University of Southern Indiana
Pott College of Science, Engineering, and Education
Engineering Department

8600 University Boulevard
Evansville, Indiana 47712

Wearable Multi-Sensor Safety Monitor

A Monitoring System for Elderly and Dementia Patients

Josh Duzan, BSEE, Jenna Hall, BSEE, Patrick Kasinger, BSEE

ENGR 491 – Senior Design
Spring 2025

Project Advisor: Dr. Ryan Integlia

ACKNOWLEDGEMENTS

In this section, we would like to acknowledge

Dr. Ryan Integlia

Roberto Batista

Jenario Johnson

Kendon Rickets

Dr Ronald Diersing

Dr. Arthur Chlebowski

Abstract

The purpose of this project was to create a safety monitoring device for dementia and elderly patients that can track a person's location and send alerts if they wander, fall, or have irregularities in activity levels. For people who need constant monitoring, an effective tracker that notifies a caretaker when falling, immobility, or leaving are detected is a great alternative solution to in-patient care homes. Knowing when a fall occurs even though a patient can't remember or knowing immediately when someone wonders off property could greatly reduce the risk of permanent injury or death. This was accomplished with a multisensory approach utilizing a wearable device, Bluetooth receivers, and a central personal computer for processing and sending alerts. The wearable sensors include a photoplethysmography (PPG) heart rate monitor, a 6-axis inertial measurement unit, and a Bluetooth emitter.

CONTENTS

Figures

Table of Tables

WEARABLE MULTI-SENSOR SAFETY MONITOR

1 INTRODUCTION

Elderly people and dementia patients often face problems that impact their safety, independence, and privacy. When living at home, they need round-the-clock monitoring to ensure they don't wander off, fall, become sedentary, or become otherwise injured. Families and caregivers do their best to give full time monitoring and care; however, most caregivers have other responsibilities and cannot devote the amount of time needed to monitor the patient 24/7. Because of this, caregivers may find themselves forced by their circumstances, to move the patient into a full-time care home. Although, even in the nursing home setting, patients may face the same challenges due to the lack of second-to-second monitoring resources available. They also face a new set of problems, including loneliness and confusion at being separated from their homes and families.

One of the most pressing problems facing elderly people today is falling. According to the CDC, 14 million Americans aged 65 and older fall each year. In 2021, of those 65 and older who fell, there were 38,000 deaths related to the injuries from the fall. Emergency departments alone reported 3 million visits due to falling in older adults. The cost of treating such injuries is expected to increase to over 101 billion by 2030 [1]. Even if a fall does not cause serious injury or death, a fall for an elderly person can impact the quality of life in a very negative way. Studies have shown that a growing number of older adults fear falling and, as a result, limit their activities and social engagements. This can result in further physical decline, depression, social isolation, and feelings of helplessness [2].

It's common for a person living with dementia to wander. At any stage of the disease, they can easily become lost or confused about their location. This confusion can cause emotional distress, and the person with dementia could wander off seeking a familiar place or person. Six in 10 people living with dementia will wander at least once; however, many do so repeatedly. Although common, wandering can be dangerous — even life-threatening — and the stress of this risk weighs heavily on caregivers and family [3].

In order to address these issues, a novel Geriatric Care Assistance System with multisensory wearable device will be constructed to create a safety monitoring system for elderly and dementia patients. The sensors located on the device include a photoplethysmography (PPG) heart rate monitor, a 6-axis inertial measurement unit (IMU), and Bluetooth emitter. This device would be used in conjunction with Bluetooth receivers and a personal computer to give caretakers a person's location and notifications if the person wandered, fell, or had irregularities in activity levels. The patient would wear the device, and they would be tracked by Bluetooth emission from the device to Bluetooth receivers. Data from the PPG monitor and IMU sensor would be then sent directly from the device to a personal computer. The tracking data would be sent via WIFI from the Bluetooth receivers to the personal computer. The personal computer receiving the data would process and record the information and send alerts to caretakers if necessary. When successfully implemented, this monitoring system could increase response time, help caretakers identify issues before accidents occur, and give peace of mind for family members.

1.1 HISTORY

According to Alert-1 [4], the concept of a personal emergency response system traces back to the early 1970s, when Wilhelm Hormann invented the first system called simply PERS (personal emergency response system). Hormann's main goal was to ensure that any sick, elderly, and disabled people living alone could have a method to call for help and receive care during any situation. His invention, shown in Figure 1 below, consisted of a pendant worn around the neck. When pressed, the pendant would signal a pre-programmed message machine connected to a rotary phone. The machine would dial a predetermined number and send the message. One of the biggest limitations of this early system is that although the caregiver would know the person was in need of aid, they didn't know exactly what the problem was because of the pre-recorded message.

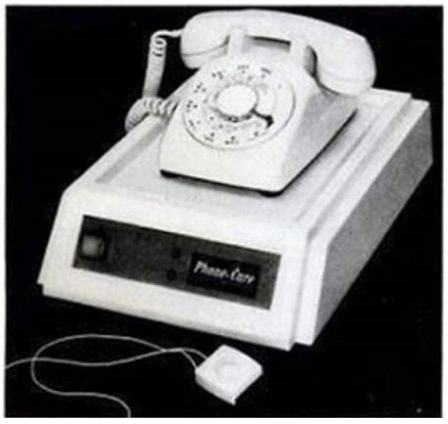
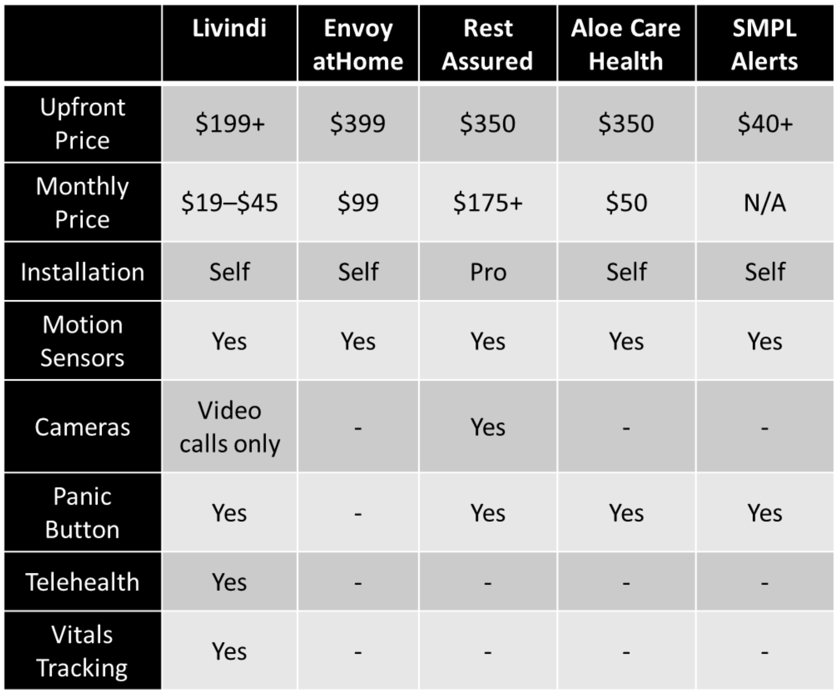


Figure 1: PERS (personal emergency response system) [4]

1.2 CURRENT SOLUTIONS/PRODUCTS

The National Council on Aging did an in-depth study on home monitoring systems in 2024. During this process they reviewed 32 medical alert systems, mystery shopped 13 medical alert systems, surveyed over 1000 medical alert system users, and reviewed independent researchers and third parties [5]. A summary of 2024's top monitoring systems for adults is shown in Table 1.

Table 1. National Council of Aging's Top 5 Medical Alert Systems



	Livindi	Envoy atHome	Rest Assured	Aloe Care Health	SMPL Alerts
Upfront Price	$199+	$399	$350	$350	$40+
Monthly Price	$19–$45	$99	$175+	$50	N/A
Installation	Self	Self	Pro	Self	Self
Motion Sensors	Yes	Yes	Yes	Yes	Yes
Cameras	Video calls only	-	Yes	-	-
Panic Button	Yes	-	Yes	Yes	Yes
Telehealth	Yes	-	-	-	-
Vitals Tracking	Yes	-	-	-	-

[5]

As shown in Table 1 above, many of the leading medical alert and monitoring systems can be expensive and require a monthly fee. Some of these companies rely on invasive camera monitoring systems to detect a patient's location down to the room level. Also, many of these companies use sensors to determine if an object moved past a certain point and not the actual room location the individual is located. Many of these resources also rely on the user to report a fall. For example, Livindi's wearable sensors include a wearable smart watch that monitors sleep, pulse, SpO2, respiration, HRV, temperature and a call button for fall detection. All of these are great, but this system does not use a wearable motion device to help determine if a fall occurred in which the person could not respond. Lastly, none of these companies use a wearable device in unison with Bluetooth location monitoring.

2 OBJECTIVE STATEMENT

We are designing a novel wearable device for an elderly patient or dementia patient that can detect the relative location of the patient with enough accuracy to discern the room the patient is in, heart rate, and event of a fall. Our intent for this device is to retain some level of independence and privacy for the individual in their daily routine while allowing staff and caretakers to react quickly to an emergency situation.

2.1 SPECIFIC AIMS

This monitoring system will utilize both an IMU sensor and pulse sensor on a wearable device for improved fall detection and decreased reaction time by alerting staff or caretake of possible fall events or heart issues through wi-fi alerts. Overall, this should increase home or nursing home safety while increasing patient independence and mobility. With this monitoring, system the collection and analysis of movement data through Bluetooth signal strength from the wearable device to separately located Bluetooth receivers can alert caretakers if the patient is sedentary for long periods of time. This monitoring can also detect and send alerts if the patient has wandered away from home. When the signal strength from their wearable device is not recorded on any of the Bluetooth receivers in the home, the patient can be assumed to have wandered out of range of the system.

2.2 REQUIREMENTS/CONSTRAINTS

There are many requirements, constraints, and concerns with any monitoring system implementation. A few identified with this system are as follows. Because this system utilizes a wearable device, the biggest hurdle for an elderly person or person with dementia will be wearing the device every day in order for monitoring to occur. Also, because falls happen in showers, a waterproof housing for the wearable device is a must have. Like any other top market monitoring system, a Wi-fi connection is required to get information to a central computer. In addition to a wearable device, the Bluetooth receiver anchors will need to be installed in every room and will need be powered by the building's power and possible back up battery during power outages. Lastly, shrinking the sensor down to fit on a wearable device without hindering the wearer will be a large design challenge as well. The following is a list of the desired requirements of the system.

1. The system shall detect falls with a false positive rate below 10% during normal daily activities.
2. The wearable device shall measure heart rate within ±20 BPM accuracy compared to a clinical-grade pulse oximeter.
3. The system shall determine the wearer's location within a room-level resolution (e.g., Bedroom, Bathroom, etc.) with at least 90% accuracy.
4. The wearable device shall transmit data (e.g., fall detection and heart rate) to a central computer or hub every 5 seconds.
5. The system shall send alerts (e.g., fall detection) to caregivers within 30 seconds of an event.
6. The wearable device shall be installable by a user or caregiver in under without special tools.
7. The wearable device shall not exceed 60 grams in weight and shall not obstruct normal arm/wrist movement during daily activities.
8. The wearable device shall include waterproof housing rated to allow safe use in showers.
9. The system shall not require any subscription service to operate or access core features.

2.3 *ETHICAL IMPACT STATEMENT*

When engineering a wearable medical alert device for elderly and dementia patients, ethical concerns are at the forefront and pivotal for the design and implementation of such a device. This paper addresses the three broad ethical concerns which must be addressed to implement hypothetical use of this device, and they are the following:

1. A failure of the device that someone is relying on could result in serious injury, prolonged response time and needless suffering, and even death.
2. The patient’s data is made available without their permission to people outside of the care staff.
3. The electronic waste generated by this product.

The biggest concern for this device is to ensure that it works and if it does not, it alerts the users of its failure. With a multisensory device there are many points for failure. If the sensors generate false positives or negatives the device is putting users at risk. One partial solution would be to have extensive testing before the design phase to get a statistical understanding of the failure rate. Also, a rigorous quality control cycle during production could help ensure the highest product integrity. Another solution would be to send updates to users and have a way to digitally check the wearables through USB C connection to a computer. Lastly, as part of a user agreement, we could require the users, staff, or possible service team to do a physical calibration or safety test check periodically to ensure the device is properly functioning. An additional possibility to limit the risk of how a failure impacts the patient, would be using the PC application to periodically make sure that the wearable devices is still communicating with the PC. If not, this could represent a failure of the device, so the care staff should be notified.

The second concern, privacy, is not only ethically concerning but also breaks HIPPA law. To address this concern, hypothetically we could encrypt the transactions between the computer and the devices as well as password protect access to the computer program’s GUI. This could satisfy HIPPA law and protect users’ information from possible data extraction and mining through wireless efforts. Keeping the data private is related to the HIPPA standards for autonomy where users can control how and why their personal health information is being used.

As more and more electronics are produced, more electronic waste is accumulating in our consumerist society. The main sources of our waste are the multiple ESP32 anchors used for Bluetooth tracking, the wearable circuitry and board, and the plastic housing for these components. To combat this waste, we could use housing for the anchors and wearables that are biodegradable and use ESP32s that are smaller and designed specifically for our product. In no way is this an exhaustive list for ethical concerns nor for solutions but a possible starting point for ways we could look closer at this issue. The device is mostly created from off the shelf parts and not custom made parts, e.g. the ESP32 dev boards. These non-custom parts could be taken from end of life systems and re-sold as budget friendly hobby hardware.

2.4 *FAILURE MODES AND EFFECT ANALYSIS*

The possible failure modes of the device present a clear risk. Any failure of a component or system could result in an inability of staff to promptly care for a patient who may have fallen.

The following failure modes are considered

Table 2 Failure Modes and Effects Analysis

	Potential Failure Mode	**Potential Failure Effects**	**SEVERITY (1-10)**	**Potential Causes**	**OCCURRENCE (1-10)**	**Current Controls**	**DETECTION (1-10)**	**RPN**
1	Battery on wearable dies	Falls not detected, location not tracked	**7**	Failure to charge battery	**10**	UI will flag wearable as non-responsive	**1**	70
2	Lack of PPG sensor Data	Heart rate tracking will not work	**3**	Improper wearing	**3**	Heart rate data in UI will show something not real (-1)	**5**	45
3	IMU failure	Falls will silently not be detected.	**10**	Excessive vibration or overstress of mems device	**1**	Motion data sent passively will cease reflecting reality	**8**	80
4	Screen Failure	User won’t be able to call for help actively or acknowledge a fall has happened	**6**	Excessive heat or cold exposure, device end of life	**3**	When a fall event occurs, it is still sent to care staff even if there is no user ack	**8**	144

The screen failure presents the highest risk priority number. Due to its relatively easy occurrence and difficult detection with a severe outcome.

3 FULL SYSTEM DESIGN

3.1 SYSTEM OVERVIEW

An overview of the system is depicted in a block diagram below in Figure 2. The system has three main components: a wearable multi-sensor device, position determining Bluetooth anchors, and a data aggregating PC. A more detailed flow chart in Figure 3 shows the components of each system and how Bluetooth signaling and Wi-fi communication interconnects all three of the main components.

The wearable health monitoring device includes a microcontroller for emitting a Bluetooth device for location detection, an inertial measurement unit (IMU) to track movement and orientation for fall detection or activity monitoring, and a photoplethysmography (PPG) sensor to measure heart rate and blood oxygen levels.

It also features Bluetooth and Wi-Fi for communication and connectivity, along with a battery charging system to sustain its operation. The positional determination anchors use Bluetooth to track the user's location within a defined area and transmit this data via Wi-Fi. This ensures that caregivers can monitor the whereabouts of the elderly individual in real time.

Finally, the data aggregator PC application processes and visualizes collected data. It provides functionalities such as text or email notifications for alerts, reports, or advanced data processing. As seen in Figure 2, Wi-fi connectivity allows it to communicate effectively with the wearable device and positional anchors. Together, these components create an integrated system that enhances health monitoring and location tracking for elderly care.

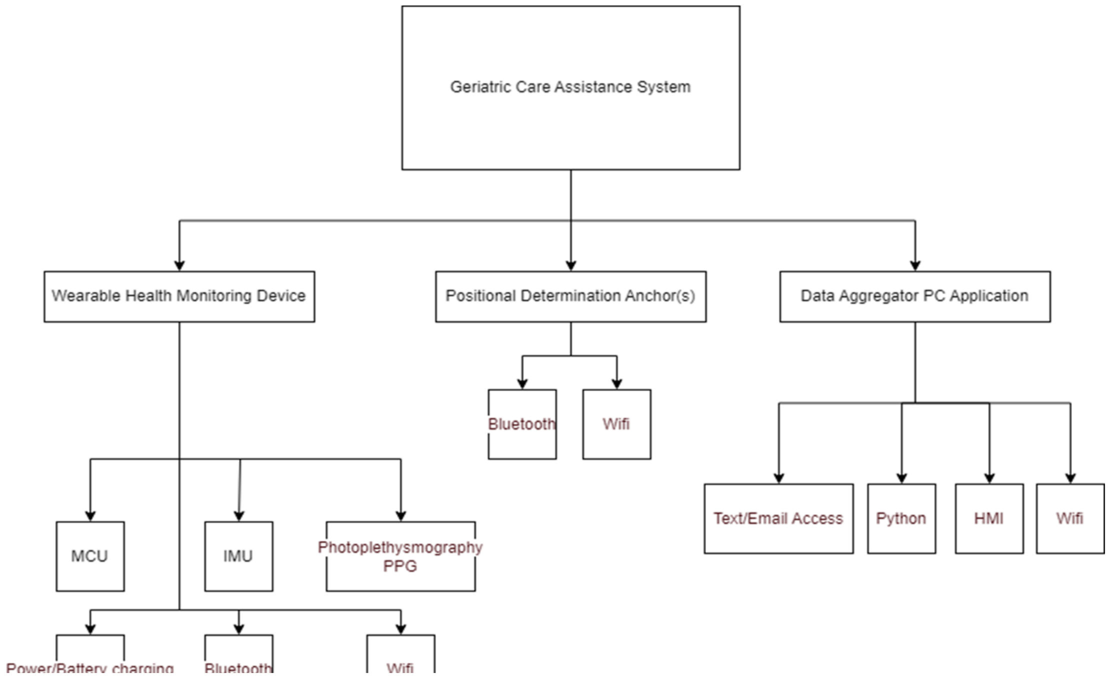

Geriatric Care Assistance System
Wearable Health Monitoring Device
Positional Determination Anchor(s)
Data Aggregator PC Application
Bluetooth
Wifi
Text/Email Access
Python
HMI
Wifi
MCU
IMU
Photoplethysmography PPG
Power/Battery charging
Bluetooth
Wifi

Figure 2: Block Diagram

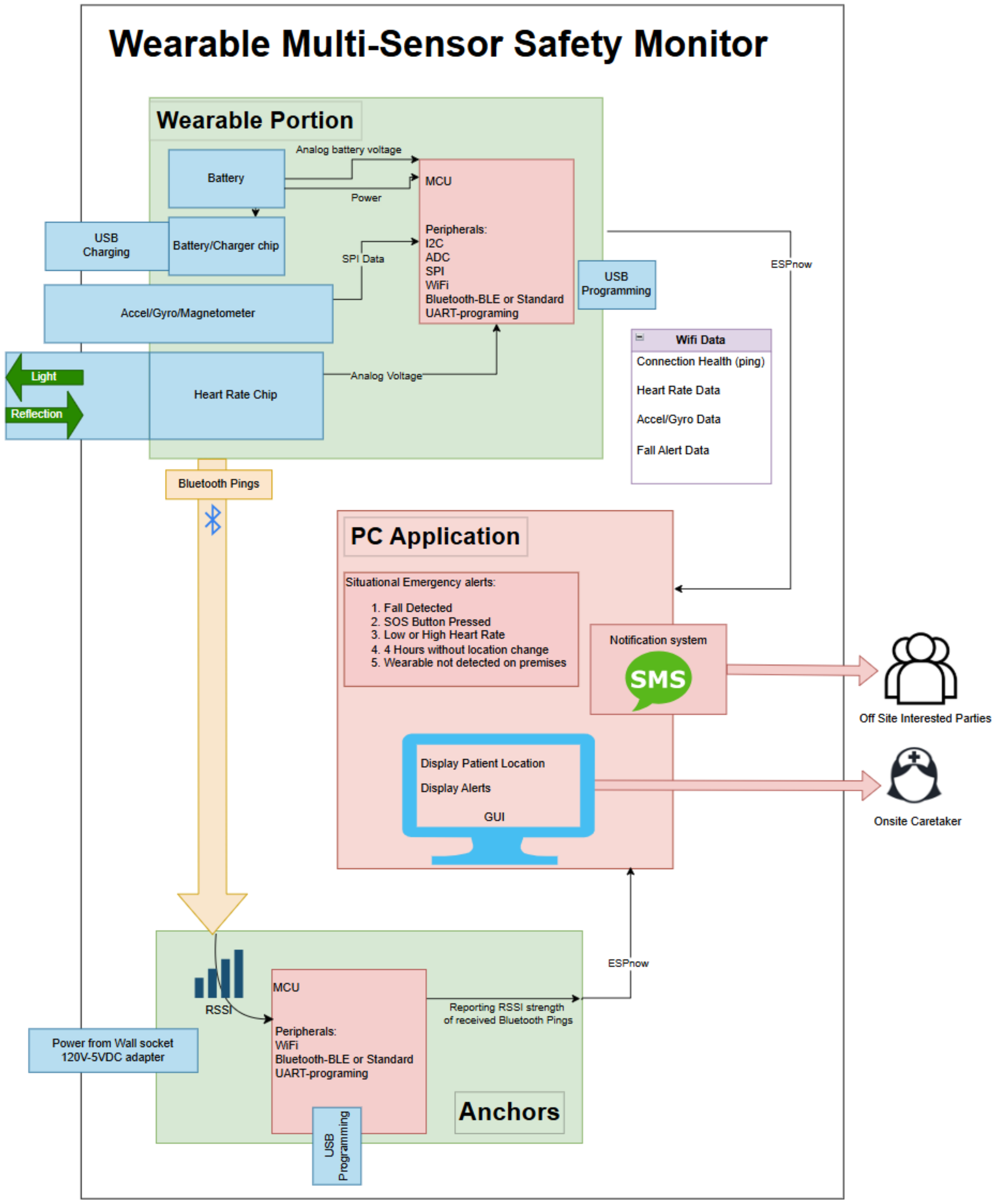


Figure 3. Flow Diagram the System

3.2 Wi-Fi implementation (ESP-NOW)

In this system, inter-component communication is achieved through the use of a vendor specific communication protocol known as ESP-NOW. ESP-NOW is an implementation of a specific kind of transaction which is included in the IEEE802.11 standard for wireless networks (Wi-Fi). The type of 802.11 transaction used is an "action frame". Action frames are used for device-to-device communications on a wireless network. In general, they are used for things like spectrum management or "quality of service" (QOS) announcements. In the case of ESP-NOW a "vendor specific" action frame is used to encapsulate the data [6]. In the case of this system, using ESP-NOW imposes some constraints and helps achieve some goals. The constraints ESP-NOW imposes are limits to the size of each transferred telegram which is limited to 250 bytes. We responded to this constraint by performing all heart rate calculations and fall detection in the wearable portion of the design. Another constraint is that ESP-NOW requires a MAC address of the target device to which the data is sent. We responded in this system by "hard coding" the MAC addresses of each of the devices used into the firmware of the other devices. The "hard coding" issue is addressed in the future work section of this paper. The way ESP-NOW helps achieve goals comes down to mainly how the firmware can specifically manage the number of wireless transmissions happening on the antenna of the wearable device. In a traditional Wi-Fi connection system, there are more transmissions needed for synchronization, sync-acknowledgement and other general connection maintenance. ESP-NOW does not do any additional connection maintenance. When an ESP-NOW transmission is sent, there is one transmission, and one reception for an acknowledgement. This allows the firmware to turn the antenna on less often than would be needed for traditional Wi-Fi which in turn saves energy and increases battery life.

3.3 Wearable Portion

The wearable device houses a battery for powering the MCU with IMU and PPG sensors and Bluetooth and Wi-Fi transmission capabilities. The wearable device will transmit Bluetooth signals to the stationed Bluetooth receivers. It also transmits IMU, PPG sensor data, and user alert data via Wi-Fi to the PC.

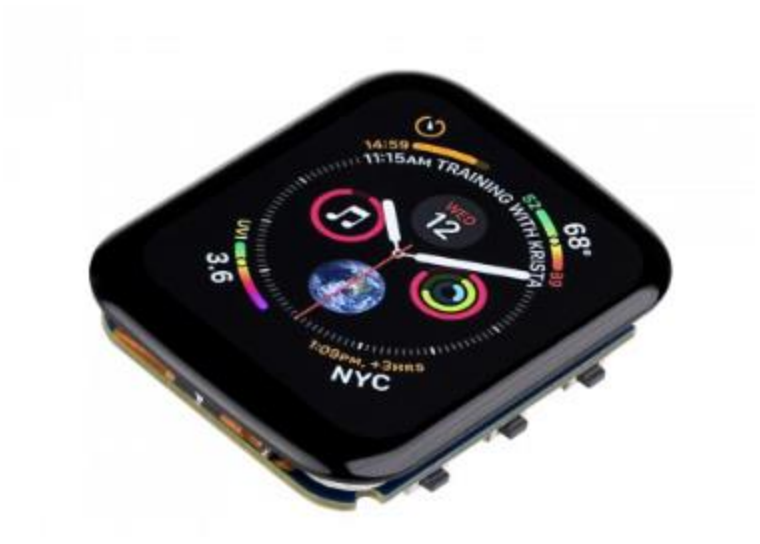


Figure 4. Waveshare ESP32-S3-Touch-LCD-1.69

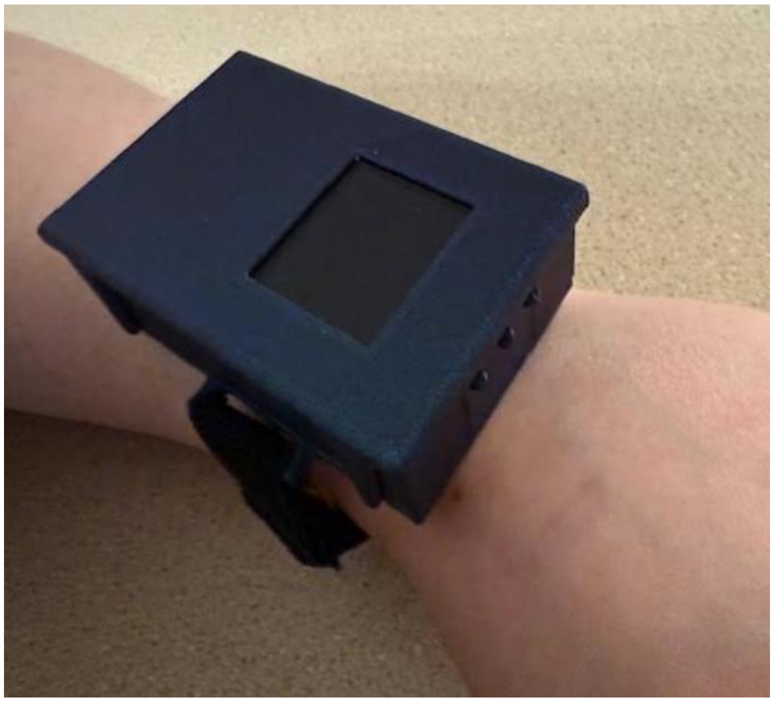


Figure 5. Wearable in Housing

The design integrates the chosen PPG sensor, the battery and the MCU package into a cohesive group that will allow a strap to be attached. The full wearable system weighs 130g. This is further discussed in the conclusions section.

3.3.1 FreeRTOS

The wearable system uses an operating system to support the design goals and functionality of the system. FreeRTOS stands for “Free Real Time Operating System”. The core of the use of FreeRTOS is creating the ability to break the program into smaller parts which can have varying degrees of interconnectedness. These smaller parts (called tasks) can also achieve concurrency where they all appear to be executing at the same time but in reality, the scheduler of FreeRTOS is switching execution between each very quickly.

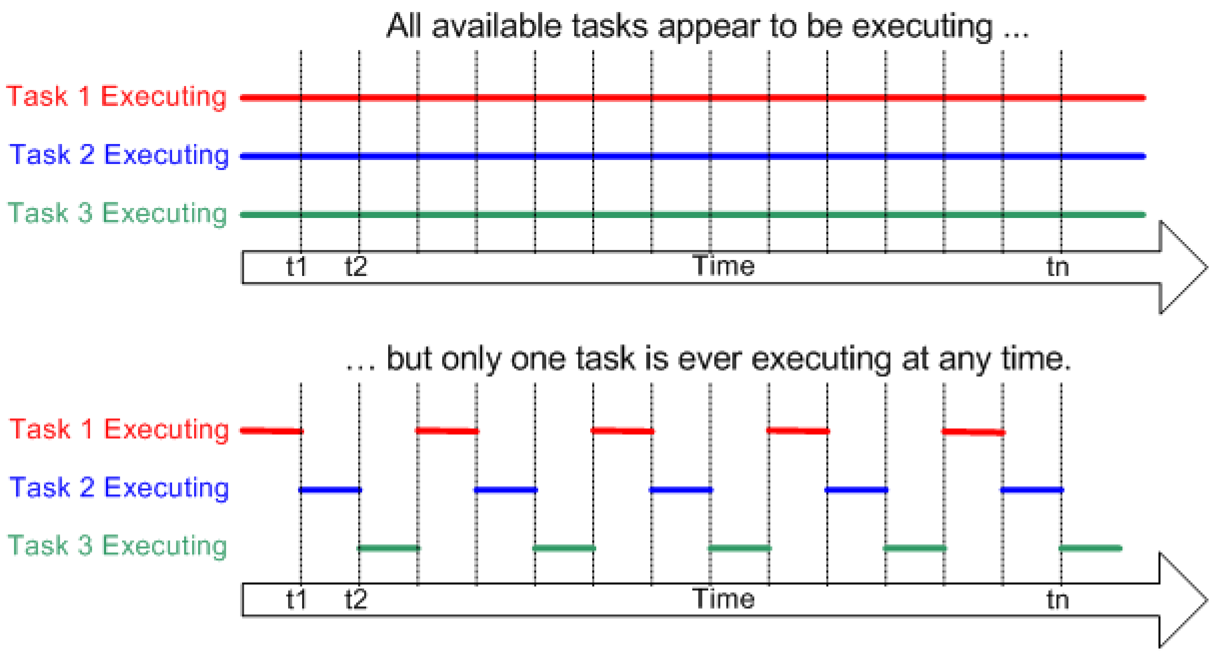


Figure 6. FreeRTOS Task Switching [7]

Each task has a priority, and its execution can be paused and yielded to the scheduler for a variety of reasons such as waiting for an interrupt, waiting for data in a queue, or simply pause for a certain amount of time. The specific implementation of FreeRTOS for ESP32 chips also accounts for the fact there are two processor cores on this chip and tasks may be distributed between the two of them. The constraints this imposes is now each task must be sure to execute quickly so that there is ample time for the other tasks to execute as well as the fact that the FreeRTOS construct itself takes up memory and time. The wearable system’s time and stack usage are shown below

Table 3 Wearable Device time and Stack Usage

Task Name	CPU Usage (%)	Stack Usage (bytes)
TaskMonitor	0.11%	1648
IDLE1	98.88%	356
IDLE0	83.39%	240
BLE Task	0.02%	1580
tiT	0.03%	3420
esp-now Task	0.02%	1492
UI Task	14.50%	5364
sys_evt	0.00%	3256
ipc1	0.00%	152
arduino_events	0.00%	2896
Tmr Svc	0.00%	2424
hciT	0.07%	1288
BTU_TASK	0.04%	6828
BTC_TASK	0.02%	6740
PPG FIFO TASK	0.12%	2512
Heart Rate Task	0.04%	11760
Fall Detection	0.12%	3888
QMI FIFO TASK	0.83%	2128
wifi	0.44%	4996
btController	0.12%	2508
esp_timer	1.24%	7504
ipc0	0.00%	152
toneTask	0.00%	2416

The tasks use about 9% of total time. This 9% includes the fact that there are multiple processors. The stack usage adds up to 75.5KB. The Processor has 512KB of internal SRAM. We make limited use of global variables in the program so about 14% of RAM is used. The dual core processors run at 240MHz

3.3.2 PPG sensor

The chosen PPG (photoplethysmography) is the MAX30101. This sensor was chosen over the MAX30102 sensor for the green led light which it integrates. The green LED light has higher energy than the red and infrared.

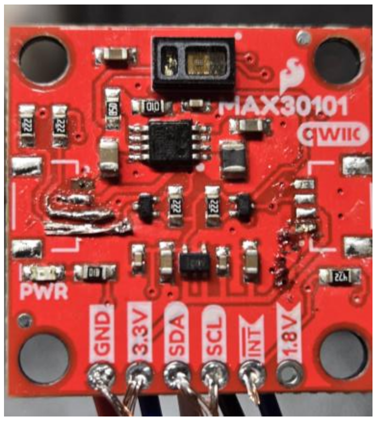


Figure 7. MAX30101 Sensor on Sparkfun development board with QWIIC connectors removed.

The purchased MAX30101 development board from Sparkfun required that the two large QWIIC connectors be removed so that the sensor itself could be placed close to the skin without interference. The I2C connection is achieved through the hole connections on the MAX30101 development board and the test points on the ESP32-S3 development board.

The PPG sensor code is implemented in a few steps; data acquisition, filtering, and finally analysis. The MAX30101 sensor is set up to sample at 200Hz, with hardware averaging of 4. This means the data is practically sampled at 50Hz. The hardware FIFO and “buffer almost full” interrupt are used to ensure no data is lost. The next filtering step is a 128 tap FIR filter implemented using the ESP32S3 SIMD instructions. The bandpass filter design (0.8Hz-5Hz) was done in MATLAB and is visited in the appendix. The last step before analysis is Z-score normalization to make the peaks more detectable. After the peaks in the signal are accentuated then the data can be evaluated to find the height of the peaks. The evaluation after the bandpass filter is completed once every 10 seconds. The code from Sparkfun for their MAX30101 development board required modifications to support the use of the interrupt and FIFO buffer.

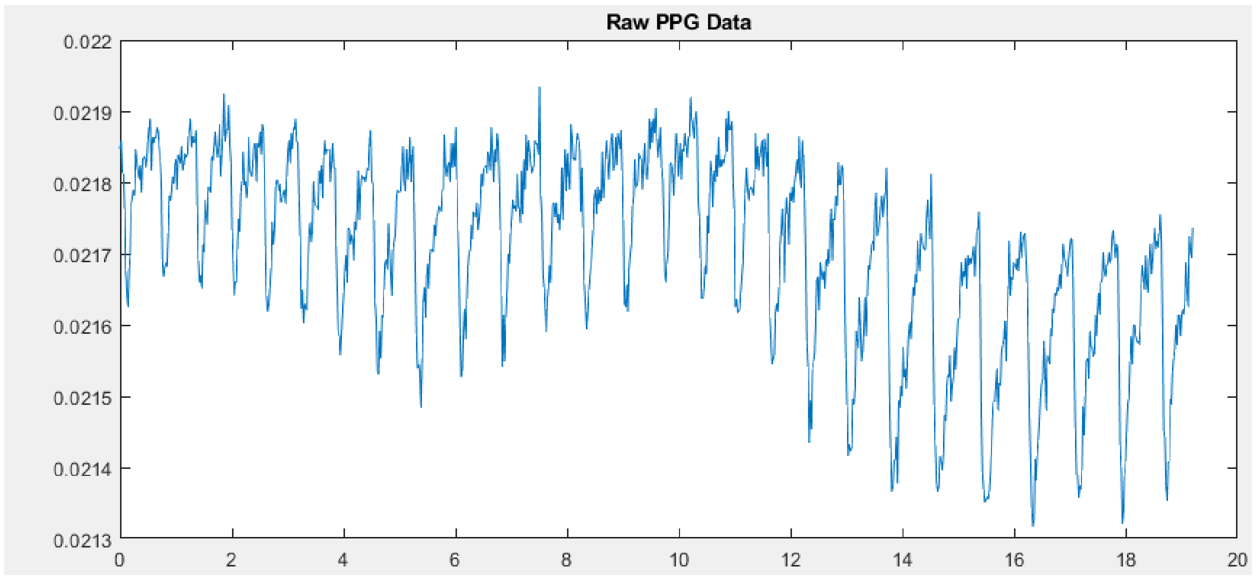


Figure 8. Raw PPG Data Pre-Filtering

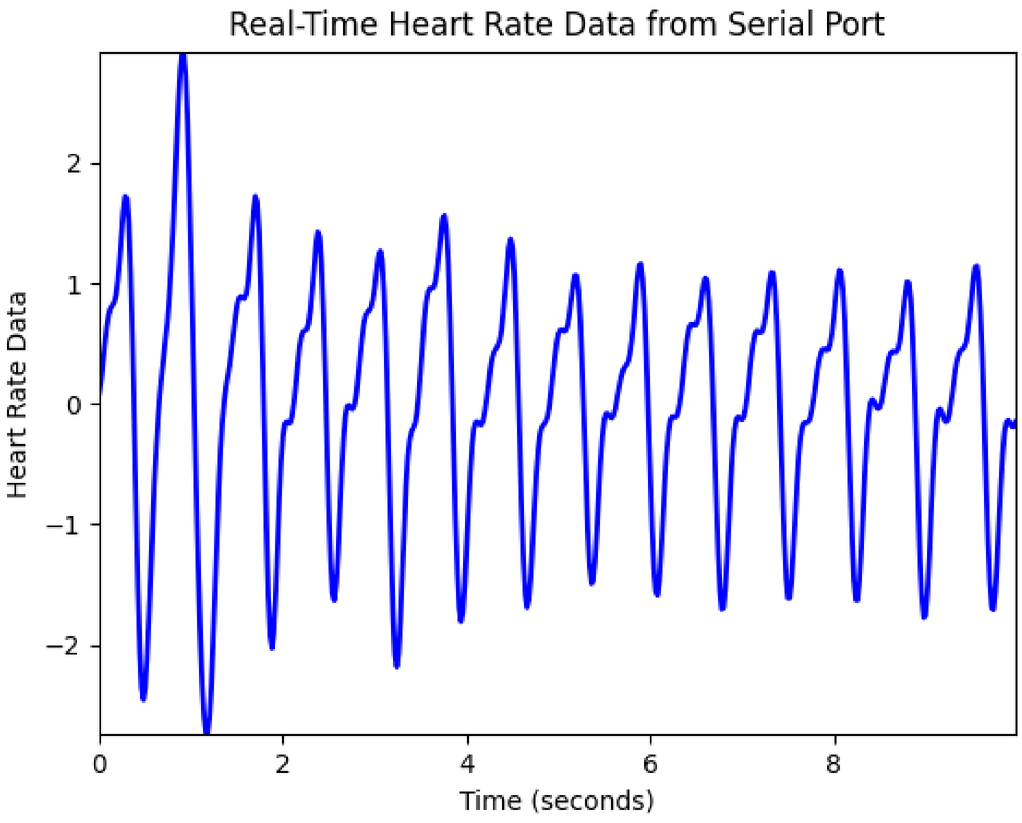


Figure 9. PPG Data Post Filtering

3.3.3 *Fall Detection Algorithm*

The fall detection algorithm was developed using the WEDA-FALL dataset from CMU Portugal [8]. This dataset was used to optimize an algorithm in python, which was translated to C to work in real time. The algorithm works in a few steps. First the 50Hz IMU sensor data is input into the Madgwick filter. The Madgwick filter is a filter which can estimate the movement of the sensor in the earth frame. The earth accelerations are clumped into 1.2 second chunks and features of these chunks are identified. These features include Average Acceleration, Average Jerk, and Total rotation. Average jerk is the average change in earth acceleration throughout the chunk. The rotation feature is developed from the IMU data directly, short circuiting the Madgwick filter. The rotation feature is meant to denote the total amount of rotation the sensor underwent during the chunk. The algorithm then looks back at previous chunks and identifies fall events based on limits and peaks of these features. The limits selected were selected through a random search differential evolution. Each iteration of the evolution is tested against the entire WEDA-Fall dataset with the following objective function:

```
score = sensitivity + 0.9 * specificity
```

This objective function selects a lighter weight for the specificity of the algorithm. In other words, it is more important for the algorithm to detect all falls than it is for the algorithm to screen motions which are not falls.

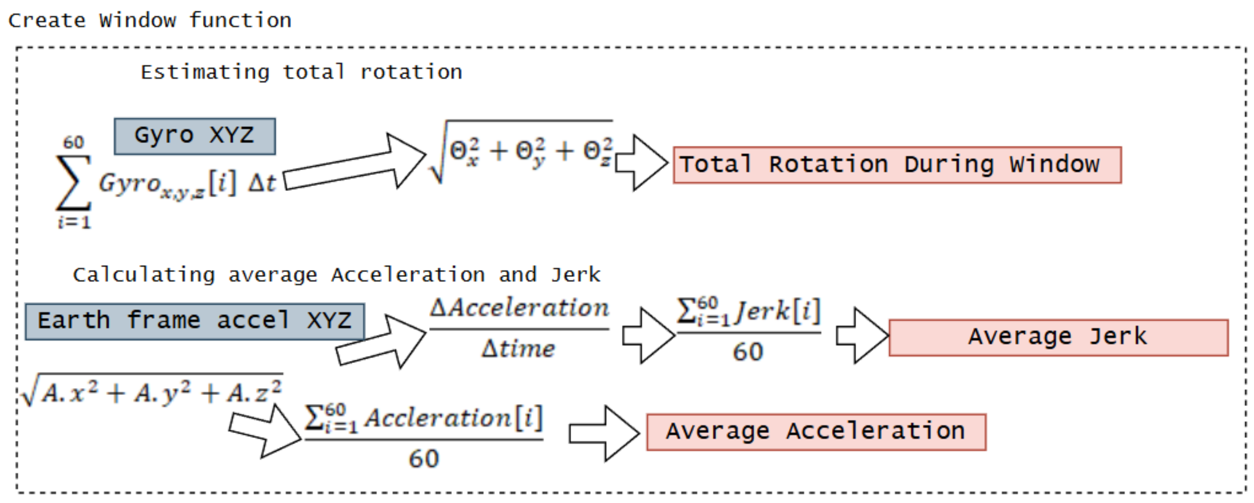
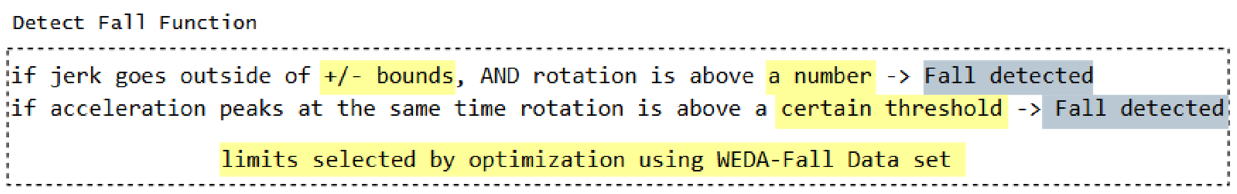

Create Window function
Estimating total rotation
Gyro XYZ
$\sum_{i=1}^{60} Gyro_{x,y,z}[i]\ \Delta t$
$\sqrt{\theta_x^2+\theta_y^2+\theta_z^2}$
Total Rotation During Window
Calculating average Acceleration and Jerk
Earth frame accel XYZ
$\frac{\Delta Acceleration}{\Delta time}$
$\frac{\sum_{i=1}^{60} Jerk[i]}{60}$
Average Jerk
$\sqrt{A.x^2+A.y^2+A.z^2}$
$\frac{\sum_{i=1}^{60} Accleration[i]}{60}$
Average Acceleration

Figure 10. Fall Detection Data Chunking Diagram



```
Detect Fall Function
if jerk goes outside of +/- bounds, AND rotation is above a number -> Fall detected
if acceleration peaks at the same time rotation is above a certain threshold -> Fall detected

                 limits selected by optimization using WEDA-Fall Data set
```

Figure 11 Fall detection function description

The function for fall detection in shown above. The function operates on the chunked data. The highlighted items were optimized using scipy differential evolution.

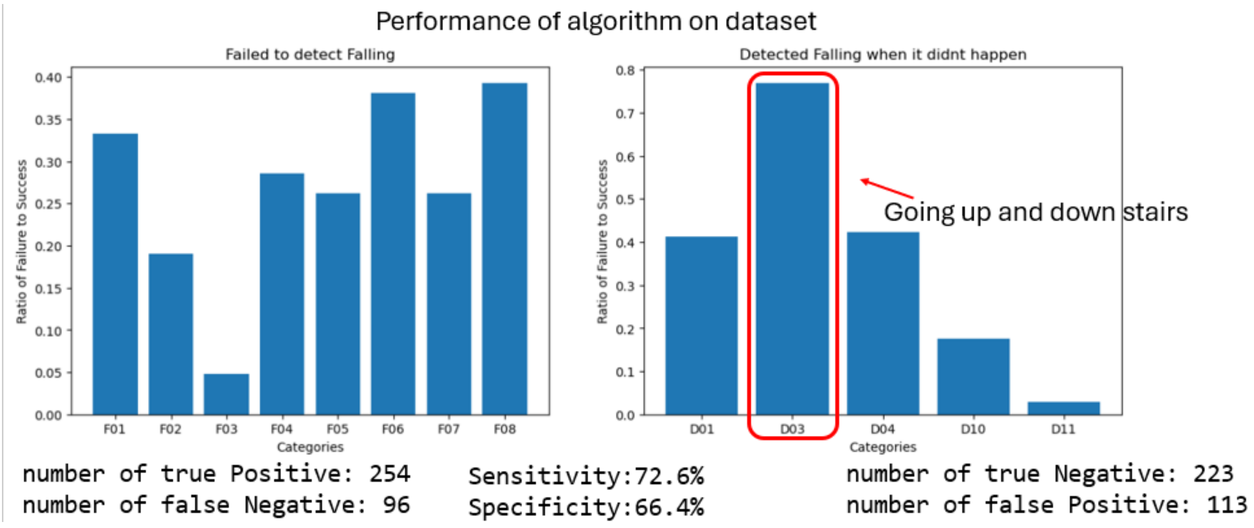
The following types of activities are included in the dataset. Several of the non-fall activities are excluded from the optimization due to their highly active nature.

Table 4. Activity Index

index	Activity
F01	Fall forward while walking caused by a slip
F02	Lateral fall while walking caused by a slip
F03	Fall backward while walking caused by a slip
F04	Fall forward while walking caused by a trip
F05	Fall backward when trying to sit down
F06	Fall forward while sitting, caused by fainting or falling asleep
F07	Fall backward while sitting, caused by fainting or falling asleep
F08	Lateral fall while sitting, caused by fainting or falling asleep
D01	Walking
D02	Jogging
D03	Walking up and downstairs
D04	Sitting on a chair, wait a moment, and get up
D05	Sitting a moment, attempt to get up and collapse into a chair
D06	Crouching (bending at the knees), tie shoes, and get up

D07	Stumble while walking
D08	Gently jump without falling (trying to reach high object)
D09	Hit table with hand
D10	Clapping Hands
D11	Opening and closing door

The excluded activities are D02, D05, D07, D08, D09.



```
number of true Positive: 254      Sensitivity:72.6%      number of true Negative: 223
number of false Negative: 96      Specificity:66.4%      number of false Positive: 113
```

Figure 12. Fall Detection Algorithm Performance

The algorithm optimization resulted in a sensitivity of 72.6% and a Specificity of 66.4%. During testing of this algorithm on people it was found that the algorithm would detect hard falls 2 out of 3 times and would very often falsely report falls when they did not happen.

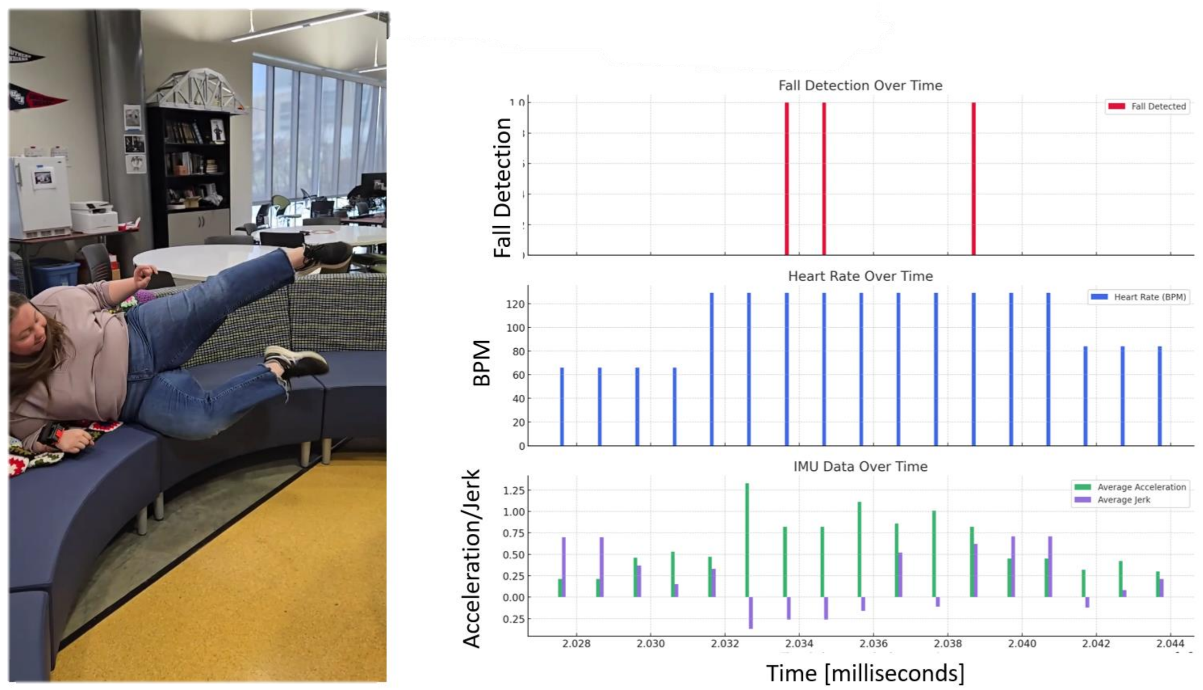


Figure 13 Fall detection testing

In the picture in the figure above a team member is falling, the data on the right shows the results of the fall detection algorithm at the top, their heart rate, and the aggregated motion data at the bottom.

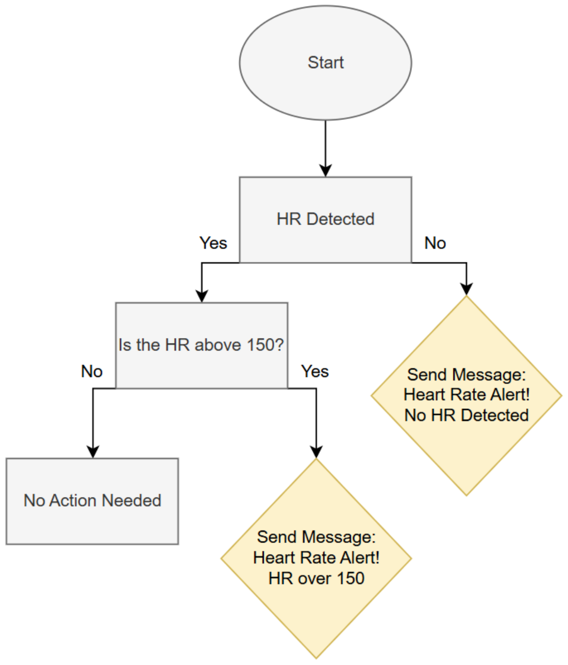
3.4 PC Application – Message Processing Center

The PC application aggregates the data obtained by both the wearable portion and the stationary portion by utilizing a python script specifically tailored to analyze the data and send alerts to caregivers. The information obtained by the wearable and anchors is sent via ESPnow to the hub, which is attached to the computer through the COM port. The Hub then consolidates the data from both and transmits it as a singular unit to the PC through serial communication. Once received that data is then communicated with the Python program using the PySerial library, which was chosen for efficiency with transfers and to aid with processing the data in real time. The Python program serves as the center for information gathering and monitoring in the system. It continuously reads the real-time data from the hub, processes the incoming sensor and RSSI data line-by-line, logs the information for later analysis in CSV files, and sends alerts and updates to caregivers.

Once the data is received by the python script, it is parsed using regular expressions and processed to update two CSV files that function as logs for the data. The first log (wearable_log.csv) stores the general data pertaining to the location (anchor and RSSI data), whether a fall was detected, if the SOS signal was used, the heart rate data, and the room location determined by the threshold logic in the program. The second (wearable_sensors.csv) stores the battery status of the wearable and all of the sensor readings used to determine whether to flag a fall. A complete breakdown of the CSV columns is included in the code in the attached appendix. Both logs contain a timestamp that is determined by the datetime module in Python. This module utilizes the datetime.now() function which uses the system time on your computer to record the current date in time in the files.

The system performs several key functions with the obtained data: it monitors for fall detection and SOS signals, abnormal heartrates, long periods with no movement, room location within the building, and whether the person is detectable within the building. It displays any updates or alerts through a graphical user interface, and It sends any warnings or alerts to the caregiver via text message. The decision trees below visually show the logic used to determine when to send alerts to the caregiver.

The first decision tree shows the heart rate logic. The heart rate information is used to determine if the user is wearing the wearable portion or if they may have taken it off. Thus, the caretaker is notified if the heart rate stops being detected. It also notifies the caretaker if there is a unusually high heart rate detected (over 150 bpm), as this may indicate that the user is in distress and needs aid.



Start
HR Detected
Yes
No
Is the HR above 150?
No
Yes
Send Message: Heart Rate Alert! No HR Detected
No Action Needed
Send Message: Heart Rate Alert! HR over 150

Figure 14: Decision Tree for Heart Rate Logic

The next decision tree shown details the immobility alert logic. This is meant to alert the caretaker if the person hasn't been detected moving between rooms for longer than a four hour period. After the alert is sent, the room change timer is reset in the code to avoid any spam massages. There is also a quiet hours functionality. When engaged, it allows the caretaker to silence all immobility alerts in order to account for sleeping overnight or longer rest periods throughout the day. The caretaker can set automatic quiet hours to engage over night; however, they can also be toggled on or off at any time using a button toggle on the interactive GUI.

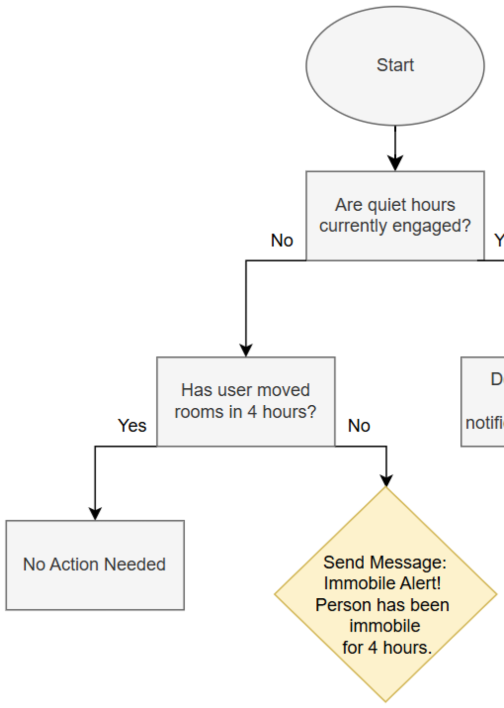

Start
Are quiet hours currently engaged?
No
Yes
Has user moved rooms in 4 hours?
Don't send any immobility notifications or alerts.
Yes
No
No Action Needed
Send Message: Immobile Alert! Person has been immobile for 4 hours.

Figure 15: Decision Tree for Immobile Detection Logic

The final decision tree shows the logic for the fall alert. It starts with wearable flagging a potential fall. If the user also presses the button to confirm the fall, the system will immediately send a fall alert to the caretaker. If the user hits the button to indicate they did not fall, no alert is sent. Otherwise, if the user fails to respond, the program monitors their location since the fall. If they move between rooms, the fall is considered a false alert and discarded. After 10 minutes if there is no movement detected between rooms, a message is sent infroming the caretaker that a fall was detected, not confirmed by the user, but that they haven't changed rooms in 10 minutes. The user continues to be monitored, and if they haven't moved between rooms after an aditional 30 minutes the caretaker is alerted that a fall was detected and the user has been immobile for 40 minutes. This is to help stop false alerts to the caretaker, while also takeing the users well being in mind.

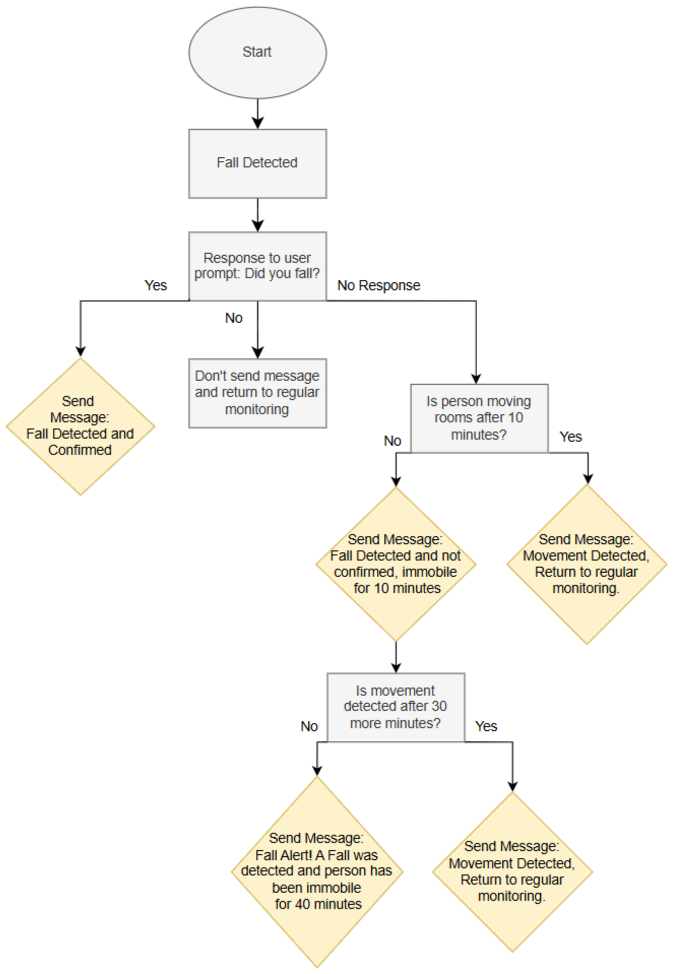
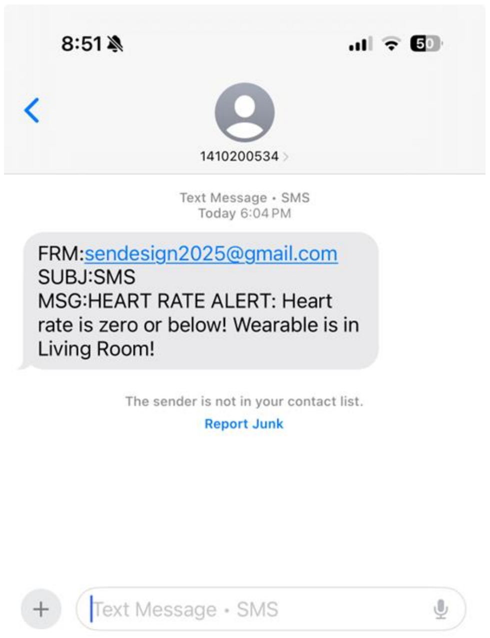

Start
Fall Detected
Response to user prompt: Did you fall?
Yes
No
No Response
Send Message: Fall Detected and Confirmed
Don't send message and return to regular monitoring
Is person moving rooms after 10 minutes?
No
Yes
Send Message: Fall Detected and not confirmed, immobile for 10 minutes
Send Message: Movement Detected, Return to regular monitoring.
Is movement detected after 30 more minutes?
No
Yes
Send Message: Fall Alert! A Fall was detected and person has been immobile for 40 minutes
Send Message: Movement Detected, Return to regular monitoring.

Figure 16: Decision Tree for Fall Detection Logic

The program also notifies the caregiver if the wearable is no longer detected within the premises. This could be the case if the person wearing the wearable has left the premises or if the battery on the wearable has died. When displaying the alert for the wearable no longer being detected, the last known room is also displayed. This is so the caretaker can see the last room the wearable was detected in, and if that room was an interior room it is more likely that the wearable battery has died.

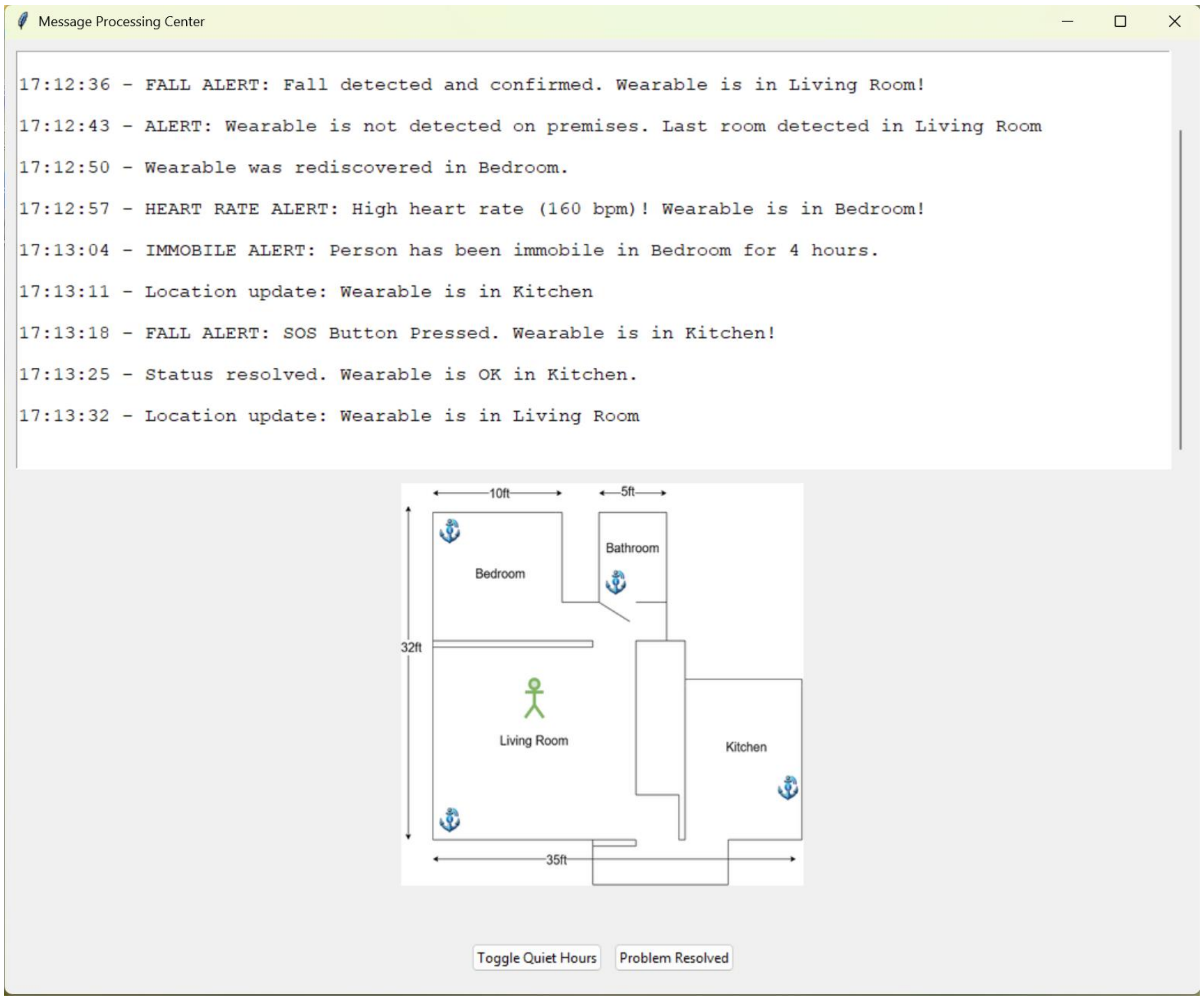
The logic detailed above was integrated into a user-friendly graphical interface called the Message Processing Center, developed using Python's Tkinter library. In Figure 19 below, the GUI is shown. It features a scrolling log of system messages with timestamps and a dynamic floor plan image that changes depending on the user's room and condition. The messages displayed include location updates, warnings, and alerts. The location updates are displayed each time the user changes rooms. The alerts and warnings are displayed depending upon the logic described above. When an alert or warning is displayed, a windows alert sound is played on the PC. There is also a text message sent to a caretaker phone number entered in the code. This allows the caretaker to still get alert and warning messages when they’re away from their computer.



FRM:sendesign2025@gmail.com
SUBJ:SMS
MSG:HEART RATE ALERT: Heart rate is zero or below! Wearable is in Living Room!

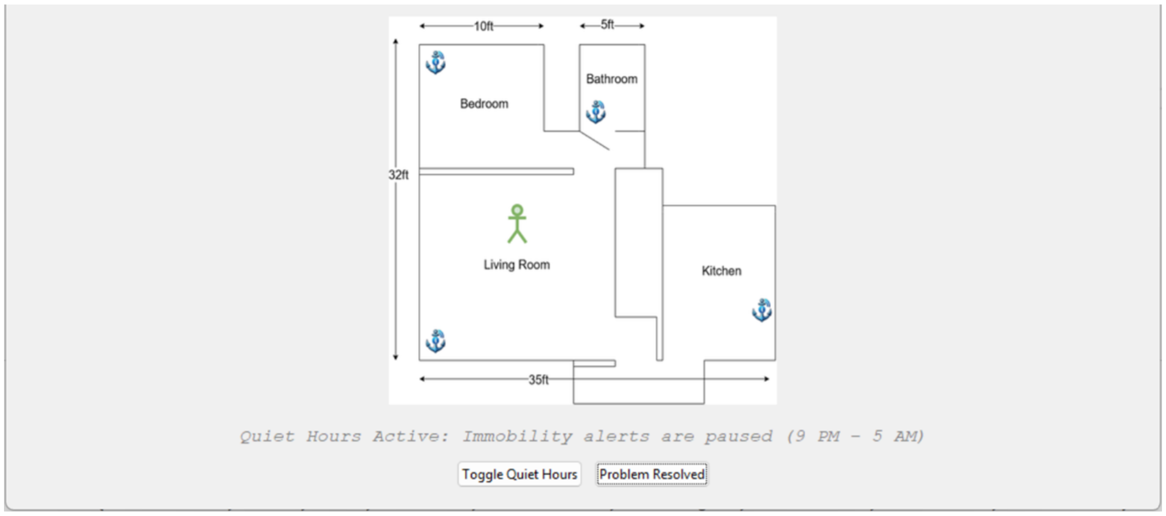
Figure 17: Text Message Alert

Each time the person moves between rooms, the green person on the image changes rooms to match the real time movements. If there is an alert or warning sent, the green person turns red, indicating the user is in distress. Below the floor plan image there is a button to toggle the quiet hours on or off. Next to the quiet hours button, there is a problem resolved button. This button resets all the alert related variables to their ok setting. It also resets the floorplan image from its red person state to the green person state. It is intended to be pressed after the wearer has been checked on and any potential problems have been resolved.



```
17:12:36 - FALL ALERT: Fall detected and confirmed. Wearable is in Living Room!
17:12:43 - ALERT: Wearable is not detected on premises. Last room detected in Living Room
17:12:50 - Wearable was rediscovered in Bedroom.
17:12:57 - HEART RATE ALERT: High heart rate (160 bpm)! Wearable is in Bedroom!
17:13:04 - IMMOBILE ALERT: Person has been immobile in Bedroom for 4 hours.
17:13:11 - Location update: Wearable is in Kitchen
17:13:18 - FALL ALERT: SOS Button Pressed. Wearable is in Kitchen!
17:13:25 - Status resolved. Wearable is OK in Kitchen.
17:13:32 - Location update: Wearable is in Living Room
```

Figure 18: Complete GUI Screen with Messages Displayed



```
Quiet Hours Active: Immobility alerts are paused (9 PM - 5 AM)
```

Figure 19: Bottom of GUI Screen with Quiet Hours Active and Displayed

3.5 Stationary Portion

Bluetooth Low Energy (BLE) is a widely adopted wireless communication protocol designed for low-power data exchange over short distances. It is particularly well-suited for Internet of Things (IoT) and wearable applications due to its energy efficiency and compatibility with smartphones and embedded devices. In BLE-based positioning systems, location is often inferred using the Received Signal Strength Indicator (RSSI), a measure of the signal power received by a device. Because signal strength typically decreases with distance, RSSI values can be used to estimate proximity to a BLE transmitter. While RSSI is inherently noisy and subject to environmental interference, it remains a popular method for coarse localization due to its simplicity and minimal hardware requirements [9], [10].

In this project, multiple Bluetooth anchor nodes were installed in each room (one per room) of the home or monitored environment. Each anchor is equipped with a Bluetooth receiver and an ESP32 microcontroller capable of communicating over ESP-NOW. These devices are powered via standard wall outlets, ensuring continuous operation without the need for battery maintenance. The primary role of each anchor is to detect the RSSI of Bluetooth advertisements transmitted by the wearable device. These RSSI values are then transmitted via ESP-NOW to a central hub ESP32 microcontroller with a serial connection to a PC for processing. By comparing signal strength across multiple anchors, the system can estimate the wearer's room-level location in real time using signal strength differentials between fixed spatial points.

The three MCU models considered for anchor Bluetooth receivers are displayed in Table 3. These ESP32s were considered because of their ability to receive and measure Bluetooth Low Energy signals and to transfer data through ESP NOW. These three were chosen to be considered due to availability by quick purchasing for initial testing. Or in the case of the Seeed Studio model, had a special design that might be incorporated.

Although the Seeed Studio XIAO ESP32C3 was initially a strong contender for both the wearable and anchor components due to its small size and the convenience of using the same microcontroller across the system, the design evolved significantly as development progressed. As requirements grew to include integrated components such as an IMU and LCD screen for the wearable, the Seeed board was phased out due to its limited I/O and lack of onboard features. It also became less favorable as a Bluetooth anchor because it lacked a built-in antenna, which is critical for consistent and accurate RSSI measurements. The ESP-WROOM-32 was ultimately selected as the Bluetooth anchor due to a combination of practical benefits: robust library and community support, excellent availability, affordability, and solid wireless performance. Its dual-core processor, broader GPIO options, and support for Bluetooth Classic and BLE made it more versatile for the demands of the system. Although alternatives like the ESP32-C3-DevKitM-1 offered slightly better power efficiency and security features, the WROOM-32 was chosen for its overall balance of features, proven reliability, and compatibility with the development environment making it the optimal choice for the final anchor implementation, as shown in Figure 11.

In the final implementation of the Bluetooth anchor system, the ESP32-WROOM-32U microcontroller paired with a 2.4GHz 6dBi indoor omni-directional Wi-Fi antenna was selected for each anchor node. This choice was made over the standard ESP32-WROOM-32 module without external antenna support to improve the reliability and range of Bluetooth signal detection. The external antenna significantly enhances both signal strength and reception sensitivity, which is critical for consistent RSSI measurements. During initial testing with modules using only internal PCB antennas, signal fluctuations and short-range inconsistencies were observed, leading to lower location detection accuracy. Switching to the ESP32-WROOM-32U with an external antenna mitigated these issues by providing a more stable signal across larger room spaces and through partial obstructions, which ultimately improved the robustness of the room-level localization system.

Table 5. Comparison of ESP32 Models

Board name	Alinan ESP-WROOM-32U with 2.4GHz 6dBi Indoor Omni-Directional WiFi Antenna	Seeed Studio XIAO ESP32C3	Espressif Systems ESP32-C3-DevKitM-1
Rank	1	3	2
Price	$5.33	$4.99	$8.00
Antenna included	No, 2.4GHz 6dBi Indoor Omni-Directional WiFi Antenna avialable	No, I-PEX Antenna connector available	Yes
Size	48 mm x 28 mm	21 x 17.8mm	50mm x 25.4
Processor	2 cores	1 core	1 core
Memory	SRAM: 520 KB FLASH: 4 MB	SRAM: 400 KB FLASH: 4 MB	SRAM 400KB FLASH: 4 MB

	ROM:448 KB	ROM: 384 KB	ROM: 384 KB
Power Efficiency	Medium	Very High	High
GPIO Connections	34	11	22
Bluetooth	Bluetooth Classic & LE	Bluetooth LE (5.0)	Bluetooth LE (5.0)
Privacy Safety Feature	Hardware Cryptography: AES, SHA, RSA, ECC Digital Signature: Yes	Hardware Cryptography: AES, SHA, RSA, ECC Digital Signature: Yes	Hardware Cryptography: AES, SHA Digital Signature: No

[11] [12] [13]

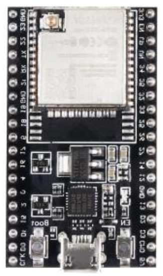


Figure 20. ESP32-WROOM-32U Board for Anchor and Hub Components. [14]

The Bluetooth anchor receiver system is illustrated in Figure 12. This system monitors which room a user is in by analyzing the signal strength (RSSI) received by the room anchors from the wearable's ESP32-S3 which continuously sends out a ping signal. Each anchor receives the same ping signal and determines the RSSI signal strength. This information is used to estimate the user's proximity to the anchors.

The RSSI values are inversely related to the distance between anchor and wearable. Additionally, obstructions such as walls decrease signal strength read by the anchors. Thus, ideally the anchor in the same room as the wearable would read the highest RSSI value out of all of the home's anchors due to it having a relatively small distance and no walls obstructing the signal. Hence, stronger RSSI values indicate closer proximity, allowing the system to determine the user's location based on the anchor with the strongest signal. For example, in the diagram, the Dining Room anchor has the strongest signal at -50dB, suggesting that the user is in the dining room. This logic is implemented in the anchor and hub microcontroller code, which is provided in Appendix D [15] [16].

This approach is effective for room-level localization, providing a practical solution for monitoring user activity in indoor spaces. By comparing RSSI values from multiple anchors, the system can infer the user’s location even in environments with walls. Proper installation would be required to ensure avoiding obstacles that would cause signal distortion. The benefits of this design is its simplicity, efficiency and ease of implementation.

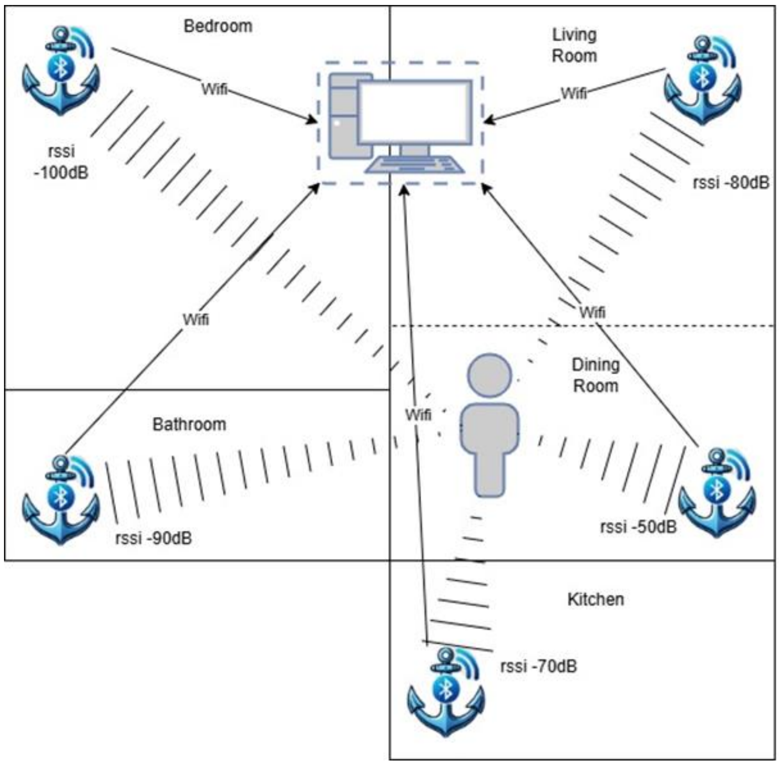


Figure 21. Bluetooth Anchor Diagram Showing How Anchors Would Interact.

To evaluate the reliability of Bluetooth signal strength (RSSI) as a basis for room-level localization, a data collection test was conducted. In this test, the wearable device remained stationary in one room for ten minutes while four ESP32-WROOM anchor devices, each located in a different room, measured RSSI values from the wearable. This process was repeated for the Bathroom, Bedroom, Kitchen, and Living Room areas. During testing, the wearable was placed at a seated position within each room, ensuring a direct line-of-sight to all anchors. This setup minimized major obstructions and allowed the RSSI measurements to reflect the best-case performance of the system under controlled, stationary conditions.

The collected RSSI data were averaged and analyzed to examine their stability and relative strength across locations. The results of this analysis with the standard onboard PCB antenna are summarized in Table 4, Table 5, Table 6 and Table 7. The data presented in Tables 4 through 7 reveal consistent trends that support the feasibility of using RSSI values for room-level localization.

In each test, the anchor located in the same room as the wearable recorded the strongest average RSSI, demonstrating clear signal dominance when the transmitter and receiver are co-located. For instance, the Living Room anchor measured an average RSSI of -54.8 dB when the wearable was stationary in the Living Room, significantly stronger than the readings from more distant anchors. This same pattern was evident in all test scenarios, including the Kitchen (-67.8 dB), Bathroom (-55.2 dB), and Bedroom (-71.6 dB) tests.

Table 6. RSSI Summary for Living Room

10-Minute RSSI Summary: Wearable Stationary in Living Room				
Anchor Room Location	**Number of Readings**	**Standard Deviation**	**Range**	**Average rssi in dB**
Bathroom	119	2.71	13	-69.7
Bedroom	120	1.58	6	-67.9
Kitchen	118	3.16	17	-81.8
Living Room	119	5.06	16	-54.8

Table 7. RSSI Summary for Bedroom

10-Minute RSSI Summary: Wearable Stationary in Bedroom				
Anchor Room Location	**Number of Readings**	**Standard Deviation**	**Range**	**Average rssi in dB**
Bathroom	120	6.26	16	-73.4
Bedroom	119	5.31	15	-71.6
Kitchen	118	2.05	8	-95.8
Living Room	120	5.55	18	-76.2

Table 8. RSSI Summary for Kitchen.

10-Minute RSSI Summary: Wearable Stationary in Kitchen				
Anchor Room Location	**Number of Readings**	**Standard Deviation**	**Range**	**Average rssi in dB**
Bathroom	119	3.39	12	-76.2
Bedroom	119	1.00	5	-82.0
Kitchen	120	6.06	18	-67.8
Living Room	120	2.42	11	-78.3

Table 9. RSSI Summary for Bathroom

10-Minute RSSI Summary: Wearable Stationary in Bathroom				
Anchor Room Location	**Number of Readings**	**Standard Deviation**	**Range**	**Average rssi in dB**

Bathroom	120	7.25	24	-55.2
Bedroom	120	2.94	9	-71.8
Kitchen	95	0.96	5	-85.4
Living Room	120	1.21	5	-80.9

The visualizations in Figure 20 reinforce the trends observed in the raw data tables, clearly showing that the room in which the wearable is placed corresponds to the highest average RSSI value across all four trials. In each subplot, the bar representing the anchor in the same room as the wearable stands significantly higher (i.e., closer to 0 dBm), supporting the effectiveness of using RSSI for room-level localization. However, the inclusion of standard deviation error bars reveals some overlap between signals from different rooms, particularly in more ambiguous cases like the Bedroom trial. For example, in subplot (b), the RSSI values received from the Bedroom and Bathroom anchors are close enough with overlapping error bars that confidently distinguishing the room becomes more difficult. This indicates that while RSSI is a strong general indicator of proximity, environmental factors and signal variability can reduce certainty, and averaging alone may not always guarantee unambiguous localization. Furthermore, the observed variability underscored the need for a more robust anchor design, such as using ESP32-WROOM modules with external antennas to improve signal stability and spatial discrimination.

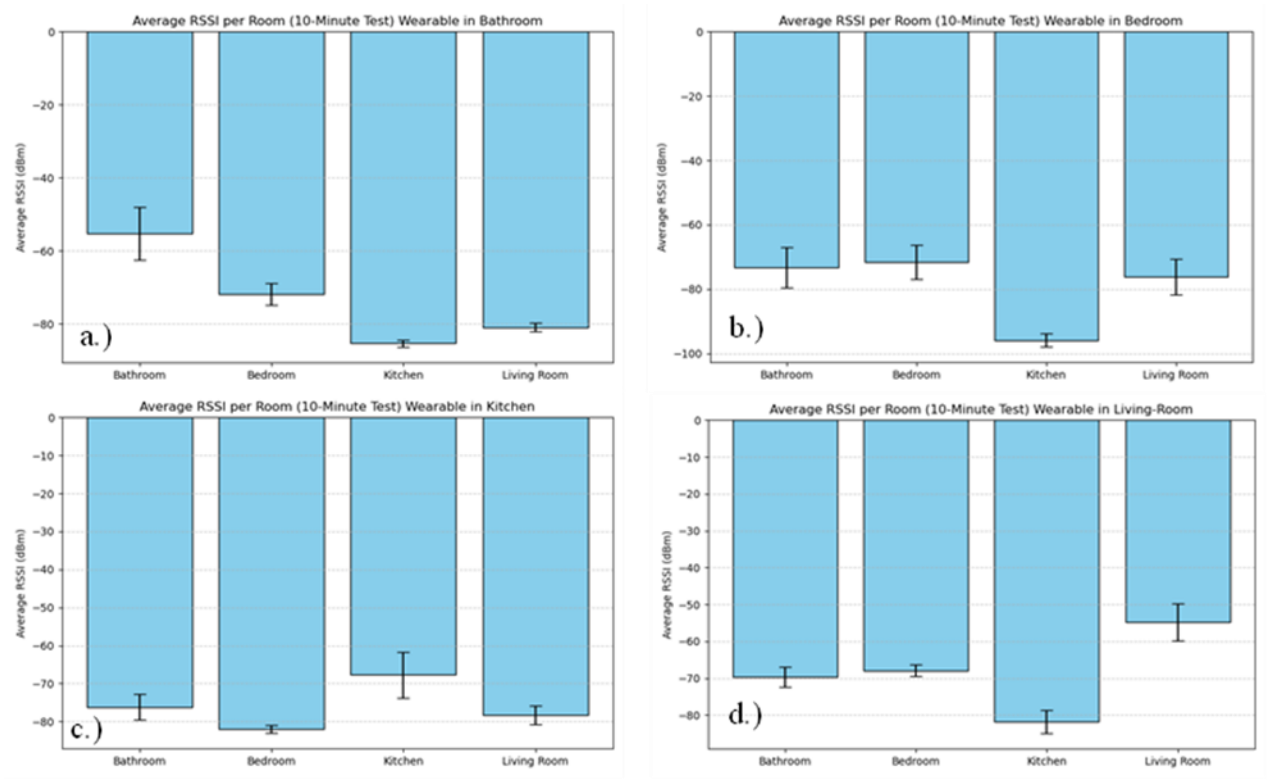


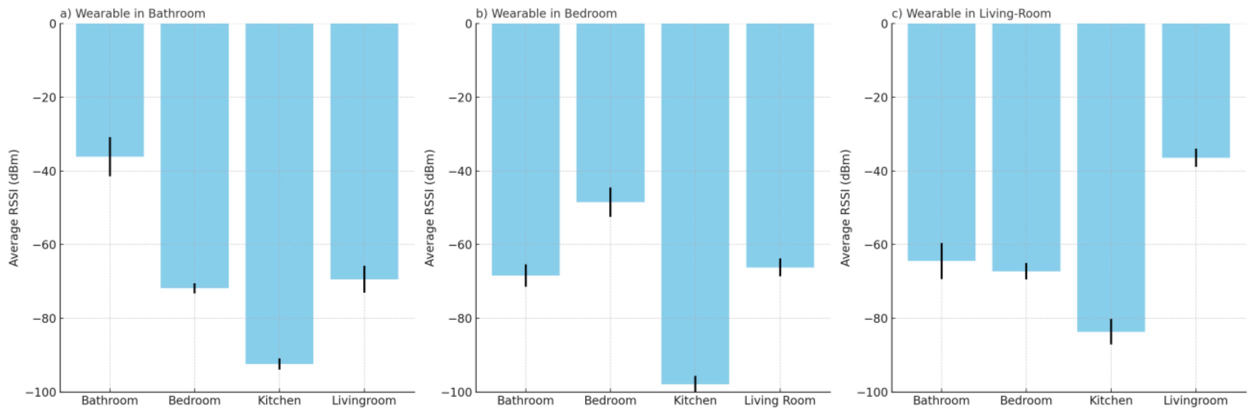
Figure 22. Bar Graphs of Average RSSI with PCB antenna.

Values per room with standard deviation error bars for each 10-minute trial. Subplots represent trials where the wearable was stationary in:(a) Bathroom, (b) Bedroom,(c) Kitchen, and (d) Living Room.

Building on the observations from the PCB antenna trials, the next set of experiments utilized ESP32-WROOM-32U modules equipped with external 2.4 GHz antennas to further improve signal stability and localization accuracy. Figure 21 presents the results of these updated trials, following the same experimental setup but replacing the internal PCB antennas with high-gain external antennas. As shown in the bar graphs, the use of external antennas led to noticeably stronger average RSSI readings and reduced overlap between error bars across different room anchors. In particular, the subplots reveal clearer separation between the in-room anchors and neighboring anchors, even in previously challenging environments such as the Bedroom. The tighter error margins and increased signal consistency demonstrate that incorporating external antennas significantly enhances the system's ability to distinguish room level locations based on RSSI. These improvements support the decision to move to an external antenna architecture for

Figure 23. Bar Graphs of Average RSSI with External Antenna.

the final system design, balancing low-cost hardware with higher localization reliability.



Examining the 30-second average RSSI data across the three room tests in Figure 22 revealed clear trends supporting room level determination. In each subplot, the anchor located in the same room as the wearable consistently shows the strongest average RSSI values throughout the 10-minute period. For instance, during the Bathroom test (subplot a), the Bathroom anchor maintains the highest RSSI, while signals from the other anchors remain significantly weaker. Similarly, in the Living Room and Bedroom tests (subplots b and c), the corresponding room anchors dominate in signal strength. Although minor fluctuations and occasional overlaps between non-dominant anchors are visible, the separation between the correct room's RSSI and others is generally sufficient for reliable classification. These results highlight the effectiveness of using averaged RSSI values over short time windows to mitigate noise and improve location detection performance.

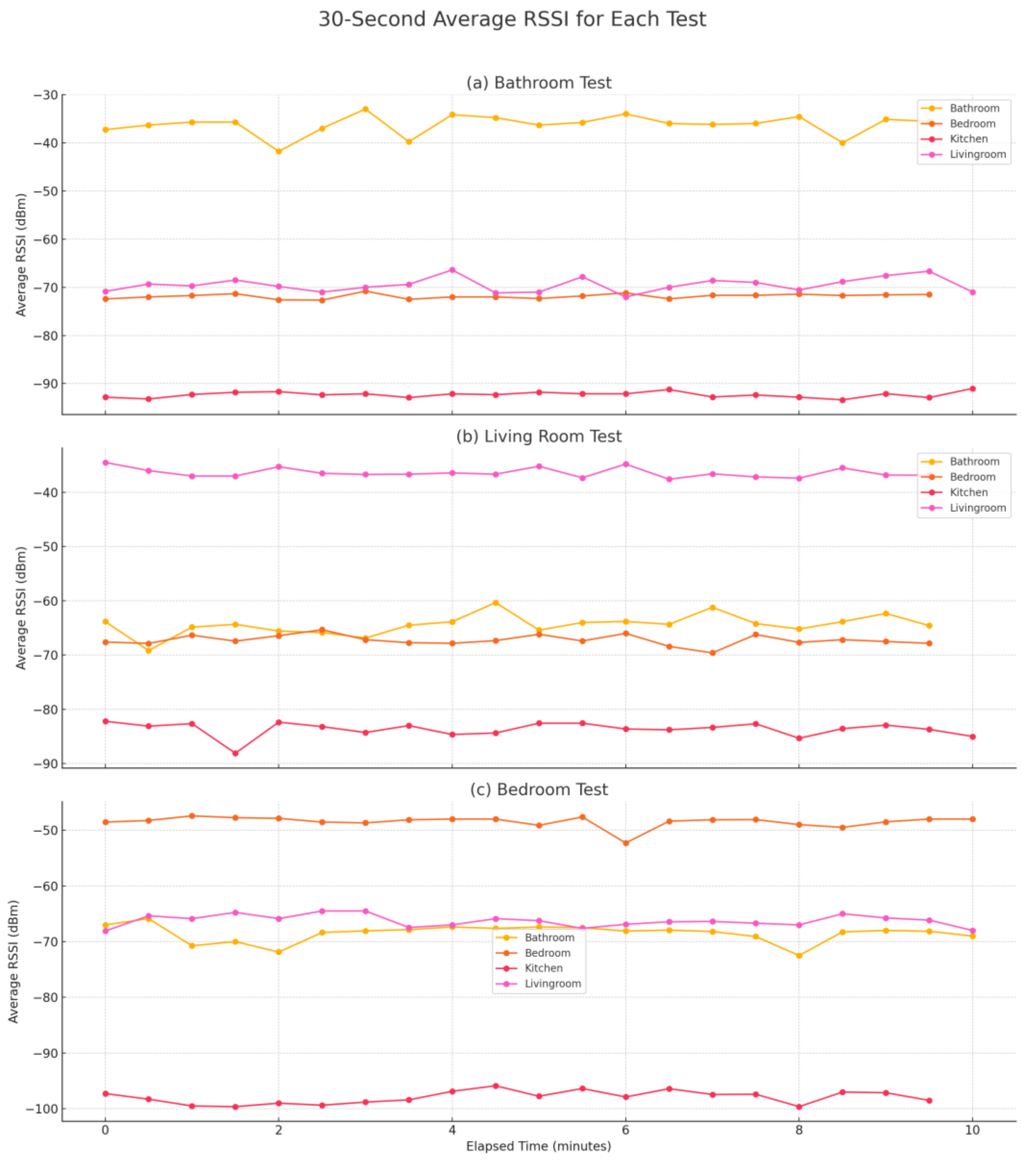


Figure 24. 30 Second Average RSSI Measurements During Each Room Test with External Antenna.

To statistically validate that the correct room anchor consistently exhibited stronger signal strength, a series of paired t-tests were performed comparing the 30 second averaged RSSI values of the correct anchor to those of the competing anchors across 20 windows per room trial. The null hypothesis for each test stated that there was no difference or that the correct anchor's RSSI was not significantly stronger.

In all comparisons, the correct anchor's average RSSI was significantly higher than the others ($p < 0.001$ across all tests). For example, in the Bathroom test shown in Table 10, the

Bathroom anchor's RSSI exceeded the Bedroom anchor's RSSI by an average of 35.8 dB with a standard deviation of 2.1 dB, yielding a t-statistic of 76.00 and a p-value of 2.27×10^{-25}. Similar levels of statistical significance were observed in the Bedroom and Living room tests. These results confirm that not only was the correct anchor consistently the strongest, but the magnitude of signal strength separation was robust and highly unlikely to have occurred by random chance.

Table 10. P Value Results with External Anchor

	Bathroom Test P values	Bedroom Test P values	Living room Test P values
Bathroom vs Bedroom	2.27E-25		
Bathroom vs Kitchen	2.79E-29		
Bathroom vs Living room	2.86E-24		
Bedroom vs Bathroom		2.16E-22	
Bedroom vs Kitchen		9.85E-30	
Bedroom vs Living room		1.38E-22	
Living room vs Bathroom			2.19E-22
Living room vs Bedroom			5.40E-27
Living room vs Kitchen			1.13E-31

It is important to note that the stationary testing was conducted under idealized conditions designed to establish baseline system performance. The wearable device was placed in a seated position within each room and maintained a direct line of sight to all anchors, minimizing the impact of obstructions, multipath reflections, or human body attenuation. As a result, the collected RSSI data reflects best-case performance scenarios. The testing did not account for more challenging situations such as wearable placement near adjoining walls between rooms, individuals falling or lying against walls, or significant interference from moving obstacles. Future evaluations will be necessary to assess system robustness under more complex and realistic usage conditions where signal overlap and environmental variability may degrade localization accuracy.

Figure 23 presents heatmaps of triangulated wearable positions using 30-second average RSSI values from multiple Bluetooth anchors, with each subplot representing a different trial where the wearable remained stationary in one of four rooms. To estimate the position of the wearable device from RSSI measurements, a commonly used mathematical model was applied to convert signal strength into distance:

Equation 2.

$$Distance = 10^{((txPower-RSSI)/(10xn))}$$

This logarithmic path loss model is widely used in BLE localization research, where txPower is the RSSI at 1 meter and n is the path-loss exponent, typically set to 2.0 for indoor environments [17]To estimate the (x, y) location of the wearable, the algorithm minimizes the sum of squared differences between the estimated distances from RSSI and the Euclidean distances to known anchor coordinates. This minimization is performed using the L-BFGS-B algorithm, a well-established quasi-Newton optimization technique suitable for bounded multivariate problems [18]. These methods were adapted for use in this project based on open-source examples and literature in BLE-based indoor positioning systems [19] and thus are not original contributions of this work. For technical details, the complete source code used for the triangulation algorithm is included in Appendix D.

In each case, the red dots indicate estimated positions, while the green “X” marks the known true location. The results show that while the triangulated points tend to cluster near the correct room, they do not consistently fall within the precise boundaries of the actual room. In some trials particularly the Bathroom and Living Room (subplots b and d) estimated positions show noticeable spatial drift, likely due to multipath interference. The anchors’ signals all had different walls and closets to pass through due to the asymmetric house build and orientation.

These results suggest that triangulation, in its current form using averaged RSSI data, may lack the precision needed for definitive room identification. However, it does demonstrate a useful degree of spatial grouping, which could be leveraged to detect relative movement over time. If a cluster of estimated points shifts from one region of the home to another, this can serve as a strong indicator that the wearer has changed location. As such, while triangulation may not be the primary method for real time room detection, it can be instrumental in identifying movement, especially when combined with other metrics such as abrupt changes in anchor RSSI rankings or IMU movement data. That said, this method requires mapping out anchor positions within a floor plan, which introduces additional setup complexity and may reduce its suitability for plug-and-play or consumer friendly installations where minimal configuration is preferred.

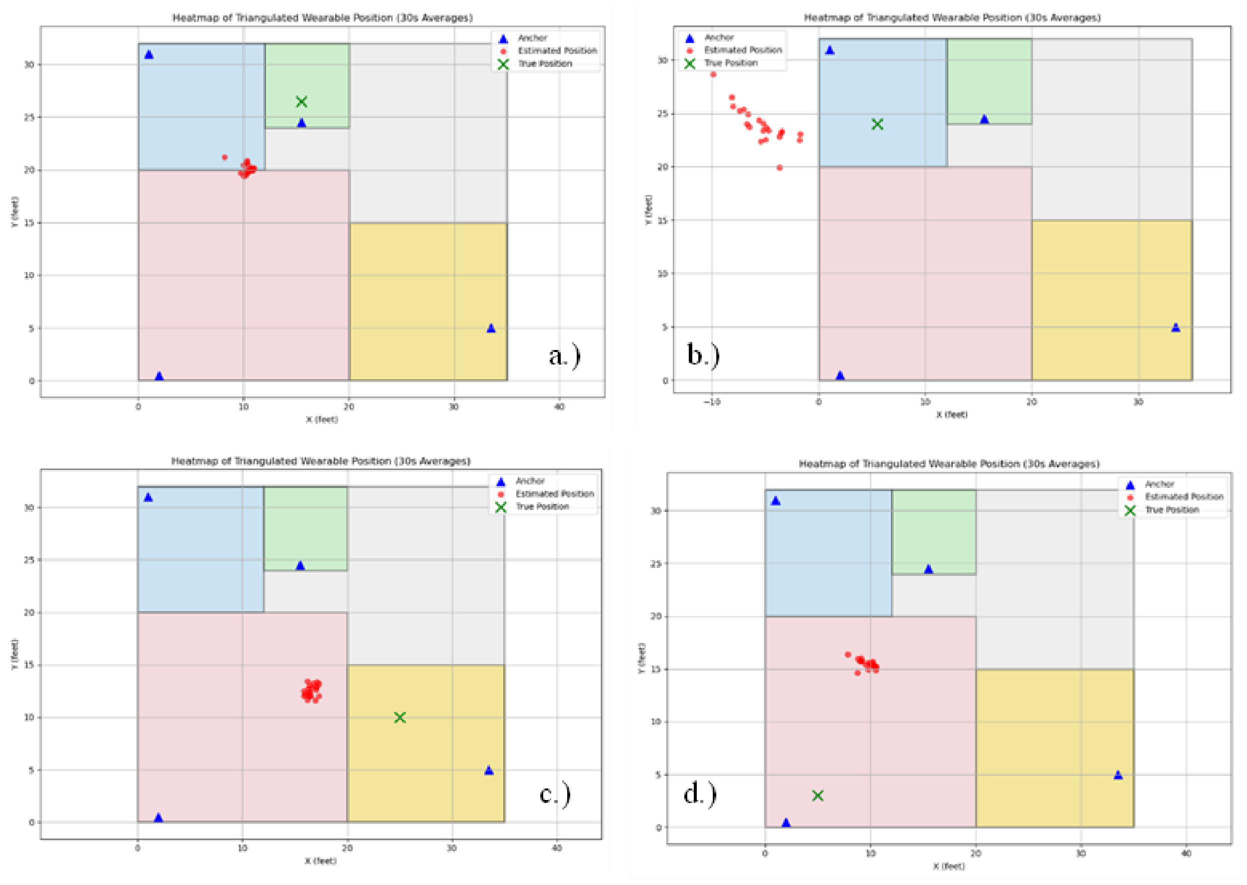


Figure 25. Heatmaps of Triangulated Wearable Positions.

Using 30-second averaged RSSI values from multiple anchors. Subplots represent trials where the wearable remained stationary in: (a) Bedroom, (b) Bathroom, (c) Kitchen, and (d) Living Room. Blue triangles indicate fixed anchor positions, green Xs represent the known true position of the wearable, and red dots show the estimated positions over time.

Our Bluetooth anchor system shares architectural similarities with several BLE-based localization and monitoring systems presented in recent literature. Kolakowski and Blachucki [20] describe a system that uses fixed-position Laird BL652 anchors with dual external antennas and applies Extended Kalman Filtering to track dementia patients' movement trajectories. While our project also relies on anchors to measure RSSI values from a wearable BLE device, we use cost-effective ESP32-WROOM modules with built-in PCB antennas. This makes our system more accessible and easier to deploy in general home environments. Though our use of internal antennas may introduce greater RSSI variability due to multipath interference, the system remains scalable and could be enhanced with directional antennas or dual-receiver configurations similar to the TELFOR study for improved signal stability.

Other systems, such as those by Surendran and Rohini [21]and De Raeve et al. [22], highlight alternative approaches to BLE-based monitoring. Surendran's system triggers alerts when a BLE beacon worn by an Alzheimer's patient moves beyond a receiver's range, inferring departure from a safe zone using single-point monitoring. De Raeve's system uses BLE advertisements from fall-detection wearables, with fixed detection nodes in each room determining location based on which node first receives the alert. While effective for event-based localization, these systems assume proximity equates to presence, which may reduce accuracy near room boundaries. In contrast, our system performs continuous location estimation by comparing RSSI values from multiple anchors. This enables real-time, room-level tracking with broader spatial awareness, making it more suitable for detailed activity monitoring in home environments.

3.6 SUBSYSTEM INTERFACING

3.6.1 PC application interface with MCUs

The interface between the PC application and the MCUs will use WIFI for the physical and data link layers. The other layers of the OSI model are covered by ESPnow. There is a direct connection between the PC and a single MCU which we call the HUB. The hub acts as a gateway to the ESPnow network for the PC and the connection between them is serial.

3.6.2 Wearable Interface with Stationary Anchors

The wearable device will emit a Bluetooth signal by setting it up in a discoverable mode. The Bluetooth anchors will continuously scan for that discoverable signal, then send the RSSI information to the PC application through the hub via ESPnow.

3.7 STANDARDS AND COMPLIANCE

Table 11. Standards and Compliance.

Name of standard	Specific action related to project needed to comply
Wi-Fi (802.11)	Use ESP-32 SOC with Wi-Fi integrated/tested for regulatory compliance. Refer to ESP-NOW section for more information
Bluetooth 5	Use ESP-32 SOC with Bluetooth integrated/tested for regulatory compliance

HIPPA	Create written business associate contracts or other arrangements with the nursing homes that establishes specifically what we have been engaged to do comply with HIPPA by protecting personally identifiable information

3.8 FUTURE WORK

3.8.1 ESP-now mesh networking

The current version of the firmware requires that the hub, each anchor, and the wearable system need to have the MAC addresses programmed into the firmware. This is not ideal because each future system will have to be programmed at the factory with the components which will be used on the system. This fails to meet the "easy installation" requirement because a user will not be able to add rooms themselves or replace broken devices. The solution to this problem would be for each device to send a pairing message to the broadcast address(FF:FF:FF:FF:FF:FF) then each device would need to identify it's type (wearable, anchor, hub) and finally the part of the code which links the MAC addresses to actual room names would need to be moved to the PC program for user configuration. This way each device could be programmed with a default firmware image then could join the network with the others and communicate.

3.8.2 Custom wearable PCBs and housing

The current wearable device uses a Waveshare dev board. This dev board's PCB contains a few subsystems which were used, like the battery monitor voltage divider, and the power on/off circuits. The issues with this board are related to mounting, and the low-quality parts used on the board. Firstly, the board's IMU (QMI8658) is listed by Waveshare as a QMI8658C when it is really a QMI8658A. The QMI8658 chip is not available for sale to consumers. Likely because no one would buy it while there are reputable IMUs on the market for low cost. This part would be replaced with one from a manufacturer like TI, TDK, STMicroelectronics, or any other normal sensor. This IMU is not special, and the requirements for it are not especially strict. It needs to output accelerometer and gyroscope data at 50Hz, have a FIFO with an interrupt-based trigger, and communicate via I2C. If a board was to be developed it could also integrate a better heart rate chip. The MAX30101 used in this project was selected mainly because it was available on a dev board from an easy-to-use supplier. There are newer, more feature rich heart rate solutions which include multiple LEDs and ADCs to measure the PPG signal on the market.

At the time of writing this, hobby development boards have not yet integrated these newer sensors. Finally, the topic of the battery size could also be addressed. If a new housing and board design was completed then a smaller, lighter battery could be integrated. The current 3000mAh battery was selected because of concerns with Wi-Fi taking a lot of energy. The Concern was we would not have enough battery time to test if we went with a reasonably sized battery.

4 BUDGETS AND FINAL SCHEDULE

4.1 POWER BUDGET

The power usage of the devices included in the wearable portion of the design is extra important because users won't wear a device that needs to be charged many times per day. The proposed Lithium Polymer Battery 3.7V 3000mAh coupled with the ETA6098 charging chip present on the Waveshare ESP-32 allow for significant extra time than needed by a wristwatch. DC current measurements show that the device draws about 100mA of current with the screen and beeper off. Long time scale testing shows that the max time the device would last is about 30h. Scope analysis shows that the screen and the beeper both pull significant amounts of current. Use of these peripherals will significantly decrease the battery life of the device.

The ESP-32 chips' current varies greatly based on the functions currently in use. For example, using the Wi-Fi functions appears to push the current as high as 340mA peak current. The Waveshare board uses an LDO from the battery voltage to 3.3V for everything on the board.

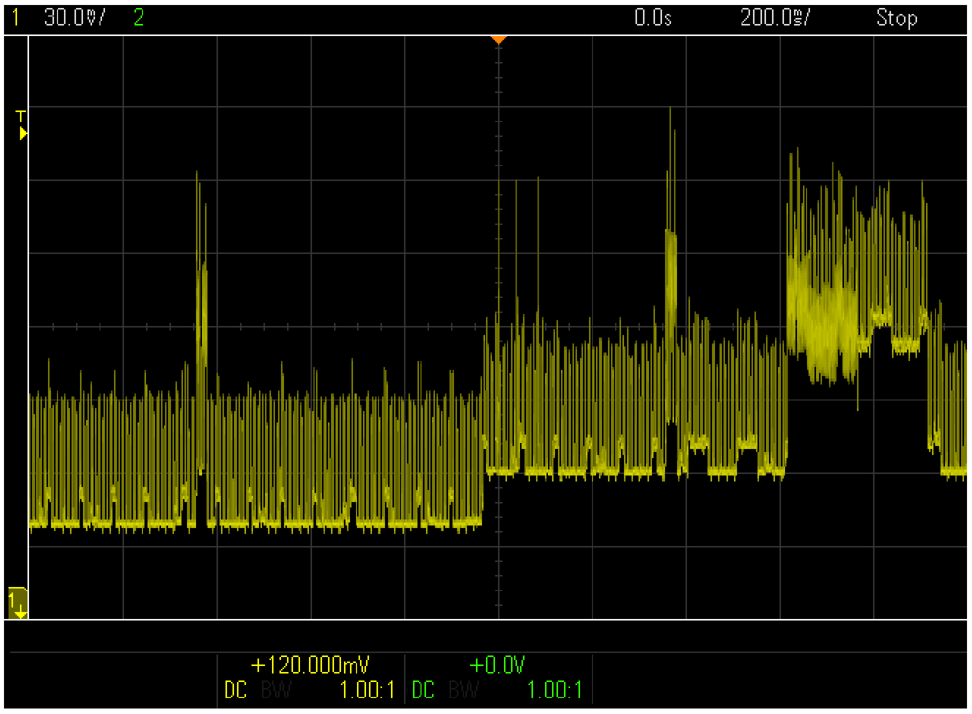


Figure 26 Wearable Current usage AC

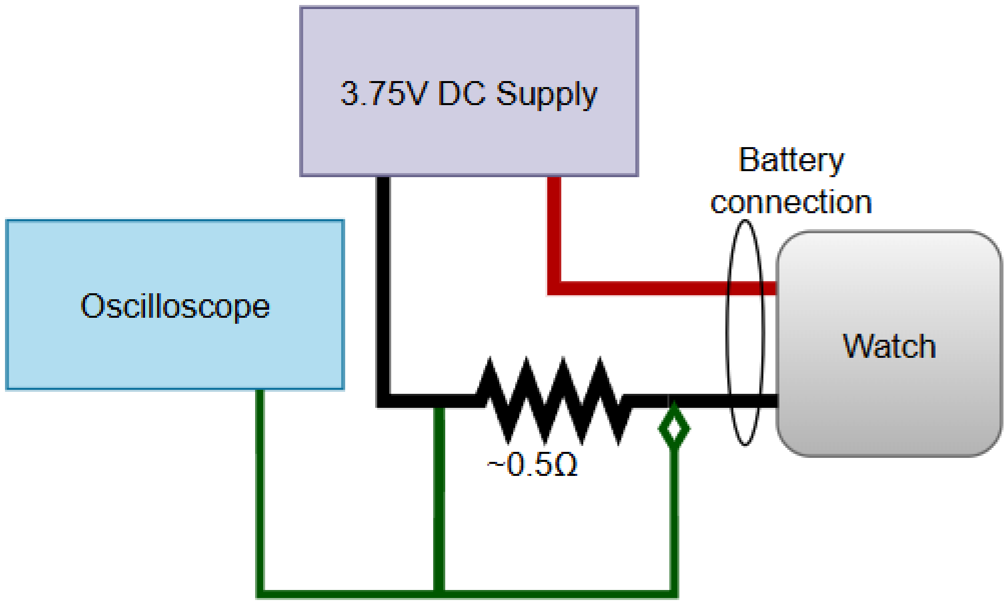


Figure 27 Wearable AC Current Test Setup

The figures above show the test setup and results of the test showing how each of the peripherals effect the current draw of the device. The first DC shift in the data is from the screen(~40mA), and the second large shift is from the buzzer (~80mA). The two large spikes are exactly one second apart and show how much energy is used for the ESP-Now tranmissions.

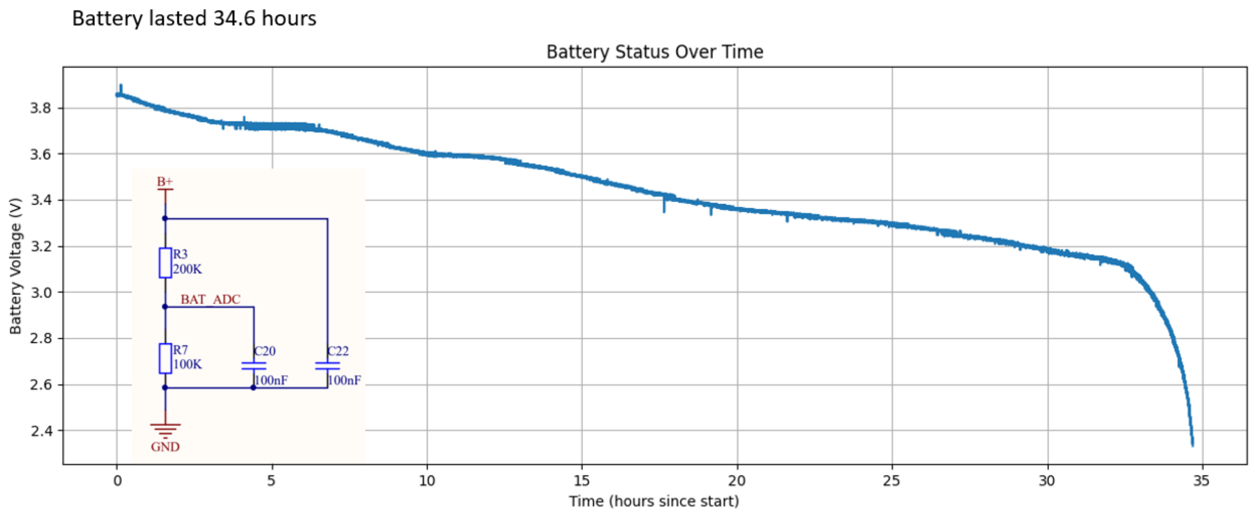


Figure 28 Long time scale battery testing

The long-time scale battery testing validates the DC current draw measurement of 100mA. The device lasted 34.6 hours on a 3000mAh battery.

4.2 Cost Breakdown

The break-even analysis presented in Table 10 and visualized in Figure 15 outlines the financial feasibility of the wearable safety monitoring system. Fixed costs total approximately $490,224, covering engineer and builder salaries, facility rent, administrative expenses, and mold manufacturing. The variable cost to produce each unit is $51.86, including components like the Waveshare ESP32, plastic housing, PPG sensor, and ESP32 anchor boards. With a sales price of $259.30 per unit—reflecting a 500% markup—the system is positioned to achieve profitability while staying competitive in the medical monitoring market.

Figure 15 clearly shows that the revenue line (yellow) surpasses the total cost line (blue) at around 2,400 units, marking the break-even point. At this point, cumulative revenue covers both fixed and variable expenses. The fixed cost line (orange) remains constant, while production costs (gray) increase gradually as units are produced. The steep incline of the revenue curve compared to total cost indicates favorable unit economics. This analysis demonstrates that after the initial investment, the project becomes increasingly profitable with each unit sold—highlighting strong potential for scalability and market sustainability.

Table 12. Break-Even Analysis

salaries 3 engineers	300,000
building rent $7/sf/y, 3000sf	21000
2 builders $27/hr full time + benfits	154224
mold manufacturing	5000
Admin	10000
sum	490,224

Part	Cost
Waveshare ESP32	27.83
plastic housing	0.05
PPG sensor	6.99
3x ESP32 dev boards for anchors	16.99
total cost to produce each part	51.86
sale price (500% industry standard markup)	259.3

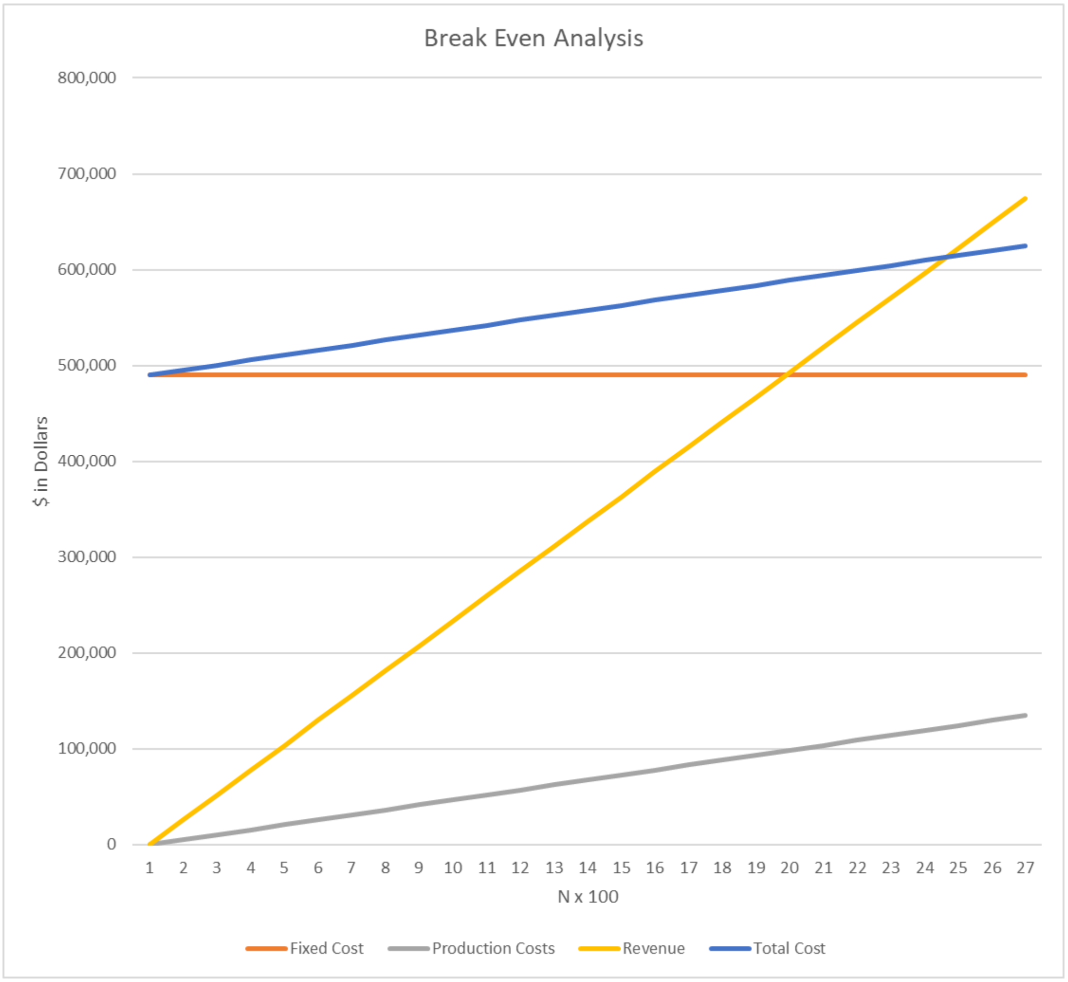


Figure 29. Break Even Graph

4.3 TEAMWORK BREAKDOWN

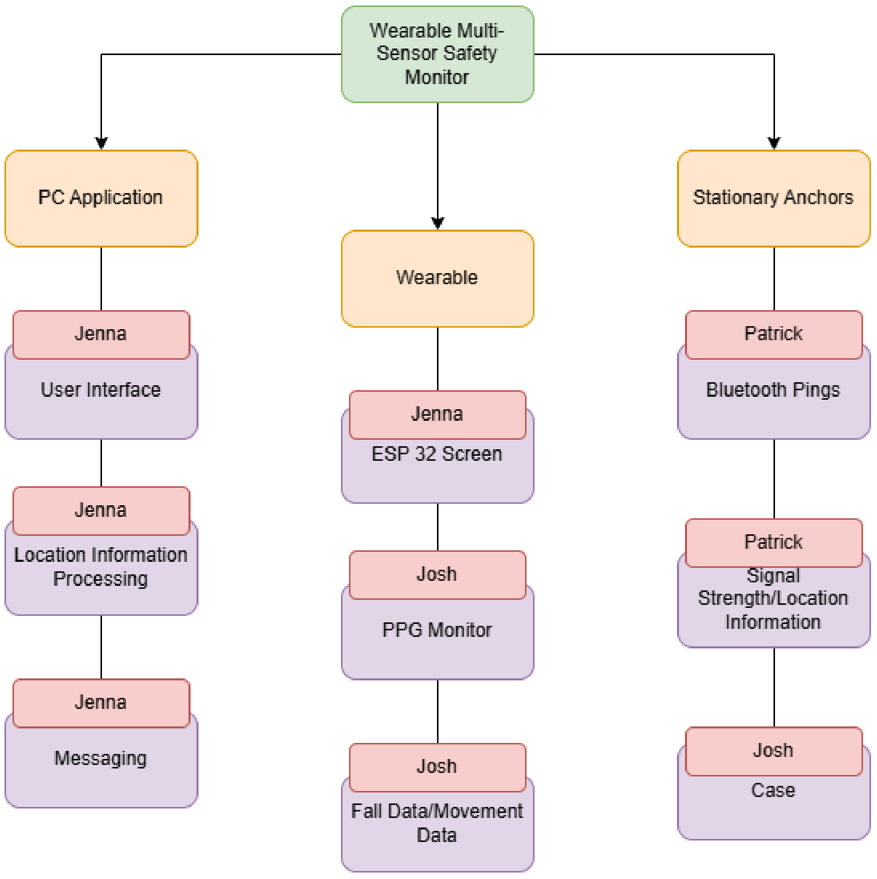


Figure 30. Teamwork Breakdown.

5 CONCLUSION

In addition to the above research and preliminary design, this group has begun testing of individual components, design of housing for the wearable, and very introductory filtering of PPG data. The next steps of the project will be three equal parts worked on at the same time. The first is to power a rudimentary version of the wearable device and have it send pings to multiple Bluetooth anchors. Then this data will be sent via Wi-Fi to a computer where it will be received. The second will be to have IMU and PPG data sent from the rudimentary device to a computer via Wi-Fi at the same time the Bluetooth anchors are sending data. The third is to create a basic user interface and data management program.

In summary, the proposed Geriatric Care Assistance System offers significant potential to improve the health and wellbeing of elderly individuals while addressing economic and social challenges within the healthcare environment. By enabling older adults to remain in their homes longer and providing tools for timely health intervention, the system fosters independence and enhances quality of life. This innovation could reduce the reliance on full-time healthcare or nursing home services, yielding substantial economic benefits for families and healthcare systems alike. The cascading effects, from reducing Medicare and Medicaid expenditures to potentially lowering insurance costs and enhancing nursing home efficiency, illustrate the broader societal impacts of this system.

While the system presents numerous advantages, it is vital to address its limitations. Over-reliance on the device's normal functioning without acknowledging its dependency on local computing and network stability could lead to critical gaps in care, such as missed alerts during system failures. Despite this limitation, the device's ability to detect and respond quickly to falls, health emergencies, or wandering incidents represents a groundbreaking step toward improving response times and health outcomes for aging populations. With proper user education and robust system design, the Geriatric Care Assistance System could revolutionize eldercare, extending life expectancy and improving care efficiency on a large scale.

REFERENCES

[1] "Older Adult Fall Prevention," Centers for Disease Control, 09 05 2024. [Online]. Available: https://www.cdc.gov/falls/data-research/facts-stats/index.html. [Accessed 09 10 2024].

[2] "Get the facts on Falls Prevention," National Council on Aging, 01 June 2024. [Online]. Available: https://www.ncoa.org/article/get-the-facts-on-falls-prevention/. [Accessed 8 10 2024].

[3] "Wandering," Alzheimer's Association, [Online]. Available: https://www.alz.org/help-support/caregiving/stages-behaviors/wandering. [Accessed 08 10 2024].

[4] Alert-1, "A Brief History of the First Personal Emergency Response System," Alert-1, [Online]. Available: https://www.alert-1.com/content/personal-emergency-response-system/789#:~:text=A%20Brief%20History%20of%20the,cared%20for%20in%20any%20situation.. [Accessed 3 10 2024].

[5] C. Habas, "National Council on Aging," National Council on Aging, 23 09 2024. [Online]. Available: https://www.ncoa.org/adviser/medical-alert-systems/best-elderly-monitoring-system-in-home-use/. [Accessed 8 10 2024].

[6] Espressif, "ESP-IDF Programming Guide," [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_now.html. [Accessed 8 4 2025].

[7] freeRTOS, "FreeRTOS," [Online]. Available: https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals. [Accessed 8 4 2025].

[8] J. Marques and P. Moreno, "Online Fall Detection Using Wrist Devices," *Sensors,* no. 23, 2023.

[9] A. F. a. R. Harle, "Location Fingerprinting with Bluetooth Low Energy Beacons," *IEEE Journal on Selected Areas in Communicaiton,* vol. 33, no. 11, pp. 2418-2428, Nov. 2015.

[10] A. G. a. K. L. F. Zafari, "A Survey on Indoor Positioning Systems for Internet of Things," *IEEE Communications Surveys and Tutorials* , vol. 21, no. no.3, pp. 2568-2599, 2019.

[11] AITRIP, "ESP-WROOM-32 Module Specifications," AITRIP, [Online]. Available: https://www.aitrip.com/.. [Accessed 8 April 2025].

[12] S. Stuidio, "XIAO ESP32C3 Documentation," [Online]. Available: https://wiki.seeedstuidio.com/XIAO_ESP32C3/.. [Accessed 8 April 2025].

[13] Espressif Systems, "ESP32-C3-DevKitM-a," Esspressif Sytesms, [Online]. Available: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32c3/esp32-c3-devkitm-1/.. [Accessed 8 April 2025].

[14] Amazon.com, "HiLetgo ESP-WROOM-32 Development Board WiFI+Bluetooth Ultra-Low Power Consumption Dual Core," [Online]. Available: https://a.co/d/6QvJC8F. [Accessed 8 April 2025].

[15] Espresssif Systems, "ESP-NOW User GUide," [Online]. Available: https://docs.esspresssif.com/projects/esp-idf/en/latest/esp32/apireference/network/esp_noew.html. [Accessed 8 April 2025].

[16] N. Kolban, "ESP32 BLE Arduino, GitHub Repository," [Online]. Available: https://github.com/nkolban/ESP32_BLE_Arduino. [Accessed 8 April 2025].

[17] A. Faragher, "Understanding the Basis fo the RSSI Path Loss Model for Localization," *Journal of Navigation,* vol. 65, no. 4, pp. 1-10, 2012.

[18] P. L. J. N. a. C. Z. R.H. Byrd, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientiic Computing,* vol. 16, no. 5, pp. 1190-1208, 1995.

[19] Tinkerman, "BLE Distance Estimation," [Online]. Available: https://tinkerman.cat/post/ble-distance-estimation. [Accessed 8 April 2025].

[20] B. B. M. Kolakowski, "Monitoring Wandering Behavior of Persons Suffering from Dimentia Using BLE Based Localization System," in *27th Telecommunications Forum*, Belgrade, Serbia, 2019.

[21] M. R. D. Surendran, "BLE Bluetooth Beacon Based Solution to Monitor Egress of Alzheimer's Disease Sufferers from Indoors," *Procedia Computer Science,* vol. 165, pp. 591-597, 2019.

[22] A. S. M. d. S. E. D. P. I. M. J. V. P. V. T. H. R. N. De Raeve, "Bluetooth-Low-Energy-Based Fall Detection and Warning System for Elderly People in Nursing Homes," *Journal of Sensors,* pp. 1-14, 2022.

[23] A. P., "Python Tkinter Tutorial," GeeksforGeeks, 2023. [Online]. Available: https://www.geeksforgeeks.org/python-tkinter-tutorial/?utm_source=chatgpt.com.. [Accessed 15 3 2025].

[24] P. M. a. jfs, "Using python's os.path, how do I go up one directory?," Stack Overflow, 26 march 2012. [Online]. Available: https://stackoverflow.com/questions/9856683/using-pythons-os-path-how-do-i-go-up-one-directory. [Accessed 19 March 2025].

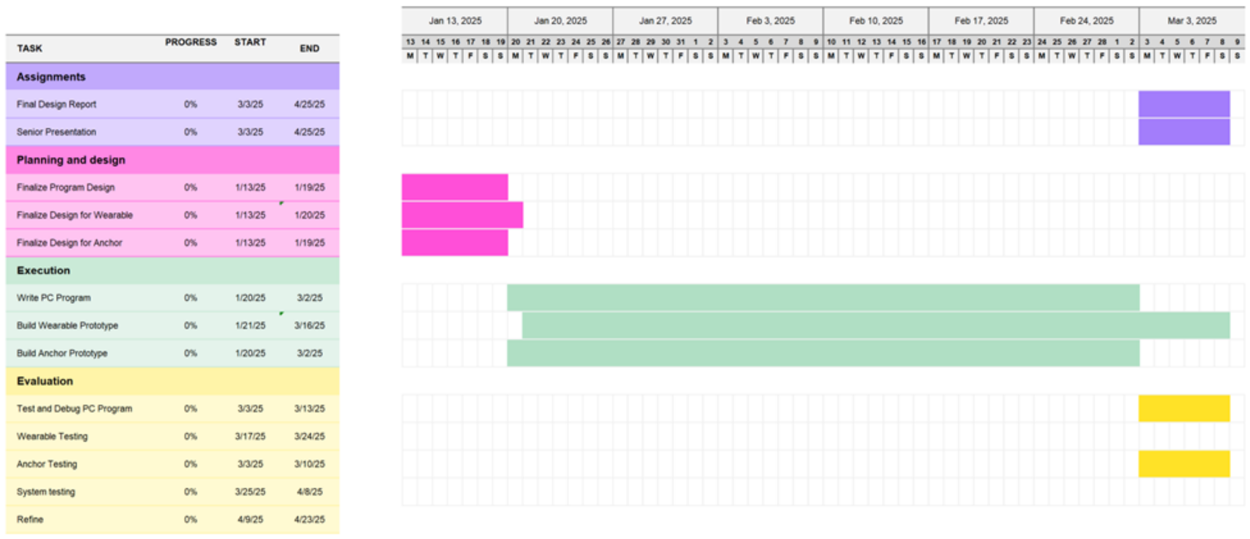
[25] user384706, "Full examples of using pyserial package," Stack Overflow, 18 July 2011. [Online]. Available: https://stackoverflow.com/questions/676172/full-examples-of-using-pyserial-package. [Accessed 27 March 2025].

[26] N. Osman, "What is the proper way to do logging to a CSV file?," Stack Overflow, 3 November 2013. [Online]. Available: https://stackoverflow.com/questions/19765139/what-is-the-proper-way-to-do-logging-to-a-csv-file. [Accessed 4 April 2025].

[2
7] Parallax Incorporated, "Store: 7.2V Motor, Bracket and Wheel Kit," [Online]. Available: http://www.parallax.com/Store/Robots/AllRobots/tabid/755/ProductID/587/List/0/Default.aspx?SortField=ProductName,ProductName. [Accessed 25 September 2011].

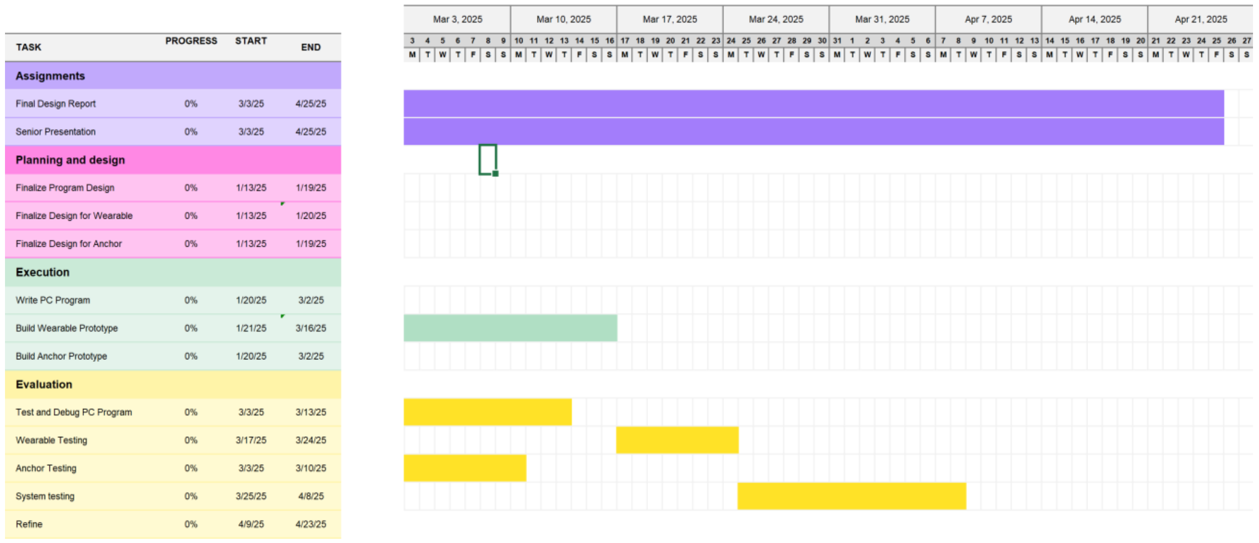
Appendix

Appendix A: Preliminary Project Schedule



TASK	PROGRESS	START	END
Assignments			
Final Design Report	0%	3/3/25	4/25/25
Senior Presentation	0%	3/3/25	4/25/25
Planning and design			
Finalize Program Design	0%	1/13/25	1/19/25
Finalize Design for Wearable	0%	1/13/25	1/20/25
Finalize Design for Anchor	0%	1/13/25	1/19/25
Execution			
Write PC Program	0%	1/20/25	3/2/25
Build Wearable Prototype	0%	1/21/25	3/16/25
Build Anchor Prototype	0%	1/20/25	3/2/25
Evaluation			
Test and Debug PC Program	0%	3/3/25	3/13/25
Wearable Testing	0%	3/17/25	3/24/25
Anchor Testing	0%	3/3/25	3/10/25
System testing	0%	3/25/25	4/8/25
Refine	0%	4/9/25	4/23/25

Figure 31. First Half of Semester Schedule



TASK	PROGRESS	START	END
Assignments			
Final Design Report	0%	3/3/25	4/25/25
Senior Presentation	0%	3/3/25	4/25/25
Planning and design			
Finalize Program Design	0%	1/13/25	1/19/25
Finalize Design for Wearable	0%	1/13/25	1/20/25
Finalize Design for Anchor	0%	1/13/25	1/19/25
Execution			
Write PC Program	0%	1/20/25	3/2/25
Build Wearable Prototype	0%	1/21/25	3/16/25
Build Anchor Prototype	0%	1/20/25	3/2/25
Evaluation			
Test and Debug PC Program	0%	3/3/25	3/13/25
Wearable Testing	0%	3/17/25	3/24/25
Anchor Testing	0%	3/3/25	3/10/25
System testing	0%	3/25/25	4/8/25
Refine	0%	4/9/25	4/23/25

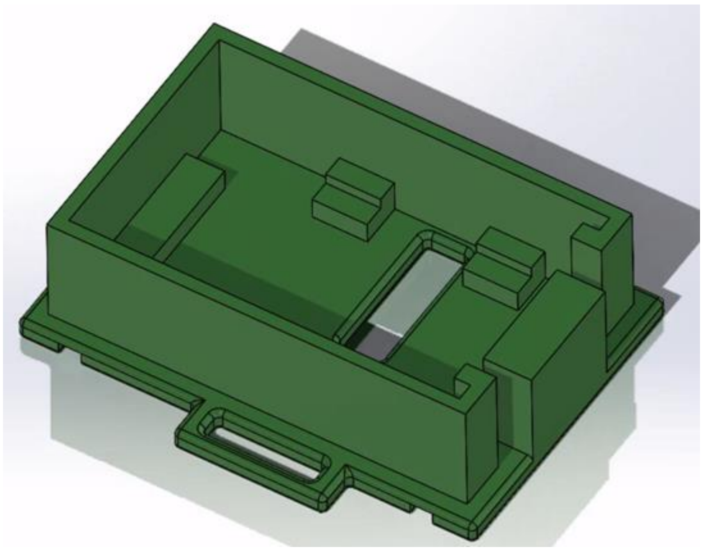
Figure 32. Second Half of Semester Schedule

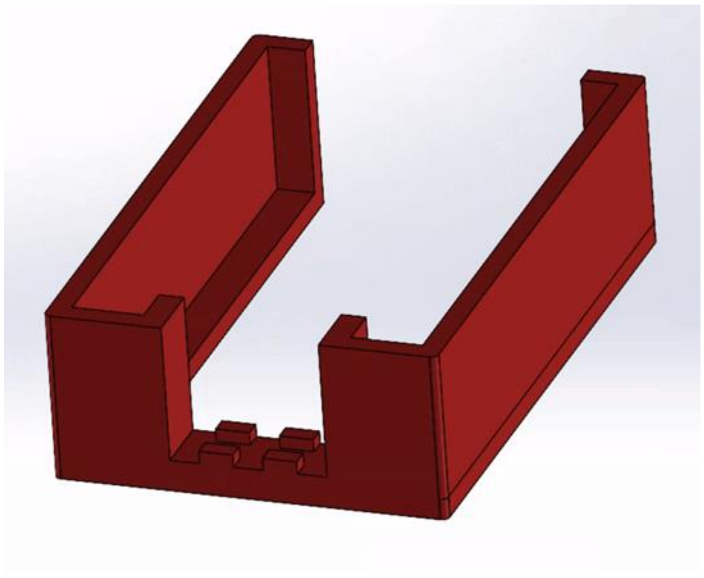
Appendix B: Bill of Materials

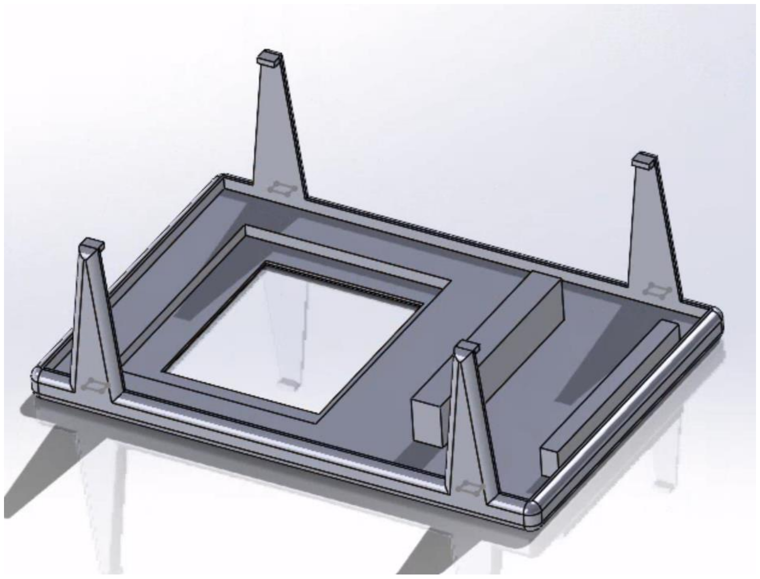
Table 13. Bill of Materials

Part	Part Vendor	Unit Cost	Quantity	Line Total
ESP32-WROOM U with External Antenna	Amazon	9.30	5	46.50
Sparkfun MAX30101	Sparkfun	22.50	1	22.50
Waveshare ESP32-S3 Development Board with 1.69 inch Touch LCD	Waveshare	20.29	1	20.29
EEMB Lithium Polymer Battery 3.7V 3000mAh	Amazon	9.99	1	9.99
Anchor and Hub Cable, USB A to Type C Charger Cord	Amazon	2.23	5	11.15
Wall Charger Cube	Amazon	1.70	5	8.5
			Total	**118.93**

Appendix C: Drawings for Custom Parts







Appendix D: Complete Listing of Code

<u>PC Application – Messaging Center Code</u>

This code was developed and compiled using Visual Studio Code. The Python Tkinter Tutorial on Geeks for Geeks [23] was very helpful when learning how to use the Tkinter library. Some other helpful sources were on Stack overflow, and they pertained to using an os.path in python [24], using pyserial packages in python [25], and logging to a CSV file in python [26].

```python
import re
import os
import csv
import serial
import threading
import smtplib
from email.mime.text import MIMEText
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import winsound
import time
from datetime import datetime


# ------------- Configuration -----------------
# ---------------------------------------------
SERIAL_PORT = "COM3"                    # Change Port as needed
BAUD_RATE = 115200                      # Set to Devices Baud Rate
CSV_FILE = "wearable_log.csv"           # Name CSV file
EXTRA_CSV_FILE = "wearable_sensors.csv" # Name CSV with wearable sensor data
RSSI_THRESHOLD = -55              # Set threshold for determining if wearer is in
room

# -------------- SMS Alert Configuration -------------
# ---------------------------------------------------
PHONE_NUMBER = "2709251332"          # Phone number of Caregiver
CARRIER_GATEWAY = "txt.att.net"      # Carrier Gateway - dependent on Caregiver
phone carrier
EMAIL = "sendesign2025@gmail.com"
EMAIL_PASSWORD = "tmiocunrhlcvxqir"

# ------------- Map Anchor names to room names -----------
# -------------------------------------------------------
room_name_map = {
    #"JHRoom": "Living Room",
    "Livingroom": "Living Room",
    "Kitchen": "Kitchen",
    "Bathroom": "Bath Room",
    "Bedroom": "Bedroom"
}

# ----------- Initialize Tracking Variables --------------
# -------------------------------------------------------
anchor_rssi = {}                              # Stores most recent RSSI values for
each anchor
```

```
last_room = None                            # Keeps track of last known room
wearer was detected in
last_room_change_time = time.time()         # Stores timestamp (in seconds since
epoch) of the last room change
last_movement_time = time.time()            # Stores time of last detected
movement
last_hr_alert_time = 0                      # Time stamp for last heart rate
alert
fall_detected_time = None                   # Keeps track of when fall was first
detected (but not confirmed)
fall_state = "none"                         # Tracks fall detection state
fall_pending_time = 0                       # When fall is first detected
fall_confirmed = False                      # Flag for fall confirmed or not
tracking
quiet_hours_manual = False                  # Flag for quiet hours being set
manually
last_wandering_alert = 0                    # The time of the last wandering
alert
was_wandering = False

# ----------------- Image Path Definition ----------------
# --------------------------------------------------------
IMAGE_DIR = os.path.join(os.path.dirname(__file__), "images")   # Direct path to
folder with images in it

# ------------------ Logging Setup -----------------------
# --------------------------------------------------------
with open(CSV_FILE, mode='w', newline='') as file:           # Create new CSV
file named CSV_FILE in write mode
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Type", "Anchor", "RSSI", "Fall", "HR",
"UserMsg", "Room"])     # CSV Column Headers

with open(EXTRA_CSV_FILE, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "batStatus", "aveAccel", "aveJerk",
"rotation"])

# ------------ wearable_log.csv Column Descriptions -----------------
# -----------------------------------------------------------------
# "Timestamp" - The date and time when data was logged (e.g., 2025-04-09
16:45:12)
# "Type"      - Data source: "anchor" or "waveshare"
# "Anchor"    - The name of the anchor that received the RSSI signal (e.g.,
Living Room, Kitchen)
```

```
# "RSSI"      - Signal strength value from the anchor (negative number; higher =
stronger)
# "Fall"      - Fall detection status: 1 = fall detected, 0 = no fall
# "HR"        - Heart rate from the wearable device in BPM (beats per minute)
# "UserMsg"   - User message or response (e.g., Confirm fall, Deny Fall, SOS
button)
# "Room"      - Determined room location based on RSSI values and threshold logic

# ----------- wearable_sensors.csv Column Descriptions ----------------
# --------------------------------------------------------------------
# "Timestamp" - The date and time when the sensor data was logged (e.g., 2025-04-
09 16:45:12)
# "batStatus" - Battery voltage or status level of the wearable device
# "aveAccel"  - Average acceleration measured (magnitude of movement)
# "aveJerk"   - Average jerk (rate of change in acceleration)
# "rotation"  - Detected body rotation (orientation or tilt angle)

# ------------- SMS Message Send Function -----------
# -------------------------------------------------
def send_sms_via_email(phone_number, carrier_gateway, message, email, password):
    to_number = f"{phone_number}@{carrier_gateway}"
    msg = MIMEText(message)
    msg["From"] = email
    msg["To"] = to_number
    msg["Subject"] = "SMS"

    try:
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.starttls()
        server.login(email, password)
        server.sendmail(email, to_number, msg.as_string())
        server.quit()
        print("Message sent successfully!")
    except Exception as e:
        print(f"Failed to send message: {e}")

# -------------- GUI Setup ---------------
# ----------------------------------------
root = tk.Tk()                        # Create the main window
root.title("Message Processing Center")      # Title window

mainframe = ttk.Frame(root, padding="10")                          # Create a
frame to hold other widgets (textbox and image)
mainframe.grid(column=0, row=0, sticky=(tk.W, tk.E, tk.N, tk.S))    # Puts frame
in window and straches it to fit
```

```
output_box = tk.Text(mainframe, height=20, width=100, wrap='word',
font=("Courier", 12))    # Create text box
output_box.grid(column=0, row=0, sticky=(tk.W, tk.E, tk.N,
tk.S))                           # Put textbox in frame and size

scrollbar = ttk.Scrollbar(mainframe, orient="vertical",
command=output_box.yview)   # Vertical scrollbar in textbox
scrollbar.grid(column=1, row=0, sticky=(tk.N, tk.S))       # Size to fit textbox
output_box['yscrollcommand'] = scrollbar.set               # Connect scrollbar to
textbox to impliment scrolling capability

# === Load Images ===
def load_image(path, size=(350, 350)):                  # Function to load image
of this size
    img = Image.open(path)                              # Use PIL to open Image
    img = img.resize(size, Image.Resampling.LANCZOS)    # Resize Image without
pixilating it
    return ImageTk.PhotoImage(img)                      # Change PIL format to
Tkinter format to display

# Define all images for floorplan
Living_img = load_image(os.path.join(IMAGE_DIR, "Person_in_LR.png"))
Bath_img = load_image(os.path.join(IMAGE_DIR, "Person_in_BT.png"))
Bedroom_img = load_image(os.path.join(IMAGE_DIR, "Person_in_BR.png"))
Kitchen_img = load_image(os.path.join(IMAGE_DIR, "Person_in_Kt.png"))
H_Living_img = load_image(os.path.join(IMAGE_DIR, "Hurt_in_LR.png"))
H_Bath_img = load_image(os.path.join(IMAGE_DIR, "Hurt_in_BT.png"))
H_Kitchen_img = load_image(os.path.join(IMAGE_DIR, "Hurt_in_KT.png"))
H_Bedroom_img = load_image(os.path.join(IMAGE_DIR, "Hurt_in_BR.png"))
Blank_img = load_image(os.path.join(IMAGE_DIR, "blank_floor_plan.png"))

# Image display label
image_label = ttk.Label(mainframe, image=Blank_img)
image_label.grid(column=0, row=1, pady=10, columnspan=2)

# Quiet Hours Label - Tells if quiet hours are on
quiet_label = ttk.Label(mainframe, text="", foreground="gray", font=("Courier",
12, "italic"))
quiet_label.grid(column=0, row=2, columnspan=2, pady=(5, 0))

# Quiet hours toggle button function
def toggle_quiet_hours():
    global quiet_hours_manual
    quiet_hours_manual = not quiet_hours_manual
    status = "ENABLED" if quiet_hours_manual else "DISABLED"
    update_display(f"Manual Quiet Hours {status}.")

# Resolve button function
def resolve_problem():
    global last_room, fall_state, fall_confirmed, fall_detected_time,
last_hr_alert_time

    if last_room:
        update_display(f"Status resolved. Wearable is OK in {last_room}.")

        # Restore green image
        if last_room == "Living Room":
            image_label.configure(image=Living_img)
        elif last_room == "Kitchen":
            image_label.configure(image=Kitchen_img)
        elif last_room == "Bathroom":
            image_label.configure(image=Bath_img)
        elif last_room == "Bedroom":
            image_label.configure(image=Bedroom_img)
        else:
            image_label.configure(image=Blank_img)

        # Reset alert flags
        fall_state = "none"
        fall_confirmed = False
        fall_detected_time = None
        last_hr_alert_time = 0

        winsound.MessageBeep(winsound.MB_OK)

# Button Frame to contain buttons in middle of GUI
button_frame = ttk.Frame(mainframe)
button_frame.grid(column=0, row=3, columnspan=2, pady=10)

# Quiet Hours Button
toggle_button = ttk.Button(button_frame, text="Toggle Quiet Hours",
command=toggle_quiet_hours)
toggle_button.pack(side="left", padx=5)

# Resolve Button
resolve_button = ttk.Button(button_frame, text="Problem Resolved",
command=resolve_problem)
resolve_button.pack(side="left", padx=5)
```

```
# --------------- Display Logic -----------------------
# -----------------------------------------------------


# Function for displaying message and changing picture on display
def update_display(message):
    timestamp = datetime.now().strftime('%H:%M:%S')     # Gets timestamp for 
message
    full_message = f"{timestamp} - {message}"           # Creates full message 
with timestamp
    output_box.insert(tk.END, full_message + "\n\n")    # Instert as the 
bottommost message in display box
    output_box.see(tk.END)                              # Scrolls to bottom of 
display box automatically

    # === Send SMS Alerts ===
    alert_keywords = ["ALERT", "Warning"]
    if any(keyword in message for keyword in alert_keywords):
       send_sms_via_email(PHONE_NUMBER, CARRIER_GATEWAY, message, EMAIL, 
EMAIL_PASSWORD)

    # === Image selection based on message content ===
    if "Living Room" in message:
        if "ALERT" in message or "Warning" in message:
            image_label.configure(image=H_Living_img)
            winsound.MessageBeep(winsound.MB_ICONHAND)
        else:
            image_label.configure(image=Living_img)

    elif "Bedroom" in message:
        if "ALERT" in message or "Warning" in message:
            image_label.configure(image=H_Bedroom_img)
            winsound.MessageBeep(winsound.MB_ICONHAND)
        else:
            image_label.configure(image=Bedroom_img)

    elif "Bathroom" in message:
        if "ALERT" in message or "Warning" in message:
            image_label.configure(image=H_Bath_img)
            winsound.MessageBeep(winsound.MB_ICONHAND)
        else:
            image_label.configure(image=Bath_img)

    elif "Kitchen" in message:
        if "ALERT" in message or "Warning" in message:
            image_label.configure(image=H_Kitchen_img)
            winsound.MessageBeep(winsound.MB_ICONHAND)
        else:
            image_label.configure(image=Kitchen_img)

    else:
        image_label.configure(image=Blank_img)


# ---------------- Logging Functions -----------------
# ----------------------------------------------------
def log_to_csv(data_row):
    with open(CSV_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(data_row)

def log_extra_data(batStatus, aveAccel, aveJerk, rotation):
    with open(EXTRA_CSV_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([datetime.now(), batStatus, aveAccel, aveJerk, rotation])

# --------------- Data Processing Logic ------------------
# ------------------------------------------------------

# A function that takes one line at a time from the serial data and processes it
def process_line(line):
    # Global variables to track the system over time
    global anchor_rssi, last_room, last_room_change_time
    global last_movement_time, fall_detected_time, fall_confirmed, was_wandering
    global last_hr_alert_time, fall_state, fall_pending_time,
last_wandering_alert

    print(f"RAW LINE: {line}")

    current_time = time.time()   # Current time stamp

    # Make sure it is anchor code
    anchor_match = re.search(r"\{Name: (\w+), RSSI: (-?\d+)\}", line)
    if anchor_match:
        raw_anchor, rssi = anchor_match.groups()
        room = room_name_map.get(raw_anchor.strip(), raw_anchor.strip())
        anchor_rssi[room] = int(rssi)
        # print(f"DEBUG: Received RSSI from {room}: {rssi}")
        log_to_csv([datetime.now(), "anchor", room, rssi, "", "", "", ""])
        # ------------- Wandering Logic -----------------
        # ---------------------------------------------
        if all(rssi <= -100 for rssi in anchor_rssi.values()):
            if not was_wandering and current_time - last_wandering_alert > 300:
                update_display(f"ALERT: Wearable is not detected on premises.
Last room detected in {last_room}")
                last_wandering_alert = current_time
                was_wandering = True
        else:
            if was_wandering:
                # Device rediscovered
                was_wandering = False
                recovered_room = max(anchor_rssi.items(), key=lambda x: x[1])[0]
                update_display(f"Wearable was rediscovered in {recovered_room}.")
                last_room_change_time = current_time
                last_room = recovered_room
                log_to_csv([datetime.now(), "anchor", recovered_room,
anchor_rssi[recovered_room], "", "", "Rediscovered", recovered_room])

        return

    # Make sure it is waveshare data
    wave_match = re.search(
        r"\{Name: Waveshare, Fall: (\d), HR: (-?\d+), Time: (\d+), batStatus:
(\d+), userMessage: (\d+), aveAccel: ([\d.]+), aveJerk: ([\d.]+), rotation:
([\d.]+)\}",
        line
    )
    if wave_match:
        fall, hr, _, bat, user_msg, aveAccel, aveJerk, rotation =
wave_match.groups()
        fall = int(fall)
        hr = int(hr)
        user_msg = int(user_msg)
        print(f"DEBUG: Fall={fall}, HR={hr}, user_msg={user_msg}")


        room = max(anchor_rssi.items(), key=lambda x: x[1])[0] if anchor_rssi
else "Unknown"  # Choose the Room with the largest rssi value


        log_to_csv([datetime.now(), "waveshare", "", "", fall, hr, user_msg,
room])
        log_extra_data(bat, aveAccel, aveJerk, rotation)
        # Room change detection
        if room != last_room:
            last_room = room
            last_room_change_time = current_time
            update_display(f"Location update: Wearable is in {room}")

        last_movement_time = current_time

        # ----------- Fall Logic --------------
        # -------------------------------------
        if user_msg == 100:
            fall_confirmed = True
            fall_state = "confirmed"
            update_display(f"FALL ALERT: SOS Button Pressed. Wearable is in
{room}!")

        if user_msg == 110:
            fall_confirmed = True
            fall_state = "confirmed"
            update_display(f"FALL ALERT: Fall detected and confirmed. Wearable is
in {room}!!")

        if fall == 1 and fall_state == "none":
            fall_state = "detected"
            fall_pending_time = current_time
            fall_detected_time = current_time
            print("DEBUG: Fall detected. Waiting for user confirmation...")

        if fall_state == "detected" and user_msg == 110:
            fall_state = "confirmed"
            fall_confirmed = True
            update_display(f"FALL ALERT: Fall detected and confirmed. Wearable is
in {room}!")

        # 5-minute warning if no confirmation
        if fall_state == "detected" and (current_time - fall_pending_time >= 5 *
60) and (current_time - fall_pending_time < 10 * 60):
            fall_state = "warning_sent"
            update_display(f"WARNING: Fall detected but not confirmed after 5
minutes. Wearable is in {room}!")

        # 10-minute escalation
        if fall_state in ["detected", "warning_sent"] and (current_time -
fall_pending_time >= 10 * 60):
```

```
            update_display(f"FALL ALERT: Fall detected but not confirmed after 10
minutes. Wearable is in {room}!")
            fall_state = "none"
            fall_confirmed = False
            fall_detected_time = None

        # Reset everything if fall value goes to 0
        if fall == 0 and fall_state == "confirmed":
            fall_state = "none"
            fall_confirmed = False
            fall_detected_time = None


        # ----------- Heart Rate Logic -------------
        # ------------------------------------------
        global last_hr_alert_time

        hr_alert_message = None
        if hr <= 0:
            hr_alert_message = (f"HEART RATE ALERT: Heart rate is zero or below!
Wearable is in {room}!")
        elif hr > 150:
            hr_alert_message = (f"HEART RATE ALERT: High heart rate ({hr} bpm)!
Wearable is in {room}!")

        if hr_alert_message:
            if time.time() - last_hr_alert_time >= 30:
                update_display(hr_alert_message)
                last_hr_alert_time = time.time()

        # ------------- Immobility Alert ----------------
        # ---------------------------------------------
        current_hour = datetime.now().hour
        is_quiet_hours_time = current_hour >= 21 or current_hour < 5
        is_quiet_hours = quiet_hours_manual or is_quiet_hours_time

        if is_quiet_hours:
            if quiet_hours_manual:
                quiet_label.config(text="Quiet Hours Active: Manually Enabled")
            else:
                quiet_label.config(text="Quiet Hours Active: Immobility alerts
are paused (9 PM - 5 AM)")
        else:
            quiet_label.config(text="")
        if not is_quiet_hours:
            if current_time - last_room_change_time >= 4 * 60 * 60:
                update_display(f"IMMOBILE ALERT: Person has been immobile in
{room} for 4 hours.")
                last_room_change_time = current_time  # reset so message doesn't
spam

# === Serial Reader Thread ===
def read_serial():
    try:
        ser = serial.Serial(port=SERIAL_PORT, baudrate=BAUD_RATE, timeout=1)
        while True:
            try:
                line = ser.readline().decode('utf-8').strip()
                if line:
                    process_line(line)
            except Exception as e:
                print(f"Error reading serial: {e}")
    except Exception as e:
        update_display(f"Serial connection failed: {e}")

# Start serial read in background
threading.Thread(target=read_serial, daemon=True).start()

# Launch the GUI
root.mainloop()
```

Anchor Code

This anchor code was developed and compiled using the Arduino IDE, targeting ESP32-WROOM devices acting as fixed-location Bluetooth anchors. The program scans for BLE advertisements from a specific wearable device (defined by MAC address) and extracts the RSSI value when the device is detected. The anchor then transmits this RSSI value to a central hub ESP32 using ESP-NOW, a lightweight, connectionless peer-to-peer protocol developed by Espressif.

To run this code, the following setup is required:

- Board Manager URL:
  https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
- Board Selection:
  In Arduino IDE, go to Tools → Board → ESP32 Arduino → Select "ESP32 Dev Module" (or your specific ESP32-WROOM variant).
- Required Libraries:

- WiFi.h (included with ESP32 core)
- esp_now.h (included with ESP32 core)
- BLEDevice.h from the ESP32 BLE Arduino library by Neil Kolban

This code leverages open-source libraries and APIs provided by Espressif and the ESP32 development community [13], [14]. The core logic is adapted from publicly available examples demonstrating BLE scanning and ESP-NOW communication, which are commonly used in indoor localization, proximity detection, and low-power wireless communication projects.

```
#include <WiFi.h>
#include <esp_now.h>
#include "BLEDevice.h"

// Target BLE MAC (Waveshare ESP32-S3)
const String targetDeviceAddress = "a0:85:e3:f0:9e:f1";
//Jenna's Wearable d8:3b:da:a0:6c:99
// Hub MAC Address (replace if needed)
uint8_t hubMAC[] = { 0xCC, 0xDB, 0xA7, 0x32, 0xFD, 0xF4 };
//Jenna's Hub Mac { 0x3C, 0x8A, 0x1F, 0xD3, 0x78, 0x30 };
BLEScan* pBLEScan;

// Only sending RSSI as a 4-byte integer
int rssiValue;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("ESP-NOW Send Status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Success" : "Fail");
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
   // Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);

  esp_now_peer_info_t peerInfo;
  memcpy(peerInfo.peer_addr, hubMAC, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    //Serial.println("Failed to add ESP-NOW peer");
    return;
  }

  BLEDevice::init("");
  pBLEScan = BLEDevice::getScan();
  pBLEScan->setActiveScan(true);

  //Serial.println("Anchor initialized. Scanning and sending RSSI...");
}

void loop() {
  BLEScanResults foundDevices = *pBLEScan->start(4);  // 2 second scan
  rssiValue = -100;  // Default

  for (int i = 0; i < foundDevices.getCount(); i++) {
    BLEAdvertisedDevice device = foundDevices.getDevice(i);
    if (device.getAddress().toString().equalsIgnoreCase(targetDeviceAddress)) {
      rssiValue = device.getRSSI();
      //Serial.print("Found Waveshare - RSSI: ");
      //Serial.println(rssiValue);
      break;
    }
  }

  pBLEScan->clearResults();

  if (esp_now_send(hubMAC, (uint8_t *)&rssiValue, sizeof(rssiValue)) == ESP_OK) {
   // Serial.println("Sent RSSI to hub");
  } else {
    //Serial.println("Failed to send RSSI");
  }

  delay(1000);  // Send every 3 seconds scan + delay
}
```

Hub Code

This hub code was developed and uploaded using the Arduino IDE to an ESP32-WROOM device, which functions as the central receiver in the system. It listens for ESP-NOW messages from fixed anchor nodes as well as the wearable device. When receiving data from an anchor, it parses the signal strength (RSSI) and identifies the source location using the anchor's MAC address. When receiving data from the wearable, the hub decodes a structured message containing fall detection certainty, heart rate, battery level, and motion metrics. These data packets are printed to the serial monitor in a structured format for easy logging and further processing by the connected PC.
Tools & Configuration:

- Arduino Board Manager URL:
  https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
- Board Selection:
  Tools → Board → ESP32 Arduino → "ESP32 Dev Module"
- Required Libraries:
  - WiFi.h and esp_now.h — included with the official ESP32 Arduino core
  - esp_wifi.h — for MAC retrieval and Wi-Fi power settings (also part of ESP32 core)

This code utilizes ESP-NOW to achieve low-latency, peer-to-peer communication between ESP32 microcontrollers. MAC addresses for all anchors are predefined, and a lookup table (array of Anchor structs) is used to associate MACs with named room locations. The implementation structure and use of esp_now_register_recv_cb() follow Espressif's open documentation and sample code [13].

```
#include <WiFi.h>
#include <esp_now.h>
#include <esp_wifi.h>
#include <map>

// ===== STRUCTS =====
int receivedRSSI;

typedef struct wearableDataOut {
  int16_t HR;
  uint8_t FallCertainty;
  uint32_t Time;
  uint32_t batStatus;
  uint8_t userMessage;
  float aveAccel;
  float aveJerk;
  float rotation;
} OutMessage_t;

OutMessage_t latestWearableData;
```

```
struct Anchor {
  const char* position;
  uint8_t mac[6];
};

Anchor anchors[] = {
  { "JHRoom",       { 0x3C, 0x8A, 0x1F, 0xD3, 0x42, 0x04 } },
  { "pkDiningRoom",    { 0xF0, 0x9E, 0x9E, 0x95, 0xDA, 0xFC } },
  { "pkBathroom",      { 0xCC, 0xDB, 0xA7, 0x3E, 0xE1, 0x04 } },
  { "pkLiving Room",   { 0xCC, 0xDB, 0xA7, 0x3E, 0x05, 0x04 } },
  { "pkKitchen",       { 0x3C, 0x8A, 0x1F, 0xA8, 0xD1, 0x0C } },
  { "pkOffice",        { 0x08, 0xF9, 0xE0, 0x1F, 0xC2, 0xB8 } },
};

String getPosition(const uint8_t *mac_addr) {
  for (Anchor anchor : anchors) {
    if (memcmp(mac_addr, anchor.mac, 6) == 0) {
      return String(anchor.position);
    }
  }
  return "Unknown";
}

// ===== BUFFERS =====
//std::map<String, int> latestAnchorRSSI;
//std::map<String, unsigned long> lastAnchorTimestamp;

// ===== CALLBACK =====
void onDataReceived(const esp_now_recv_info_t *info, const uint8_t *data, int
data_len) {
  if (data_len == sizeof(OutMessage_t)) {
    memcpy(&latestWearableData, data, sizeof(OutMessage_t));
    Serial.print("{Name: Waveshare");
    Serial.print(", Fall: ");
    Serial.print(latestWearableData.FallCertainty);
    Serial.print(", HR: ");
    Serial.print(latestWearableData.HR);
    Serial.print(", Time: ");
    Serial.print(latestWearableData.Time);
    Serial.print(", batStatus: ");
    Serial.print(latestWearableData.batStatus);
    Serial.print(", userMessage: ");
    Serial.print(latestWearableData.userMessage);
    Serial.print(", aveAccel: ");
    Serial.print(latestWearableData.aveAccel, 2);
```

```
      Serial.print(", aveJerk: ");
      Serial.print(latestWearableData.aveJerk, 2);
      Serial.print(", rotation: ");
      Serial.print(latestWearableData.rotation, 2);
      Serial.println("}");
    } else if (data_len == sizeof(int)) {
      int rssiVal;
      memcpy(&rssiVal, data, sizeof(int));
      String position = getPosition(info->src_addr);
      //latestAnchorRSSI[position] = rssiVal;
      //lastAnchorTimestamp[position] = millis();

      Serial.print("{Name: ");
      Serial.print(position);
      Serial.print(", RSSI: ");
      Serial.print(rssiVal);
      Serial.println("}");
    }
  }
// ===== SETUP =====
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  esp_wifi_set_ps(WIFI_PS_NONE);  // keeps Wi-Fi radio fully awake

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_recv_cb(onDataReceived);
  uint8_t mac[6];
  esp_wifi_get_mac(WIFI_IF_STA, mac);
  Serial.printf("Hub MAC address: %02X:%02X:%02X:%02X:%02X:%02X\n",
                mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
  Serial.println("ESP-NOW Hub Initialized.");
}
// ===== LOOP =====
void loop() {
  // Nothing needed — printing happens immediately in callback
}
```

<u>Triangulation Code</u>

This Python script was developed for post-processing hub RSSI logs and estimating the wearable's location through triangulation. The script reads RSSI data from anchor nodes, averages them over 30-second intervals, and converts signal strength to estimated distances using

a log-distance path loss model. It then applies nonlinear optimization to compute the most likely (x, y) position of the wearable device, and plots these positions over an approximate room layout.
This code can be run in any Python 3.x environment (e.g., Anaconda, VS Code, or Jupyter Notebook), with the following libraries installed:
numpy — for vector math and numerical operations
matplotlib — for plotting spatial heatmaps
scipy — specifically scipy.optimize.minimize for nonlinear optimization
pandas — for organizing timestamped RSSI data
re, datetime — for parsing and organizing hub log files
You can install required libraries via pip: pip install numpy pandas matplotlib scipy
Functionality Overview:
RSSI Parsing: The log file is parsed using a regular expression that extracts timestamp, anchor ID, and RSSI value.

RSSI to Distance Conversion: RSSI is converted to estimated distance using:
d = 10^((txPower − RSSI) / (10 × n))
where txPower is the RSSI at 1 meter (typically -59 dBm), and n is the path-loss exponent, assumed to be 2.0 for indoor environments [17].

Triangulation: Using 3 or more anchor distances, the script minimizes the error between estimated and actual distances via the L-BFGS-B optimization algorithm [18] to output the most probable position estimate.

Plotting: The resulting estimated positions are plotted over a simplified floor plan, alongside anchor markers and the known true wearable location.

This script uses open-source scientific computing libraries and widely known RSSI localization models adapted from public examples and BLE localization literature [19].

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import re
from datetime import datetime
import pandas as pd

# === Anchor positions (in feet)
anchor_positions = {
    "pkLiving Room": (2, 0.5),
    "pkBedroom": (1, 31),
    "pkBathroom": (15.5, 24.5),
    "pkKitchen": (33.5, 5),
}

# === True wearable position for this trial
true_wearable_pos = (15.5, 26.5)  # <-- Change for Bathroom
```

```
#true_wearable_pos = (5, 3)  # <-- Change for Living Room
#true_wearable_pos = (5.5, 24)  # <-- Change for Bedroom
#true_wearable_pos = (25, 10)  # <-- Change for Kitchen

# === RSSI to estimated distance
def rssi_to_distance(rssi, tx_power=-59, n=2.0):
    return 10 ** ((tx_power - rssi) / (10 * n))

# === Triangulation optimization
def triangulate(rssi_dict):
    distances = {a: rssi_to_distance(rssi) for a, rssi in rssi_dict.items()}

    def objective(pos):
        x, y = pos
        return sum((np.linalg.norm([x - anchor_positions[a][0], y - anchor_positions[a][1]]) - d) **
2
                   for a, d in distances.items())

    result = minimize(objective, x0=(10, 10), method='L-BFGS-B')
    return result.x

# === Parse your hub .txt file
def parse_log_file(filepath):
    pattern = re.compile(r"(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}), {Name: ([^,]+), RSSI: (-
?\d+)}")
    rows = []

    with open(filepath, "r") as file:
        for line in file:
            match = pattern.search(line)
            if match:
                timestamp = datetime.strptime(match.group(1), "%Y-%m-%d %H:%M:%S")
                anchor = match.group(2).strip()
                rssi = int(match.group(3))
                if anchor in anchor_positions:
                    rows.append((timestamp, anchor, rssi))

    df = pd.DataFrame(rows, columns=["timestamp", "anchor", "rssi"])
    return df

# === Main function
def main():
    log_file = "AnchorTest_SittingBathroom10min.txt"  # 👈 Update this as needed
    df = parse_log_file(log_file)

    df["timestamp"] = pd.to_datetime(df["timestamp"])
    df.set_index("timestamp", inplace=True)

    rssi_windows = df.groupby("anchor").resample("30s")["rssi"].mean().unstack(level=0)

    positions = []
    for time_idx, row in rssi_windows.iterrows():
        rssi_snapshot = row.dropna().to_dict()
        if len(rssi_snapshot) >= 3:
            pos = triangulate(rssi_snapshot)
            positions.append(pos)

    # === Plotting
    plt.figure(figsize=(10, 7))

    # Draw walls (approximate layout)
    walls = [
    ((0, 0), 35, 32, "#D0D0D0"),      # Outer wall (light gray)
    ((0, 20), 12, 12, "#87CEFA"),    # Bedroom (sky blue), now meets living room
    ((12, 24), 8, 8, "#90EE90"),     # Bathroom (light green)
    ((0, 0), 20, 20, "#FFB6C1"),     # Living Room (light pink)
    ((20, 0), 15, 15, "#FFD700"),    # Kitchen (gold), now extends to y = 15
]
    for (x, y), w, h, color in walls:
        plt.gca().add_patch(
            plt.Rectangle(
                (x, y), w, h,
                edgecolor='black',
                facecolor=color,
                linewidth=2.0,
                alpha=0.35
        )
    )

    # Plot anchors (legend-friendly)
    anchor_label_plotted = False
    for anchor, (x, y) in anchor_positions.items():
        if not anchor_label_plotted:
            plt.scatter(x, y, c='blue', marker='^', s=100, label='Anchor')
            anchor_label_plotted = True
        else:
            plt.scatter(x, y, c='blue', marker='^', s=100)

    # Plot estimated wearable positions
    if positions:
        positions = np.array(positions)
        plt.scatter(positions[:, 0], positions[:, 1], c='red', alpha=0.6, label='Estimated Position')
    # Plot true position
    plt.scatter(*true_wearable_pos, c='green', marker='x', s=150, linewidths=2, label='True
Position')

    plt.title('Heatmap of Triangulated Wearable Position (30s Averages)')
    plt.xlabel('X (feet)')
    plt.ylabel('Y (feet)')
    plt.grid(True)
    plt.legend()
    plt.axis("equal")
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

Wearable Code Description

This code is compiled in Arduino using the esp32 board packages which includes the waveshare 1.69in touch screen dev board. SPIFFS must be adjusted to fit this program.

Libraries used:

xioTechnologies/Fusion – Madgwick filter commit(082ef7b)

    File structure adjusted to compile for Arduino

sparkfun/SparkFun_MAX3010x_Sensor_Library commit(72d5308)

    Functions added to support FIFO reading and interrupt capability for the ESP32S3

lvgl/lvgl

    library used as is (9.2.2)

lewisxhe/SensorLib commit(f0102d4…103476e)

    library updated in place to support interrupt based reading of the QMI8658C IMU sensor data

WearableMainV1_RawCompile.ino

```
#include "sdkconfig.h"
#include "esp_system.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/timers.h"
```

```
#include "freertos/event_groups.h"

#include <Arduino.h>




//RF Includes
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <esp_coexist.h>
#include <esp_now.h>
#include <WiFi.h>
#include "time.h"




//my includes

#include "WearableTaskHandles.h"
#include "WearableQmiFifoRead.h"
#include "WearableESPnow.h"
#include "WearableBLE.h"
#include "WearableFallDetection.h"
#include "WearableMaxFifoRead.h"
#include "WearableHeartRate.h"
#include "WearableTaskMonitor.h"
#include "WearableUI.h"



//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"
// fix this define here:
HWCDC USBSerial;

SensorQMI8658 qmi;

/*=======================================
TODO LIST
  time setting
  ESP Now sending/receiving           SENDING DONE
  Accel/gyro get data buffer          DONE
  Fall detection                      DONE
  BLE broadcasting                    DONE
  Get HR data  GPIO15 SDA GPI016 SCL    %100
  calculate HR                        DONE
  Display time, get alert from user
*/

// Define the global array of task handles
```

```
TaskHandle_t TaskHandles[TASK_COUNT] = {NULL};

//global debug Mutex, this is needed to keep print statements from being interrupted
SemaphoreHandle_t serialMutex;
portMUX_TYPE serialSpinlock = portMUX_INITIALIZER_UNLOCKED;//ESP32 spinlock object.

//regular queues which need to hold multiple values
QueueHandle_t IMU_TO_FALLDETECT = NULL;
QueueHandle_t MAXFIFO_TO_HR = NULL;

//mailbox queues which hold one value only.
QueueHandle_t MailBox[MailboxCount] = {NULL};

void setup() {
 // put your setup code here, to run once:
 serialMutex = xSemaphoreCreateMutex();  // Create the mutex
 // Initialize Serial Monitor
 USBSerial.setTxBufferSize(1024);  // Increase buffer size
 DBEGIN(115200);

 DPRINT("serial init done\n");
 //Create Queues
 IMU_TO_FALLDETECT = xQueueCreate(12, sizeof(IMUCombined_t));
 if (IMU_TO_FALLDETECT == NULL) {
   DPRINT("Error: IMU_TO_FALLDETECT creation failed!\n");
   while (1);
 }
 MAXFIFO_TO_HR = xQueueCreate(40, sizeof(uint32_t));
 if (MAXFIFO_TO_HR == NULL) {
   DPRINT("Error: MAXFIFO_TO_HR creation failed!\n");
   while (1);
 }

 //create Mailboxes
 for(int i=0; i<MailboxCount;i++){
  switch(i){
   case Mailidx_FALL_TO_ESPNOW:
    MailBox[i] = xQueueCreate(1, sizeof(MailFall_t));
    break;
   case Mailidx_HR_TO_ESPNOW:
    MailBox[i] = xQueueCreate(1, sizeof(MailHR_t));
    break;
   case Mailidx_HR_TO_UI:
    MailBox[i] = xQueueCreate(1, sizeof(MailHR_t));
    break;
   case Mailidx_FALL_TO_UI:
    MailBox[i] = xQueueCreate(1, sizeof(MailFall_t));
    break;
   case Mailidx_UI_TO_ESPNOW_periodic:
    MailBox[i] = xQueueCreate(1, sizeof(MailUI_periodic_t));
    break;
   case Mailidx_ESPNOW_TO_UI:
    MailBox[i] = xQueueCreate(1, sizeof(int32_t));
```

```
      break;
     case Mailidx_UI_TO_ESPNOW_user:
      MailBox[i] = xQueueCreate(1, sizeof(MailUI_User_t));
      break;
    }
    if (MailBox[i] == NULL) {
     DPRINT("Error: mailbox %d creation failed!\n",i);
     while (1);
    }

  }



  // Create Send Data Task
  xTaskCreate(
    BLEtask,            // Task function
    "BLE Task",         // Name of the task
    4096,               // Stack size
    NULL,               // Parameters
    1,                  // Priority
    &TaskHandles[BLETask_INDEX]   // Task handle
  );


  startUITask(&TaskHandles[QMIfifoTask_INDEX]);
  QMIFifosetup(&TaskHandles[QMIfifoTask_INDEX],IMU_TO_FALLDETECT);
  MaxFIFOReadSetup(&TaskHandles[MAXfifoTask_INDEX],MAXFIFO_TO_HR);
  startFallDetectionTask(&TaskHandles[Falldetection_INDEX],IMU_TO_FALLDETECT);
  startESP_NowTask(&TaskHandles[espNowTask_INDEX]);
  startHeartRateTask(&TaskHandles[HeartRateTask_INDEX],MAXFIFO_TO_HR);
  StartMonitoringTask(&TaskHandles[HeartRateTask_INDEX]);
}

void loop() {
  // put your main code here, to run repeatedly:
  vTaskDelete(NULL);//permanently delete loop task.
}
```

WearableTaskHandles.h

```
#include <Arduino.h>

#ifndef TASK_HANDLES_H
#define TASK_HANDLES_H

#endif

// FreeRTOS Task Handles
enum {
    BLETask_INDEX,
    espNowTask_INDEX,
    QMIfifoTask_INDEX,
    Falldetection_INDEX,
    MAXfifoTask_INDEX,
    HeartRateTask_INDEX,
    UITask_INDEX,
    TASK_COUNT
};


enum {
  Mailidx_FALL_TO_ESPNOW,
  Mailidx_HR_TO_ESPNOW,
  Mailidx_UI_TO_ESPNOW_periodic,
  Mailidx_HR_TO_UI,
  Mailidx_FALL_TO_UI,
  Mailidx_ESPNOW_TO_UI,
  Mailidx_UI_TO_ESPNOW_user,
  MailboxCount
};

typedef struct {
    float aveAccel;
    float aveJerk;
    float rotation;
    uint8_t fallyn;
} MailFall_t;

typedef struct {
    uint32_t BatteryLevel;
} MailUI_periodic_t;


typedef struct {
    uint8_t UserMessage;
} MailUI_User_t;

typedef struct {
    int16_t HRValue;
    uint8_t HRValid;
} MailHR_t;



extern TaskHandle_t TaskHandles[TASK_COUNT];
extern QueueHandle_t MailBox[MailboxCount];
```

WearableQmiFifoRead.h

```
#pragma once
#include <Arduino.h>
#include <Wire.h>
#include "SensorQMI8658.hpp"

//#include "pin_config.h"
```

```
#include <HWCDC.h>

//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"

#define USE_WIRE
#define QMI_SDA  11
#define QMI_SCL  10
#define QMI_SENSOR_IRQ  38  // Pin 38 as interrupt source

extern SensorQMI8658 qmi;  // Declare qmi (defined in main.ino)

typedef struct {
    float Ax;
    float Ay;
    float Az;
    float Gx;
    float Gy;
    float Gz;
} IMUCombined_t;

void QMIFifosetup(TaskHandle_t* fifoTaskHandle, QueueHandle_t QueueToFallDetect);
```

WearableQmiFifoRead.cpp

```
#include "WearableQmiFifoRead.h"
#include "WearableTaskHandles.h"

SemaphoreHandle_t imuSemaphore;

// Interrupt Service Routine (ISR)
void IRAM_ATTR imuISR() {
    detachInterrupt(QMI_SENSOR_IRQ);  // Prevent retriggering
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR(imuSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

// FreeRTOS Task
void _QMIFifoTask(void *pvParameters) {
    QueueHandle_t queue = (QueueHandle_t)pvParameters;
    uint8_t data[128];
    IMUdata acc[11];
    IMUdata gyr[11];
    IMUCombined_t buffer[11];
    uint32_t tnow, tlast = 0;
    DPRINT("QMI_FIFO: entered freeRTOS task\n");
    while (true) {
        if (xSemaphoreTake(imuSemaphore, portMAX_DELAY) == pdTRUE) {
            tnow = micros();
            uint16_t nsamples = qmi.readFromFifo_New(acc, gyr, data);
            DPRINT("QMI_FIFO: ReceivedSamples TimeSinceLastRead:%dus 
PracticalFreq:%0.2f\n",tnow-tlast,((float)1/(float)(tnow-
tlast))*1e6*(float)nsamples);
```

```
            tlast = tnow;
            for (int i=0; i<nsamples;i++){
                buffer[i].Ax=acc[i].x;
                buffer[i].Ay=acc[i].y;
                buffer[i].Az=acc[i].z;
                buffer[i].Gx=gyr[i].x;
                buffer[i].Gy=gyr[i].y;
                buffer[i].Gz=gyr[i].z;
                if (xQueueSend(queue, &buffer[i], 0) != pdTRUE) {
                  DPRINT("QMI_FIFO: Queue full! Data lost.\n");
                }
            }


            vTaskDelay(pdMS_TO_TICKS(10)); // Prevent over-polling
            attachInterrupt(QMI_SENSOR_IRQ, imuISR, RISING);
        }
    }
}

void QMIFifosetup(TaskHandle_t* fifoTaskHandle, QueueHandle_t QueueToFallDetect) {
    pinMode(QMI_SENSOR_IRQ, INPUT);

    DPRINT("QMI_FIFO: IRQ digital input setup done\n");

    if (!qmi.begin(Wire, QMI8658_L_SLAVE_ADDRESS, QMI_SDA, QMI_SCL)) {
        DPRINT("QMI_FIFO: Failed to find QMI8658 - check your wiring!\n");
        while (1) { delay(1000); }
    }

    DPRINT("QMI_FIFO: Device ID:");
    DPRINT("%x\n",qmi.getChipID());
    qmi.configAccelerometer(
      SensorQMI8658::ACC_RANGE_4G,
      SensorQMI8658::ACC_ODR_62_5Hz,//matched to GYRO ODR. no touchy
      SensorQMI8658::LPF_MODE_3,
      true);

    qmi.configGyroscope(
      SensorQMI8658::GYR_RANGE_512DPS,
      SensorQMI8658::GYR_ODR_56_05Hz,//matched to ACC ODR. no touchy
      SensorQMI8658::LPF_MODE_3,
      true);

    qmi.configFIFO(
      SensorQMI8658::FIFO_MODE_FIFO,
      SensorQMI8658::FIFO_SAMPLES_16,
      SensorQMI8658::IntPin1,
      10);// watermark level.  I2C buffer is only 128bytes long, each sample is
12bytes. increase past 10 at your own risk.
    qmi.enableGyroscope();
    qmi.enableAccelerometer();

    qmi.dumpCtrlRegister();
    delay(1000);
    DPRINT("QMI_FIFO: ConfigDone\n");


    imuSemaphore = xSemaphoreCreateBinary();
    if (imuSemaphore != NULL) {
      xTaskCreate(
            _QMIFifoTask,          // Task function
            "QMI FIFO TASK",       // Name of the task
            4096,                  // Stack size
            QueueToFallDetect,     // Parameters
            2,                     // Priority
            fifoTaskHandle        // Task handle
      );
      delay(100);

    }else{DPRINT("QMI_FIFO: IMU semaphore null\n");}
    qmi.resetFifo_New();
    xSemaphoreGive(imuSemaphore);
    attachInterrupt(QMI_SENSOR_IRQ, imuISR, RISING);
}
```

WearableESPnow.h

```
#pragma once
#include <Arduino.h>
#include <HWCDC.h>
#include <esp_now.h>
#include <esp_wifi.h>
#include <WiFi.h>

//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"

typedef struct wearableDataOut {
    int16_t HR;
    uint8_t FallCertainty;
    uint32_t Time;
    uint32_t batStatus;
    uint8_t userMessage;
    float aveAccel;
    float aveJerk;
    float rotation;
} OutMessage_t;


typedef struct wearableDataIn {
    uint32_t Time;
    uint8_t userMessage;
} InMessage_t;

//define hub address
#define HUB_ADDRESS {0x70, 0xB8, 0xF6, 0x97, 0xBA, 0xB8}

#define SEND_DELAY 1000
```

```
void startESP_NowTask(TaskHandle_t* TaskHandle);
```

WearableESPnow.cpp

```
#include "WearableESPnow.h"
#include "WearableTaskHandles.h"

static InMessage_t myData;

//private functions


void _OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  DPRINT("\nespnow: Last Packet Send Status:\t");
  DPRINT(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success\n" : "Delivery Fail\n");
  if (status == ESP_NOW_SEND_SUCCESS){
    int32_t stat=1;
    xQueueOverwrite(MailBox[Mailidx_ESPNOW_TO_UI],&stat);
  }else{
    int32_t stat=0;
    xQueueOverwrite(MailBox[Mailidx_ESPNOW_TO_UI],&stat);
  }

}

void _OnDataRecv(const esp_now_recv_info_t* info, const uint8_t* data, int len){
  const uint8_t* mac = info->src_addr;
  //copy in incomming data to myData
  memcpy(&myData, data, sizeof(myData));
  DPRINT("DataReceived\n");

  //TODO Set time
}

bool _espnow_Create(){
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  esp_wifi_set_protocol( WIFI_IF_STA , WIFI_PROTOCOL_LR);
  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    DPRINT("Error initializing ESP-NOW\n");
    return false;
  }
  else{
    esp_now_register_send_cb(_OnDataSent);
    esp_now_register_recv_cb(_OnDataRecv);
    return true;
    }
}

// Task to send/receive data over espnow
void espnowTask(void* parameter) {

  //Setup
  DPRINT("espnow: setting up\n");
  OutMessage_t DataOut;
  esp_now_peer_info_t peerInfo;

  _espnow_Create();
  uint8_t broadcastAddress[] = HUB_ADDRESS;
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);  // Copy MAC address
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  peerInfo.ifidx = WIFI_IF_STA;
  while (esp_now_add_peer(&peerInfo) != ESP_OK){
    DPRINT("espnow: Failed to add peer\n");
    vTaskDelay(SEND_DELAY*3);
    DPRINT("espnow: Retrying Peer Add\n");
  }
  MailFall_t FallMail;
  MailHR_t HR;
  MailUI_periodic_t UIperiodicMail;
  MailUI_User_t UIUserMail;
  while(true){
    //Loop area

    //Get data from fall detection,
    xQueueReceive(MailBox[Mailidx_FALL_TO_ESPNOW], &FallMail, 0);

    //Get data from HR,
    xQueueReceive(MailBox[Mailidx_HR_TO_ESPNOW], &HR, 0);

    //Get user data---- must default to 0 if it is not sent
    if(xQueueReceive(MailBox[Mailidx_UI_TO_ESPNOW_user], &UIUserMail, 0) == pdTRUE ){

    }else{
      UIUserMail.UserMessage=0;
    }

    //get battery data (periodic)
    xQueueReceive(MailBox[Mailidx_UI_TO_ESPNOW_periodic], &UIperiodicMail, 0);

    //GET DATA
    DataOut.HR = (HR.HRValid==1) ? HR.HRValue : -1;
    DataOut.FallCertainty=FallMail.fallyn;
    DataOut.Time=millis();
    DataOut.batStatus=UIperiodicMail.BatteryLevel;
    DataOut.userMessage=UIUserMail.UserMessage;

    DataOut.aveAccel=FallMail.aveAccel;
    DataOut.aveJerk=FallMail.aveJerk;
    DataOut.rotation=FallMail.rotation;
    DPRINT("espnow: DATA BEING SENT ESPNOW:\n");
    DPRINT("espnow: HR\t\t%d\n",DataOut.HR);
    DPRINT("espnow: FallCertainty\t%d\n",DataOut.FallCertainty);
    DPRINT("espnow: Time\t\t%lu\n",DataOut.Time);
    DPRINT("espnow: batStatus\t%d\n",DataOut.batStatus);
    DPRINT("espnow: userMessage\t%d\n",DataOut.userMessage);
    DPRINT("espnow: movdata\tacc:%0.2f jerk:%0.2f
rot:%0.2f\n",DataOut.aveAccel,DataOut.aveJerk,DataOut.rotation);
```

```
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &DataOut,
sizeof(DataOut));
    if (result == ESP_OK) {
      DPRINT("espnow: Sent with success\n");
    }
    else {
      DPRINT("espnow: Error sending the data\n");
    }
    DPRINT("\n\n");
    vTaskDelay(SEND_DELAY);
  }

}

void startESP_NowTask(TaskHandle_t* TaskHandle){
    // Create Send Data Task
  xTaskCreatePinnedToCore(
    espnowTask,             // Task function
    "esp-now Task",         // Name of the task
    4096,                   // Stack size
    (void*)NULL,            // Parameters
    2,                      // Priority
    TaskHandle,             // Task handle
    0                       // PinTask to core 0
  );
}
```

<u>WearableBLE.h</u>

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <esp_coexist.h>
#include <HWCDC.h>

//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS

#include "WearableDebug.h"
#define BLE_ADVERTISE_TIME 100
#define BLE_SLEEP_TIME 2000



void BLEtask(void* parameter);
```

<u>WearableBLE.cpp</u>

```
#include "WearableBLE.h"
#include "WearableTaskHandles.h"

//Disconnect and start advertising anytime someone acts up.
class NoConnectCallback : public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
        pServer->disconnect(0); // Reject connection immediately
        DPRINT("ble: Someone Tried to connect\n");
    }
    void onDisconnect(BLEServer* pServer) {
        pServer->getAdvertising()->start(); // Ensure advertising restarts
        DPRINT("ble: Restarting advertising after connection\n");
    }
};

void BLEtask(void* parameter) {


    // BLE Configuration
    BLEServer* pServer = NULL;
    BLEAdvertising* pAdvertising = NULL;
    // Initialize Bluetooth
    // Initialize BLE
    BLEDevice::init("WearableDevice");
    pServer = BLEDevice::createServer();
    pServer->setCallbacks(new NoConnectCallback());

    //ask RF monitor to prefer coexistence.
    esp_coex_preference_set(ESP_COEX_PREFER_BALANCE);

    // Configure advertising
    pAdvertising = pServer->getAdvertising();
    pAdvertising->addServiceUUID("12345678-1234-1234-1234-1234567890ab"); // Example
UUID
    pAdvertising->setScanResponse(false); // No scan response needed
    pAdvertising->setMinPreferred(0x00);  // Disable connection preferences
    pAdvertising->setMaxPreferred(0x00);  // Disable connection preferences

    DPRINT("ble: init done\n");
    while (1){
        TickType_t xLastWakeTime;
        xLastWakeTime = xTaskGetTickCount();
        const TickType_t xFrequency = pdMS_TO_TICKS(BLE_SLEEP_TIME); // 1-second
interval
        // Start advertising
        pAdvertising->start();
        DPRINT("ble: advertising started\n");
        vTaskDelay(BLE_ADVERTISE_TIME);
        pAdvertising->stop();
        DPRINT("ble: advertising Stopped\n");
        vTaskDelayUntil(&xLastWakeTime, xFrequency);
    }



    }
```

WearableFallDetection.h

```
#pragma once
//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include <math.h>
```

```
#include <Fusion.h>
#include "WearableDebug.h"

typedef struct{
    float aveAccel;
    float aveJerk;
    float rotation;

}WindowInfo_t;

void startFallDetectionTask(TaskHandle_t* TaskHandle,QueueHandle_t Queue);
```

WearableFallDetection.cpp

```
#include "WearableFallDetection.h"
#include "WearableQmiFifoRead.h"  // Include here for the struct definition
#include "WearableTaskHandles.h"

#define STORAGE_SIZE 128 //defining number of accel/gyro samples to hold in memory
#define WINDOW_WIDTH 60
#define N_HISTORIC_WINDOWS 5 //define the number of window historic data is kept.

#define SAMPLE_RATE (56.05f)

#define CONSTANTS {78.48903426f, 162.72863795f, 0.68059264f, 60.86678552, -
62.10738956, 61.22120614}//result of optimization

// Initialise algorithms
FusionOffset offset;
FusionAhrs ahrs;

const FusionAhrsSettings settings = {
        .convention = FusionConventionNwu,
        .gain = 0.5f,
        .gyroscopeRange = 512.0f, /* replace this with actual gyroscope range in
degrees/s */
        .accelerationRejection = 10.0f,
        .magneticRejection = 10.0f,
        .recoveryTriggerPeriod = (uint16_t)(5 * SAMPLE_RATE), /* 5 seconds */
};


//window info:
//float aveAccel;
//float aveJerk;
//float rotation;

void CreateWindow(IMUCombined_t* storage ,uint16_t idx,WindowInfo_t* WindowInfo){
    float accSum=0;
    float jerkSum=0;
    float thetaX=0;
    float thetaY=0;
    float thetaZ=0;
    float accLast=0;
    for(int16_t i=idx-1; i>=0; i--){
      FusionVector gyroscope =      {storage[i].Gx,storage[i].Gy,storage[i].Gz}; //
replace this with actual gyroscope data in degrees/s
      FusionVector accelerometer =  {storage[i].Ax,storage[i].Ay,storage[i].Az};
      gyroscope = FusionOffsetUpdate(&offset, gyroscope);

      float deltaTime = (float)1/SAMPLE_RATE;
      //do the thing
      FusionAhrsUpdateNoMagnetometer(&ahrs, gyroscope, accelerometer, deltaTime);
      const FusionVector earth = FusionAhrsGetEarthAcceleration(&ahrs);

      //sum rotation * dt --- integration
      thetaX += deltaTime * storage[i].Gx;
      thetaY += deltaTime * storage[i].Gy;
      thetaZ += deltaTime * storage[i].Gz;


      //Compute Earth frame acceleration Magitude
      const float
accnow=sqrt(pow(earth.axis.x,2)+pow(earth.axis.y,2)+pow(earth.axis.z,2));
      accSum+=accnow;

      //Compute Derivative
      if (i==idx-1){
        accLast=accnow;//save first accel
        continue;
        }//dropping the first

      jerkSum+=(accnow-accLast)/deltaTime;
      accLast=accnow;
    }
    //estimate total rotation during frame
    WindowInfo->rotation=sqrt(pow(thetaX,2)+pow(thetaY,2)+pow(thetaZ,2));

    //compute average accel by dividing sum by N points
    WindowInfo->aveAccel=accSum/idx;

    //compute average jerk by dividing sum by N points
    WindowInfo->aveJerk=jerkSum/idx;
    return ;
}
bool detectFall(WindowInfo_t* Windows){
  const float constants[]=CONSTANTS;
  /*in words
  if statement 1:
    if jerk goes outside of +/- bounds, AND rotation is above a number ITS A FALL.
  if statment 2:
    if acceleration peaks At the same time rotation is above a certain threshold ITS
A FALL.
  */

  if ((Windows[0].aveJerk > constants[0] || Windows[0].aveJerk > constants[4]) &&
Windows[0].rotation > constants[1])
  {return true;}
  if (Windows[1].aveAccel > constants[2] && Windows[1].rotation > constants[3] &&
Windows[0].aveAccel < Windows[1].aveAccel && Windows[2].aveAccel <
Windows[1].aveAccel)
  {return true;}

  return false;
}

void ShiftWindows(WindowInfo_t* Windows){
    //shift windows so window 0-3 is now 1-4.
    //old data in window[4] after doing this is gone (Stale data)
    //window[0] will be a duplicate of window[1], everything overwritten by create
window
    memmove(&Windows[1], &Windows[0], (N_HISTORIC_WINDOWS - 1) *
sizeof(WindowInfo_t));
}

void FallDetectTask(void *parameter) {
    QueueHandle_t imuQ =(QueueHandle_t)parameter;
    if (imuQ == NULL) {
        DPRINT("Fall Detect: Error Queue handle is NULL!\n");
        vTaskDelete(NULL);  // Terminate the task if queue is invalid
    }
    IMUCombined_t buffer;
    IMUCombined_t dataStorage[STORAGE_SIZE];
    WindowInfo_t WindowData[N_HISTORIC_WINDOWS];
    uint16_t index = 0;
    UBaseType_t itemCount;
    FusionAhrsSetSettings(&ahrs, &settings);
    FusionOffsetInitialise(&offset, SAMPLE_RATE);
    FusionAhrsInitialise(&ahrs);
    DPRINT("Fall Detect: Entering task\n");
    while (1) {
      if (xQueueReceive(imuQ, &buffer, portMAX_DELAY) == pdTRUE) {
        itemCount = uxQueueMessagesWaiting(imuQ);
        dataStorage[index] = buffer;
        for (int i = 0; i < itemCount; i++) {
            if (xQueueReceive(imuQ, &buffer, 0)==pdTRUE){
              dataStorage[index] = buffer;
              index++;
              //DPRINT("Fall Detect: Ax:%0.3f Ay:%0.3f Az:%0.3f Gx:%0.3f Gy:%0.3f
Gz:%0.3f\n",dataStorage[index].Ax,dataStorage[index].Ay,dataStorage[index].Az,dataSto
rage[index].Gx,dataStorage[index].Gy,dataStorage[index].Gz);
            }
        }

        if (index >= WINDOW_WIDTH){
            //DPRINT("Fall Detect: getting window info\n");

            ShiftWindows(WindowData);
            CreateWindow(dataStorage,index,&WindowData[0]);
            index-=WINDOW_WIDTH;//pull back the index by the window width to keep the
buffer from filling.
            MailFall_t outdata;
            outdata.aveAccel=WindowData[0].aveAccel;
            outdata.aveJerk=WindowData[0].aveJerk;
            outdata.rotation=WindowData[0].rotation;
            if (detectFall(WindowData)){
              DPRINT("\n\nFall Detect: FALL DETECTED\n");
              DPRINT("Fall Detect:\t\t\tAveAcc\t\tAveJerk\t\tRotation\n");
              DPRINT("Fall Detect: WindowData[0]\t%8.3f\t%8.3f\t%8.3f\n",
                    WindowData[0].aveAccel, WindowData[0].aveJerk,
WindowData[0].rotation);
              DPRINT("Fall Detect: WindowData[1]\t%8.3f\t%8.3f\t%8.3f\n",
                    WindowData[1].aveAccel, WindowData[1].aveJerk,
WindowData[1].rotation);
              DPRINT("Fall Detect: WindowData[2]\t%8.3f\t%8.3f\t%8.3f\n",
                    WindowData[2].aveAccel, WindowData[2].aveJerk,
WindowData[2].rotation);
              DPRINT("Fall Detect: WindowData[3]\t%8.3f\t%8.3f\t%8.3f\n",
                    WindowData[3].aveAccel, WindowData[3].aveJerk,
WindowData[3].rotation);
              DPRINT("Fall Detect: WindowData[4]\t%8.3f\t%8.3f\t%8.3f\n",
                    WindowData[4].aveAccel, WindowData[4].aveJerk,
WindowData[4].rotation);
              outdata.fallyn=1;
              xQueueOverwrite(MailBox[Mailidx_FALL_TO_ESPNOW],&outdata);
              xQueueOverwrite(MailBox[Mailidx_FALL_TO_UI],&outdata);
              }
            else{
              outdata.fallyn=0;
              xQueueOverwrite(MailBox[Mailidx_FALL_TO_ESPNOW],&outdata);
              xQueueOverwrite(MailBox[Mailidx_FALL_TO_UI],&outdata);
            }
        }
        else{
            //DPRINT("Fall Detect: have %d datapoints.\n",index);

        }
      }
    }
}

void startFallDetectionTask(TaskHandle_t* TaskHandle,QueueHandle_t Queue) {

  xTaskCreate(
      FallDetectTask,             // Task function
      "Fall Detection Task",      // Name of the task
      8192,                       // Stack size
      Queue,                      // Parameters
      1,                          // Priority
      TaskHandle                 // Task handle
    );
}
```

WearableMaxFifoRead.h

```
#pragma once
#include <Wire.h>
```

```
#include "MAX30105.h"

//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"

#define sclPin 18
#define sdaPin 17
#define intPin 3


void MaxFIFOReadSetup(TaskHandle_t* TaskHandle, QueueHandle_t QueueToHRTask);
```

WearableMaxFifoRead.cpp

```
#include "WearableMaxFifoRead.h"
#include "WearableTaskHandles.h"

MAX30105 particleSensor;

SemaphoreHandle_t PPGSemaphore;

void IRAM_ATTR PPG_ISR_Handler() {
    detachInterrupt(intPin);  // Prevent retriggering
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR(PPGSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}



// FreeRTOS Task: Waits for the ISR notification
void _MAXFifoTask(void *pvParameters) {
    QueueHandle_t queue = (QueueHandle_t)pvParameters;
    uint32_t tstart;
    uint32_t tlast;
    uint32_t tfifoReadDone;
    uint32_t FIFO_Green[32] = {0};
    uint32_t FIFOir[32] = {0};
    int16_t nsamples;
    particleSensor.ReadFIFO(&nsamples,&FIFOir[0],&FIFO_Green[0],1,15);
      while (1) {
        if (xSemaphoreTake(PPGSemaphore, portMAX_DELAY) == pdTRUE){
            uint8_t int1=particleSensor.getINT1();
            //DPRINT("FIFO_MAX: Task start");
            tstart=micros();
            //DPRINT("FIFO_MAX: Overflow= 0x%02x
\n",particleSensor.readRegister8(MAX30105_ADDRESS, 0x05));
            particleSensor.ReadFIFO(&nsamples,&FIFOir[0],&FIFO_Green[0],1,18);
            tfifoReadDone=micros();
            for (uint8_t i=0 ; i<nsamples;i++){

              DPRINT("FIFO_MAX: %lu\n",FIFO_Green[i]);
              // Convert ADC uint32_t to float (adjust based on ADC resolution)
              //input[i] = (float)FIFO_Green[i] / 2621143.0f;
              if (xQueueSend(queue, &FIFO_Green[i], 0) != pdTRUE) {
                DPRINT("FIFO_MAX: QMI_FIFO: Queue full! Data lost.\n");
              }
          }

            //DPRINT("FIFO_MAX: fifo read time:%luus, nsamples:%d, int1:0x%02x
ovf:0x%02x, practicalFreq:%0.2f\n",tfifoReadDone-
tstart,nsamples,int1,particleSensor.readRegister8(MAX30105_ADDRESS,0x05),1.0f/(float)
(tstart-tlast)*1e6*nsamples);//read time ~5ms for 25samples
            //fifo read time:2709us, nsamples:30, int1:0x80 ovf:0x00,
practicalFreq:49.84 EXAMPLE
            tlast=tstart;
            //DPRINT("FIFO_MAX: time since last= %d \nTotalSamples=%d\n",tnow-
tlast,samplesTaken);
            //vTaskDelayUntil(&xLastWakeTime, xFrequency);
        }
        attachInterrupt(intPin, PPG_ISR_Handler, FALLING);
      }
  }

void MaxFIFOReadSetup(TaskHandle_t* TaskHandle, QueueHandle_t QueueToHRTask){

    pinMode(intPin,INPUT_PULLUP);

    vTaskDelay(500);

    //while(1){}
    // Initialize sensor
    if (Wire1.setPins(sdaPin, sclPin) == false){
        DPRINT("FIFO_MAX: set pins failed");
        while (1);
    }

    if (particleSensor.begin(Wire1, I2C_SPEED_FAST) == false) //Use 2nd I2C port,
400kHz speed
    {
        DPRINT("FIFO_MAX: MAX30105 was not found. Please check wiring/power. ");
        while (1);
    }
    //Setup to sense up to 18 inches, max LED brightness
    byte ledBrightness = 128; //Options: 0=Off to 255=50mA  WAS 10 3/1/25
    byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
    byte ledMode = 1; //Options: 1 =green only, EDITS TO LIB MEAN THIS SHOULD ONLY BE
1!
    int sampleRate = 200; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
    int pulseWidth = 411; //Options: 69, 118, 215, 411  THIS AFFECTS THE RESOLUTION
and the number of bits. fix the FIFO read function if you change this from 69.
    int adcRange = 2048; //Options: 2048, 4096, 8192, 16384

    particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate,
pulseWidth, adcRange); //Configure sensor with these settings

    particleSensor.writeRegister8(MAX30105_ADDRESS, 0x2, 0x80);//write int enable reg
    particleSensor.setFIFOAlmostFull(2);//write int enable reg
    DPRINT("FIFO_MAX: FIFO CONFIG= 0x%02x
\n",particleSensor.readRegister8(MAX30105_ADDRESS, 0x08));
```

```
    DPRINT("FIFO_MAX: INTstat1= 0x%02x \n",particleSensor.getINT1());
    DPRINT("FIFO_MAX: INTstat2= 0x%02x \n",particleSensor.getINT1());

    DPRINT("FIFO_MAX: INTEN1= 0x%02x 
\n",particleSensor.readRegister8(MAX30105_ADDRESS, 0x02));
    DPRINT("FIFO_MAX: INTEN2= 0x%02x 
\n",particleSensor.readRegister8(MAX30105_ADDRESS, 0x03));

    PPGSemaphore = xSemaphoreCreateBinary();
    if (PPGSemaphore != NULL) {
      xTaskCreate(
            _MAXFifoTask,         // Task function
            "PPG FIFO TASK",      // Name of the task
            4096,                 // Stack size
            QueueToHRTask,        // Parameters
            2,                    // Priority
            TaskHandle            // Task handle
      );
      vTaskDelay(10);
      xSemaphoreGive(PPGSemaphore);
      attachInterrupt(intPin, PPG_ISR_Handler, FALLING);
    }
}
```

<u>WearableHeartRate.h</u>

```
#pragma once

#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS

#include <math.h>
#include "esp_dsp.h"
#include "WearableDebug.h"

#define CONFIG_IDF_TARGET_ESP32S3 1

void startHeartRateTask(TaskHandle_t* TaskHandle,QueueHandle_t Queue);
```

<u>WearableHeartRate.cpp</u>

```
#include "WearableHeartRate.h"
#include "WearableTaskHandles.h"

#define SAMPLE_RATE 50
#define MIN_BPM 30                    // Minimum valid heart rate
#define MAX_BPM 230                   // Maximum valid heart rate
#define MIN_PEAK_HEIGHT -5            // Threshold for peak detection
#define MAX_PEAK_HEIGHT  5            // Threshold for peak detection
#define MIN_PEAK_PROMINENCE 0.7       // Minimum prominence for a valid peak
#define MAX_PEAK_PROMINENCE 5         // Maximum prominence for a valid peak
#define MIN_PEAK_WIDTH 1              // Minimum width of a peak
#define MAX_PEAK_WIDTH 100            // Maximum width of a peak
```

```
#define OUTLIER_STD_DEV 3           // std deviation outside of which the data is
tossed (for intervals)
#define NBEATS_TO_USE 3             //number of peaks needed to use this data chunk
#define MOVING_AVG_WINDOW 50        // Window size for moving average filter

#define AVERAGE_RAW_FOR_VALID 15000 //average raw value needed to call it a valid
data chunk.

#define STORAGE_SIZE 10*SAMPLE_RATE //10 seconds worth of data.

#define FIR_COEFFS
{
                                                                  \
  -0.00041271,     -0.00040991,     -0.00035275,     -0.00023662,     -
0.00006122,     0.00016790,     0.00043785,     0.00072761,     0.00100868,     0.001
24749,         \
  0.00140981,     0.00146668,     0.00140111,     0.00121426,     0.00092964,     0.0
0059407,     0.00027427,     0.00004900,     -
0.00000290,     0.00018304,            \
  0.00064200,     0.00136740,     0.00230333,     0.00334432,     0.00434368,     0.0
0513030,     0.00553256,     0.00540610,     0.00466154,     0.00328732,
\
  0.00136309,     -0.00093986,     -0.00337307,     -0.00563996,     -
0.00743688,     -0.00850083,     -0.00865745,     -0.00786146,     -0.00622229,     -
0.00400902,    \
  -0.00163120,     0.00040446,     0.00155805,     0.00133395,     -0.00063789,     -
0.00452819,     -0.01025358,     -0.01745043,     -0.02548402,     -
0.03349516,      \
  -0.04048161,     -0.04540652,     -0.04732198,     -0.04549278,     -
0.03950466,     -0.02934210,     -
0.01542412,     0.00140891,     0.01995834,     0.03877491,      \
  0.05629982,     0.07102361,     0.08164449,     0.08720742,     0.08720742,     0.0
8164449,     0.07102361,     0.05629982,     0.03877491,     0.01995834,
\
  0.00140891,     -0.01542412,     -0.02934210,     -0.03950466,     -
0.04549278,     -0.04732198,     -0.04540652,     -0.04048161,     -0.03349516,     -
0.02548402,    \
  -0.01745043,     -0.01025358,     -0.00452819,     -
0.00063789,     0.00133395,     0.00155805,     0.00040446,     -0.00163120,     -
0.00400902,     -0.00622229,      \
  -0.00786146,     -0.00865745,     -0.00850083,     -0.00743688,     -
0.00563996,     -0.00337307,     -
0.00093986,     0.00136309,     0.00328732,     0.00466154,      \
  0.00540610,     0.00553256,     0.00513030,     0.00434368,     0.00334432,     0.0
0230333,     0.00136740,     0.00064200,     0.00018304,     -
0.00000290,            \
  0.00004900,     0.00027427,     0.00059407,     0.00092964,     0.00121426,     0.0
0140111,     0.00146668,     0.00140981,     0.00124749,     0.00100868,
\
  0.00072761,     0.00043785,     0.00016790,     -0.00006122,     -0.00023662,     -
0.00035275,     -0.00040991,     -
0.00041271                                \
}//128 coeffs.

// Function to remove DC bias using a moving average filter with initial fill-in
```

```
void removeDCBias(std::vector<float> &data) {
    std::vector<float> filteredData(data.size());
    float sum = 0;

    // Compute initial mean for the first MOVING_AVG_WINDOW samples
    int initialWindow = std::min(MOVING_AVG_WINDOW, (int)data.size());
    for (int i = 0; i < initialWindow; i++) {
        sum += data[i];
    }
    float initialMean = sum / initialWindow;

    for (size_t i = 0; i < data.size(); i++) {
        if (i < MOVING_AVG_WINDOW) {
            filteredData[i] = data[i] - initialMean;  // Use initial mean for first
values
        } else {
            sum += data[i] - data[i - MOVING_AVG_WINDOW];
            filteredData[i] = data[i] - (sum / MOVING_AVG_WINDOW);
        }
    }
    data = filteredData;
}
void zScoreNormalize(std::vector<float> &data) {
    float mean = 0, stddev = 0;
    for (float val : data) mean += val;
    mean /= data.size();
    for (float val : data) stddev += pow(val - mean, 2);
    stddev = sqrt(stddev / data.size());
    if (stddev == 0) return;
    for (float &val : data) val = (val - mean) / stddev;
}

bool detectHeartbeat(std::vector<float> &data, float &detectedBPM) {

    std::vector<int> peaks;
    int dataSize = data.size();

    if (dataSize < SAMPLE_RATE) return false;  // Need at least 1 second of data

    float minPeakHeight = std::numeric_limits<float>::max();
    float maxPeakHeight = std::numeric_limits<float>::lowest();
    float minPeakProminence = std::numeric_limits<float>::max();
    float maxPeakProminence = std::numeric_limits<float>::lowest();
    int minPeakWidth = std::numeric_limits<int>::max();
    int maxPeakWidth = std::numeric_limits<int>::lowest();

    // Detect peaks with prominence and width filtering
    for (int i = 1; i < dataSize - 1; i++) {
        if (data[i] > data[i - 1] && data[i] > data[i + 1] && data[i] >
MIN_PEAK_HEIGHT && data[i] < MAX_PEAK_HEIGHT) {
            int left = i, right = i;
            while (left > 0 && data[left] > data[left - 1]) {left--;}
            while (right < dataSize - 1 && data[right] > data[right + 1]) {right++;}
            int width = right - left;
            if (width < MIN_PEAK_WIDTH || width > MAX_PEAK_WIDTH) continue;
            // Find last turning points on both sides
            float leftMin = data[left];
            float rightMin = data[right];
            for (int j = left; j < i; j++) {
                if (data[j] < leftMin) leftMin = data[j];
            }
            for (int j = right; j > i; j--) {
                if (data[j] < rightMin) rightMin = data[j];
            }

            // Use the smaller of the two turning points for prominence calculation
            float prominence = data[i] - std::min(leftMin, rightMin);
            if (prominence < MIN_PEAK_PROMINENCE || prominence > MAX_PEAK_PROMINENCE)
continue;

            peaks.push_back(i);

            // Update min/max values
            if (data[i] < minPeakHeight) minPeakHeight = data[i];
            if (data[i] > maxPeakHeight) maxPeakHeight = data[i];
            if (prominence < minPeakProminence) minPeakProminence = prominence;
            if (prominence > maxPeakProminence) maxPeakProminence = prominence;
            if (width < minPeakWidth) minPeakWidth = width;
            if (width > maxPeakWidth) maxPeakWidth = width;
        }
    }
    // Print detected min/max values
    if (peaks.size() > 1){
      DPRINT("Heart Rate: Min Peak Height: %0.6f\n",minPeakHeight);
      DPRINT("Heart Rate: Max Peak Height: %0.6f\n",maxPeakHeight);
      DPRINT("Heart Rate: Min Peak Prominence: %0.6f\n",minPeakProminence);
      DPRINT("Heart Rate: Max Peak Prominence: %0.6f\n",maxPeakProminence);
      DPRINT("Heart Rate: Min Peak Width: %d\n",minPeakWidth);
      DPRINT("Heart Rate: Max Peak Width: %d\n",maxPeakWidth);
    }
    DPRINT("Heart Rate: Peak cnt: %d\n",peaks.size());
    // Validate peak count
    if (peaks.size() < NBEATS_TO_USE) return false;

    // Compute time intervals between peaks (in seconds)
    std::vector<float> intervals;
    for (size_t i = 1; i < peaks.size(); i++) {
        float interval = (peaks[i] - peaks[i - 1]) / (float)SAMPLE_RATE;
        if (interval > 0) intervals.push_back(interval);
    }

    if (intervals.empty()) return false;

     // Compute mean and standard deviation
    float mean = 0, stddev = 0;
    for (float interval : intervals) mean += interval;
    mean /= intervals.size();
    for (float interval : intervals) stddev += pow(interval - mean, 2);
    stddev = sqrt(stddev / intervals.size());
    // Filter out outliers beyond OUTLIER_STD_DEV standard deviations
    std::vector<float> filteredIntervals;
    for (float interval : intervals) {
        if (fabs(interval - mean) <= OUTLIER_STD_DEV * stddev) {
            filteredIntervals.push_back(interval);
        }
    }

    if (filteredIntervals.empty()) return false;

    // Compute average BPM with filtered intervals
    float avgInterval = 0;
    for (float interval : filteredIntervals) {
        avgInterval += interval;
    }
    avgInterval /= filteredIntervals.size();

    detectedBPM = 60.0 / avgInterval;

    // Validate BPM range
    return (detectedBPM >= MIN_BPM && detectedBPM <= MAX_BPM);
}

void HeartRateTask(void *parameter) {
    QueueHandle_t HRQ =(QueueHandle_t)parameter;
    if (HRQ == NULL) {
        DPRINT("Heart Rate: Error Queue handle is NULL!\n");
        vTaskDelete(NULL);  // Terminate the task if queue is invalid
    }
    uint32_t buffer;
    float inputdata[40]__attribute__((aligned(16)));
    float outputdata[40]__attribute__((aligned(16)));
    float DelayLine[132]__attribute__((aligned(16)));
    float dataStorage[STORAGE_SIZE];
    uint16_t IdxQueue = 0;
    uint16_t IdxStorage = 0;
    uint32_t aveInput = 0;
    fir_f32_t fir;
    float coeffs[128] __attribute__((aligned(16))) = FIR_COEFFS;
    esp_err_t err = dsps_fir_init_f32(&fir,&coeffs[0],DelayLine,128);
    if (err != ESP_OK){
      while (1) {
        DPRINT("Heart Rate: ERROR INIT ESP-DSP");
        vTaskDelay(1000);
        }
    }
    UBaseType_t itemCount;
    DPRINT("Heart Rate: Entering task\n");
    while (1) {
      if (xQueueReceive(HRQ, &buffer, portMAX_DELAY) == pdTRUE) {
        IdxQueue=0;
        itemCount = uxQueueMessagesWaiting(HRQ);
        inputdata[IdxQueue] = (float)buffer / 262143.0;
        IdxQueue++;
        aveInput+=buffer;
        for (int i = 0; i < itemCount; i++) {//receive data
            if (xQueueReceive(HRQ, &buffer, 0)==pdTRUE){
              inputdata[IdxQueue] = (float)buffer / 262143.0 ;
              IdxQueue++;
              aveInput+=buffer;
            }
        }
        err = dsps_fir_f32_aes3(&fir,&inputdata[0], &outputdata[0], IdxQueue);
        if (err != ESP_OK){
          while (1) {
            DPRINT("Heart Rate: ERROR ESP-DSP failed to run");
            vTaskDelay(1000);
            }
        }
        if(IdxStorage+IdxQueue<STORAGE_SIZE){
          memcpy(&dataStorage[IdxStorage],&outputdata[0],IdxQueue *
sizeof(float));//if there is enough room, copy the whole chunk
          IdxStorage+=IdxQueue;
        }
        else{
          uint8_t ncopy = STORAGE_SIZE - IdxStorage;
          memcpy(&dataStorage[IdxStorage],&outputdata[0],ncopy * sizeof(float));
          IdxStorage+=ncopy;
        }//fill 'er up

        if(IdxStorage>=STORAGE_SIZE){
        aveInput/=IdxStorage;
        float bpm=0;
        MailHR_t HRdata;
        DPRINT("Heart Rate: aveRaw:%ld\n",aveInput);
        std::vector<float> dataVector(dataStorage, dataStorage + STORAGE_SIZE);
        removeDCBias(dataVector);  // Remove DC bias first
        zScoreNormalize(dataVector);  // Apply Z-score normalization

        for (int i = 0; i < STORAGE_SIZE; i++){
            DPRINT("Heart Rate: data:%0.5f\n",dataVector.at(i));
          }

        if(detectHeartbeat(dataVector,bpm)&&aveInput>AVERAGE_RAW_FOR_VALID){
          DPRINT("Heart Rate: bpm:%0.2f\n",bpm);
          HRdata.HRValid=1;
          HRdata.HRValue=(uint8_t)(bpm+0.5);
          xQueueOverwrite(MailBox[Mailidx_HR_TO_ESPNOW],&HRdata);
          xQueueOverwrite(MailBox[Mailidx_HR_TO_UI],&HRdata);
        }
        else{
          DPRINT("Heart Rate: invalid bpm:%0.2f\n",bpm);
          HRdata.HRValid=0;
          HRdata.HRValue=0;
          xQueueOverwrite(MailBox[Mailidx_HR_TO_ESPNOW],&HRdata);
          xQueueOverwrite(MailBox[Mailidx_HR_TO_UI],&HRdata);
        }
        aveInput=0;
        IdxStorage=0;//after consuming the data, set the storage pointer back to 0.
        }
      }
    }
}




void startHeartRateTask(TaskHandle_t* TaskHandle,QueueHandle_t Queue) {

  xTaskCreate(
      HeartRateTask,             // Task function
      "Heart Rate Task",      // Name of the task
      16384,                       // Stack size
      Queue,                      // Parameters
      1,                          // Priority
      TaskHandle                // Task handle
    );
}
```

WearableTaskMonitor.h

```
#pragma once
//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"
#include "Arduino.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_private/esp_clk.h"
void StartMonitoringTask(TaskHandle_t* TaskHandle);
```

WearableTaskMonitor.cpp

```

#include "WearableTaskMonitor.h"
#include "WearableTaskHandles.h"

void TaskMonitor(void *parameter) {
    const TickType_t delayTime = pdMS_TO_TICKS(5000); // Update every 5 seconds
    char buffer[512]; // Buffer for task statistics
    TaskStatus_t *taskArray;
    UBaseType_t taskCount;
    uint32_t totalRunTime;

    while (1) {
        // Get the number of tasks
        taskCount = uxTaskGetNumberOfTasks();
        taskArray = (TaskStatus_t *)pvPortMalloc(taskCount * sizeof(TaskStatus_t));

        if (taskArray != NULL) {
            // Get system task details
            taskCount = uxTaskGetSystemState(taskArray, taskCount, &totalRunTime);

            if (totalRunTime > 0) {
                DPRINT("\nTask Name\tCPU Usage (%%)\tStack Usage (bytes)\n");
                DPRINT("------------------------------------------------\n");

                for (UBaseType_t i = 0; i < taskCount; i++) {
                    float cpuUsage = (taskArray[i].ulRunTimeCounter * 100.0) /
totalRunTime;
                    uint32_t stackUsed = taskArray[i].usStackHighWaterMark *
sizeof(StackType_t);

                    DPRINT("%-12s\t%6.2f%%\t%8d\n", taskArray[i].pcTaskName,
cpuUsage, stackUsed);
                }
            }
            vPortFree(taskArray);
        }
        DPRINT("CPU Clock Speed: %d MHz\n", esp_clk_cpu_freq() / 1000000);
        vTaskDelay(delayTime);
    }
}
void StartMonitoringTask(TaskHandle_t* TaskHandle){
    xTaskCreate(TaskMonitor, "TaskMonitor", 4096, NULL, 1, NULL);
}
```

WearableUI.h

```
#pragma once
//#define DEBUG     //TURN THIS ON IF YOU WANT PRINT STATEMENTS
#include "WearableDebug.h"
#include <lvgl.h>
#include "Arduino_GFX_Library.h"
#include "Arduino_DriveBus_Library.h"
#include "lv_conf.h"
#include "demos/lv_demos.h"
#include "pin_config.h"
#include <Wire.h>
#include "esp32-hal-ledc.h"

#define EXAMPLE_LVGL_TICK_PERIOD_MS 2
#define USE_NEW_PIN_CONFIG 0

#define POWER_DOWN_BUTTON_HOLD_TIME 1000
#define BACKLIGHT_DURATION 3000
#define TOUCH_DEBOUNCE_MS 200  // Time to ignore touches after backlight turns on

void startUITask(TaskHandle_t* TaskHandle);
```

WearableUI.cpp

```
#include "WearableUI.h"
#include "WearableTaskHandles.h"

//pin configuration
const int inputPin = 40;   // Define GPIO40 as input
const int outputPin = 41;  // Define GPIO41 as output
const int beePin = 42;     // Define GPIO42 for buzzer

const int voltageDividerPin = 1;  // GPIO1 pin
float vRef = 3.3;                 // Power supply voltage of ESP32-S3 (unit: volts)
float R1 = 200000.0;              // Resistance value of the first resistor (unit: 
ohms)
float R2 = 100000.0;              // Resistance value of the second resistor (unit: 
ohms)

bool buttonState = false;
bool lastButtonState = false;

/*Change to your screen resolution*/
static const uint16_t screenWidth = 240;
static const uint16_t screenHeight = 280;

static lv_disp_draw_buf_t draw_buf;
static lv_color_t buf[screenWidth * screenHeight / 10];

static const MailUI_User_t helpReqUnprompted = {
  .UserMessage = 100
};
static const MailUI_User_t helpReqPrompted = {
  .UserMessage = 110
};
static const MailUI_User_t nohelpPrompted = {
  .UserMessage = 120
};

uint8_t firstscan=1;//Power sequence/sound control globals
uint8_t buttonReleasedInit=0;
uint8_t PowerDownStarted=0;
uint32_t PowerDownStartTime=0;

uint32_t waketouchTime=0;
uint32_t lasttouchTime=0;

uint32_t tnow;
uint32_t tlast;

char HRCharbuff[30];
char espnowCharbuff[30];
char BatteryCharbuff[30];
lv_obj_t *labelHR_Main, *labelESPnow_Main, *labelBatt_Main;  //global label objects
lv_obj_t *labelHR_Conf, *labelESPnow_Conf, *labelBatt_Conf;  //global label objects
lv_obj_t *scrMain, *scrConf;
lv_obj_t *labelQuestion;
Arduino_DataBus *bus = new Arduino_ESP32SPI(LCD_DC, LCD_CS, LCD_SCK, LCD_MOSI);

Arduino_GFX *gfx = new Arduino_ST7789(bus, LCD_RST /* RST */,
                                      0 /* rotation */, true /* IPS */, LCD_WIDTH, 
LCD_HEIGHT, 0, 20, 0, 0);

std::shared_ptr<Arduino_IIC_DriveBus> IIC_Bus =
  std::make_shared<Arduino_HWIIC>(IIC_SDA, IIC_SCL, &Wire);

void Arduino_IIC_Touch_Interrupt(void);

std::unique_ptr<Arduino_IIC> CST816T(new Arduino_CST816x(IIC_Bus,
CST816T_DEVICE_ADDRESS,
                                                         TP_RST, TP_INT,
Arduino_IIC_Touch_Interrupt));

void Arduino_IIC_Touch_Interrupt(void) {
  CST816T->IIC_Interrupt_Flag = true;
}
//TOUCH CONTROL
void my_touchpad_read(lv_indev_drv_t *indev_driver, lv_indev_data_t *data) {
    static bool backlightWasOff = false;  // Track if backlight was off before the
touch

    int32_t touchX = CST816T->IIC_Read_Device_Value(CST816T-
>Arduino_IIC_Touch::Value_Information::TOUCH_COORDINATE_X);
    int32_t touchY = CST816T->IIC_Read_Device_Value(CST816T-
>Arduino_IIC_Touch::Value_Information::TOUCH_COORDINATE_Y);

    if (CST816T->IIC_Interrupt_Flag == true) {
        CST816T->IIC_Interrupt_Flag = false;

        // If backlight is off, only turn it on and don't process touch
        if (digitalRead(LCD_BL) == LOW) {
            digitalWrite(LCD_BL, HIGH);   // Turn on backlight
            waketouchTime = millis();
            lasttouchTime = waketouchTime;     // Reset backlight timeout
            backlightWasOff = true;       // Mark that we only woke the screen
        }
        else {
            // If backlight was already on, process touch normally
            if (touchX >= 0 && touchY >= 0  && millis() - waketouchTime >
TOUCH_DEBOUNCE_MS) {
                lasttouchTime = millis();     // Reset backlight timeout
                if (!backlightWasOff) {  // Ignore the first touch after waking
                    data->state = LV_INDEV_STATE_PR;
                    data->point.x = touchX;
                    data->point.y = touchY;
                } else {
                    data->state = LV_INDEV_STATE_REL; // Ignore first touch
                    backlightWasOff = false;         // Reset flag for next touches
                }
            }
        }
    } else {
        data->state = LV_INDEV_STATE_REL;
    }
}

/* Display flushing */
void my_disp_flush(lv_disp_drv_t *disp, const lv_area_t *area, lv_color_t *color_p) {
  uint32_t w = (area->x2 - area->x1 + 1);
  uint32_t h = (area->y2 - area->y1 + 1);

#if (LV_COLOR_16_SWAP != 0)
  gfx->draw16bitBeRGBBitmap(area->x1, area->y1, (uint16_t *)&color_p->full, w, h);
#else
  gfx->draw16bitRGBBitmap(area->x1, area->y1, (uint16_t *)&color_p->full, w, h);
#endif

  lv_disp_flush_ready(disp);
}

//TIMER INTERRUPT FUNCTION
void example_increase_lvgl_tick(void *arg) {
  /* Tell LVGL how many milliseconds has elapsed */
  tnow=millis();
  lv_tick_inc(tnow-tlast);
  tlast=tnow;
}

 //SOUND FUNCTIONS
void PowerUpSound(){
    noTone(beePin);
    tone(beePin, 2500,100);
    tone(beePin, 3000,100);
    tone(beePin, 5000,100);
}

void PowerDownSound(){
    noTone(beePin);
    tone(beePin, 5000,100);
    tone(beePin, 3000,100);
    tone(beePin, 2500,100);
}

void PowerDownStartSound(){
    noTone(beePin);
    tone(beePin, 7000,300);
}

void HelpRequestSound(){
    noTone(beePin);
    tone(beePin, 5500,150);
    tone(beePin, 7500,150);
}

void HelpDenySound(){
    noTone(beePin);
    tone(beePin, 2500,50);
    tone(beePin, 3500,50);
    tone(beePin, 4500,50);
    tone(beePin, 5500,50);
    tone(beePin, 3500,100);
}

  void PossibleFallSound(){
    noTone(beePin);
    tone(beePin, 5000,100);
    tone(beePin, 7000,100);
    tone(beePin, 5000,100);
    tone(beePin, 7000,100);
    tone(beePin, 5000,100);
    tone(beePin, 7000,100);
}

//BUTTON EVENT HANDLING FUNTIONS

void MainHelpHandler(lv_event_t* event)
{
    if(lv_event_get_code(event) == LV_EVENT_CLICKED) {
        DPRINT("Clicked HELP\n");
        HelpRequestSound();
        xQueueOverwrite(MailBox[Mailidx_UI_TO_ESPNOW_user],&helpReqUnprompted);//send
unprompted Help to ESPnow
    }
}

void ConfirmHelpHandler(lv_event_t* event)
{
    if(lv_event_get_code(event) == LV_EVENT_CLICKED) {
        DPRINT("Clicked Help Confirmed\n");
        HelpRequestSound();
        xQueueOverwrite(MailBox[Mailidx_UI_TO_ESPNOW_user],&helpReqPrompted);//send
unprompted Help to ESPnow
        lv_scr_load(scrMain);
    }
}
void DenyHelpHandler(lv_event_t* event)
{
    if(lv_event_get_code(event) == LV_EVENT_CLICKED) {
        DPRINT("Clicked No Help Needed\n");
        HelpDenySound();
        xQueueOverwrite(MailBox[Mailidx_UI_TO_ESPNOW_user],&nohelpPrompted);//send
unprompted Help to ESPnow
        lv_scr_load(scrMain);
    }
}



//BUTTON CREATION FUNCTIONS
void CreateMainHelpButton(lv_obj_t* scr){
  lv_obj_t * label;
  lv_obj_t * btn1 = lv_btn_create(scr);
  lv_obj_add_event_cb(btn1, MainHelpHandler, LV_EVENT_CLICKED, NULL);
  lv_obj_set_height(btn1, lv_pct(70));
  lv_obj_set_width(btn1, lv_pct(100));
  lv_obj_align(btn1, LV_ALIGN_BOTTOM_MID, 0, 0);
  lv_obj_set_style_bg_color(btn1, lv_palette_main(LV_PALETTE_RED), LV_PART_MAIN); //
Change background color
```

```
  label = lv_label_create(btn1);
  lv_label_set_text(label, "HELP!");
}

void CreateConfirmHelpButtons(lv_obj_t* scr){
  lv_obj_t * label1;
  lv_obj_t * btn1 = lv_btn_create(scr);
  lv_obj_add_event_cb(btn1, ConfirmHelpHandler, LV_EVENT_CLICKED, NULL);
  lv_obj_set_height(btn1, lv_pct(70));
  lv_obj_set_width(btn1, lv_pct(50));
  lv_obj_align(btn1, LV_ALIGN_BOTTOM_LEFT, 0, 0);
  lv_obj_set_style_bg_color(btn1, lv_palette_main(LV_PALETTE_RED), LV_PART_MAIN); //
Change background color

  label1 = lv_label_create(btn1);
  lv_label_set_text(label1, "YES\n\rSEND HELP");

  lv_obj_t * label2;
  lv_obj_t * btn2 = lv_btn_create(scr);
  lv_obj_add_event_cb(btn2, DenyHelpHandler, LV_EVENT_CLICKED, NULL);
  lv_obj_set_height(btn2, lv_pct(70));
  lv_obj_set_width(btn2, lv_pct(50));
  lv_obj_align(btn2, LV_ALIGN_BOTTOM_RIGHT, 0, 0);
  lv_obj_set_style_bg_color(btn2, lv_palette_main(LV_PALETTE_GREEN), LV_PART_MAIN);
// Change background color

  label2 = lv_label_create(btn2);
  lv_label_set_text(label2, "NO\n\rI'm OK");
}

//SCREEN CREATION FUNCTIONS
void CreateMainScreen(lv_obj_t*& scr,lv_obj_t*& labelHR,lv_obj_t*&
labelESPnow,lv_obj_t*& labelBatt){
    //MAIN Screen Creation
    scr = lv_obj_create(NULL);
    CreateMainHelpButton(scr);

    labelHR = lv_label_create(scr);
    lv_label_set_text(labelHR, "HR: initmain");
    lv_obj_align(labelHR, LV_ALIGN_TOP_LEFT, 5, 25);

    labelESPnow = lv_label_create(scr);
    lv_label_set_text(labelESPnow, "ESPnow: initmain");
    lv_obj_align(labelESPnow, LV_ALIGN_TOP_LEFT, 5, 50);

    labelBatt = lv_label_create(scr);
    lv_label_set_text(labelBatt, "Charge:ini1");
    lv_obj_align(labelBatt, LV_ALIGN_TOP_RIGHT, -10, 25);
}

void CreateConfirmScreen(lv_obj_t*& scr,lv_obj_t*& labelHR,lv_obj_t*&
labelESPnow,lv_obj_t*& labelBatt){
    //MAIN Screen Creation
    scr = lv_obj_create(NULL);
    CreateConfirmHelpButtons(scr);
    labelHR = lv_label_create(scr);
    lv_label_set_text(labelHR, "HR: initsecond");
    lv_obj_align(labelHR, LV_ALIGN_TOP_LEFT, 5, 25);

    labelESPnow = lv_label_create(scr);
    lv_label_set_text(labelESPnow, "ESPnow: initsecond");
    lv_obj_align(labelESPnow, LV_ALIGN_TOP_LEFT, 5, 50);

    labelBatt = lv_label_create(scr);
    lv_label_set_text(labelBatt, "Charge:ini2");
    lv_obj_align(labelBatt, LV_ALIGN_TOP_RIGHT, -5, 25);

    labelQuestion = lv_label_create(scr);
    lv_label_set_text(labelQuestion, "DID YOU FALL?");
    lv_obj_align(labelQuestion, LV_ALIGN_CENTER, 0, -65);
}

  void UItask(void *parameter) {
    MailHR_t HR;
    MailFall_t FallMail;
    int32_t NowStat;
    while(1){
      lv_timer_handler(); /* let the GUI do its work */
      vTaskDelay(60);

      //keep tone pin low while not in use.
      uint32_t nTones = ledcRead(beePin);
      if (nTones==0){
        digitalWrite(beePin, LOW);
      }

      //BACKLIGHT DISABLING CONTROL
      if (millis()-lasttouchTime>BACKLIGHT_DURATION){
        digitalWrite(LCD_BL, LOW);
      }


      // Read ADC value
      int adcValue = analogRead(voltageDividerPin);
      // Convert to voltage
      float voltage = (float)adcValue * (vRef / 4095.0);
      // Apply the voltage divider formula to calculate the actual voltage
      float actualVoltage = voltage * ((R1 + R2) / R2);
      MailUI_periodic_t periodicMessage = {.BatteryLevel =
actualVoltage*1000};//sending battery voltage in units of mV
      xQueueOverwrite(MailBox[Mailidx_UI_TO_ESPNOW_periodic],&periodicMessage);

      //Receive Messages
      if (xQueueReceive(MailBox[Mailidx_FALL_TO_UI], &FallMail, 0) == pdTRUE){
        if (FallMail.fallyn==1){
          PossibleFallSound();
          lv_scr_load(scrConf);
          digitalWrite(LCD_BL, HIGH);   // Turn on backlight
          lasttouchTime = millis(); // Reset backlight timeout
        }
      }

      xQueueReceive(MailBox[Mailidx_HR_TO_UI], &HR, 0);
      int16_t HRdisp = (HR.HRValid==1) ? HR.HRValue : -1;

      xQueueReceive(MailBox[Mailidx_ESPNOW_TO_UI], &NowStat, 0);

      //label printing
      (NowStat==1) ? sprintf(espnowCharbuff, "ESPnow: OK") : sprintf(espnowCharbuff,
"ESPnow: XX");
      //sprintf(espnowCharbuff, "ESPnow: OK %d", 37);
      sprintf(HRCharbuff, "HR: %dbpm", HRdisp);
      sprintf(BatteryCharbuff, "Batt:%4.2fV",actualVoltage);
      if (lv_scr_act() == scrMain) {
            lv_label_set_text(labelHR_Main, HRCharbuff);
            lv_label_set_text(labelESPnow_Main, espnowCharbuff);
            lv_label_set_text(labelBatt_Main,BatteryCharbuff);
      }else if (lv_scr_act() == scrConf) {
            lv_label_set_text(labelHR_Conf, HRCharbuff);
            lv_label_set_text(labelESPnow_Conf, espnowCharbuff);
            lv_label_set_text(labelBatt_Conf,BatteryCharbuff);
      }



      //POWER Control
      if (firstscan){
        PowerUpSound();
        waketouchTime = millis();
        lasttouchTime = waketouchTime;     // Reset backlight timeout
        firstscan=0;
      }

      if (buttonReleasedInit){//INITIAL POWER BUTTON RELEASE
        if (!digitalRead(inputPin) && PowerDownStarted==0){//POWER BUTTON PUSHED
          PowerDownStartSound();
          PowerDownStarted=1;
          PowerDownStartTime = millis();
        }
        else if (digitalRead(inputPin) && PowerDownStarted==1 ){//button released
          PowerDownStarted=0;
          if (millis()-PowerDownStartTime > POWER_DOWN_BUTTON_HOLD_TIME){
            PowerDownSound();
            vTaskDelay(10);
            while(ledcRead(beePin)!=0){
              vTaskDelay(1);
            }
            digitalWrite(outputPin, LOW);  // Shut Down
          }
        }
      }else{
        if (digitalRead(inputPin)){// wait until button reads released before
executing power down detection code
          buttonReleasedInit=1;
        }
      }
    }
  }

    void startUITask(TaskHandle_t* TaskHandle) {
    USBSerial.begin(115200); /* prepare for possible serial debug */
    pinMode(inputPin, INPUT);
    pinMode(outputPin, OUTPUT);
    pinMode(beePin, OUTPUT);

    digitalWrite(outputPin, HIGH);  // Initialize output pin to HIGH
    digitalWrite(beePin, LOW);
    pinMode(voltageDividerPin, INPUT);


    while (CST816T->begin() == false) {
        DPRINT("CST816T initialization fail");
      delay(2000);
    }
    DPRINT("CST816T initialization successfully");

    CST816T->IIC_Write_Device_State(CST816T-
>Arduino_IIC_Touch::Device::TOUCH_DEVICE_INTERRUPT_MODE,
                                    CST816T-
>Arduino_IIC_Touch::Device_Mode::TOUCH_DEVICE_INTERRUPT_PERIODIC);

    gfx->begin();
    pinMode(LCD_BL, OUTPUT);
    digitalWrite(LCD_BL, HIGH);

    String LVGL_Arduino = "Hello Arduino! ";
    LVGL_Arduino += String('V') + lv_version_major() + "." + lv_version_minor() + "."
+ lv_version_patch();

    DPRINT("UI: %s\n",LVGL_Arduino.c_str());
    DPRINT("UI: I am LVGL_Arduino\n");

    lv_init();

    lv_disp_draw_buf_init(&draw_buf, buf, NULL, screenWidth * screenHeight / 10);

    /*Initialize the display*/
    static lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);
    /*Change the following line to your display resolution*/
    disp_drv.hor_res = screenWidth;
    disp_drv.ver_res = screenHeight;
    disp_drv.flush_cb = my_disp_flush;
    disp_drv.draw_buf = &draw_buf;
    lv_disp_drv_register(&disp_drv);


    /*Initialize the (dummy) input device driver*/
    static lv_indev_drv_t indev_drv;
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = my_touchpad_read;
    lv_indev_drv_register(&indev_drv);

    //start periodic task to update lvgl with milliseconds.
    const esp_timer_create_args_t lvgl_tick_timer_args = {
      .callback = &example_increase_lvgl_tick,
      .name = "lvgl_tick"
    };

    esp_timer_handle_t lvgl_tick_timer = NULL;
    esp_timer_create(&lvgl_tick_timer_args, &lvgl_tick_timer);
    esp_timer_start_periodic(lvgl_tick_timer, EXAMPLE_LVGL_TICK_PERIOD_MS * 1000);


    CreateMainScreen(scrMain,labelHR_Main,labelESPnow_Main,labelBatt_Main);
    CreateConfirmScreen(scrConf,labelHR_Conf,labelESPnow_Conf,labelBatt_Conf);
    lv_scr_load(scrConf);
    //lv_disp_set_refresh_period(NULL, 15);
    xTaskCreate(
        UItask,                    // Task function
        "UI Task",                 // Name of the task
        8192,                      // Stack size
        (void*)NULL,               // Parameters
        3,                         // Priority
        TaskHandle                 // Task handle
      );
    DPRINT("UI: Setup done\n");
  }
```

WearableDebug.h

```
#include <Arduino.h>
#include <HWCDC.h>
extern HWCDC USBSerial;  // Declare USBSerial (defined in main.ino)

#ifdef DBEGIN
#undef DBEGIN
#endif

#define DBEGIN(...)       USBSerial.begin(__VA_ARGS__)

#ifdef DPRINT
#undef DPRINT
#endif

#ifdef DEBUG
//OR, #define DPRINT(args...)    Serial.print(args)


#define DPRINT(...)                                                   \
    do {                                                              \
        if (xSemaphoreTake(serialMutex, portMAX_DELAY) == pdTRUE) {   \
            USBSerial.printf(__VA_ARGS__);                            \
            xSemaphoreGive(serialMutex);                              \
        }                                                             \
    } while (0)

extern SemaphoreHandle_t serialMutex;
extern portMUX_TYPE serialSpinlock;
#else
#define DPRINT(...)     //blank line
#define DPRINTLN(...)   //blank line
#define DPRINTF(...)    //blank line
#define DPRINTLNF(...)  //blank line
#endif
```

<u>SensorQMI8658.hpp</u>

From lewisxhe/SensorLib UPDATES START AT LINE 1394

```
bool resetFifo_New(){
      uint8_t stat;
      uint8_t cmd = 0x04;
      writeRegister(QMI8658_REG_CTRL9, cmd);
      do{readRegister(QMI8658_REG_STATUSINT,&stat,1);
      }while(stat>>7!=1);

      writeRegister(QMI8658_REG_CTRL9, 0x0);

      do{readRegister(QMI8658_REG_STATUSINT,&stat,1);
      }while(stat>>7!=0);

      return true;
    }

    uint16_t readFromFifo_New(IMUdata *AccBuffer,IMUdata *GyrBuffer,uint8_t*
databuffer){
      //assume this function was called from an interrupt, also assume that both the
acclerometer and gyro are enabled.
      /*
      1. Get FIFO watermark interrupt by INT pin or polling the FIFO_STATUS register
(FIFO_WTM and/or FIFO_FULL).
      2. Read the FIFO_SMPL_CNT and FIFO_STATUS registers, to calculate the level of
FIFO content data, refer to 8.4
         FIFO Sample Count.
      3. Send CTRL_CMD_REQ_FIFO (0x05) by CTRL9 command, to enable FIFO read mode.
Refer to
         CTRL_CMD_REQ_FIFO for details.
      4. Read from the FIFO_DATA register per FIFO_Sample_Count.
      5. Disable the FIFO Read Mode by setting FIFO_CTRL.FIFO_rd_mode to 0. New data
will be filled into FIFO
         afterwards.
            */
      //step 2
      uint8_t cnt;
      uint8_t stat;
      readRegister(QMI8658_REG_FIFOCOUNT,&cnt,1);
      readRegister(QMI8658_REG_FIFOSTATUS,&stat,1);//2 LSBs of this value make up the
2 MSB of the actual count value.

      uint16_t nbytes=2*(uint16_t)(((uint16_t)stat & 0x03)*(uint16_t)256 +
(uint16_t)cnt);//fifo sample count in bytes
      uint8_t nsamp=nbytes/12;

      if (nsamp==0){return 0;}
      //step 3
      uint8_t cmd = 0x05;
      writeRegister(QMI8658_REG_CTRL9, cmd);
      do{readRegister(QMI8658_REG_STATUSINT,&stat,1);
      }while(stat>>7!=1);
      writeRegister(QMI8658_REG_CTRL9, 0x0);

      do{readRegister(QMI8658_REG_STATUSINT,&stat,1);
      }while(stat>>7!=0);

      //step4
      readRegister(QMI8658_REG_FIFODATA,databuffer,nbytes);

      //step5
      readRegister(QMI8658_REG_FIFOCTRL,&stat,1);
      writeRegister(QMI8658_REG_FIFOCTRL, stat & 0x7F);//reset fifo_RD_MODE which was
set automatically by the ctrl 9 process

      //write to sample registers
      for (int i=0;i<nsamp;i++){
        uint8_t samplen=12;
        int16_t rawBuffer[3];
        //accel reading
        rawBuffer[0] = (int16_t)(*(databuffer + samplen*i + 1) << 8) | (*(databuffer
+ samplen*i  + 0));
        rawBuffer[1] = (int16_t)(*(databuffer + samplen*i + 3) << 8) | (*(databuffer
+ samplen*i  + 2));
        rawBuffer[2] = (int16_t)(*(databuffer + samplen*i + 5) << 8) | (*(databuffer
+ samplen*i  + 4));
        AccBuffer[i].x = rawBuffer[0] * accelScales;
        AccBuffer[i].y = rawBuffer[1] * accelScales;
        AccBuffer[i].z = rawBuffer[2] * accelScales;
        //gyro reading
        rawBuffer[0] = (int16_t)(*(databuffer + samplen*i +  7) << 8) | (*(databuffer
+ samplen*i +  6));
        rawBuffer[1] = (int16_t)(*(databuffer + samplen*i +  9) << 8) | (*(databuffer
+ samplen*i +  8));
        rawBuffer[2] = (int16_t)(*(databuffer + samplen*i + 11) << 8) | (*(databuffer
+ samplen*i + 10));
        GyrBuffer[i].x = rawBuffer[0] * gyroScales;
        GyrBuffer[i].y = rawBuffer[1] * gyroScales;
        GyrBuffer[i].z = rawBuffer[2] * gyroScales;
      }

      return nsamp;
```

```
    }
```

MAX30105.cpp

from sparkfun/SparkFun_MAX3010x_Sensor_Library

updates start on line 740

```
int8_t MAX30105::getWritePointer2() {
  _i2cPort->beginTransmission(MAX30105_ADDRESS);
  _i2cPort->write(0x04);  // Register for FIFO_WR_PTR
  _i2cPort->endTransmission(false);

  _i2cPort->requestFrom(MAX30105_ADDRESS, (uint8_t)1);
  if (_i2cPort->available()) {
    return _i2cPort->read();
  } else {
    Serial.println("ERROR: Failed to read Write Pointer");
    return 0xFF;  // Return an invalid value
  }
}
uint8_t MAX30105::getReadPointer2() {
  _i2cPort->beginTransmission(MAX30105_ADDRESS);
  _i2cPort->write(0x06);  // Register for FIFO_rd_PTR
  _i2cPort->endTransmission(false);

  _i2cPort->requestFrom(MAX30105_ADDRESS, (uint8_t)1);
  if (_i2cPort->available()) {
    return _i2cPort->read();
  } else {
    Serial.println("ERROR: Failed to read Write Pointer");
    return 0xFF;  // Return an invalid value
  }
}

void MAX30105::ReadFIFO(int16_t* nsamples, uint32_t* FifoBufferIR, uint32_t*
FifoBufferRed, uint8_t ledmode,uint8_t bitwidth) {

  uint8_t wrptr = getWritePointer2();
  _i2cPort->endTransmission(true);
  uint8_t rdptr = getReadPointer2();
  _i2cPort->endTransmission(true);

  if (wrptr == 0xFF || rdptr == 0xFF) {  // Check for I2C errors
    Serial.println("ERROR: Invalid FIFO Pointers");
    return;
  }

  //Serial.printf("wrptr:0x%02X ",wrptr);
  //Serial.printf("rdptr:0x%02X\n",rdptr);
  *nsamples = (wrptr >= rdptr) ? (wrptr - rdptr) : ((32 - rdptr) + wrptr);

  //Serial.printf("nSamples=%d\n", *nsamples);
  //Serial.printf("wrptr=%d\n", wrptr);
  //Serial.printf("rdptr=%d\n", rdptr);

  if (*nsamples == 0) {
    //its probs full, dinner is ready lets eat.
    *nsamples=32;
  }

  _i2cPort->beginTransmission(MAX30105_ADDRESS);
  _i2cPort->write(0x07);             // MAX30105_FIFODATA
  _i2cPort->endTransmission(false);  // Restart condition

  _i2cPort->requestFrom(MAX30105_ADDRESS, *nsamples * 3);  // Request the correct
amount of data
  uint8_t LED1_23_16;
  uint8_t LED1_15_8;
  uint8_t LED1_7_0;
  uint8_t LED2_23_16;
  uint8_t LED2_15_8;
  uint8_t LED2_7_0;
  for (uint8_t i = 0; i < *nsamples; i++) {
    if (_i2cPort->available() < 3) {
      Serial.println("ERROR: Not enough data available");
      break;
    }

    LED1_23_16 = _i2cPort->read();
    LED1_15_8 = _i2cPort->read();
    LED1_7_0 = _i2cPort->read();

    //uint8_t dummy = _i2cPort->read();  // Extra read per sample
    *(FifoBufferRed + i)=0;
    *(FifoBufferRed + i) = (((((uint32_t)LED1_23_16 << 16) | ((uint32_t)LED1_15_8 <<
8) | (uint32_t)LED1_7_0) & 0x03FFFF) >> (18 - bitwidth));// bit shift due to ADC
resolution from LED_PW
    *(FifoBufferIR + i)=0;


  }
}
```

Appendix E: Design Considerations and Standards

Appendix N:

Table 14 N.1 Design Factors Considered

Design Factor	Page number, or reason not applicable
Public health safety, and welfare	Pages 1-4, 7, 14, 26: Device is designed to prevent injury/death from falls and wandering.
Global	Not applicable: The design is intended for localized use in U.S. residential/nursing homes.
Cultural	Not applicable: No cultural adaptation is needed; the solution is generally universal.
Social	Pages 1–2, 26: Reduces social isolation by enabling elderly to stay at home longer.
Environmental	Pages21 21, 26: Discusses electronic waste mitigation and use of biodegradable housings.
Economic	Pages 3, 22-24, 26: Reduces long-term costs of nursing home care; includes cost breakdown.
Ethical & Professional	Pages 4, 21, 25-26: Includes data privacy (HIPAA), device reliability, and ethical concerns.
Reference for Standards	Pages 6, 21, 25: Cites standards like IEEE 802.11 (Wi-Fi), Bluetooth 5, HIPAA compliance.