Smart Insole

ECE 445 Design Document - Spring 2024

Project # 41

Ramsey van der Meer, Alyssa Huang, Tony Leapo

Professor: Jonathon Schuh

TA: Selva Subramaniam

1 Introduction

1.1 Problem

Many people enjoy hiking since it allows for people of all fitness levels to experience the outdoors. However, oftentimes the constant repetitive pounding on hikers feet can lead to soreness or even injury. Many factors contribute to the injury risk factor including a hiker's gait, fitness level, the amount of weight carried, terrain, and much more. Currently, there are no products on the market which can deliver personalized feedback on foot stresses experienced over the duration of a hike. This information can be crucial in selecting appropriate footwear or even improving walking techniques to prevent injuries. Additionally, this information could be repurposed to provide a metric to measure the difficulties of hikes, as trails that place a lot of pressure on your feet can be shared amongst avid hikers.

1.2 Solution

Our solution is to develop an insertable insole equipped with many integrated pressure sensors and external accelerometers, and gyroscopes. These sensors will help monitor the dynamics of the foot during a hike by capturing data on the distribution of pressure across the foot, as well as the intensity of impacts, and the foot's orientation and movements.

The insole will be constructed with durable but comfortable materials to ensure it does not alter the hiking experience negatively. It will connect to an external part that clips to the outside of the shoe. This external portion will contain the microcontroller and any other sensors such as a gyroscope and an accelerometer. The device will be able to connect wirelessly through BlueTooth to a smartphone interface, enabling hikers to receive real-time feedback of the sensor data during their hike. After the hike, the interface will provide a comprehensive summary of the collected data, presenting insights into areas of the foot that experienced the most stress and impact, as well as other data collected about the user’s walking habits. This summary will include visual representations such as heat maps and graphs, illustrating the pressure points and movement patterns.

Additionally, the interface will offer personalized recommendations based on the collected data. These could include suggestions for foot exercises, guidance on improving hiking techniques, and advice on selecting the right type of hiking footwear for individual needs.

By providing hikers with this detailed and personalized information, our solution aims to enhance the hiking experience, reduce the risk of foot injuries, and contribute to the overall well-being of hiking enthusiasts. The insole will be designed to ensure compatibility with a range of different types of shoes, and the type of data we will be collecting can be generalized to solve other orthotic issues.

1.3 Visual Aid

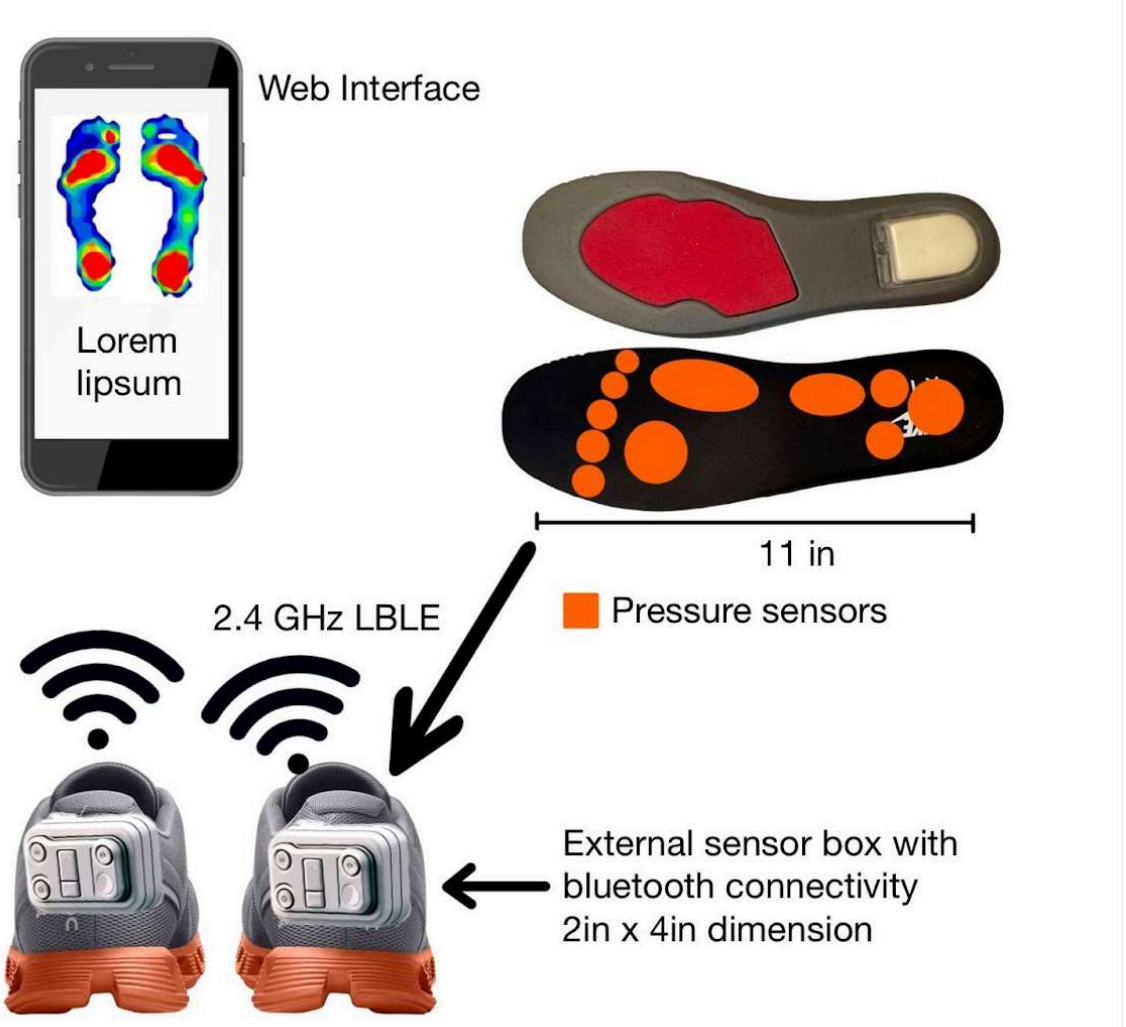

Figure 1: Visual aid demonstrating basic physical features of our product

1.4 High Level Requirements

Accurate Pressure and Sensor Values:

- Integrate the pressure sensor to be able to track pressure changes in distinct regions of the foot, these regions having a maximum area of 3 inches squared.
- The sensors will need to track foot pressure distribution, impact intensity, and foot motion using the accelerometer, pressure sensor, and gyroscope. We expect sensors to be accurate within 10%.

Accurate and Intuitive Data Integration:

- Once we properly collect the data from the physical foot exerting pressure, we want to make sure we can properly store, transport, and intuitively represent the data on our web application. This will require us to make proper use of the data we retrieve through sensible calculations and graphs.
- For storing and transporting, we will either store it within the microcontroller using an attached micro-SD card and send it in real time over Bluetooth to a separate device. We want real time transmission to not be significantly lossy, and only drop less than 5% of packets.
- For visualizing the data, we have chosen a heat map on our user interface to graphically show where the pressure distribution is. Other attributes regarding the overall hike, such as average pace and average orientation of the foot, can be extracted from the accelerometer and gyroscope data and displayed as normal graphs.

Wearable/Modular Physical Implementation:

- We want our device to be adaptable to various types of hiking boots, which is why we chose an insole that can be implemented very easily.
- We want our device to be wearable and enable our users to have 100% range of motion so that it does not deter from the hiking experience at all. Our objective of tracking analytics of the hike should augment and not bring down the hiking experience.

2 Design

2.1 Physical Design

Figure 2: Description of the clip-on device

Top Middle: Start Hike

Bottom Middle End Hike

Upper Left: Power Button

Upper Right: Bluetooth Pairing Button

Bottom Left: Status LED

Bottom Right: Status LED

The smart insole will have two main body of components: 1) The insole, which is a soft foam insole with embedded sensors that will collect pressure data about the feet, and 2) The clip-on, which will be a small clip-on device that connects to the insole through wires, and will contain the the main

microcontroller and communication components, as well as other sensors such as gyroscopes and accelerometers.

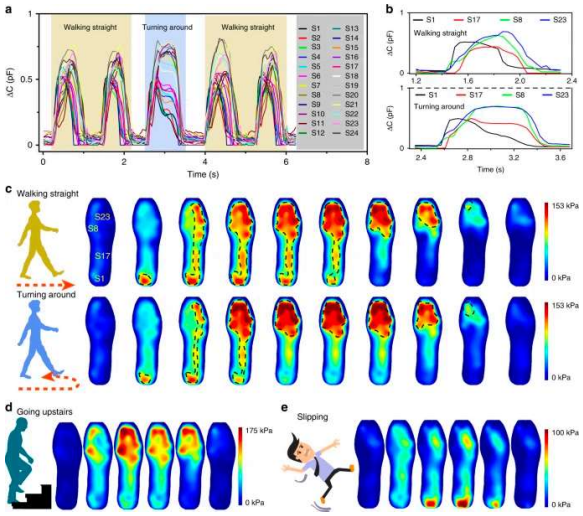

Figure 3: Real-time pressure mapping smart insole system based on a controllable dielectric layer [5][1]

The distribution of sensors is concentrated around the toes and the heel as we did some external research and found studies of the pressure distribution which showed highest concentration around the heel and ball of the foot. Larger sensors are also placed at less concentrated pressure areas so we can additionally monitor these areas.

The actual device will have the start hike, end hike, bluetooth pairing, and power buttons to control functionality of the device. It will also have status LED's on the control panel so users can at a glance see what the status of the device is.

[1] Tao, J., Dong, M., Li, L. *et al.* Real-time pressure mapping smart insole system based on a controllable vertical pore dielectric layer. *Microsyst Nanoeng* 6, 62 (2020). https://doi.org/10.1038/s41378-020-0171-1

2.2 Block Diagram

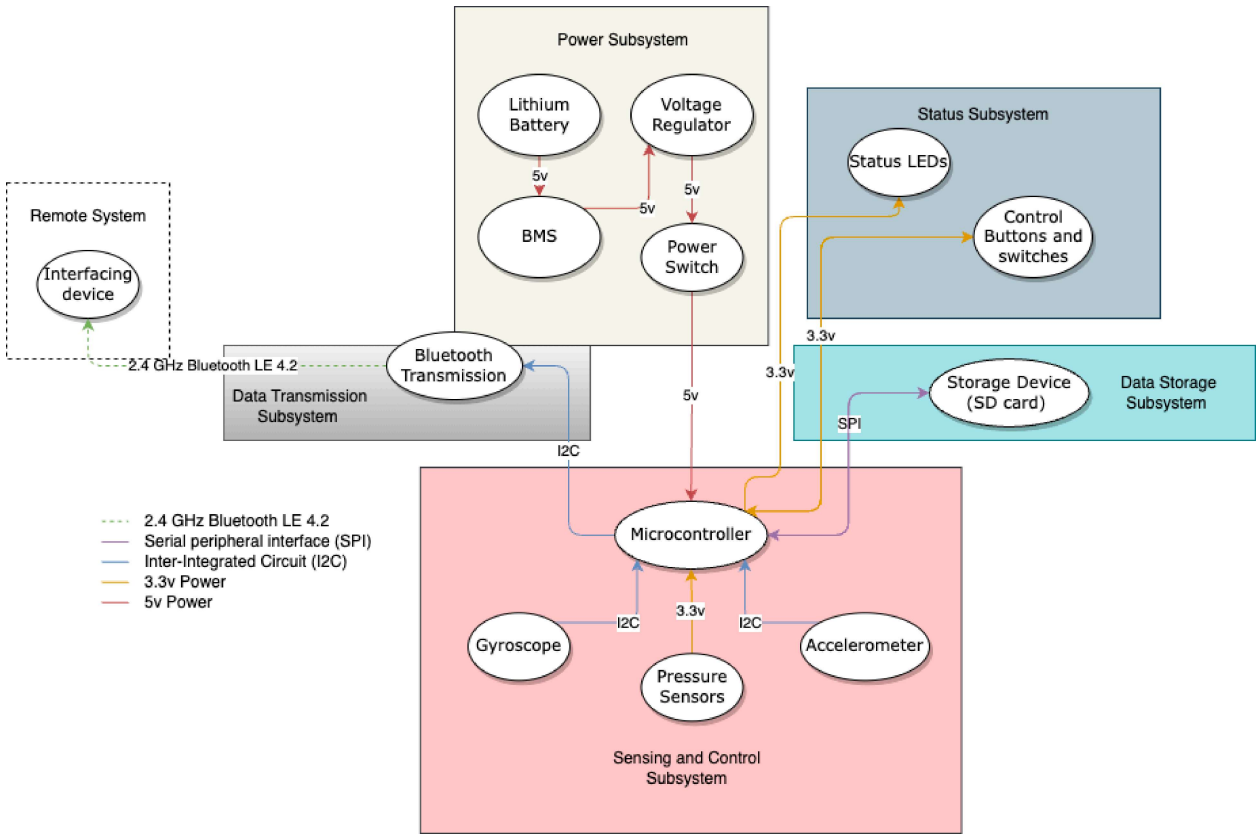

Figure 4: Block Diagram of each subsystem

2.3 Functional Overview

2.3.1 Remote Interface Subsystem

The Remote Interface Subsystem for the smart insole leverages the comprehensive data collected by the Sensing Subsystem to provide a user-friendly interface for monitoring and analyzing foot dynamics during hikes. The Sensing Subsystem, which includes an accelerometer, gyroscope, and pressure sensors, collects valuable data on foot movements, orientation, and pressure distribution. This data is essential for understanding the wearer's gait, the impact on different parts of the foot, and identifying potential areas of discomfort or injury. This subsystem will interface with the Data Transmission Subsystem through Bluetooth, sending information to the device such as connection status and receiving sensor information from the device.

Requirements	Verification
• Receive data transmitted from data transmission subsystem • Low latency	• View data acquired through bluetooth connection. • Data acquired for real-time display within 4 seconds
• Display data to users	• Show users acquired data in heat map, elevation gain and average pace. For these charts, the x dimension is time and the y dimension are the listed attributes. • Data must be accurate. We can test this by applying pressure to one side of the shoe and then monitoring its corresponding display on our interface. If the data is not accurate we will explore data post processing options to reduce noise, such as applying filters or taking a weighted average of numerous measurements.

2.3.2 Data Transmission Subsystem

The Data Transmission Subsystem is needed to facilitate the seamless transfer of data from the insole's Sensing Subsystem to the Remote Interface Subsystem for further analysis and user interaction. This subsystem ensures that the comprehensive foot dynamics data collected by the insole's sensors is reliably and securely communicated to the user's smartphone or another

designated device for real-time monitoring and historical data analysis. It will communicate with the microcontroller through I2C in the Sensing Subsystem to receive and transmit data between the remote system and the device. This includes sensor information from the microcontroller and connection status from the remote system.

Requirements	Verification
• Data can be transmitted to another device • Information not lost when transferring from sensor -> microcontroller -> bluetooth	• Validate bluetooth packets are being sent with programs like wireshark. • Validate that all data points are collected and sent by applying pressure to various different parts of the insole rapidly and then cross referencing the received output with your known input.

2.3.3 Data Storage Subsystem

The data storage subsystem will be in place to collect and store data that isn't able to be immediately transferred to the remote interface. The microcontroller will be able to read and write data into an SD card. It will store sensor data for future Bluetooth transfer by the microcontroller. This subsystem will be put to use in the case where the microcontroller disconnects from the interfacing device or as a buffering system if Bluetooth is unable to transfer the data being collected fast enough as sensor data is read. It will offer read/write access to the microcontroller through SPI. The reason we are using a SD card is because we anticipate needing approximately 50 MB of nonvolatile storage necessary for proper data collection, and according to the ESP32-S2 datasheet, only about 2 MB of flash is embedded, but the ESP32-C6 can support up to 1GB of external flash. [2]

[2][3] "Esp32-C6 Series Datasheet." *ESP32-C6 Series Datasheet*, 25 July 2023, www.espressif.com/sites/default/files/documentation/esp32-c6_datasheet_en.pdf.

Requirements	Verification
• Data can be stored on device	• Record data for a few minutes on the deceive. • Turn off the device and turn it back on to make sure no information is lost.
• Data can be read off device	• Make sure data can be read by the microcontroller for later transmission. • The data should be stored to a microSD card, so then take out that card and upload the data to our laptop to verify there is some external storage.
• Data storage is large enough to store multiple full hikes	• Sensors should poll at 64 Hz and downsample 8 Hz per sensor. Data on the high end would be a list of floats for each sensor. 4 bytes per float * ~54 sensors (50 pressure sensor, 2 gyroscope, and 2 accelerometer) * 5 Hz * 60 sec/min * 60 min/hour * 6 hours * 4 hikes = 93.312 Megabytes. • Validate the SD card used is at least 1 Gigabyte

2.3.4 Sensing Subsystem

For the insole, we will integrate a combination of sensors to accurately track and analyze foot movements and pressures during hikes. These sensors will include an accelerometer, gyroscope, and pressure sensors.

- Accelerometer: This sensor we will use to measure movements that users will make as well as sudden changes to motion to better get a sense of where and when impacts happen.
- Gyroscope: The gyroscope sensor will measure the rotational movements and orientation of the foot. This would provide insight into how the foot moves during a hike.
- Pressure Sensors: These sensors will be distributed across different areas on the insole to map the pressure exerted on different parts of the foot. This data is crucial for identifying high-stress areas and potential points of discomfort or injury. We could use thin and flexible pressure sensors like a Velostat conductive sheet.. This sensor works by increasing resistance as the sheet bends are applied to it, which we can measure with a voltage divider and see a change in voltage.

The data from these sensors will be collected and processed by a microcontroller unit external from the insole. This microcontroller would have to be capable of handling multiple inputs simultaneously from different sensors. We think the ESP32 fits the bill for a low-power, efficient microcontroller. This also includes Bluetooth for wireless data transmission to a smartphone interface. The microcontroller receives power from the power subsystem and communicates with the Data Storage Subsystem (read/write sensor data), Data Transmission Subsystem (Bluetooth transmission data), and Status Subsystem (User control and device status data).

Requirements	Verification
• Accelerometer can track velocity and acceleration	• Move accelerometer around • Make sure that the movement is read by checking accelerometer output • Accurate to within 5%, can verify with gravity measurements.
• Pressure sensors can track pressure in relation to each other accurately.	• Use set of 3 different weights • Place weights on all permutations of pressure pads on the insole.

	• Make sure that in the interface each pad is calibrated right so that the weights in relation to each other are right. • Only want them to be able to measure pressure accurately in relation to other sensors as sensors work when bent. Since some are already bent a bit to be placed on an insole just need to calibrate them accordingly.
• Gyroscope can track direction	• Change the orientation of the device. • Make sure that this is picked up by the microcontroller. • Accurate to within 5%, we can verify by rotating the shoe 90 degrees and verifying that the gyroscope registers this exactly.
• Processing delay can keep up with sensor read rate	• Validate that all sensor data is being written to SD card and none is lost due to overflow of write buffer due to reading too fast. ○ To verify we can record for 60 seconds specified in a test software program and check we have 60 * 8 samples for each sensor
• The enclosure that the sensing subsystem will be housed in can withstand the hiking	• Validate that the sensors do not get damaged and the sensor box can withstand small samples of water

experience/slightly wet and non-ideal weather	being poured on it

2.3.5 Status Subsystem

LEDs will be added to provide clear, visual indications of various statuses. We would include a power status LED indicating when the device is running. This LED could be repurposed for power status, and change to a green color when the insole is charging. It might flash red when the power is low. We could also incorporate LEDs for other statuses, such as Bluetooth connectivity (whether or not bluetooth is activity paired or if it is in pairing mode), or a warning LED for sensor malfunction or disconnection. These LEDs will not only provide an additional interface for users to look at and easily understand the status of their device. This would also have the benefit of having much less power draw than a screen interface. Additionally this subsystem will be in charge of coordinating device functions. We will need it to specify if we want to be recording data for a hike or if we just want to be offloading data to our external web interface. Additionally we will need an end hike button to allow users to end a hike once they finish. The last button functionality we think we need is a bluetooth connection button so we can connect to our external web interface. This subsystem will communicate its button data with the microcontroller, and it will also light up status LEDs via control from the microcontroller in the Sensing Subsystem.

Requirements	Verification
• Users can tell status of device	• Make sure LEDs light up at the right times to indicate status of device • Red for low power • Flashing blue for bluetooth pairing • Flashing green for recording hike • Orange for idle and not doing anything • Yellow for offloading data to external

	device
• Start and end hike button	• Make sure data isn't being recorded by probing SD card connection and shouldn't be any data written • Press start • Make sure data is being recorded by making sure a new file is being created and written to • Press end • Make sure data isn't being recorded and a hike file has been written to SD card and completed with sensor data included.
• Bluetooth button	• Make sure we can press this button and connect from our web interface to our device.

2.3.6 Power Delivery Subsystem

The power subsystem would contain a lithium-ion battery. Due to its compact size, rechargeability, and widespread availability, we find lithium-ion batteries to be the best battery type to integrate. We would mount this battery externally from the insole to power the device. Considering the power requirements of the sensors (accelerometers, pressure sensors, and gyroscopes), the microcontroller, LEDs, and the Bluetooth module for data transmission, a battery capacity in the range of 200-300mAh for a six hour charge would likely be sufficient. For reference, a FitBit sense worn on the wrist has a battery of about 266 mAh at 3.85 V. This capacity should provide enough power for a hike (approximately 4-6 hours) on a single charge, assuming moderate data recording and transmission frequency. The battery would be placed

away from the insole. Our goal was to provide about 24 hours on a single charge, and the reverse engineered calculation for that is available in the tolerance section.

The insole device will primarily operate on two voltage levels: 3.3V and 5V. Different parts of the system, such as sensors, the microcontroller, and communication modules, will require differing voltage levels to operate optimally. In the case where components with different operating voltages need to communicate or interact, such as a microcontroller interfacing with sensors or LEDs, a level shifting circuit will be necessary to ensure compatibility.

To manage and regulate the power supply, we've implemented the Power Subsystem, which includes a Battery Management System (BMS) and a voltage regulator. The BMS will help monitor the state of the lithium-ion battery, collecting information such as battery level, and also protect the battery from damage from sub-ideal electrical conditions such as low voltage, high current, high temperature, or short-circuiting. In the case that these occur, the BMS will be able to shut off the battery to protect it from operating in unsafe conditions. Incorporating features like low-power modes for inactive periods and efficient power conversion circuits would enhance the overall efficiency and longevity of the battery, crucial for sustaining the device throughout the duration of a hike. The battery management system will interface with the microcontroller to ensure proper battery status updates are received to the insole device and the remote interface.

The remaining electrical components of the insole device, including the SD Card, sensors, and Bluetooth module, will be powered by the microcontroller. The microcontroller will serve as the central hub for power distribution and data management across the various subsystems. It will receive power from the voltage regulator and manage its distribution of power to all other subsystems.

Requirements	Verification
• Consistent power delivery	• Measure voltage delivered to microcontroller and validate that it is

	consistent at 5v without too much fluctuations by probing the ground and live pads with a voltmeter.

• Rechargeability	• Use the device until it runs out of battery
	  ○ Measure voltage of battery and validate it is dead
	• Charge the battery until fully charged
	  ○ Measure the voltage of the battery and make sure it is fully charged according to the data sheet.
	• Show that we can run the device on the recharged battery

• Last all hike	• Boot up device, start a hike function and run device until it stops working. At this point charge it. Once changed, verify a hike file has been properly created and saved to SD card. To do this try to offload the file through the web interface. • Make sure during this testing at the minimum 6 hours, and ideally can last 24 hours. Our current capacity battery that we purchased is 2000 mAh which should be able to provide more than 24 hours, and more details on this is in the tolerance section. • Another way to verify is to monitor

	the power draw of the system while running and calculate the total time it can run. To do this voltage*current*time running = watt rating of lithium ion battery.

2.4 Hardware Design

2.4.1 Microcontroller

Microcontroller: ESP32-C6-WROOM-1-N4

Our design will be using the ESP32 as our microcontroller. It will interface directly with the MPU-6050 gyroscope and accelerometer, pressure sensors, status LEDs, SD Card, and control buttons, as well as receive power from the power supply. These acquired measurements will then be sent wirelessly through Bluetooth to the external device

The ESP32 will be powered by a +3.7V battery regulated by an LP2953 voltage regulator outputting +3.3V. The ESP32 will provide power to all peripheral sensors and modules.

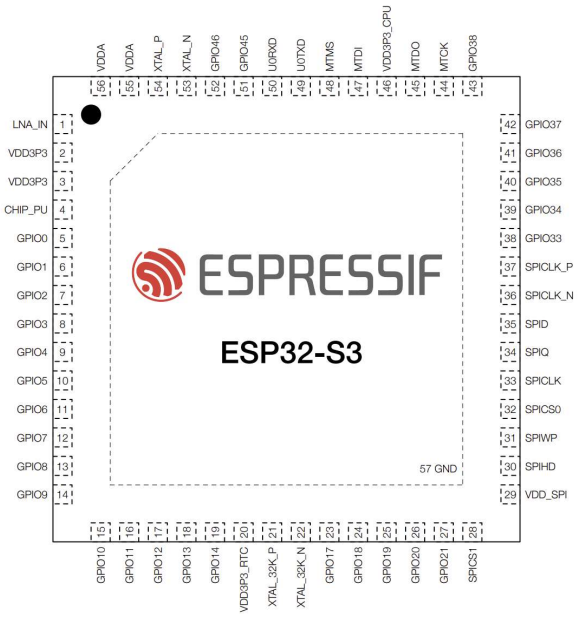

Figure 5: Top Level View of ESP32-S3 Pin Layout [3]

2.4.2 Accelerometer and Gyroscope

Accelerometer and Gyroscope: MPU-6050

To measure movement and rotation, we will be using the MPU-6050 which is a 6-axis sensor with both a gyroscope and an accelerometer. It will receive +3.3V power via the ESP32 microcontroller.

Accelerometer and gyroscope data will be sampled at 64 samples/second and digitized by the MPU-6050 before being sent to the ESP32 through I2C which will then downsample to 8 samples/second in an attempt to remove some noise.

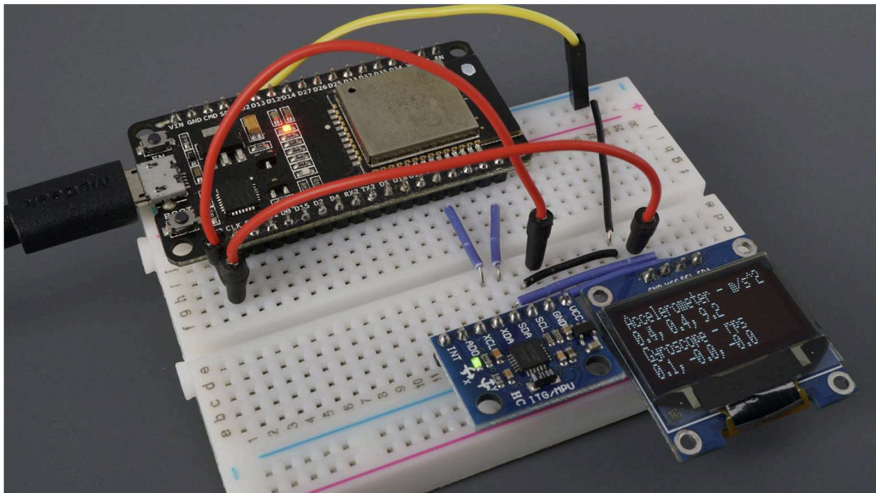

Figure 6: Sample picture of our accelerometer/gyroscope integrated with the ESP32 microcontroller.

2.4.3 Pressure-Sensing Insole

Pressure-Sensitive Conductive Sheet: Velostat

Conductive Copper Tape: Copper Tape

Base Layer: Shoe Insole

We will be designing the pressure-sensing insole using Velostat, a pressure sensitive material, and conductive copper tape due to it being thin enough to be comfortable to wear. The pressure insole will comprise of two layers, one layer containing small, circular cutouts of Velostat which will be spread out around the insole to provide pressure data at different points on the foot. Copper tape will be on this layer so we can measure the voltage drop across each Velostat point. The top layer will have only copper tape to provide voltage to the Velostat.

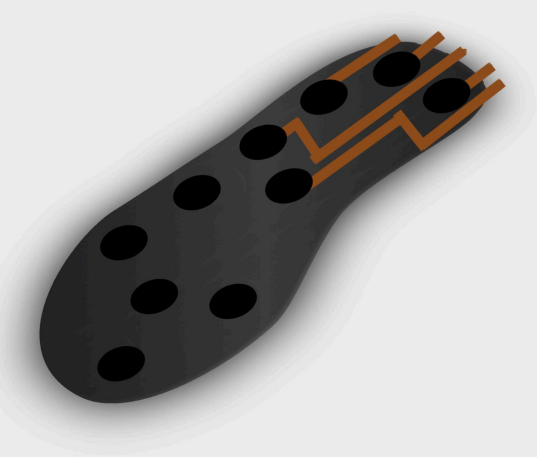

Figure 7: Sample picture of how the Velostat will be wired on the insole.

Figure 7 above illustrates an example sensor (black) layout for the bottom insole layer along with copper (brown) conducting tape. The rest of the sensors can be wired through the bottom of the insole. Then every exposed copper region will be covered in insulating tape to ensure that the sensor data do not interfere with each other. The top insole layer will have copper tape to provide a voltage source to each of the sensors in parallel.

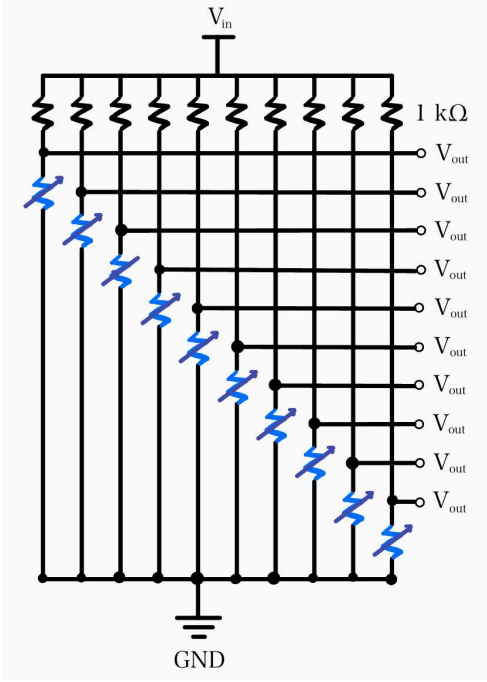

Figure 8: Schematic for pressure sensor.

We will design our pressure sensor circuit according to the above schematic. Essentially, we will measure Vout and use the voltage divider rule to measure the resistance of the Velostat. Since Velostat increases resistivity as it is pressurized, the resistances of the blue resistors will increase, which will

increase the corresponding Vout value. We will have to calibrate these voltages with a bias as some sensors will already be bent when placed on the insole. But overall, it is a very simple circuit design for the pressure sensor.

2.4.4 Power

Battery: Lithium-Ion Battery

Voltage Regulator: LP2953

The battery will be a rechargeable lithium ion battery that outputs +3.7V. The battery includes protection circuitry that will act as the BMS, keeping the battery from overcharging or overuse, and protecting the battery cells against shorts from the output. The battery will also be connected to a LP2953 voltage regulator that will output two levels of voltage, +3.3V and +5V to power the rest of the electronics.

Below are the power requirements for the components of the smart insole device:

Component	Voltage Requirement
ESP32-C6-WROOM-1-N4 Microcontroller	Operating Voltage: +2.2V - +3.6V
MPU-6050 Accelerometer and Gyroscope	VDD: +2.3V - +3.4V VLOGIC: +1.8V or VDD
Lithium Ion Battery	Output Voltage: +3.7V
Micro-SD Card Adapter	Operating Voltage: +3.3V
LP2953 Voltage Regulator	Input Voltage: -20V to +30V Output Voltage: +1.2V to +29V Fixed Output: +3.3V or +5V

2.4.5 Memory

Micro-SD Card Adapter: Micro-SD Card Module

SD Card: Micro-SD Card

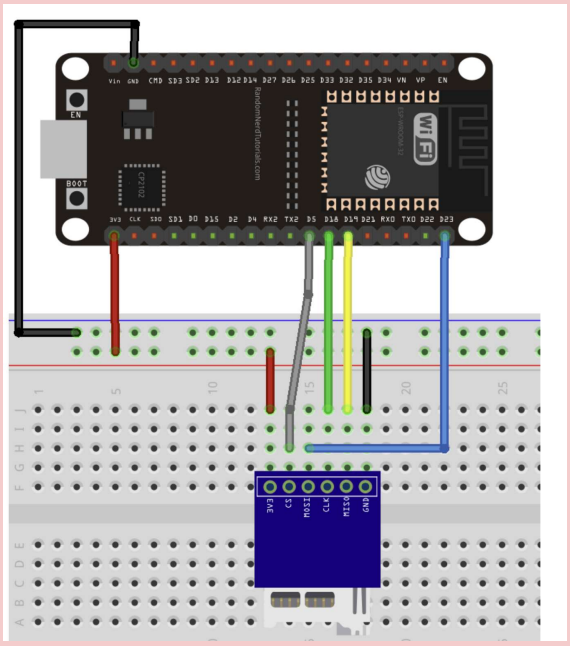

Figure 9: This indicates how the ESP 32 is connected to
and the micro-SD card adapter. [4]

The linked SD card is capable of holding up to 16GB of flash and RAM memory, but the ESP32-S2 is limited to supporting only 1GB of external flash. This is well within our requirements as we only needed about 50MB of non-volatile memory.

2.4.6 Buttons

The buttons in the Status Subsystem will be physical buttons that the user can push from the outside of the shoe. They will be active user input of the device and will be wired directly into the GPIO pins of the ESP32. The following buttons will be on the device for the user to press:

On/Off Button

This button will serve as an On/Off button for the insole device. When pressed, it will toggle the circuit on or off depending on the previous state.

Start Recording/Stop Recording

These two buttons will trigger the microcontroller to start or stop recording data from the sensors and sending them to the remote device.

2.4.7 Status LEDs

The smart insole device will also contain status LEDs that are wired to the GPIO pins of the ESP32 and will serve to tell the user what state the device is in.

Green LEDs

One green LED will be used to indicate if the device is on or off. If the green LED is blinking, it will indicate Bluetooth connecting status. A steady light will mean that the smart insole is connected to the remote device.

Blue LED

One blue LED will be used to indicate recording status, meaning that the device is actively transmitting data to the remote device.

Orange LED

One orange LED will be used to indicate the low battery status of the device, which will turn on once the device reaches 15% battery.

2.5 Software Design

2.5.1 Data Collection, Aggregation, and Transmission

Figure 10: Software Flowchart

The software outlined in the flowchart is designed for managing sensor data recording and Bluetooth connectivity for a hiking activity tracker. It begins by determining whether it should be in Bluetooth pairing mode. If so, it checks for a connected device and, if not found, enters a retry loop until a timeout is reached. Once a device is connected, the software transitions to the main process.

In the main operational phase, the software checks if the recording switch is enabled. If the recording is active, it captures sensor data from the hike and writes this information to storage. The recording continues unless the user decides to end the hike, in which case the software completes the data writing process.

Simultaneously, in a process that runs asynchronously to the main task, the software sends data packets to another device connected through a web interface. This could be for real-time tracking or backup purposes or just to review old completed hikes.

Upon completion of a hike, the software queries if all data recording is complete. If affirmative, it proceeds to delete the current hike file and prepares to start recording a new hike. The software also performs a final check to determine if all hikes have been processed and if there is no ongoing data recording. If both conditions are met, it safely terminates the Bluetooth connection, effectively concluding its operations. Below is some preliminary code and pseudocode for how we will be interfacing with the sensors:

```
//basic pseudocode for pressure sensor:
for velostat_sensor_x in velostat_sensors:
  current_sensor_draw = reading_from_sensor_x
  pressure_for_region_x = fitting_current_to_curve(current_sensor_draw)
  //each sensor will have a unique calibration curve that should increase
  linearly but will require calibration, so we will functionalize it
```

Figure 11: Pseudocode for the Velostat sensor on the insole.

```
'''
Helper function for gyroscope sensor values to help with rotation math
'''
def axis_angle_to_rotation_matrix(axis, angle):
    axis = axis / np.linalg.norm(axis)  # Normalize axis
    x, y, z = axis
    cos = np.cos(angle)
    sin = np.sin(angle)
    rotation_matrix = np.array([[cos + x**2*(1-cos), x*y*(1-cos) - z*sin, x*z*(1-cos) + y*sin],
                                [y*x*(1-cos) + z*sin, cos + y**2*(1-cos), y*z*(1-cos) - x*sin],
                                [z*x*(1-cos) - y*sin, z*y*(1-cos) + x*sin, cos + z**2*(1-cos)]])
    return rotation_matrix
'''
Function that takes in gyroscope sensor values (wx, wy, wz) as input and outputs a determined final orientation of the sensor.
Orientation will be helpful if you want to determine what direction the foot is facing for most of the hike (measure of uphillness and difficulty)
'''
def track_orientation(gyro_data):
    # Your implementation starts here:
    dt = 1/100 #depending on sampling rate
    df = pd.read_csv(gyro_data)
    #pretend it is a CSV for now, how we extract the data from Bluetooth or SSd can vary, could just be an array
    data = df.values
    # Initialize total 3D rotation matrix
    R = np.eye(3)

    for i in range(len(data)):
        omega_x, omega_y, omega_z = data[i]
        delta_theta = np.sqrt(omega_x**2 + omega_y**2 + omega_z**2) * dt

        rotation_axis = np.array([omega_x, omega_y, omega_z])

        rotation_axis_global = np.dot(R, rotation_axis)

        delta_R = axis_angle_to_rotation_matrix(rotation_axis_global, delta_theta)
        #this is a self defined function that converts the gyroscope values to a rotation

        R = np.dot(delta_R, R)
```

Figure 12: Preliminary code for interfacing with the gyroscope data to get the orientation of the shoe

```
'''
Helper function to get the typical length of a step
'''
def get_step_length():
    height=1.75 # in meters
    return 0.415*height

'''
Function for the accelerometer data to calculate the number of steps a person has taken, which will be useful for displacement.
'''
def calculate_steps(accel_data):
    # deal with the DC offset
    accel_data['x'] -= accel_data['x'].mean()
    accel_data['y'] -= accel_data['y'].mean()
    accel_data['z'] -= accel_data['z'].mean()

    #extract the data readings
    accel_data['magnitude'] = np.linalg.norm(accel_data[['x','y','z']].values, axis=1)
    #smooth out some of the points
    accel_data['filtered_magnitude'] = accel_data['magnitude'].rolling(window=10, min_periods=1).mean()

    #adjust parameters for peak
    sampling_rate = 100 # Sampling rate of the accelerometer (need to verify)
    min_peak_distance_seconds = 0.5  # Minimum separation between adjacent peaks in seconds
    min_distance = int(min_peak_distance_seconds * sampling_rate)  # Convert to number of samples
    peaks, _ = find_peaks(accel_data['filtered_magnitude'], distance=min_distance)

    #extract the actual peak indices
    step_timestamps = accel_data.index[peaks].values

    #put in the format they want
    step_lengths = [get_step_length()] * len(step_timestamps)
    steps = pd.DataFrame({"timestamp": step_timestamps, "steplength": step_lengths})

    return steps

'''
Used to calculate the final ending spot for the person, and the accelerometer.
We can also use the accelerometer data in a different way, as just measuring the amount of force the person exerts on their foot/sensor throughout the hike,
and using that as valuable insight. The total displacement is jsut an example metric of what we can do with the accelerometer data.
'''
def calculate_final_position(data_from_step_function, start_position):

    #find what direction each step is being taken in using the walking direction angle, x is cos(theta) and y is sin(theta)
    dx = np.sum(data_from_step_function * np.cos(orientation_from_gyroscope_calculation))
    dy = np.sum(data_from_step_function * np.sin(orientation_from_gyroscope_calculation))

    displacement = (dx, dy)

    return displacement
```

Figure 13: Preliminary code for calculating the overall displacement of the shoe using accelerometer data

2.5.2 Web Interface

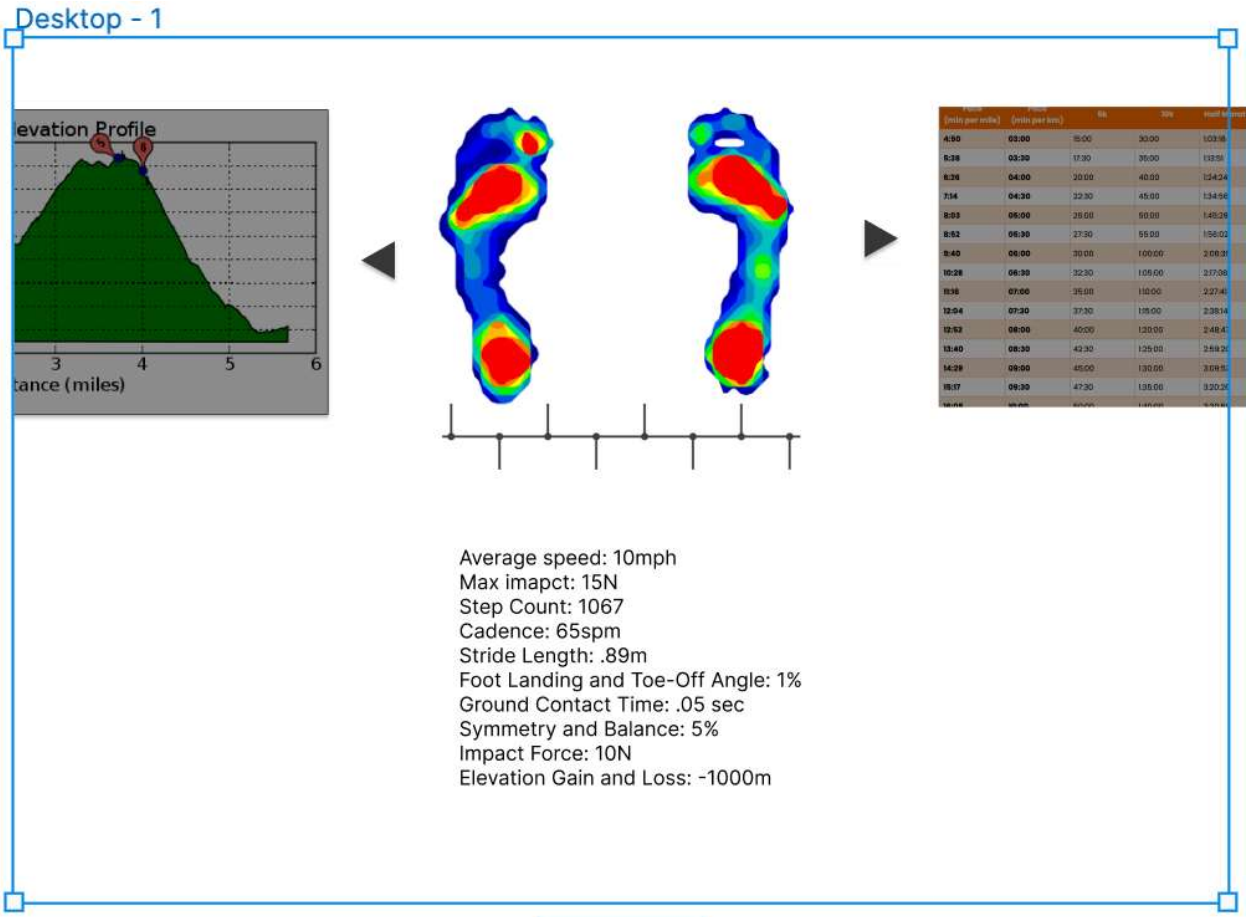

Figure 14: Sample snapshot of web interface

We plan to create a web interface that can be used to offload data from our esp32 device. To do this we have to set up our ESP32 to have bluetooth services that our web application can interface with. To do this we need to use the modules BLEServer, BLEService, and BLECharacteristic. BLEServer is the main module through which logic is run. It handels communication, including connections, disconnections, and data transfers with BLE clients. The BLESevice handels collections of characteristics and behaviors that perform a specific function or feature of the device. These characteristics are defined within services, carrying the actual data. They have unique UUIDs and properties that define how the characteristic can be used (e.g., read, write, notify). The web application uses the Web Bluetooth API to request access to BLE devices, connect to the ESP32, and interact with its services and characteristics. The application can read from or write to the characteristics based on their properties, enabling two-way communication. By setting up our ESP32 device with bluetooth logic we can then connect to it though our web interface. To summarize the process and handshake the devices will go through a Discovery phase where the Next.js application discovers the ESP32 through BLE advertising. Then it will establish a

connection phase where the application requests a connection to the ESP32. Once connected, it can discover the ESP32's services and characteristics. Finally we can move to the data transfer phase where the application reads from or writes to the characteristics, depending on the implementation. For example, it can subscribe to notifications for a characteristic to receive updates whenever the ESP32 changes the characteristic's value. This will allow us to receive data from our sensors on our insoles though these characteristic data packets. After which we can update one of these characteristics to acknowledge a successful data retrieval at which point it will send new data. After receiving this data we plan to display and summarize the hiking statistics through charts and graphs. We plan to use next.js which is a react framework, from which we can build off of and install useful modules to create these charts. Some NPM modules which may be of use are react-grid-heatmap, chart.js, and kepler.gl.

2.6 Tolerance Analysis

The largest potential risk would be the ability of the microcontroller to maintain the draw of everything it is connected to. The microcontroller we plan on using is an ESP32-S2 which operates within 3.3-5V, and on average draws 8-40 mA at 3V according to its datasheet [3]. This should be enough current draw to supply our sensors and LEDs at this voltage. The sensors will draw approximately 3 mA per sensor at 3.3V, and the LED should draw approximately 15 mA at 3.3V according to their respective datasheets. Since we have 3 sensors (gyroscope, accelerometer, and pressure sensor), and a single multi-colored LED, this places the total current draw at roughly 24 mA at 3.3V. This is within the operational range for the ESP32-S2. One aspect that may need further analysis is whether the microcontroller can withstand adding the SD card storage system to it, but it should be able to as the average current draw for the ESP32-S2 is about 150-200 mA away from its maximum current draw. We are not worried about overheating as the microcontroller isn’t near max draw and physically won’t be confined to an extremely small space, as it will rest on the shoe near or around the tongue.

Let us perform some calculations to derive our required battery capacity for supporting this type of system for 24 hours. The total current draw of the system is the microcontroller (8-40 mA) plus the three sensors (3 * 3mA) plus the LED (15ma) which gives us an average current draw of 32-64 mA. For a battery life of 24 hours, that would require a 32-64 mA * 24 hours = 768-1536 mAh capacity. We found a battery that operates at 2000 mAh at 3.7V, so assuming it provides above 3.3V 80% of the time it will provide a consistent 1600 mAh which is perfectly what we need.

2.7 Cost Analysis

2.7.1 Labor

Assuming minimum wage of $50/hour, labor costs per person would amount up to $50/hour (approximate salary for a ECE graduate) * 2.5 (overhead factor) * 125 hours (estimate number based on previous semesters workloads) = $15,625 per person. This totals to $46,875 worth of total labor costs throughout the semester for all group members.

2.7.2 Parts

Part	Description	Manufacturer	Qty	Cost	Link
3D Printing + Filament	3D print an enclosure for the electronics.	Using school supplies: SCD, OpenLab	1	$20 for a spool	
Gyroscope + Accelerometer	Dual accelerometer and gyroscope MPU-6050 which has known integration with the ESP32 chips	HiLetGo	2	$6.49 per	https://www.amazon.com/HiLetgo-MPU-6050-Accelerometer-Gyroscope-Converter/dp/B078SS8NQV
Velostat	Pressure-Sensitive Conductive Sheet	Adafruit	1	$4.95	https://www.adafruit.com/product/1361

Copper Tape	Conductive copper tape	Sparkfun	1	$3.95	https://www.sparkfun.com/products/10561
Electrical Tape	Insulating tape	Amazon	1	$4.99	Electrical Tape Link
Shoe Insole	Soft insole that will be used as base for device	Amazon/Sintege	1	$7.99	Insole Link
Lithium Ion Battery	3.7-4.3V Battery with 2000 mAh	Adafruit	2	$12.50 Lithium Ion Battery - 3.7V 2000mAh : ID 2011 : $12.50 : Adafruit Industries, Unique & fun DIY electronics and kits	https://www.adafruit.com/product/2011
Battery Management	Battery charge board with voltage protection	Amazon	2	$4.99 for 2	https://www.amaz

System					on.com/JacobsParts-Smallest-Charging-Protection-Indicators/dp/B0CBNQH836?source=ps-sl-shoppingads-lpcontext&ref_=fplfs&smid=A2TWRNEH2276JE&th=1
Microcontroller	ESP-32-C6-WROOM-1-N4	Adafruit	2	$2.95 per	https://www.adafruit.com/product/5670
PCB Board	To be ordered				
Resistive Sheet	Pressure-sensitive conductive sheet	Adafruit	2	$4.95 per	https://www.adafruit.com/product/1361

Resistors/Switches/Capacitors/Buttons/LED	Need one for each area of the pressure sensor we are tracking, set at 8 now.	Get from ECEB	>8	Free	N/A
Micro-SD Card Adapter	A module used to help the microcontroller interface with the SD card	MakerAdvisor	2	$1.69 per	https://makeradvisor.com/tools/sd-card-module/
Micro-SD Card	A storage device for external nonvolatile memory	MakerAdvisor	2	$6.97 per	https://makeradvisor.com/tools/microsd-card-raspberry-pi-16gb-class-10/

2.8 Schedule

Week	Task	Team Member
1/15	Initial Web Board Post	Everyone
1/22	Laboratory safety training CAD assignment	Everyone
1/29	Project approval	Everyone

2/5	Proposals Team Contract	Everyone
2/12	Design Document Start	Everyone
2/19	Design Document Due	Everyone
2/26	PDB Design Sensor prototyping and testing • Stress sensor • Accelerometer • Gyroscope ESP32 Storage testing	Everyone Everyone Alyssa Tony Ramsey Ramsey + Tony
3/4	3/5 PCB Order Due for only sensors Sensor integration with microcontroller Get test data for web interface	Alyssa Tony + Ramsey Tony + Ramsey
3/11	Spring break	
3/18	2nd PCB order due + control subsystem Data storage with microcontroller and sensor data Start prototyping interface with test data	Alyssa Tony + Ramsey Tony + Ramsey
3/25	Data transmission from microcontroller side Data receiving from interface side PCB Testing 3rd PCB order due	Tony + Ramsey Tony + Ramsey Alyssa Alyssa
4/1	Integrate interface with real data 4th PCB order due	Tony Alyssa
4/8	Make interface clearer 5th PCB order due	Ramsey Alyssa
4/15	Mock Demo	Everyone

4/22	Final Demo	Everyone
4/29	Final Presentation	Everyone

3 Ethics and Safety

3.1 Ethical Concerns:

- Privacy: Privacy is of vital importance for all devices integrated into the daily life of the users. For securing privacy, we will have a very clear user interface and set of LEDs that will indicate when the device is recording data, and there will be physical buttons or a clear button in the user interface to end the recording. We will be sure to comply with the IEEE standards on data privacy, specifically those on personal data[3].
- Data: We will ensure that user data is not being tracked when not permitted by the user. We also will not track any data that is not necessary to the operation of the device. All the accelerometer and gyroscope data will be the relative motion of the foot, as there will be no location tracking with our device. We will be compliant with the IEEE standards on

[3] [6] “IEEE Privacy Policy.” *IEEE*, www.ieee.org/security-privacy.html.

wearable electronics and their security objectives, specifically the objectives regarding data processing terminal application software and wireless communication[4].

3.2 Safety Concerns:

- Physical Safety: Physical safety is our top concern as we want to ensure that our device does not impede on the hiker's ability to conduct their hike. This will be achieved through our design and placement of the sensors to ensure the hiker has full range of motion while walking.
- Electrical Safety: Since the electronics in the design of this project are relatively low voltage and current, there are not major safety concerns. When assembling the circuitry we will follow relevant safety standards and ensure all wires are properly grounded, not exposed, or close to shearing[5]. Accidental/incidental misuse of our device should not result in significant harm, as the maximum voltage on this system is 5V. We will make sure our system is compliant with the IEEE standards on wearable electronics, specifically the minimum requirements for our type of lithium cell and battery and their discharge rate. To address water leakage, we wrap our full electrical subsystem in an enclosure and we can waterproof that enclosure. This can be achieved by adding rubber gaskets that we have available to create a watertight seal. We can additionally layer the inside with nylon to further prevent leakage. Preventing against submersion and dampness is important for our device since hiking will expose it to multiple terrains.

---

[4] [2] "IEEE Standard for Wearable Consumer Electronic Devices--Overview and Architecture," in IEEE Std 360-2022 , vol., no., pp.1-35, 25 April 2022, doi: 10.1109/IEEESTD.2022.9762855.
[5] [2] "IEEE Standard for Wearable Consumer Electronic Devices--Overview and Architecture," in IEEE Std 360-2022 , vol., no., pp.1-35, 25 April 2022, doi: 10.1109/IEEESTD.2022.9762855.

4 References

[1] MozDevNet. “Web Bluetooth API - Web Apis: MDN.” *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API. Accessed 22 Feb. 2024.

[2] "IEEE Standard for Wearable Consumer Electronic Devices--Overview and Architecture," in IEEE Std 360-2022 , vol., no., pp.1-35, 25 April 2022, doi: 10.1109/IEEESTD.2022.9762855.

[3] “Esp32-C6 Series Datasheet.” *ESP32-C6 Series Datasheet*, 25 July 2023, www.espressif.com/sites/default/files/documentation/esp32-c6_datasheet_en.pdf.

[4] Beaver, I., Baars, J., Loveman, J., Santos, S., Rose, Evans, D., Navarro, B., B., E., Hlr, Brody, A., Seysen, C., Peteghem, D. V., Aguilera, C. A., Tekad, Luiz, Jegan, Ilker, Ilker, Zuly, … Hietala, H. (2023, July 31). *ESP32: Guide for microsd card module arduino*. Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-microsd-card-arduino/

[5] Tao, J., Dong, M., Li, L. *et al.* Real-time pressure mapping smart insole system based on a controllable vertical pore dielectric layer. *Microsyst Nanoeng* 6, 62 (2020). https://doi.org/10.1038/s41378-020-0171-1

[6]“IEEE Privacy Policy.” *IEEE*, www.ieee.org/security-privacy.html.