS.readXYZ(x, y, z);
  LIS.mgScale(x, y, z);
  xVal = x; //updates current x position
}
void countingStep() {
  delay(200);
  acceleration(); // attain xVal and yVal
  if (abs(xVal - xMax) >= xT or abs(xVal - xMin) >= xT){ //check if threshold is met
      stepCount++;

      if (DEBUG == true){
        Serial.print("Step Count: ");
        Serial.println(stepCount);
      }
      else {
        Serial.write(2); //send data to MIT app
      }
  }
  if (stepCount == stepGoal) { // STEP COUNT GOAL CHECK
          digitalWrite(vibM, HIGH); // 1 buzz for step goal reached
          delay(500);
          digitalWrite(vibM, LOW);
        }
}

void stairClimbed() {
   float altitude = bmp388.readAltitude();

   if (abs(altitude - altInit) >= stairT){ //check if threshold is met
      stairCount++;
      altInit = altitude;
      if (stairCount == stairGoal){  //STAIR COUNT GOAL CHECK
        digitalWrite(vibM, HIGH);
        delay(100);
        digitalWrite(vibM, LOW);
        digitalWrite(vibM, HIGH);  //two buzzes for stair count goal reached
        delay(100);
        digitalWrite(vibM, LOW);
      }
     if (DEBUG == false) {
      Serial.write(4); //send data tso MIT app
     }
   }
}
```

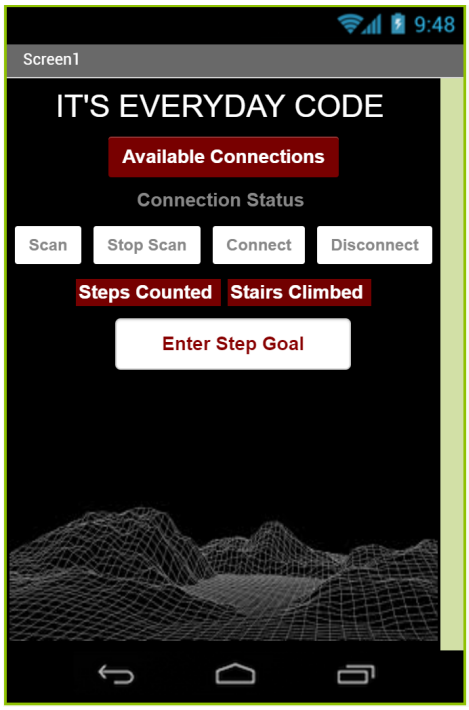
```
int steadyState(void) {
  int j = 0;
  int16_t x1, y1, z1;
  for (j = 0; j < 10; j++) {
    LIS.readXYZ(x1, y1, z1);
    LIS.mgScale(x1, y1, z1);
    if (x1 < xMin) {
      xMin = x1;
    }
    if (x1 > xMax) {
      xMax = x1;
    }
    float altInit = bmp388.readAltitude();
  }
  return (xMin, xMax, altInit);
  }
```

Appendix D: MIT App Inventor Designer View and Block Code




when Screen1 .Initialize
do set Connection_Status . Text to " Select a Bluetooth Device "
when ButtonScan .Click
do call BluetoothLE1 .StartScanning
set Connection_Status . Text to " Scanning "
set Available_Connections . Visible to true
when ButtonStopScan .Click
do call BluetoothLE1 .StopScanning
set Connection_Status . Text to " Stopped Scanning "
when BluetoothLE1 .DeviceFound
do set Available_Connections . ElementsFromString to BluetoothLE1 . DeviceList
when ButtonDisconnect .Click
do call BluetoothLE1 .Disconnect
when BluetoothLE1 .Disconnected
do set Connection_Status . Text to " Disconnected "


when BluetoothLE1 .Connected
do set Connection_Status . Text to " Connected "
set Available_Connections . ElementsFromString to call BluetoothLE1 .SupportedServices
set BluetoothLE1 . AutoReconnect to BluetoothLE1 . AutoReconnect
call BluetoothLE1 .RegisterForBytes
serviceUuid " 0000dfb0-0000-1000-8000-00805f9b34fb "
characteristicUuid " 0000dfb1-0000-1000-8000-00805f9b34fb "
signed false
when BluetoothLE1 .BytesReceived
serviceUuid characteristicUuid byteValues
do if select list item list get byteValues = 2
index 1
then set global StepsCounted to get global StepsCounted + 1
set Steps_Counted . Text to get global StepsCounted
if select list item list get byteValues = 4
index 1
then set global StairsClimbed to get global StairsClimbed + 1
set Stairs_Climbed . Text to get global StairsClimbed
if Steps_Counted . Text = 10
then call Notifier1 .ShowAlert
notice " Goal Complete "
initialize global StepsCounted to 0
initialize global StairsClimbed to 0
initialize global StepGoal to 10


9:48
Screen1
IT'S EVERYDAY CODE
Available Connections
Connection Status
Scan
Stop Scan
Connect
Disconnect
Steps Counted
Stairs Climbed
Enter Step Goal

Appendix E: References

Andreas. "What Is a Fitness Tracker?" *Improve Your Health with Technology*. New Fitness, Health & Sports Gadgets, 21 Oct. 2016. Web. 21 Mar. 2019.

Peckham, James. "Best Fitness Tracker 2019: The Top 10 Activity Bands on the Planet." *TechRadar*. TechRadar The Source for Tech Buying Advice, 05 Mar. 2019. Web. 21 Mar. 2019.

Winchester, Henry. "A Brief History of Wearable Tech." *Wareable*. N.p., 06 May 2015. Web. 21 Mar. 2019.