Athletic Tracking Sensor

By

J.D. Armedilla

Ryan Horstman

Ethan Pizarro

Final Report for ECE 445, Senior Design, Spring 2025

TA: Jiankun Yang

7 May 2025

Project No. 38

Abstract

We designed and implemented a compact wearable athletic tracking sensor that measures and analyzes velocity and form during weightlifting exercises. Using a 9-axis sensor and Bluetooth Low Energy, the device provides instantaneous feedback to alert users when they meet or exceed velocity or angle thresholds. Our developed iPhone app displays velocity and angle data and allows users to set exercise parameters. Verification demonstrated that the system measured vertical velocity to within hundredths of a meter per second and detected form deviations within 0.5 seconds, meeting our high level requirements. The device operated reliably for two hours on battery power and successfully communicated with the mobile app, allowing users to receive insight on how their exercises are going relative to their goals. While we were unable to have the sensor working on PCB, the Athletic Tracking Sensor is an effective, accessible solution for velocity-based training, providing immediate feedback and detailed exercise tracking.

Contents

1. Introduction

1.1 Problem Statement

Weightlifting is a key method of staying active and maintaining fitness for many people. Currently, the main metric of progress in weightlifting for everyday weightlifters is varying the weight and the number of repetitions. For example, one could raise 10 pounds (weight) 10 times in a row (repetitions). But, there is also value in (and workouts designed around) moving the weight at an appropriate speed, known as Velocity-Based Training (VBT) [1]. Continuing our example, instead of just counting how many times the weight is lifted, one tries to raise the weight slowly, never exceeding two meters per second. Effective velocity-based training emphasizes lifting heavy, while simultaneously ensuring one handles the weight well and with the speed desired.

However, velocity-based training is not accessible because available sensors for tracking velocity are costly and therefore infeasible for everyday weightlifters. If a lifter does buy an expensive sensor, those available lack some key features for an optimal workout. For example, some sensors assist with tracking velocity but do not assist with tracking form. Also, current sensors offer feedback that consists of the lifter doing their exercise and then checking their results on their phones. Since this feedback is not necessarily "real-time", these sensors cannot indicate to a user to adjust their velocity between each repetition. This results in wasted repetitions, and, for form tracking sensors, users are not informed immediately if bad form is used during the workout, increasing the risk of injury [2-3].

1.2 Solution

We propose a compact wearable device that takes and transmits workout data to a phone via Bluetooth. It will utilize a 9-axis sensor (acceleration, gyroscope, and magnetometer). However, in addition to sending data to a phone, it will internally process data taken during the workout and provide immediate feedback to the user through haptic signaling and LED feedback. Before starting the workout, the user can indicate on his/her phone the workout they are performing and any desired constraints. This data will be sent to the device. The device will track the user's form and acceleration, alerting him/her if a desired constraint is not being met so that it can be immediately corrected mid-set. It would be small enough to velcro around a user's wrist, hang on a necklace worn by the user, velcro-strap around a weight set, or attach it to a desired object. Using the acceleration data given by the tracking sensor, the app will display the collected velocity graphically and the lifting form so that the user can visually see his/her progress.

1.3 High-Level Requirements

- IMU readings on-chip will be able to determine velocity in the vertical direction with precision in the hundredths of a meter per second for bench press and back squat workouts. The device will alert the user immediately when a goal velocity is met/exceeded.
- The athletic sensor must be able to send and receive data via Bluetooth to/from our developed iPhone app, which will then use that data to graphically represent the athlete's velocity at a certain load.
- For form applications, our gyroscope will be able to detect degrees in a given direction at the precision of 3 degrees and alert the user within a half second of entering bad form.

1.4 Block Diagram

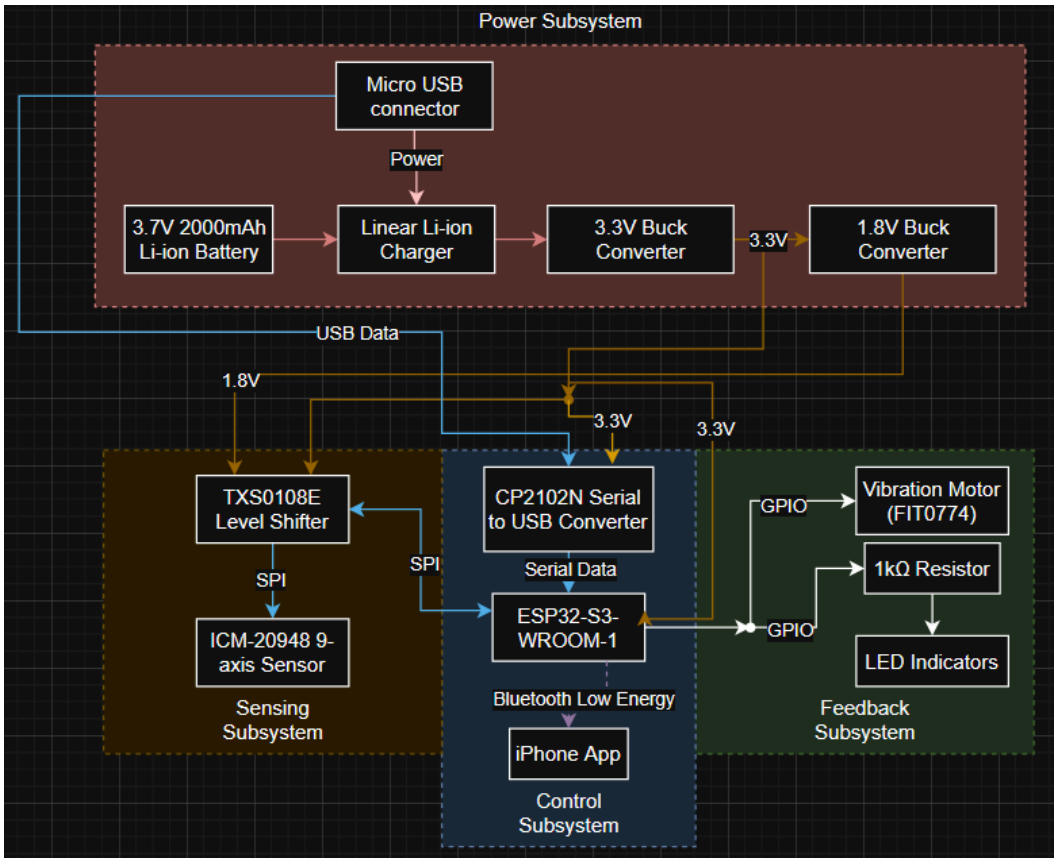

Figure 1: Final block diagram

Our device consists of 4 subsystems: power, control, sensing, and feedback. The power subsystem supplies power to all other subsystems and charges the sensor's battery. The control subsystem manages programming the microcontroller as well as communication between it and our developed iPhone app. The sensing subsystem collects accelerometer and gyroscope data via a 9-axis sensor so we can determine the velocity and angle at which an exercise is being performed. The feedback subsystem provides instantaneous feedback to the user, allowing them to adjust how they are doing the exercise immediately.

2. Design

2.1 Physical Design

The PCB is enclosed in a 3D-printed rectangular enclosure. Printed with PLA filament, it includes space to insert a velcro strap that would be used to attach to either the user or the barbell involved in the exercise. PLA was used as it was readily available and cheap, retaining device accessibility. M3 screws hold the PCB in place, as well as the lid on the top. The wall near the antennae of our microcontroller is thinner than the rest of the walls, allowing for a better Bluetooth connection. The lid includes space for the feedback subsystem's components, including the LEDs, recording button, and power switch.

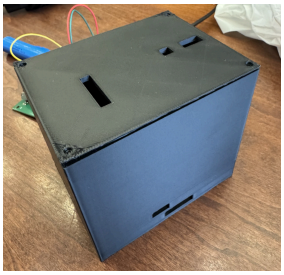

Figure 2: PCB enclosure

2.2 Power Subsystem

2.2.1 Hardware Design Overview

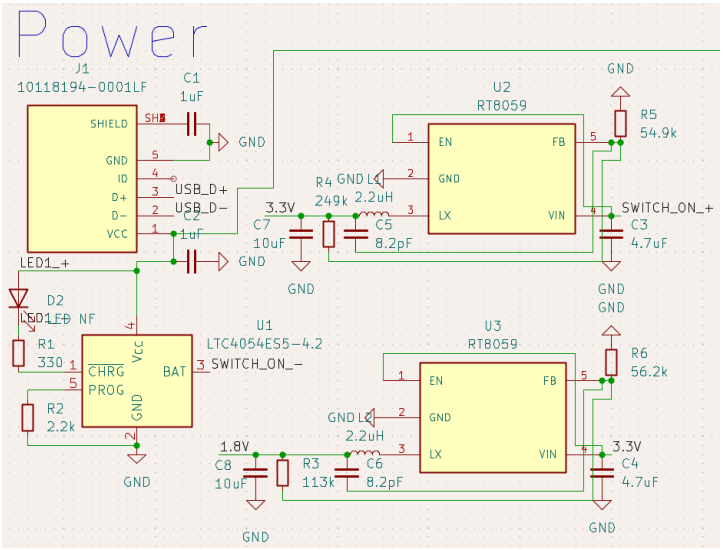

Figure 3: Power Subsystem Schematic

The power subsystem has two main purposes: supplying power with recharging capabilities and delivering power to the rest of the sensor. The main power supply is delivered by a 3.7V 2000mAh lithium-ion battery. It is connected to a simple battery charging IC, whose charging input voltage is taken

from the micro USB connector. The battery can be connected to the first of 2 buck converters based on the power switch. The first buck converter steps the battery voltage down to 3.3V. The output of the first buck converter is sent to the second, which steps the 3.3 volts down to 1.8V. Both 3.3V and 1.8V power is then delivered to the rest of the device.

2.2.2 Functionality & Contribution

The main goal of the power subsystem is to provide a rechargeable power supply and deliver reliable 3.3V and 1.8V power to the rest of the sensor. The following components contribute to this goal:

- LTC4054ES5-4.2#TRMPBF Li-ion charger: Enables charging, including an LED to indicate when said charging is occurring.
- RT8059GJ5 buck converters: Converts the battery's 3.7V output to 3.3V and 1.8V.
- 10118194-0001LF Micro USB-B connector: Handles USB charging and allows for data transfer to the USB-Serial converter.

The power subsystem converts the battery's higher voltage to lower voltages that our components and ICs can reliably use. The addition of a battery charger increases convenience for the user while allowing them to understand when their device is charging.

2.2.3 Interfaces

Inputs:

- 3.7V 2000mAh Li-ion battery
    - Voltage range: 3.0 (discharged) to 4.2 (full charge)
    - Input to battery charger: 3.7V
- 3.3V output from first buck converter: input to second buck converter to convert to 1.8V

Outputs:

- 3.3V output from first buck converter: powers the ESP32-S3-WROOM-1, USB-Serial converter, 9-axis sensor, the level shifter, and various throughhole components of the feedback subsystem
- 1.8V output from second buck converter: powers the 9-axis sensor's I/O voltage and the level shifter
- Charging LED: indicates whether the battery is being charged or not at any given time

2.2.4 Design Decisions

The 3.7V 2000mAh Li-ion battery was chosen because it provided enough capacity for a 2 hour battery life at 1 A operation. Being lithium ion also made our device capable of recharging, which is important for wearable technology. Originally, our battery charger IC was small and had pins that were difficult to solder. It also included extra features that we were not using. The LTC4054ES5-4.2#TRMPBF is a simpler IC, allowing for easier soldering and simplifying the connections we need to make on our PCB. Additionally, stepping down our voltage was originally done with two separate linear voltage regulators, with one for 3.3V and one for 1.8V respectively. Buck converters were swapped in because their output

current was higher, which gave the sensor more leeway to draw current as needed. We also made our first designs using a standard USB-A port for both charging and programming. This was swapped out for a Micro USB-B port, as they are readily available in the ECE department's E-Shop while offering the same capabilities.

2.3 Control Subsystem

2.3.1 Hardware Design Overview

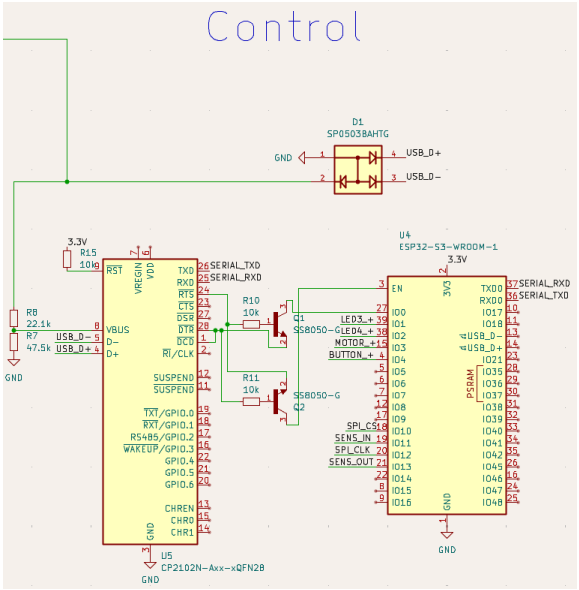

Figure 4: Control Subsystem Schematic

The control subsystem consists of our microcontroller (the ESP32-S3-WROOM-1) and a USB-Serial data converter in order to program it. As the main computational component of our device, the ESP32 will be responsible for processing the data from our 9-axis sensor via SPI. Utilizing SPI allows for faster data transfer, which will allow our device to experience less latency when it comes to alerting users of bad form. The embedded Bluetooth module will be used to communicate with our mobile phones so that we can display the data collected and set the parameters the ESP32 should trigger at.

2.3.2 Software Design Overview

The goal of the software on the ESP-32 is to process data from the 9-axis sensor and communicate via Bluetooth with our developed iPhone app. Because we used Arduino IDE for programming, functionality is split between a setup phase and a loop phase. The setup phase initializes the 9-axis sensor, calibrates it, and configures GPIO pins for the LEDs, button, and vibration motor. It also creates a Bluetooth Low Energy service and characteristic, beginning advertising once the sensor is set up. The loop phase is controlled by the user, as the program idles until a button is pressed and an exercise and velocity threshold have been received from the iPhone app. Once said button is pressed, a delay of 7 seconds begins, which gives the user time to position themselves before they start their exercise. Once the delay is over, the motor and an indicator LED are turned on by the program, communicating that recording is starting.

While recording, at a rate of 100 readings per second, sensor data is read to compute exercise velocity and angle. The data is also sent over via Bluetooth to the iPhone. If any threshold is exceeded, the accompanying alert is sent via the vibration motor and indicator LEDs. Once the button is pressed again, recording stops and the program idles to wait for a button press and exercise/velocity threshold once again.

2.3.3 iPhone App Overview

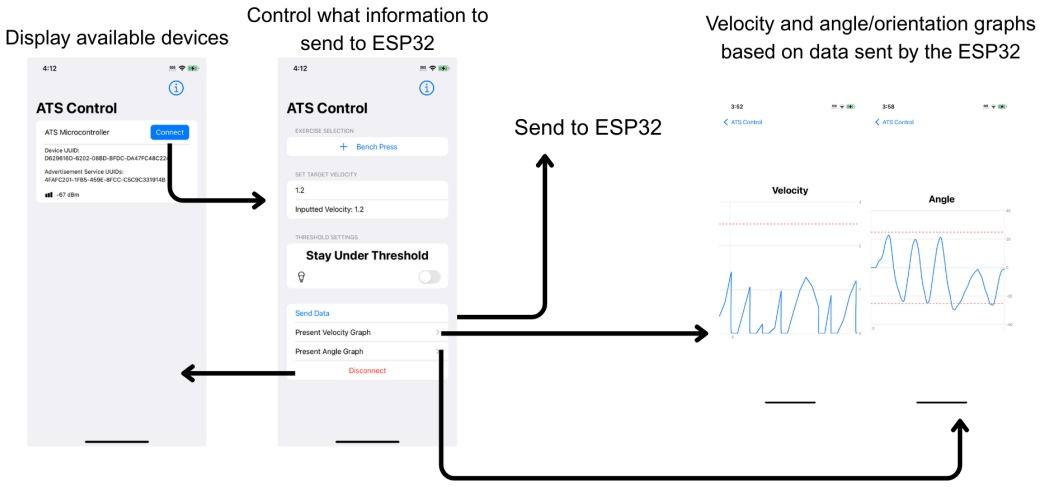

Figure 5: Final Control Flow for Application

To satisfy the second high level requirement, we developed an iPhone application. This application is the main method through which the user interacts with the Athletic Tracking Sensor. The main goal of the application is to send parameters for the desired exercise, the threshold velocity, and whether the user should go above or below the threshold. Using Bluetooth Low Energy Protocols seen in Appendix B.1 Figure_, the application searches for the ESP32's advertised service. Once the service has been detected, the application and the ESP32 can connect. From there, the application can perform read/write operations to the ESP32 by reading/writing data from/to the ESP32's characteristic.

2.3.4 Functionality & Contribution

Operation of the Athletic Tracking Sensor is initiated via a "Record Data" button. The microcontroller idles until the button is pressed and records until the button is pressed again. This process is repeat as the device is on. This system ensures a boost in efficiency, as we will not always have our device operating at full power when it does not need to do so.

Once recording begins, the ESP32 handles data processing. The 9-axis sensor sends its data via SPI to the ESP32. Once the sensor's data has been sent over, velocity and acceleration are calculated, which is sent over Bluetooth to our mobile app for display. When thresholds are exceeded, the ESP32 triggers parts of the feedback subsystem as needed. This includes turning on vibration motors or associated LEDs via the GPIO pins. For example, if an angle threshold is exceeded, the microcontroller flashes a red LED and buzzes the vibration motor, sending the user a warning. When a velocity threshold is exceeded, a green

LED turns on instead, rewarding the user. This ensures all communication and feedback signals are functional and controlled by the “Record Data” button.

2.3.5 Interfaces

Inputs:

- 9-axis sensor data via SPI
- The 3.3V power source from the power subsystem
- USB data from the Micro USB-B connector in the power subsystem

Outputs:

- Bluetooth Low Energy protocol sending velocity and acceleration data to the device hosting the mobile app
- GPIO signals to activate the feedback subsystem when relevant

2.3.6 Design Decisions

Utilizing a button to initiate and then end workouts allowed us to make our device more efficient. It is not necessary that data is transmitted when no exercises are currently being done. By utilizing the button to start an exercise, it lets the sensor have a lower power draw. By minimizing the amount of time that components are running at full power, it ensures the device’s battery lasts longer.

Additionally, utilizing the Micro USB-B connector for both charging and data purposes allows the PCB to be smaller. Having two connectors would complicate the design unnecessarily, as there would be two separate USB-B ports, with one handling charging the battery and the other handling microcontroller programming. By combining both purposes into one connector, it simplifies how the port should be used by users and prevents a waste of resources and space by having two of the same connector.

Regarding the ESP32 software, we originally had issues because we unknowingly fried our sensor in the beginning. Because of this, a lot of debugging was done without realizing that we were at a dead end. Once we got a new sensor, we decided to use the ICM20948_WE library [7]. This allowed us to get accelerometer and gyroscope data for processing. For Bluetooth communication, we utilized starter code from ArtsemiR’s ESP32 BLE Remote Control Demo [8]. This gave us the foundations for Bluetooth communication so we could adapt it for the messages we wanted to process.

For the iPhone application, the first iteration of the application had the following control flow seen in Figure _: inputting the parameters, sending the parameters to the ESP32 and waiting for a confirmation, recording the workout data and sending it to the application, and plotting the data.

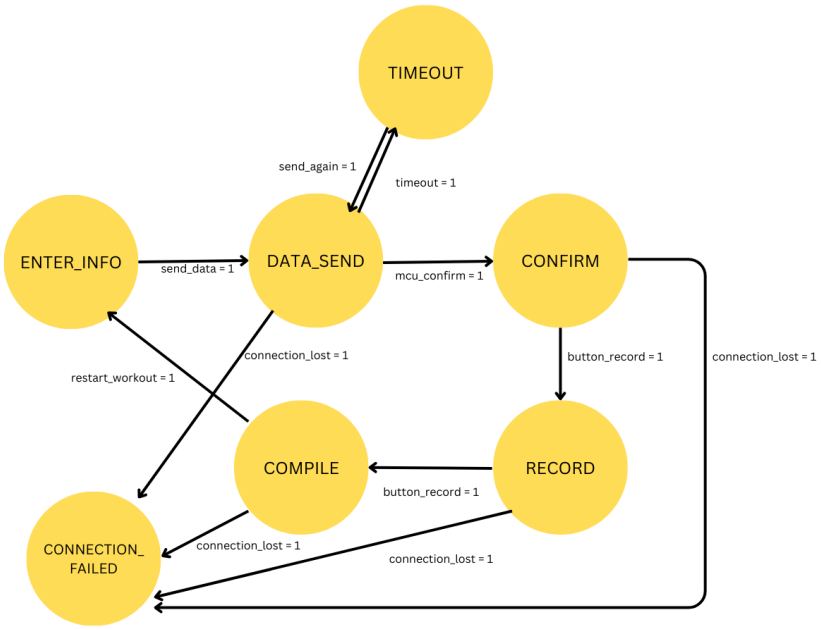

Figure 6: Original Control Flow for Application

However, there were a few issues with this original control flow. First, the CONNECTION_FAILED state is a dead state that does not flow into a state that resets the application. Next, for user convenience, the user should have access to the graph data up until the point that new data is sent by the ESP32. Finally, from a usability standpoint, it makes more sense for the application to connect to the ESP32 first before being able to attempt read/write operations from/to the ESP32. With these considerations in mind, the control flow of the application was modified, as seen in Figure _: The new control flow is as such: connect to the Athletic Tracking Sensor using Bluetooth Low Energy, input parameters, send the parameters, perform the workout, send recorded data to the application, and plot that data into its respective velocity/orientation graphs. The code used in the application was adapted from a Github source that specifically implemented communication between an iPhone application and an ESP32 [6].

2.4 Sensing Subsystem

2.4.1 Hardware Design Overview

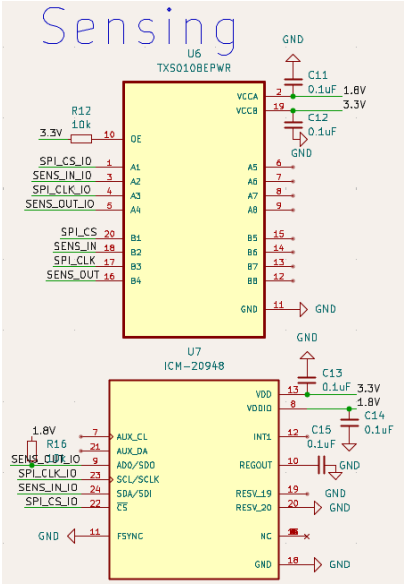

Figure 7: Sensing Subsystem Schematic

Our sensing subsystem currently consists of two main components, seen in Figure 7. The first is the ICM-20948, a 9-axis sensor composed of an accelerometer, a magnetometer, and a gyroscope. From this component, the accelerometer is utilized to collect acceleration data in meters-per-second-squared, and the gyroscope is utilized to collect rotational velocity data in radians-per-second. The second main conponent is the TX5010BEPWR Level Shifter. This component shifts the microcontroller's 3.3V GPIO outputs down to 1.8V so the signals can be read by the 9-axis sensor, and vice versa for the sensor's responses. This is crucial because the ICM-20948 operates on 1.8V signals.

2.4.2 Functionality & Contribution

The sensing subsystem is where all of the data that dictates our device comes from. To perform SPI communication and get the raw data out of the sensors, we utilized the ICM20948_WE library. From this library, we utilized functions that initialized the sensor, read all nine axes of the sensor and stored each in registers, and pulled individual raw data from those registers. The code that pulled this data from the sensors is found for reference in Appendix _ under figure _.

Once the raw data was collected, it had to be processed on the microcontroller in order to produce the angle and velocity data required. Starting with obtaining velocity values, the first step is removing the gravity component of acceleration from the. Upon powering on the device, it is laid flat and logs the angle and direction of gravity's acceleration. Now, given any angle the device is at, it can remove the acceleration vector with the following equations. First, the degree measurements are converted into radians, seen in equation 1:

$$angle\ in\ radians\ =\ angle\ in\ degrees\ *\ \boldsymbol{\pi}/180 \quad (1$$

Next, based on the x and y angles in degrees and utilizing trigonometry, the components of gravity for each cartesian vector can be found, seen in equations 2 - 4:

$$gravity_x\ =\ 9.81\ *\ sin(angle_x) \quad (2$$

$$gravity_y\ =\ 9.81\ *\ sin(angle_y) \quad (3$$

$$gravity_z\ =\ 9.81\ *\ cos(angle_x)\ *\ cos(angle_y) \quad (4$$

Now, this value of acceleration due to gravity is simply subtracted from the sensor output for each axis. The code for removing gravity acceleration is also shown in Appendix _ figure _. Now that gravity is removed we have raw acceleration data from movement only. Next we deal with the issue of noise. We apply a deadband filter to the accelerometer output because small readings due to vibrations and noise, if read as movement, will throw our velocity reading off. This is known as drift and should be minimized. The deadband filter simply looks at the accelerometer data and if it is below a given value, it sets that data to zero. The pseudocode is given below:

$$if\ (abs|a_{without\ gravity}|\ <\ threshold\ )\ a_{without\ gravity}\ =\ 0 \quad (5$$

Next, to get velocity, we must numerically integrate the acceleration values. We calculate velocity for each of the three cartesian vectors, and each is initialized to zero. The integration is as follows:

$$v_{new}\ =\ v_{old}\ +\ a_{without\ gravity}\ *\ (t_{current}\ -\ t_{last\ sample}) \quad (6$$

Here, ‘v’ indicates a velocity value, ‘a’ indicates acceleration, and ‘t’ indicates a time. The code for computing velocity so far is shown in Appendix _ figure _. The last step for velocity calculation is getting an absolute velocity magnitude, to account for the user not pushing in the absolute vertical direction. To accomplish this, we applied Pythagoras’ Theorem to calculate the magnitude of the total velocity vector:

$$|V|\ =\ sqrt(v_x^{\ 2}\ +\ v_y^{\ 2}\ +\ v_z^{\ 2}) \quad (7$$

The process for calculating angle from raw gyroscope data is very similar to the process just described. A second deadband filter is applied to the gyroscope outputs to prevent drift, and then the data is put through numerical integration:

$$\boldsymbol{\phi}_{new}\ =\ \boldsymbol{\phi}_{old}\ +\ v_{angular}\ *\ (t_{current}\ -\ t_{last\ sample}) \quad (number$$

Now, the control subsystem can utilize both angle and velocity data to provide a service to the user.

2.4.3 Interfaces

The microcontroller communicated with the ICM-20948 utilizing SPI communication. As a result, the microcontroller had four channels used to facilitate this communication: chip select, clock, MOSI (master-out slave-in), and MISO (master-in slave-out). The first three channels are inputs to the subsystem, and the MISO is the only output.

2.4.4 Design Decisions

There were several key decisions made in designing the sensing subsystem. The first of which was picking which sensor to use. We had the option of using two separate integrated circuits for the gyroscope and the accelerometer, but we decided to use a 9-axis sensor so as to decrease the area of the overall device. Since our tracking sensor is a wearable device, we wanted the outcome to be as comfortable to wear as possible, and thus as small as possible.

Another key decision was choosing the SPI communication protocol over I2C, which was the other protocol option for the ICM. We chose SPI because it allows the device to operate on a 7 MHz clock, as opposed to a 400 kHz clock with I2C. This allowed a much faster data rate, and since our data algorithm relies on numerical integration, more data points per-second results in a more accurate integration.

The ICM-20948 has registers that hold values indicating how many sensor outputs should be averaged together before the data out register is updated with a new value. Upon testing with different values for the acceleration sensor we found increasing this value from one gave no significant benefit and only slowed down the output rate. With the gyroscope sensor, however, we found that increasing the value up to five samples brought about much more accurate results, and so we set the gyroscope to averaging 5 consecutive data points. This setting is also seen in Appendix_ under figure_.

The last main design decision came from experiencing velocity drift, an inevitability when using accelerometers over a larger interval of time. We found that raising and then lowering the device did not always return velocity to absolute zero, and that drift compounded per repetition. Therefore, we decided that since maximum velocity was required, not necessarily the tracking of velocity back to zero, the device should reset velocity to zero whenever acceleration readings read zero, meaning that there was no current motion. This solved our problem of velocity drift, and allowed us to maintain our full functionality.

2.5 Feedback Subsystem

2.5.1 Hardware Design Overview

The feedback subsystem consisted of five main components. The first of which is a vibration motor. The remaining components consist of four LEDs. The vibration motor is connected directly between the microcontroller and ground, and the LEDs are connected directly between the microcontroller and either a resistor or ground. Shown in the schematic above also are connections for the start-workout button and the on-off switch for the entire device.

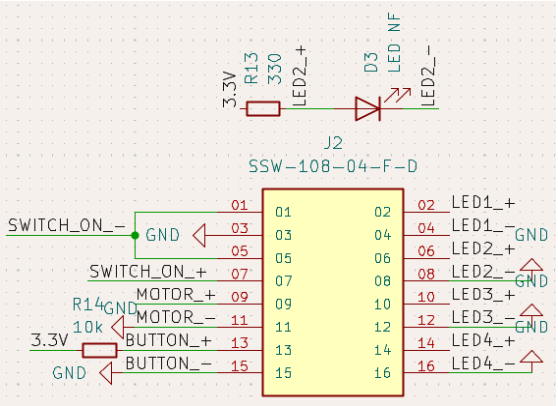

Figure 8: Feedback Subsystem Schematic

2.4.2 Functionality & Contribution

This motor provides the haptic feedback to the user, alerting him/her if adjustments need to be made. The four LEDs each have their own functionality. Firstly, there is a green LED that indicates whether the device as a whole is turned on or not, which is controlled by a switch that the user will turn on and off. If the user flips the switch and the green LED does not turn on, the user knows that the battery is dead and needs charging. The second LED is yellow, and indicates whether or not the battery of the device is charging. When the user plugs in the charging port, this LED will turn on. The third LED is also yellow and indicates the state of the device. If the device is waiting for the user to start his/her workout, it will be blinking. Once the user presses the start button and begins the workout, the state LED will be solidly on. The last LED is a red one that indicates threshold updates. When the motor actuates, so does this LED. It blinks along with the vibration motor if the user is using dangerous form, and also lights up for a pulse if the velocity threshold is exceeded.

2.4.3 Interfaces

The microcontroller actuates each of the feedback components directly using its GPIO pins. Setting the GPIOs to high turns on the components, and low turns them off. The output of the subsystem is what the user feels from the vibration motor and sees from the LEDs.

2.4.4 Design Decisions

We chose to utilize a vibration motor in order to achieve our goal of updating the user during his/her workout instead of after. The haptic notification allows the user to be aware of his/her progress without setting down the weights.

The LEDs were implemented to create an easier user experience. The user can know both power information of the device as well as what state the device is in without knowing any of the underlying technology. The LED that lights up in tandem with the motor is not completely necessary but is implemented mainly for demonstration purposes.

3. Design Verification

3.1 Power Subsystem

3.1.1 Verification Plan

As shown in Appendix A, our requirements for the power subsystem were to have a 6 hour charging time at max and a battery life of at least 2 hours. To test this, the PCB with the power subsystem implemented was left to run with the power switch flipped on. Time started when it was turned on and ended when the green LED that indicated power delivery shut off. Charging time was determined by measuring battery voltage after power was fully drained. Time began when charging began and time ended when the battery reached its max charging voltage.

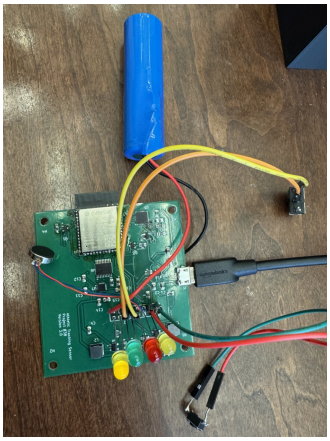

Figure 9: PCB charging with LED indicator (far left) on

3.1.2 Verification Results

Table 1 Power Subsystem Verification

Metric	Time
Charging	5 hours, 26 minutes
Battery Life	2 hours, 17 minutes

Referring to Table 1, we were able to verify both our charging and battery life requirements successfully.

3.2 Control Subsystem

3.2.1 Verification Plan

As shown in Appendix B, our control subsystem is in charge of starting and stopping recording using the "Record Data" button, alerting users as needed. Testing this is functionally equivalent to ensuring that our sensor's core functionality works, so verifying this was akin to ensuring our device worked.

3.2.2 Verification Results

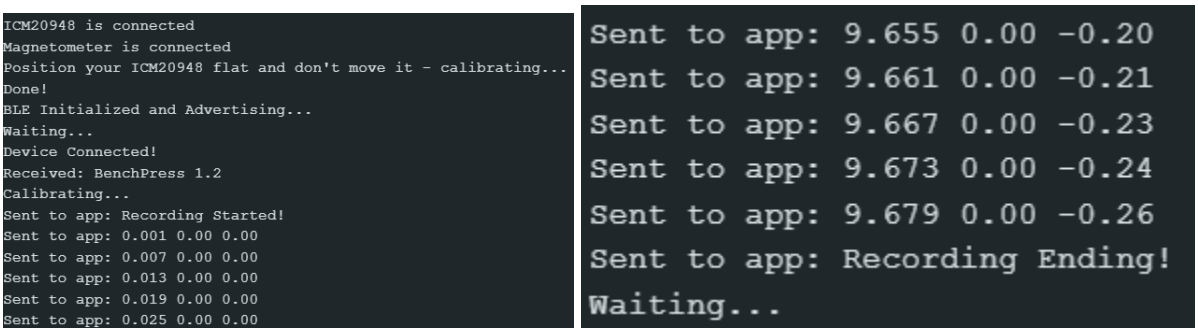

Figure 10: Terminal verification of sensor connection, Bluetooth connection and message receiving, button press detection, sending sensor data, and recording ending

Figure 10 displays the terminal output in Arduino IDE when certain events, such as iPhone connection and recording, begin and end. The first two messages ensure that the 9-axis sensor is detected by our microcontroller. The "BLE Initialized and Advertising…" message ensures our Bluetooth service on the ESP32 is being broadcast to other devices. The program also prints a message ensuring another device is connected. The next line then verifies that an exercise ("BenchPress") and a velocity threshold ("1.2") is processed. The "Calibrating…" prints after our "Record Data" button is pressed, with the messages being sent confirming that our data is being sent. The second image in the figure shows our recording ending as soon as we press the button again. Refer to Appendix B.1 for examples of graphs generated from our testing. Combined, this testing ensures our control subsystem is working properly by exemplifying successful control over our project.

3.3 Sensing Subsystem

3.3.1 Verification Plan

To test the sensing subsystem, we first have the microcontroller communicate with the sensor using SPI protocol. If the sensor returns representative values of both angular velocity and acceleration, then the sensor is functioning properly. We then run our data analysis code on the sensor output, and if representative velocity and angle readings are produced, then the data collection is verified. We will check the data through the app graphing function.

To verify the data rate of our device, we output on the serial monitor the difference in time used in the integration of acceleration values. This value is the time between each sample. If the time is below .2 seconds, we meet our goal of 5 gyroscope readings per second.

3.3.2 Verification Results

As seen in figure 12 and 13 (in appendix B), the graphical representation of some of our device's runs, our device produces data representative of velocity and angle orientation, and therefore our data collection is verified.

In outputting the time difference between samples, the result alternated between .01 and .02 seconds. This means we are collecting between 50 and 100 samples per second, far exceeding our 5 sample goal for gyroscope data. This is also a good indicator for our integration of acceleration as velocity is more accurate with a higher sample rate. In all, our sensing subsystem passed each verification check.

3.4 Feedback Subsystem

3.4.1 Verification Plan

Verification for the Feedback subsystem requires running through each component and making sure that it is operational given an applied voltage. Using a power supply, apply 3.3 volts across the vibration motor. If it vibrates, then it is functional. Next apply .7V to each of the LEDs. If they turn on, the LEDs are functional.

3.4.2 Verification Results

Upon testing, each of the LEDs and the vibration motor turned on as expected, and therefore the feedback subsystem is verified and will operate according to the control subsystem's outputs.

4. Cost Analysis

4.1 Parts

Table 1: Parts List for Athletic Tracking Sensor

Item	# Quantity	Cost per Item	Total Cost
LTC4054ES5-4.2#TRMPBF Battery Charger	1	$6.14	$6.14
RT8059GJ5 Buck Converter	2	$0.73	$1.46
LQH5BPN2R2NT0L 2.2uH inductor	2	$0.34	$0.68
CBR08C829BCGAC 8.2uF C	2	$1.47	$2.94
ERJ-6ENF2493V 249k R	1	$0.11	$0.11
AC0805FR-0754K9L 54.9k R	1	$0.11	$0.11
CR0805-FX-1133ELF 113k R	1	$0.10	$0.10
RC0805FR-0756K2L 56.2k R	1	$0.10	$0.10
10118194-0001LF Micro USB-B connector	1	$0.41	$0.41
CL21B105KBFNNNG 1uF C	2	$0.11	$0.22
RMCF0805JT2K20 2.2k R	1	$0.10	$0.10
RK73H2ATTD3300F 330 R	1	$0.10	$0.10
CL21A475KAQNNNE 4.7uF C	2	$0.10	$0.20
GRM21BR61H106ME43L 10uF C	2	$0.28	$0.56
3.7V 2600mAh Li-ion Battery	1	$11.99	$11.99
CP2102N-A02-GQFN28 USB-to-Serial converter	1	$5.79	$5.79
ERA6AEB2212V 22.1k R	1	$0.10	$0.10
ERJ6ENF4752V 47.5k R	1	$0.11	$0.11
SP0503BAHTG ESD protection	1	$0.63	$0.63
RMCF0805JG10K0 10k R	3	$0.10	$0.30
SS8050-G transistors	2	$0.24	$0.48
ESP32-S3-WROOM-1	1	$5.49	$5.49
SLW-1276864-4A-D On/Off Switch	1	$0.83	$0.83
PTS645SL43SMTR92 LFS Button	1	$0.35	$0.35
RK73H2ATTD3300F 330 R	1	$0.10	$0.10
RMCF0805JG10K0 10k R	2	$0.10	$0.20
SSW-108-04-F-D 16-pin connector	1	$2.43	$2.43
316040001 Vibration Motor	1	$2.43	$2.43
Green LED	2	$0.24	$0.48
Yellow LED	1	$0.28	$0.28
Red LED	1	$0.18	$0.18

This brings the parts costs before tax to $45.40. Sales tax is 6.25%. This brings the parts cost to $48.24.

4.2 Labor

Assuming that an average ECE graduate makes around $37.50/hr [2], and each member will work around 80 hours for the course of this project, we estimate that labor costs for one person will be:

$$labor = 37.50 * 2.5 * 80 = 7500$$

4.3 Total Cost

Overall, the total cost will be labor * 3 + cost of parts. This will bring the total cost to $22548.24.

5. Conclusion

5.1 Accomplishments

Overall, we were able to create a working rechargeable battery power supply for a wearable device, calculate the velocity and orientation of a user from the 9-axis sensor's raw data, provide live feedback during exercises, and develop an application that utilizes Bluetooth communication to display data.

5.2 Uncertainties

The biggest issue with our project was with our PCB. Mainly, due to a misunderstanding of the datasheet in which V_BUS would provide power to the ESP32, the V_REGIN and V_DD pins on the ESP32 were left unconnected. This meant that we could not program our ESP32, rendering our PCB useless. Given more time, we would've fixed the issue with our PCB and been able to have a fully functioning control subsystem.

5.3 Ethical Considerations

Due to our use of Bluetooth, we need to ensure that users have their exercise data secured, as per the IEEE Code of Ethics #1. Practically, this means implementing a form of encryption to protect against anyone who would want to intercept the data being sent. While the athletic tracking sensor does not explicitly require personal data, it is still imperative that we protect users' privacy in any way possible. The Bluetooth controller on the ESP32 supports LE Privacy 1.2. This feature ensures that the message authentication code (MAC) address associated with the ESP32 will be randomized at certain intervals [3]. Any malicious device that wishes to intercept the data sent by the tracker will be unable to determine the true MAC address of the tracker. Apple devices also support address randomization [4]. Thus, the two devices will be able to connect while still being able to ensure data privacy from malicious devices. To alleviate user concerns about the app, we will inform users of the mobile app about the data that is being collected and sent to the device.

Because our device is wearable and will be used while performing athletic exercises, safety precautions must be taken to ensure no one is injured during testing, as per IEEE Code of Ethics #9 [5]. Since we are testing with heavy lifts, including squats and the bench press, we need to allocate adequate space for those lifts to be performed. This is especially important as testing will occur in UIUC's recreation centers available to all students, so others will be around us constantly. Additionally, we need to make sure the wearable design does not inhibit movement or cause injury, especially due to heat. Ensuring that our design does not hurt the wearer is of utmost importance, as a device that enhances exercise should not prevent the user from doing so. We will address this by keeping the device compact and ensuring the location of the device will not interfere with workout movements. Generation of heat will also be minimized amongst our components, with airflow holes being implemented into the PCB's enclosure to reduce the risk of any injuries happening due to heat.

The project itself fits #1 and #2 in the IEEE Code of Ethics [5]. The tracking sensor aims to enhance athletic training for anyone, supporting their well-being and promoting exercise for all. It also introduces users to the idea that their timing while exercising is important. Different purposes of training offer

different benefits, so gaining direct knowledge of how the user's workout is going is important for both beginners and advanced lifters.

5.4 Future Work

Our future work involves modifying the schematic so that the Athletic Tracking Sensor can work on the PCB, and redesigning the enclosure such that the overall device is compact so as not to inhibit the user's movement or the movement of others. The application can also be improved to make reading data on the graphs more user-friendly. One feature that can implement this idea is when the user presses down on a point on the graph, the application will show the velocity/orientation of the user at that specific point in time.

References

[1] O. Walker, “Velocity-Based Training.” scienceforsport.com. https://www.scienceforsport.com/velocity-based-training/ (accessed Feb. 10, 2025).

[2] Grainger, “Salary and Hiring Data Portal,” Illinois.edu, 2024. https://ecs.grainger.illinois.edu/salary-data/data-portal

[3] “Bluetooth TechnologyProtecting Your Privacy,” Bluetooth® Technology Website, Apr. 02, 2015. https://www.bluetooth.com/blog/bluetooth-technology-protecting-your-privacy/

[4] “Bluetooth security,” Apple Support. https://support.apple.com/guide/security/bluetooth-security-sec82597d97e/web

[5] IEEE, “IEEE Code of Ethics,” ieee.org. https://www.ieee.org/about/corporate/governance/p7-8.html (accessed Feb. 11, 2025).

[6] ArtsemiR, “GitHub - ArtsemiR/Swift-ESP32-BLE-Remote-Control-Demo: Swift ESP32 BLE Remote Control Demo: A SwiftUI + Swift demo project illustrating capabilities for BLE-based remote control, including examples of data transmission between devices.,” *GitHub*, 2024. https://github.com/ArtsemiR/Swift-ESP32-BLE-Remote-Control-Demo (accessed April 21, 2025).

[7] wollewald, “ICM20948_WE,” *GitHub*, 2025. https://github.com/wollewald/ICM20948_WE/tree/main (accessed April 21, 2025).

[8] ArtsemiR, “ESP32 BLE Remote Control Demo,” *GitHub*, 2024. https://github.com/ArtsemiR/ESP32-BLE-Remote-Control-Demo/tree/main (accessed April 21, 2025).

Appendix A Power Subsystem

Table 3: Requirements and Verifications for Power Subsystem

Requirement	Verification	Verification status (Y or N)
The power subsystem must recharge the battery within 6 hours.	• Discharge the battery until its voltage is equivalent to 3.0V, or its discharged voltage • Plug the device into a USB power source • Check the voltage of the battery every hour until it reaches 4.2V, or its voltage when fully charged • Repeat at least once to ensure consistency	
The power subsystem must supply continuous power to the rest of the system for at least 2 hours from a fully charged battery.	• Fully charge the battery and begin operating the athletic tracking sensor, including gathering measurements to be sent to the mobile app • Utilize a stopwatch to track how long the device stays operational • Ensure the operational time the device has under one charge exceeds 2 hours • Repeat at least once to ensure consistency	
The power subsystem must not be overloaded by the device's components, failing to supply enough current for it to operate	• Fully charge the battery and begin operating the athletic tracking sensor • Verify that the correct voltage is being outputted by both regulators in the subsystem • Use a multimeter to measure total current consumption during both exercises used in testing • Verify that current draw does not exceed 1 A at any point during operation • Repeat at least once to ensure consistency	

Appendix B Control Subsystem

Table 4: System Requirements and Verifications for Control Subsystem

Requirement	Verification	Verification status (Y or N)
When the "Record Data" button is pressed, the control subsystem should determine if the user is hitting the necessary threshold for the velocity the user inputted.	• Have the user input a desirable and reasonable velocity into the mobile app. Observe if the MCU stores this value into its memory. • Have the user be in a proper position and press the button on the device • The user should then try to go way below the threshold. Observe if the boolean value for hitting the threshold is False. • The user should then be within the proper threshold. Observe if the boolean value for hitting the threshold is True. • The user should then be way over the threshold. Observe if the boolean value for hitting the threshold is False. • Alternate between these three conditions, observing the boolean value is in the proper state.	
When the "Record Data" button is pressed, the control subsystem should determine if the user is using the proper form.	• Have the user input the desired exercise as the weighted squat into the mobile app. Observe if the MCU stores this information in its memory. • Have the user place the device in the proper location on the back and press the button on the device, indicating the start of the workout • Observe if the MCU records the initial position of the user, and that the thresholds for proper form at certain locations are implemented. • Have the user be within the threshold. Observe if the boolean indicating if the user is using the proper form is True.	

	• Have the user be outside of the threshold. Observe if the boolean indicating if the using proper form is False. Have the user re-enter the thresholds. Observe if the boolean is True. • Have the user press the button on the device, indicating the end of the workout. • Have the user input the desired exercise as the bench press. Observe if the MCU stores this information in its memory. • Have the user place the device on the center of the barbell and press the button on the device, indicating the start of the workout. • Observe if the MCU records the initial position of the user, and that the thresholds for proper form at certain locations are implemented. • Have the user be within the threshold. Observe if the boolean indicating if the user is using the proper form is True. • Have the user be outside of the threshold. Observe if the boolean indicating if the using proper form is False. Have the user re-enter the thresholds. Observe if the boolean is True. • Have the user press the button on the device, indicating the end of the workout.	

B.1 Application Design and Results

```
    // Called when a peripheral is discovered during scanning.
    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String: Any], rssi
        RSSI: NSNumber) {
//        print(peripheral)
        if peripheral.name?.contains("ATS Microcontroller") ?? false {
            let adsServiceUUIDs = (advertisementData[CBAdvertisementDataServiceUUIDsKey] as? [CBUUID])?.compactMap({ data in
                data.uuidString
            })

            let newPeripheral = Peripheral(id: peripheral.identifier,
                                           name: peripheral.name ?? "Unknown",
                                           rssi: RSSI.intValue,
                                           advertisementServiceUUIDs: adsServiceUUIDs,
                                           peripheral: peripheral)
            if !peripherals.contains(where: { $0.id == newPeripheral.id }) {
                peripherals.append(newPeripheral)
            }
        }
    }

    // Called when a connection is successfully established with a peripheral.
    func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
        isConnected = true
        esp32Peripheral?.discoverServices([CBUUID(string: ESP32Constants.serviceUUID)])
        centralManager.stopScan()
    }


    // Function to initiate a connection to a chosen peripheral.
    func connectPeripheral(peripheral: Peripheral) {
        guard let foundPeripheral = peripherals.first(where: { $0.id == peripheral.id })?.peripheral else { return }
        esp32Peripheral = foundPeripheral
        esp32Peripheral?.delegate = self
        centralManager.connect(foundPeripheral, options: nil)
    }
```

Figure 11: Source Code for Connecting Application to ESP32

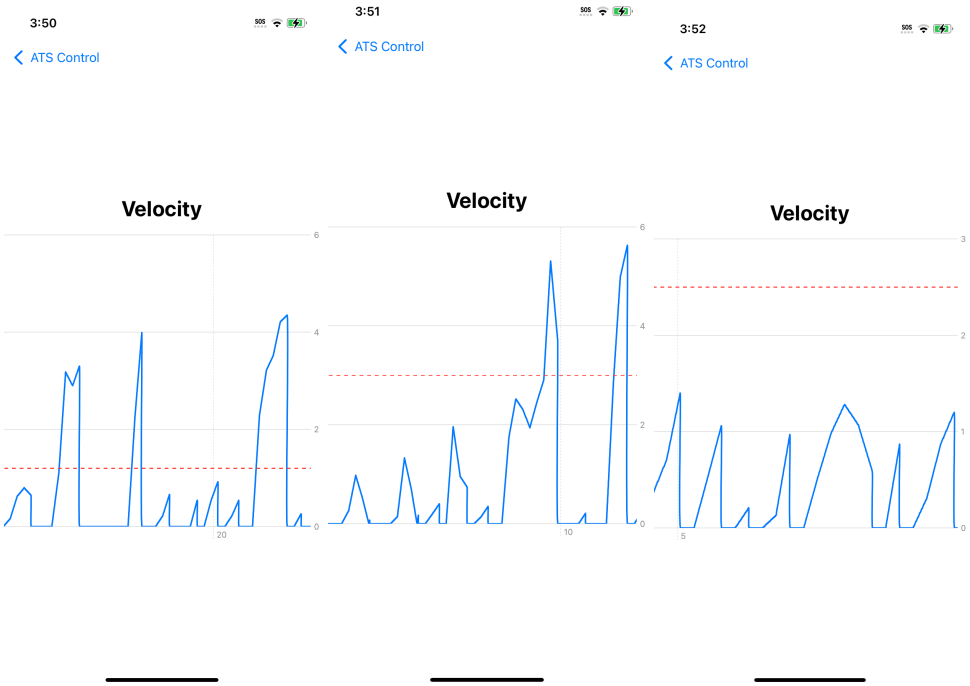

Figure 12: Velocity Graphs of the User at Different Thresholds

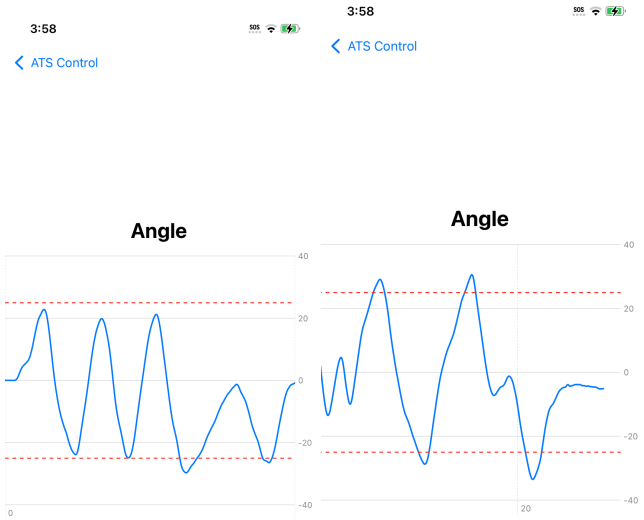

Figure 13: Orientation Graphs of the User

Appendix C Sensing Subsystem

Table 5: System Requirements and Verifications for Sensing Subsystem

Requirement	Verification	Verification status (Y or N)
The sensing subsystem must start recording once the ESP32 sends the proper signal.	• Have the ESP32 send the necessary signal that initializes the sensor and tells it to start recording. • Observe if the sensing subsystem starts sending data to the MCU.	
The sensing subsystem must collect gyroscope data often enough to alert the user of bad form usage within 0.5±0.2 seconds.	• Have the sensor record data. • Ensure that the data includes a gyroscope reading at least three times per second.	

C.1 Sensing Subsystem Design

```
#include <ICM20948_WE.h>

/* ICM-20948 Definitions */
#define CS_PIN 10   // Chip Select Pin

/* ICM-20948 Global Variables */
bool spi = true;
ICM20948_WE myIMU = ICM20948_WE(&SPI, CS_PIN, spi);

void setup(){
  /* ICM-20948 setup */
  Serial.begin(115200);
  while(!Serial) {}

  if(!myIMU.init()){
    Serial.println("ICM20948 does not respond");
  }
  else{
    Serial.println("ICM20948 is connected");
  }

  myIMU.setSPIClockSpeed(7000000);

  Serial.println("Position your ICM20948 flat and don't move it - calibrating...");
  delay(1000);
  myIMU.autoOffsets();
  Serial.println("Done!");

  myIMU.setAccRange(ICM20948_ACC_RANGE_2G);
  myIMU.setAccDLPF(ICM20948_DLPF_6);
  myIMU.setAccSampleRateDivider(1);
  myIMU.setGyrDLPF(ICM20948_DLPF_6);
  myIMU.setGyrSampleRateDivider(5);
```

Figure 14*: ICM-20948 Initialization Code Utilizing ICM20948_*WE Library

```
xyzFloat removeGravity(xyzFloat accel, float x_angle, float y_angle) {
  // Assume pitch and roll are in degrees; convert to radians
  float yRad = y_angle * PI / 180.0;
  float xRad = x_angle * PI / 180.0;

  // Gravity(in G's)
  float gravity = .99;

  //Find component of gravity per axis value
  xyzFloat gravityComp;
  gravityComp.x = gravity * sin(xRad);
  gravityComp.y = gravity * sin(yRad);
  gravityComp.z = gravity * cos(yRad) * cos(xRad);
  if(accel.z < 0) gravityComp.z = gravityComp.z*-1;

  // Subtract gravity component from measured acceleration
  xyzFloat linearAccel;
  linearAccel.x = accel.x + gravityComp.y;
  linearAccel.y = accel.y - gravityComp.x;
  linearAccel.z = accel.z - gravityComp.z;

  return linearAccel;
}
```

Figure 15 *: Code for Removing Gravity from Acceleration Values*

```
VelocityAndAccel computeVelocity(xyzFloat acceleration, xyzFloat angle) {
  unsigned long currentTime = millis();
  float dt = (currentTime - lastUpdateTime) / 1000.0; // Convert ms to s
  lastUpdateTime = currentTime;

  // Deadband to reduce noise-induced drift
  const float accelDeadband = 0.06;//2* 0.03;
  xyzFloat correctedAccel = removeGravity(acceleration, angle.x, angle.y);

  // Apply deadband
  if (abs(correctedAccel.x) < accelDeadband) correctedAccel.x = 0;
  if (abs(correctedAccel.y) < accelDeadband) correctedAccel.y = 0;
  if (abs(correctedAccel.z) < accelDeadband) correctedAccel.z = 0;

  // Integrate acceleration to get velocity
  velocity.x += correctedAccel.x * dt *9.81;
  velocity.y += correctedAccel.y * dt * 9.81;
  velocity.z += correctedAccel.z * dt * 9.81;

  VelocityAndAccel result;
  result.velocity = velocity;
  result.correctedAccel = correctedAccel;
  return result;
}
```

Figure 16 *: Code for Calculating Velocity for each Cartesian Vector*

Appendix D Feedback Subsystem

Table 6: System Requirements and Verifications for Feedback Subsystem

Requirement	Verification	Verification status (Y or N)
• The feedback subsystem's LEDs	• Turn on the device. Observe if the	

should be able to indicate the proper state of the device.	green LED turns on. Turn off the device. Observe if the green LED turns off. ● Place the MCU in a state where it waits for user input. Observe if the respective yellow LED is blinking. ● Press the button on the device. Observe if the yellow LED is solid. ● Place the MCU into data recording mode. Have it record an initial position and develop the thresholds for good form. Have the user stay within these thresholds. Observe if the red LED is off. ● Have the user go outside of the thresholds. Observe if the red LED turns on within (a certain time). ● Have the user re-enter the thresholds. Observe if the red LED is off within (a certain time).	
● The feedback system should activate/deactivate the vibration motors	● Place the MCU into data recording mode. Have it record an initial position and develop the thresholds for good form. Have the user stay within these thresholds. Observe if the motor is off. ● Have the user go outside the thresholds. Observe if the motor turns on within (a certain time). ● Have the user re-enter the thresholds. Observe if the red LED is off within (a certain time).	