Body Buddy: Fall Detection Device for Elders

Jacob Hoffman, Sojeong Lee, Yujun Lee, Max Lutwak
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Body Buddy is a fall detection device on a Raspberry Pi connected to a mobile application. It uses IMU sensors to collect 3-axis accelerations, and runs a machine learning algorithm on the data to classify user's activity as either a fall or a normal activity. When a fall is detected, it sends an alert to the user-stored contacts through the mobile application. Falling poses a substantial risk to elderly, and we hope to mitigate the damage done by falls through this project.**

***Index Terms*— Alert System, Fall Detection, Inertial Measurement Unit (IMU) sensor, Machine Learning Algorithm, Mobile Application, Raspberry Pi, Support Vector Machine (SVM)**

1 INTRODUCTION

As the elderly population increases rapidly, there are concerns for them who are facing a medical situation or get lost without any assistance near them. The problem we identified was that a fall can cause serious injuries if no immediate assistance is taken, and the fear of fall limits independent activities and social engagement of the elders. To solve such a problem, we decided to create a fall detection device that users carry in their pockets and sends alerts to the first responders through a mobile application when a fall is detected. There are fall detection devices that are available on the current market, and one example is the fall detection functionality on the Apple Watch. However, Apple Watch is expensive and it includes too many unnecessary functionalities. As we target the elderly users, we want to provide an affordable solution that is easily accessible to them. Philips also sells a fall detection device called GoSafe2 [4] that allows a user to press a button to contact an operator through the 2-way voice system. One drawback of this device is that the users have to pay a monthly subscription fee for an interface that can be shared with friends and family. Body Buddy lets users set up the system much easier by simply downloading a mobile application on their smartphones.

Our goal for this project is to accurately detect falls and promptly notify the users. We want to achieve at least 90% accuracy for the fall detection algorithm. Body Buddy collects 3-axis acceleration with an IMU sensor, performs a real-time fall detection with a machine learning algorithm, and sends a result to a mobile application through Bluetooth from a Raspberry Pi. The processing time of the data will take less than 2 seconds to be able to provide immediate help in case of an emergency. To prevent a false alarm, the alert system in the mobile application allows 2 minutes for users to cancel the alarm. Otherwise, it sends a text message or an email to the saved contacts with the user's current location information. We believe our project Body Buddy can bring an effective way to assist elders with a reliable alert system driven by technology.

2 DESIGN REQUIREMENTS

For a successful fall detection and usability of the Body Buddy device, we have come up with the following requirements.

A. Hardware Requirements

The hardware component of Body Buddy should have an enclosure to protect the embedded system from the falls and to make sure that the device is easily accessible to the users. The enclosure should fit in a user's pocket, which is 6 inches wide and 9 inches deep on average. The device should also be lightweight, as it should not be a burden for the elders to carry around. We have set the weight of the device to be less than 5oz. This number is around the average weight of a smartphone, so we believe that it would be a reasonable weight for a pocket device. We expect the users to use Body Buddy daily, which brings the battery life is also important. We set the requirement for battery life to be at least 10 hours to ensure that the users can use the device for a full day without charging.

B. Fall Detection Requirements

The most important component of this project is the fall detection algorithm. We want the algorithm to detect falls with at least 90% overall accuracy to avoid extraneous warnings or missing a fall. The accuracy will be calculated by running our algorithm with the test data and counting the number of true positives, true negatives, false positives, and false negatives. We believe that 90% accuracy is possible to achieve because the preliminary data that we collected for falls and normal activities show a clear difference in the two categories. We are aiming to focus more on reducing the false negatives than the false positives because false positives can easily be handled by the users, who will be given two minutes to cancel the alarm before it is automatically sent to their contacts.

The latency of the algorithm is another requirement. Although we are giving some time for the users to cancel the alarm, the algorithm should still be able to detect falls as soon as possible in case the falls lead to emergency situations. We want to be able to process the IMU data and send the result through Bluetooth within two seconds, because our alert system should take much less time than the time it takes for a nearby person noticing the fall and calling 911. The latency will be calculated by adding up the time it takes to run the algorithm and communicate

the result to the mobile application. The runtime of the algorithm can be obtained by measuring the execution time on the Python script. The communication latency can be calculated by logging the time that the algorithm finished running and the time that the result was received on the mobile application. The two timestamps can then be subtracted to obtain the time elapsed.

C. Mobile Application Requirements

The main functionalities of the mobile application is to send alert messages to the contacts, and to allow the users to cancel the alarm before sending the alerts to prevent false positives. Although we want our fall detection algorithm to have high accuracy, we do not want to depend solely on the algorithm, so the mobile app will also include an emergency button that allows the users to ask for help whenever they need. For the mobile application, user interface is crucial because our target users are elders. The interface should be intuitive and easy enough for older adults to use. For example, font and button sizes should be large for the users who have vision impairments. The number of buttons and the number of steps that need to be taken to get to different pages of the application should be minimized to help those who might have memory issues. Sending an app notification to remind the users to cancel the alarm is also required to make sure that they do not forget it. We are planning to do user testing on our elderly relatives to evaluate the usability of our mobile interface.

3 ARCHITECTURE OVERVIEW

Below is the block diagram for the overall architecture of our system. The hardware systems are colored green and the software systems are colored yellow.

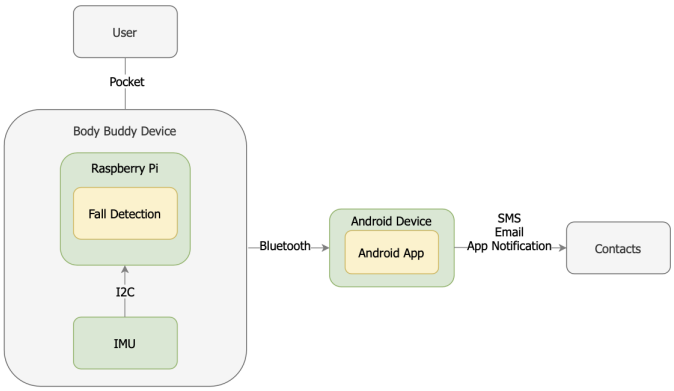


Figure 1: Block Diagram.

The physical system has two components: A Raspberry Pi collecting data from a 3-axis accelerometer over I2C, and a Bluetooth-connected Android smartphone managing the user interface and contacts. The Raspberry Pi is the Zero W model, which is the lowest-power and smallest-size version with WiFi and Bluetooth, powered by a consumer 3000mAh battery.

The software system also has two components: a machine learning system written in Python performing real-time fall detection on the Raspberry Pi, and an Android App providing the user interface as well as sending a variety of alerts when a fall is detected.

We chose Python as a programming language for the machine learning component because it provides libraries such as NumPy and Scikit that are easy to use for the machine learning tasks. It is also the language that we are all comfortable with. Because we wanted to run a Python code, we decided to put the machine learning system on the Raspberry Pi instead of on the mobile app, which can be harder to integrate the Python program.

The machine learning system pipeline consists of four steps, as shown in Figure 2. The input IMU data will first be preprocessed to filter noise that can be caused by the user holding the device or the device moving inside the pocket. Then, the filtered data will be segmented into a window of size 50 because we need to capture a time series to be able to detect a fall. The ADXL345 accelerometer that we are using has a data rate of 100Hz, so the window size of 50 will capture 0.5 seconds of user activity. From our preliminary data collection, this interval was enough to capture the falls, but we can easily change the window size in the future if we find any fall data that does not fit within this interval as we collect more data set. The segmented data is then converted into feature vectors. We are currently experimenting with three different features: the tuple of x, y, z accelerations, the magnitude of acceleration and the angle of the acceleration. For high accuracy, we expect that we will have to combine these feature vectors into a multi-dimensional vector instead of only using one of them. These vectors are the inputs to the Support Vector Machine, and we used the Python Scikit library for the SVM. [2] If the SVM detects a fall, it will then communicate to the Android application through Bluetooth.

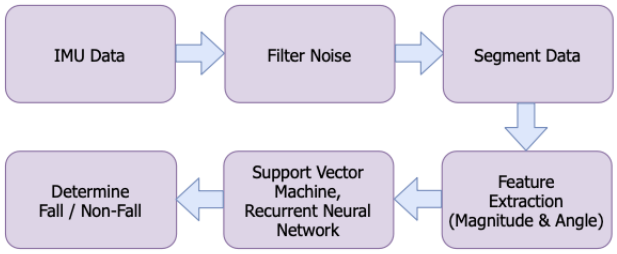


Figure 2: Fall Detection System Flow Chart.

The Android mobile application interface needs to be simple and intuitive as the main users are elders. In the main page, a user can get to all the pages of the application and press a emergency help button whenever they need even if they did not fall and , as shown in Figure 3.

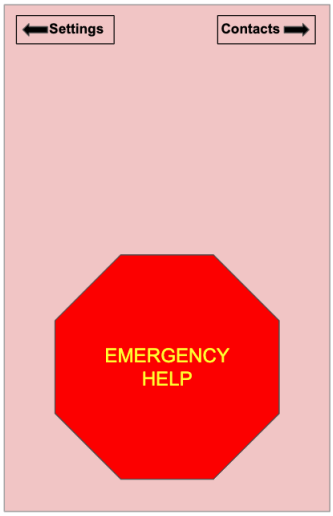


Figure 3: Application Main Page

The settings page is for putting a user's information and showing instructions how to connect a smartphone with the physical device through bluetooth. To send an alert to the user's first responders automatically, the application asks the user to save the contact information, which includes name and phone number. When the application detects a fall sent by the Raspberry Pi or the user pressed emergency button in the main page, it shows a page where user can send an alert to the saved contacts immediately or cancel it in 2 minutes to prevent a false alarm, as shown in Figure 4. If the user pressed the send button or did not cancel the alert within the given time , the application sends a text message to the saved contacts with the user's current location information.

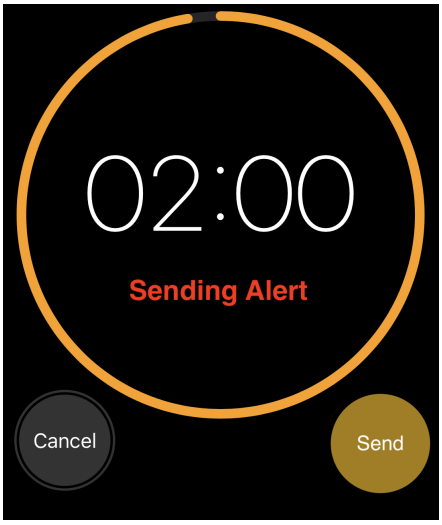


Figure 4: Application Sending Alert Page

4 DESIGN TRADE STUDIES

4.1 ML Algorithm Trade Studies

In the selection of our machine learning algorithm, we considered potential to correctly classify falls as well as wattage required to run our algorithms. After researching the power consumption of RNN's on embedded chips and SVM's on embedded chips, we found that SVM's are 100% more power efficient. SVM's are also faster on embedded chips.

- Power Consumption : WINNER - SVM
- Algorithm latency : WINNER - SVM

Because SVM's were superior in these metrics, we decided to switch our focus solely on SVM's for fall detection.

To increase the performance of the SVM's we will incorporate PCA compression before the SVM, STFT features, and wavelet features

4.2 Design Specification of an RNN on Embedded Chips

RNN's are a class of neural networks designed to preserve temporal information of data sets. Like other deep learning techniques, they can reach very high accuracy but require large amounts of data to train. As well, RNN's have significant overhead on an embedded chip at runtime since inputs need to be propogated through various layers.

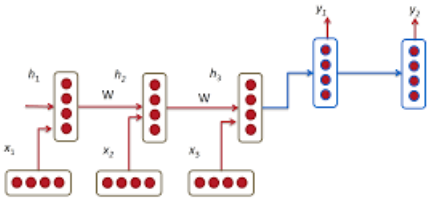


Figure 5: An RNN is so expensive because it is a combination of neural nets with state machines. Neural nets consist of multiple layers of linear transformations that require a lot of wattage to propogate inputs through for classification

Experimentally Found Results :
Latency : 4 Predictions per second [3]
Estimated Wattage : Approximately 4W [3]

4.3 Design Specifications of an SVM on Embedded Chips

In contrast to NN's, SVM's are capable on training on smaller data sets. They handle high dimensional data very well, and are lower power to run on embedded chips.

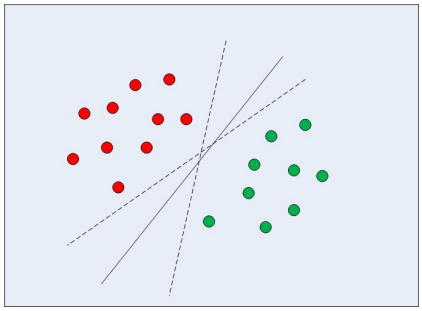


Figure 6: SVM's comparatively require a smaller number of linear transformations to make a classification, They only need to determine the position of features relative to a boundary. They require less operations per classification resulting in lower wattage.

5 SYSTEM DESCRIPTION

Provide one or more overall system and subsystem figures. This should drop into more specific detail compared to the functional diagram described in section III. Specific chips, sensors, interconnects, etc. should be described in this section.

Concisely describe and, if appropriate, depict each major subsystem.

If you use code or algorithms from other sources, make sure to cite them.

5.1 Embedded Subsystem

The hardware subsystem is built on the Raspberry Pi Zero W. This board provides the minimum subset of features to allow Bluetooth connectivity while running a full operating system. Using the Linux System Management Bus (SMBus) protocol we talk over I2C to a board containing the ADXL345 3-axis accelerometer. This device provides data at up to 3KHz [1], which is far faster than we need: we configure it to run at 100Hz. The data is read in through a Python program which either logs the data (for training/tagging) or runs the SVM to perfrom fall detection. Communication to the Android app is accomplished through Bluetooth.

5.2 Fall Detection Subsystem

The fall detection subsystem will detect and classify the falls and normal activities shown in Figure 5. We decided to define the falls and normal activities to limit the kinds of activities that we are going to detect. We also wanted to ease the testing by having clear definitions of the data that we are collecting.

Falls	Normal Activities
Falling forward / backward	Walking
Falling sideways	Running
Falling from stairs	Jumping
Falling on an incline	Lying down
Falling on a decline	Sitting / Bending Down

Figure 7: Fall and Normal Activity Categories.

For feature extraction, we calculated the total magnitude of the acceleration and the angle between the acceleration vectors. The magnitude was calculated using the following equation:

$$\sqrt{a_x^2 + a_y^2 + a_z^2} \tag{1}$$

For the angle, we chose to use the angle between the xy-plane and the z-axis, because this angle changes the most drastically when a person falls. The angle was calculated by first calculating the total magnitude of the x and y accelerations and finding the tangent angle with the z acceleration.

$$\tan^{-1}\left(\frac{a_z}{\sqrt{a_x^2 + a_y^2}}\right) \tag{2}$$

We have collected some sample data using the iPhone 8 Plus and graphed them to show how the falls and normal activities differ and to figure out how the feature vectors are related to the raw data.

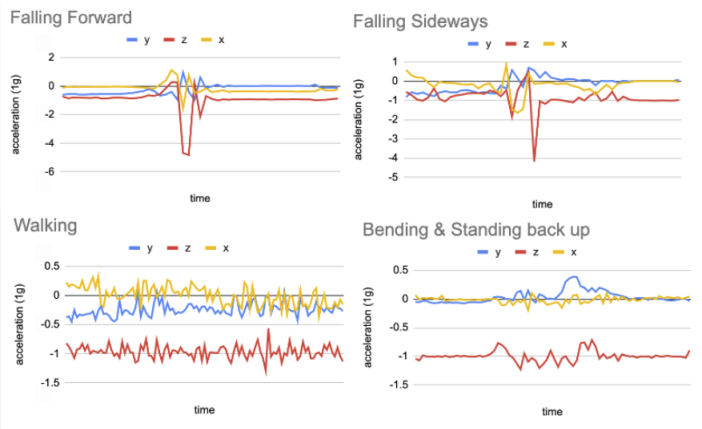


Figure 8: Raw Acceleration Data.

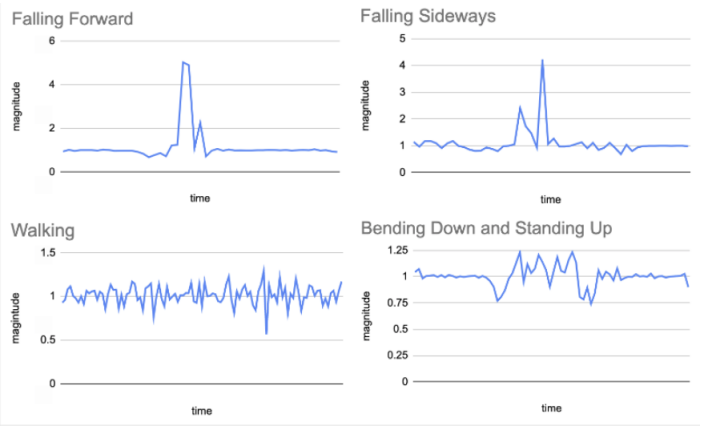


Figure 9: Total Magnitude of 3-axis Acceleration.

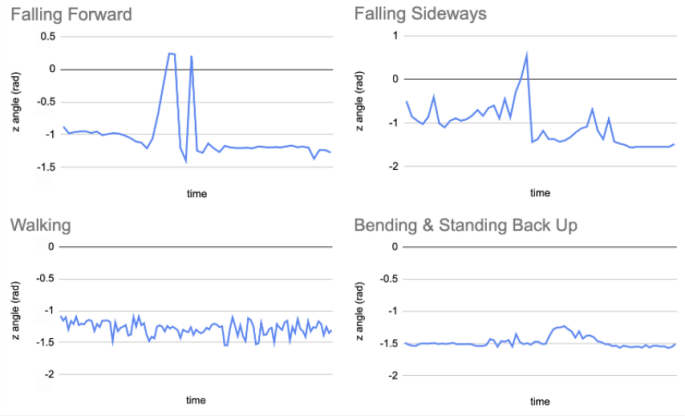


Figure 10: Angle between XY Acceleration and Z Acceleration.

From the graphs, we could easily identify that the falls and normal activities have different shapes, which indicate that machine learning algorithm can also differentiate the two categories. The graphs of the total magnitude and the angle look similar to the raw acceleration graphs, and this shows us that these two features correctly represent the raw data. Although the magnitude and the angle graphs look similar, the peaks of the normal activities in the angle graphs are much smaller. This implies that using the angle as a feature might present better results than using the magnitude.

6 PROJECT MANAGEMENT

6.1 Schedule

Our milestones for each week is described on the Gantt Chart on Appendix A. We had to modify our schedule after the proposal because Jacob joined our team as a new member. We left the weeks before the interim demo and before the final demo as slack to be able to combine as a whole team and integrate our work.

6.2 Team Member Responsibilities

Our team of four has divided responsibilities into three categories: the hardware system, the machine learning system, and the mobile application.

Sojeong is responsible for the machine learning system of the project. She wrote Python code for data segmentation and the SVM classification. She is also working on finding the combination of features that achieves the highest accuracy of the algorithm.

Jacob is responsible for looking into other techniques besides SVM's as a method of fall classification (RNN's) , as well as supporting Sojeong in providing her additional frequency domain features and compression tools to pre process features to improve the SVM accuracy.

Max is responsible for the hardware platform. So far, he has brought up the basic Pi/IMU system and is currently working on establishing a better means of programming the device as well as bringing up Bluetooth.

Yujun is primary focusing for the mobile application system. He is working on the main features of the application and implements a method to alert the first responders effectively after fall is detected by the hardware and machine learning algorithm.

6.3 Budget

Refer to the Bill of Materials on Appendix B. The items that we purchased are for the hardware system and data collection, and we used free libraries for the software system.

6.4 Risk Management

Our major concern with the project is collecting the data set that is large enough to train our machine learning algorithm. In order to do this, we ordered a dummy to easily collect a large data set without us having to fall all the time. Dummy is an easy way to collect large data, but there is a risk that it might deviate from the data collected from actual humans. In order to mitigate this risk, we are also collecting the data of us falling on a gym mat. To make sure that the fall detection works for any user, we are going to collect the test data from people with different weights and heights.

Another concern was the low accuracy of the machine learning algorithm. In order to reduce the accuracy risks, we compared the SVM and RNN approaches to fall detection. We will also try training our model with different set of features (tuple of raw acceleration, magnitude, and angle) and find the combination that achieves the best accuracy.

7 RELATED WORK

Testing method and feature selection for the SVM algorithm was inspired by the paper "SVM-based fall detection method for elderly people using Android low-cost smartphones" [5]. In the paper, they provide formulae for the features that they extracted from their data set. For the IMU, they used an accelereometer on Android devices and achieved 97.7% accuracy for their SVM classification.

References

[1] *ADXL345 Datasheet*. 2008. URL: https://www.overleaf.com/learn/latex/Bibliography_management_with_bibtex#The_bibliography_file.

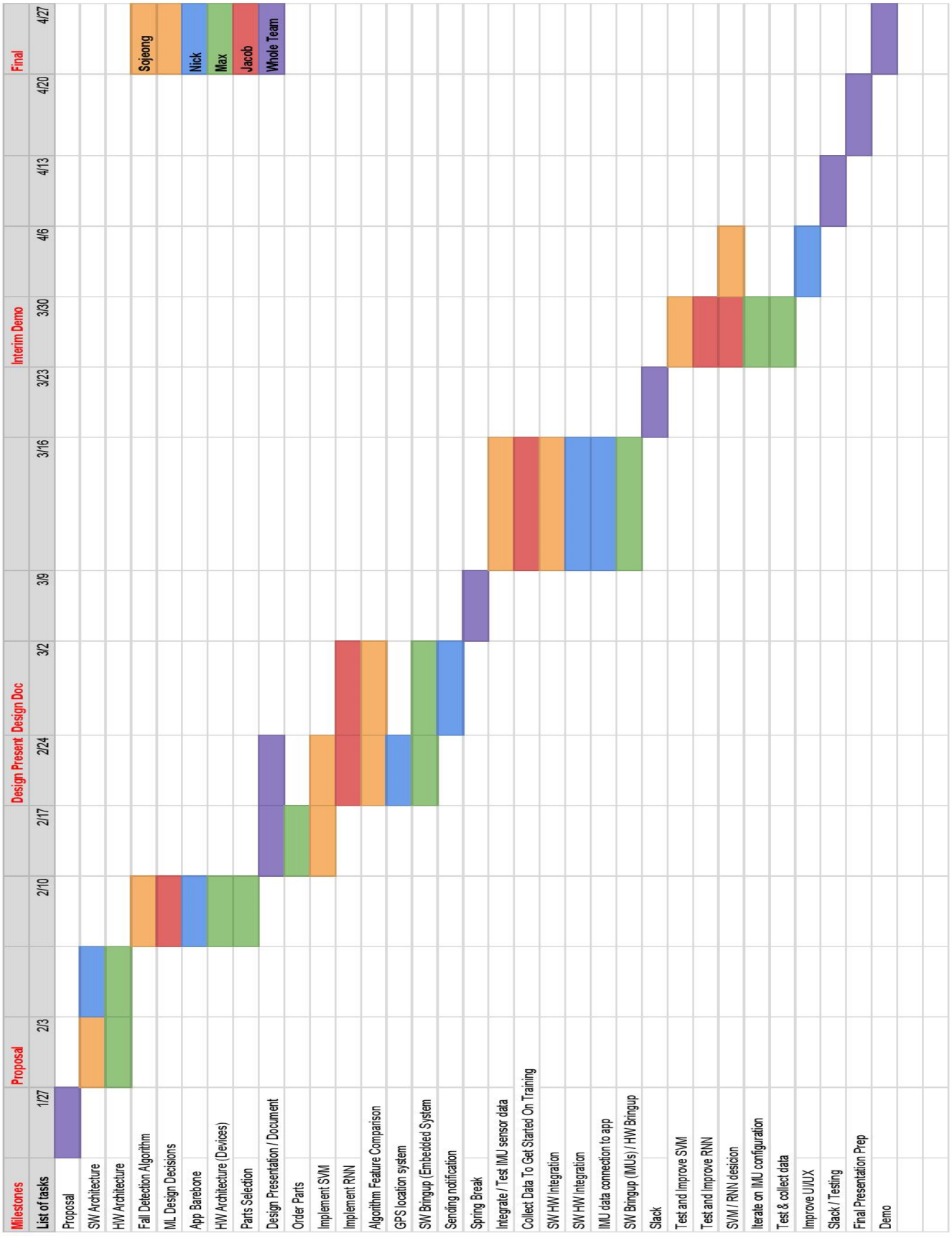
[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] *Performance Analysis of Real-Time DNN Inference on Raspberry Pi*. URL: https://digital.csic.es/bitstream/10261/163973/1/Performance_Analysis_of_Real_Time_DNN_on_RPi.pdf.

[4] *Philips GoSafe2*. URL: https://www.lifeline.philips.com/medical-alert-systems/gosafe-2.html.

[5] P. Pierleoni et al. "SVM-based fall detection method for elderly people using Android low-cost smartphones". In: *2015 IEEE Sensors Applications Symposium (SAS)* (2015), pp. 1–5.

Appendix A - Gantt Chart



,

Appendix B - Bill of Materials

No.	Item	Description	Cost
1	Raspberry Pi	Raspberry Pi Zero W v1.1	$10*
2	Micro SD Card	Embedded Storage	$8*
3	Micro SD Reader	System Setup	$7**
4	IMU Sensor	SunFounder Digital Accelerometer ADXL345 Module	$7*
5	Enclosure	Raspberry Pi Zero W Case	$7*
6	Battery		$26
7	Crash Dummy	From Amazon, for Data Collection	$40
8	Gym Mat	From Amazon, for Data Collection	$20
		TOTAL	$125

* From Capstone Inventory
** Personal Property

,