Self-Activating Fall Alert

Bradley University Department of Electrical and Computer Engineering

BRADLEY
University

Hayley Langley and Sean Miller
Advisor: Dr. Mohammad Imtiaz

Abstract

With the ever-growing population of elderly citizens, the rates of the elderly that are being treated for a fall is increasing rapidly. Every 11 seconds an elderly person is treated in a hospital for having fallen. An elderly person dies every 19 minutes from falling and not having proper medical attention. According to the Center for Disease Control and Prevention, 1 out of every 5 falls, in the elderly population, results in a serious injury. From these statistics, it is quite relevant that falling as an elderly person and not getting medical treatment right away could end with severe consequences. This indicates that this is an urgent problem that is needs a solution. Current solutions to the problem of elderly citizens falling relies heavily on the wearer to remain conscious. These products, such as Life Alert, need the elderly person to be able to interact with this device. Our Self-Activating Device takes away the need for the elderly person to interact with the device to get the immediate help that is required. The self-activating fall alert device is a wearable system that is dependable, user-friendly, self-activating and provides post-fall processing. The system aims to minimize the time to receive help after falling. The project aims to provide enough information from the data collection process to give to medical professionals who will be able to help with corrective measures for the wearer of the device. The device system aims to be able to make the distinguish between someone who has become unconscious and non-mobile due to a fall and someone who is just resting.

Acknowledgements

The completion of our project would not have been possible without our wonderful advisor. Thank you, Dr. Mohammad Imtiaz for your continuous support, patience, trust and encouragement throughout the entire project. We are grateful for your skills and expertise which have guided us during this process.

We would also like to thank Mr. Mattus and Mr. Schmidt for ordering hardware components and soldering pins onto the components. As well as Mrs. Polen for always being around and opening the labs.

Table of Contents

1. Introduction **8**
1.1 Problem background 8
1.2 Problem Statement 8
1.3 Problem Constraints 8
1.4 Related Work 9
2.1 Hardware System Block Diagram 10
2.2 Software System Block Diagram 10
2.3 High-level system flowchart 11

3. Hardware Components **12**
3.1 Inertial Measurement Unit 13
3.2 Bluetooth Mate Gold 13
3.3 Lithium Ion Battery and Charger 14

4. Method of Solution **14**
4.1 Software Design 14
4.1.1 MATLAB Code 15
4.1.2 Python Code 16
4.1.3 Sending SMS Alerts 16
4.1.4 Comparison of Previous and Current Detection Algorithm 17
4.2 Software Difficulties 17
4.3 Hardware Design 17
4.3.1 Bluetooth 18
4.4 Hardware Difficulties 18

5. Results **18**
5.1 Overall Results 18
5.2 Bluetooth Device Streaming 19
5.3 Real-Time Fall Detection 19
5.4 Fall Alert Messages 19
5.5 Data Logging - Circular Buffer 19
5.6 Power Management 20
5.7 Neural Network 20
5.7.1 Deep Neural Networks 20
5.7.2 Recurrent Neural Networks 20

6. Discussion **21**

7. Conclusion **21**

8. Reference **23**

9. Appendices **24**

Appendix A: 24

Data Logging to microSD Card 24

Importing of Data 25

Post Processing Fall Detection Algorithm 26

Phone Alert 26

Plotting IMU Sensor Data 27

Plotting of Post Processing Fall Detection 28

Appendix B: 28

Walking 28

Walking with a Fall 29

Standing with a Fall 31

Sitting 32

Sitting with Minimal Movement 32

Table of Figures

Figure 1 - System Hardware Block Diagram 10
Figure 2 - System Software Block Diagram 11
Figure 3 - System Flowchart 12
Figure 4 - 9 Degrees of Freedom Inertial Measurement Unit 13
Figure 5 - Bluetooth Mate Gold 13
Figure 6 - Lithium Ion Battery and LiPo Charger 14
Figure 7 - Text File from microSD Card 15
Figure 8 - Raw Accelerometer Data 15
Figure 9 - Magnitude of Normalized Accelerometer Data 16
Figure 10 - Fall Detection based on Accelerometer Data 16
Figure 11 - Fall Detection System and Attachable Case 19
Figure 12 - General Layout of a Three Hidden Layer Neural Network 20
Figure 13 - Example of a Recurrent Neural Network 21
Figure 14 - Walking - Raw Accelerometer Sensor Data 28
Figure 15 - Walking - Raw Magnetometer Sensor Data 28
Figure 16 - Walking - Magnitude of Accelerometer Sensor Data 29
Figure 17 - Walking - Magnitude of Magnetometer Sensor Data 29
Figure 18 - Walking with a Fall - Raw Accelerometer Sensor Data 29
Figure 19 - Walking with a Fall - Raw Magnetometer Sensor Data 29
Figure 20 - Walking with a Fall - Magnitude of Accelerometer Sensor Data 29
Figure 21 - Walking with a Fall - Magnitude of Magnetometer Sensor Data 29
Figure 22 - Walking with a Fall - Fall Detection 30
Figure 23 - Standing - Raw Accelerometer Sensor Data 30
Figure 24 - Standing - Raw Magnetometer Sensor Data 30
Figure 25 - Standing - Magnitude of Accelerometer Sensor Data 30
Figure 26 - Standing - Magnitude of Magnetometer Sensor Data 30
Figure 27 - Standing with a Fall - Raw Accelerometer Sensor Data 31
Figure 28 - Standing with a Fall - Raw Magnetometer Sensor Data 31
Figure 29 - Standing with a Fal - Magnitude of Accelerometer Sensor Data 31
Figure 30 - Standing with a Fall - Magnitude of Magnetometer Sensor Data 31
Figure 31 - Standing with a Fall - Fall Detection 31
Figure 32 - Sitting - Raw Accelerometer Sensor Data 32
Figure 33 - Sitting - Raw Magnetometer Sensor Data 32
Figure 34 - Sitting - Magnitude of Accelerometer Sensor Data 32
Figure 35 - Sitting - Magnitude of Magnetometer Sensor Data 32
Figure 36 - Sitting with Minimal Movement - Raw Accelerometer Sensor Data 32

Figure 37 - Sitting with Minimal Movement - Raw Magnetometer Sensor Data **32**
Figure 38 - Sitting with Minimal Movement - Magnitude of Accelerometer Sensor Data **33**
Figure 39 - Sitting with Minimal Movement - Magnitude of Magnetometer Sensor Data **33**

Table of Tables

Table 1 - Project Constraints **9**

1. Introduction

In the following sections, the problem and solution will be discussed. There will also be discussion about the constraints that are faced.

1.1 Problem background

Falls present a very serious issue for senior citizens with one fifth of all falls causing a serious injury. These injuries cost a large amount of both time and money. Annually 2.8 million people are seen in the emergency room for falls; 800,000 of them will need to be hospitalized for injuries received from that fall. These falls cost 31 billion dollars annually. Additionally, these falls cause serious injuries to senior citizens as 95 percent of hip fractures are caused from falling. These falls could also cause some sort of traumatic brain injury [1]. Furthermore, one in three seniors has Alzheimer's disease or some other form of dementia at the time of their death. There are more than 5 million Americans currently living with Alzheimer's and this number is projected to approach 16 million by 2050. These dementia patients are just as susceptible to falling as any other senior [2]. Medical devices, such as Life Alert, naturally assumes the user knows that they are wearing the device, and hence have the ability to interact with it. In the case of dementia patients, this assumption fails. Medical devices have been developed to contact assistance for fallen seniors, but these devices provide no post processing. Our ultimate objective is to add diagnostic capabilities to an autonomous fall detection algorithm. In doing so, we believe that we may significantly lower the amount of time, and money that society spends treating falls.

1.2 Problem Statement

Due to modern medicine, people are living longer than before. Which has resulted in a drastic increase in the senior citizen population. There exists a considerable population of senior citizens who have dementia or some other disease that affects their balance and coherence. If these senior citizens fall then they may need medical attention, as such it becomes necessary for them to be able to contact help. Several systems exist today that allow a fallen senior citizen to contact assistance. However, several of these citizens also have conditions, which make them unaware of their surroundings, and therefore they may be incapable of contacting help. Additionally, there is substantial evidence suggesting the longer a senior citizen is left unaided after a fall, their injuries will become more severe. As such, it becomes necessary to design systems that can detect when someone falls, and automatically contacts assistance, therefore minimizing their time left unaided.

1.3 Problem Constraints

The self-activating fall alert constraints are shown on Table 1 on page 9. The IMU must contain an onboard microcontroller which can process the data from the inertial sensors and connect to Bluetooth. The sensors need to be a triple-axis accelerometer, triple-axis gyroscope, and triple-axis magnetometer. It also needs to have a microSD card slot to allow for data logging. The Bluetooth must be able to stay connected to the IMU and transmit serial data at a range of 100 feet. The Lithium Ion Battery must supply a 3.3-volt supply to the IMU, last more than one hour, and it needs to be rechargeable.

Table 1 - Project Constraints

Constraints	
Inertial Measurement Unit (IMU)	Microcontroller, MicroSD, Accelerometer, Gyroscope, and Magnetometer
Bluetooth Connectivity	At least 100 feet of range
Lithium Ion Battery with Charger	Provide 3.3 V and rechargeable

1.4 Related Work

Through the research that was done previously and current research, it was apparent that a triple-axis acceleration, gyroscope, and magnetometer sensor would be the major components of the device. The utilization of both the acceleration and magnetic field data would be the foundation of the fall detection algorithm.

The research showed that the analysis was done using the ADXL345, which includes several built-in features such as motion status detection, flexible interrupts, three-axes accelerometer, GPS service, and GSM communication, to create an algorithm to detect falls [5,6]. The location of the ADXL345 to get the most accurate data was found to be on the waist [5]. The comparison of the different acceleration for the different daily movements, such as sitting down, standing up, and falling, was one of the more important aspects for the research. Being able to identify the changes in the acceleration between the daily motions helps distinguish what type of data is generated during a fall versus a non-fall. For example, sitting down had a change in acceleration that would be sudden and drastic, whereas a walking has a minimal change in acceleration since the movement of the elderly movement was relatively slow [5]. From the research, it was observed that a fall has a sharp peak when the fall occurs.

A fall-detection algorithm is based upon a threshold calculated through the summation of the three acceleration axes (magnitude), |a|, and the rotation angle [6]. A fall can be identified by this threshold. However, only using the magnitude to detect falls would be insufficient and give false positives. A few examples of motions done by a person wearing the device that can produce a high peak that has the potential to reach the threshold include jumping or sitting. From this, it means another layer is needed for a proper fall diagnose. This layer should include the angle calculated based on the acceleration measurement. “Separating the gravity component from before and after a fall using a low pass filter, can be used to calculate the rotation angle of the accelerometer coordinate in the 3D space. The rotation angle of the accelerometer is equal to the rotation angle of the gravity vector, g, relative to a fixed coordinate system.” [6]

After the device start to continuously collect data, the threshold magnitude is compared to the real-time magnitude. So, if $|a_{current}| \geq |a_{threshold}|$, a fall potentially has occurred. The next information to analyze is the rotation angle. If the rotation angle proves that the user’s orientation is toward the ground, then the device will send out an SMS for help.

2.Technical Specifications

In the following sections, the hardware, software, and high-level flowchart are discussed.

2.1 Hardware System Block Diagram

The system block diagram of the self-activating fall alert, shows the connections between the hardware components. The overall block diagram of the self-activating fall alarm is shown in Figure 1 below.

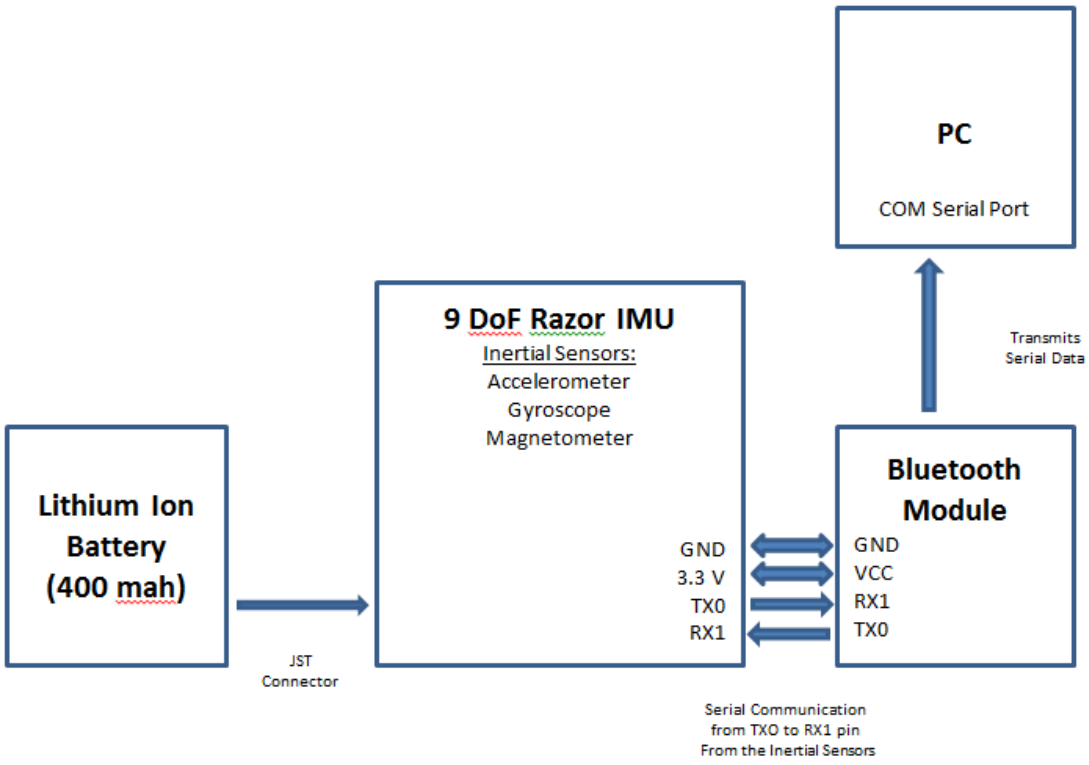


Figure 1 - System Hardware Block Diagram

2.2 Software System Block Diagram

The software block diagram of the self-activating fall alert is shown on the next page in Figure 2. This block diagram displays the hardware components and how the components are interfaced with the software. The system connects to the Bluetooth module using the Serial Monitor of Arduino Sketch. It uses a specified serial COM Port on the computer; this establishes the transmitting connection from the IMU to the computer. This connection allows for the data to be displayed onto the computer. There is also a built-in microSD card that the data gets logged and saved upon. Once the data is saved to the SD card, the data is taken off and placed onto the computer. From here the data is imported into a MATLAB program which plots the data from the inertial sensors. The data is also imported into a Python program which processes the data and detects the falls.

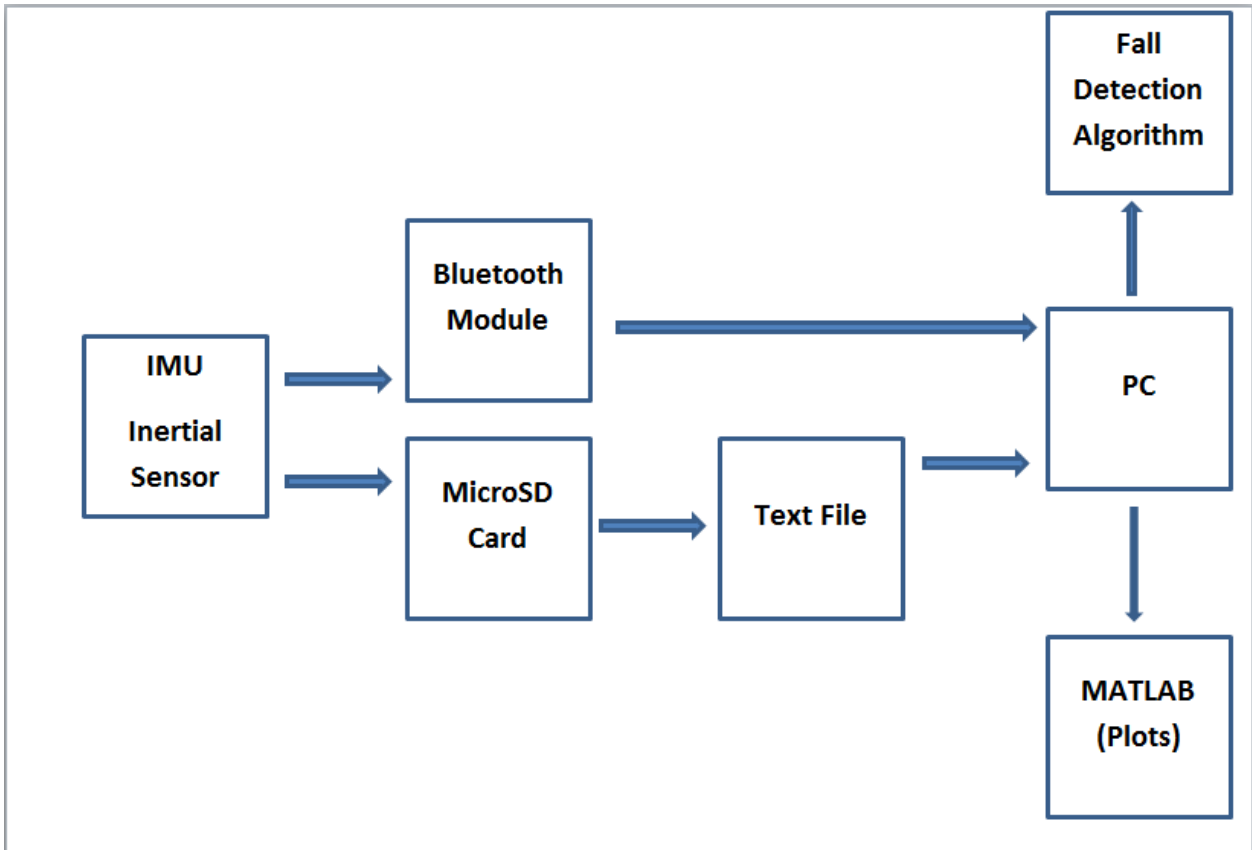


Figure 2 - System Software Block Diagram

2.3 High-level system flowchart

The high-level system flowchart shown in Figure 3 on the next page, displays the logic process of the self-activating fall alert. When the IMU is powered up, the Bluetooth will create a connection to the computer. Once the serial monitor is open and the proper command (t, a, m, g) is sent the IMU starts to transmit and save the data. The following portions are of the algorithm and plotting of the data.

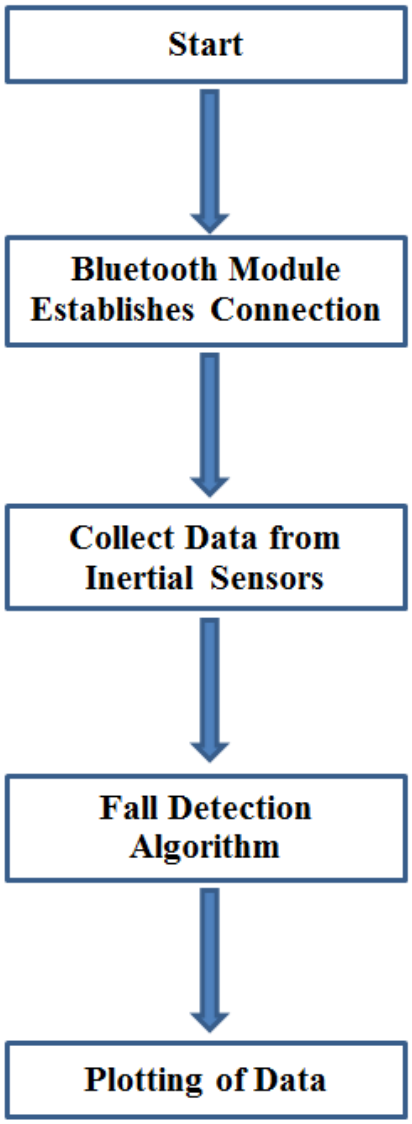


Figure 3 - System Flowchart

3. Hardware Components

The Hardware Components in the project are the Inertial Measurement Unit, Bluetooth Mate Gold, and the Lithium Ion Battery. Also included are the LiPo Charger, various wires for connecting the IMU and Bluetooth Mate, and a breadboard.

3.1 Inertial Measurement Unit

The SparkFun 9DoF (9 Degrees of Freedom) Razor IMU includes a SAMD21 microprocessor with an MPU-9250 9DoF sensor. The MPU-9250 has three 3-axis sensors that are an accelerometer, gyroscope, and magnetometer. The output of these sensors give data in linear acceleration, angular rotation velocity, and magnetic field vectors. The unit also includes a microSD card socket, which will give the ability to save the data to the device.

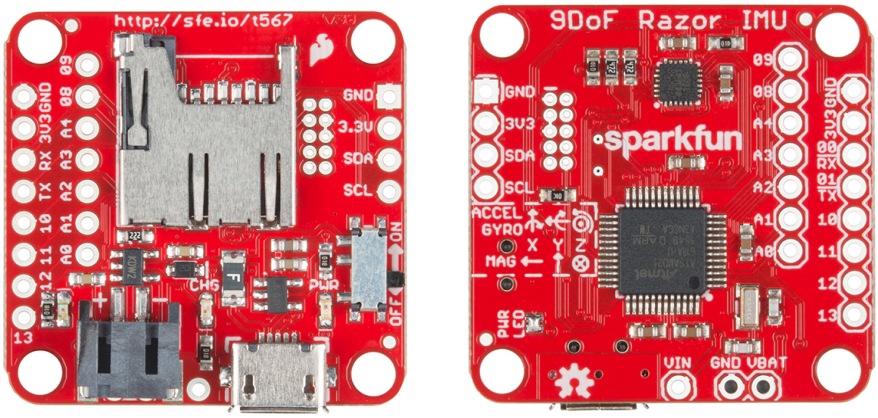


Figure 4 - 9 Degrees of Freedom Inertial Measurement Unit

3.2 Bluetooth Mate Gold

The RN41 Bluetooth Mate Gold, has a 2.4 GHz frequency, is 13.4mm by 25.88mm by 0.2mm with a baud rate of 1200 bps up to 921 Kbps. When connected the onboard LED blinks at various speeds which imply the state of connections [3]. The Bluetooth Mate Gold transmits the data from the IMU sensor to the computer using wireless serial communication. The Bluetooth Module connect to the RX/TX pins of the IMU and then connects to the computer's COM serial port to link the IMU to the computer.

Figure 5 - Bluetooth Mate Gold

3.3 Lithium Ion Battery and Charger

The device system contains a Lithium Ion Battery that allows the system to be wireless and charged by a LiPo battery charger. The battery is lightweight and small with a Lithium Ion chemistry with a high energy density. Each cell outputs a nominal 3.7V at 400 mAh [3]. The battery has a 2-pin JST-PII connector with a 2mm spacing between pins. The LiPo charge is a simple charging circuit that allows the user to charge the 3.7V LiPo cells at a rate of 100mA or 500mA. The charge is designed to charge single-cell Li-Ion or Li-Polymer batteries [3].

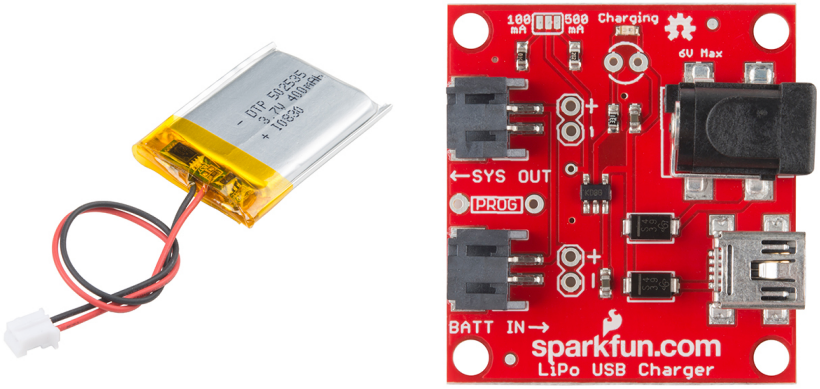


Figure 6 - Lithium Ion Battery and LiPo Charger

4. Method of Solution

The following sections are about the difficulties faced when working with the software and the hardware. As well as the design for the software and hardware.

4.1 Software Design

The first portion of this project was to get the previous year's code back to working. To do this the firmware for the Inertial Measurement Unit needed to be uploaded to the IMU from the using a FTDI cable. The Razor AHRS Firmware, was found on a GitHub, is an Arduino based software package that programs the on-board inertial sensors within the IMU. Once the change to the new device was made, the same process was used to get the device to the point of being able to use for data collection.

After the firmware was uploaded using Arduino, the serial monitors within the Arduino Console Window it displays the serial output coming from the IMU. The data is also saved onto the microSD code. This data output can be seen in the figure on the next page.

```
time_ms,  ACC_x,  ACC_y,  ACC_z, MAG_x,  MAG_y,  MAG_z, GYR_x,  GYR_y,  GYR_z
  63391583,  -53.0, -232.0,   41.0,   278.0,  -512.0,   217.0,   -14.0,    32.0,    -9.0
  63391615,  -54.0, -231.0,   41.0,   279.0,  -529.0,   221.0,   -14.0,    32.0,    -8.0
  63391649,  -54.0, -231.0,   40.0,   280.0,  -510.0,   220.0,   -15.0,    33.0,    -8.0
  63391651,  -53.0, -231.0,   41.0,   281.0,  -513.0,   219.0,   -14.0,    33.0,    -9.0
  63391685,  -52.0, -231.0,   41.0,   278.0,  -511.0,   219.0,   -14.0,    33.0,    -9.0
  63391688,  -54.0, -230.0,   42.0,   280.0,  -514.0,   220.0,   -15.0,    32.0,    -8.0
  63391728,  -53.0, -231.0,   41.0,   280.0,  -509.0,   220.0,   -14.0,    32.0,    -9.0
  63391730,  -53.0, -232.0,   41.0,   274.0,  -516.0,   224.0,   -15.0,    31.0,    -8.0
  63391797,  -54.0, -231.0,   41.0,   283.0,  -509.0,   220.0,   -14.0,    31.0,    -8.0
  63391799,  -53.0, -230.0,   42.0,   283.0,  -511.0,   220.0,   -13.0,    32.0,    -9.0
  63391800,  -54.0, -230.0,   41.0,   281.0,  -513.0,   222.0,   -15.0,    32.0,    -9.0
  63391849,  -53.0, -232.0,   42.0,   280.0,  -514.0,   222.0,   -16.0,    32.0,    -8.0
  63391863,  -53.0, -230.0,   41.0,   278.0,  -535.0,   222.0,   -13.0,    32.0,    -9.0
  63391863,  -53.0, -231.0,   42.0,   279.0,  -512.0,   221.0,   -14.0,    32.0,    -9.0
```

Figure 7 - Text File from microSD card: Time, Accelerometer, Magnetometer, and Gyroscope Values

4.1.1 MATLAB Code

The MATLAB script takes in the data from the saved files from the microSD card. The data then gets reduced for better plotting and data analysis. The reduction of the data is less than 10% of the original data. The data is then separated into time and the three sensors respectively. This allows for the data to be analyzed separately. After the separation, the data is analyzed into the normalized values, magnitude values, and the raw data. The next portion of the code is the fall detection. This algorithm is used only in post processing since there is no real-time values at this point. The fall detection algorithm takes the magnitude of the accelerometer data and compares it to a calculated threshold. The threshold is based on the maximum and minimum of the data. The last portion of the script is the SMS being sent out. This is based on whether the script detected a fall. The fall is detected by comparing the smallest slope value of the data to the calculated threshold. In the following figures it can be seen the raw, magnitude, and fall detection of the post processing results. For code, view appendix A.

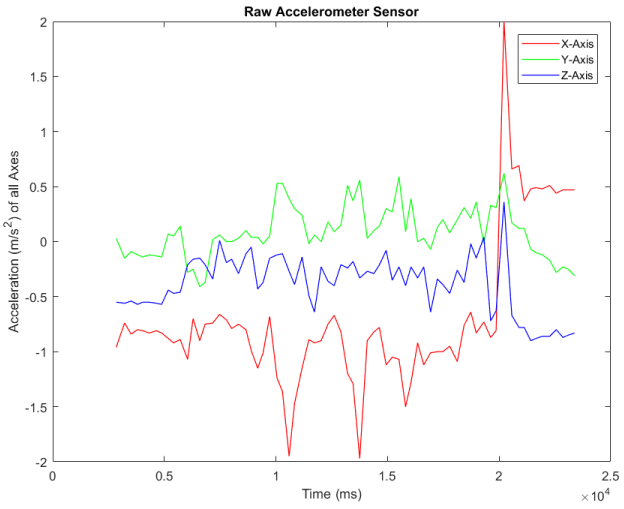


Figure 8 - Raw Accelerometer Data

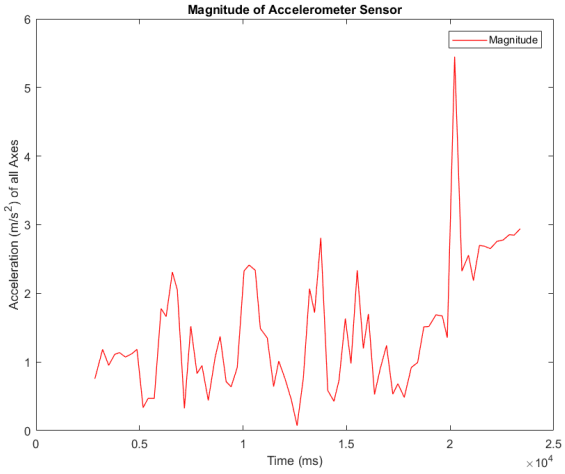


Figure 9 - Magnitude of Normalized Accelerometer Data

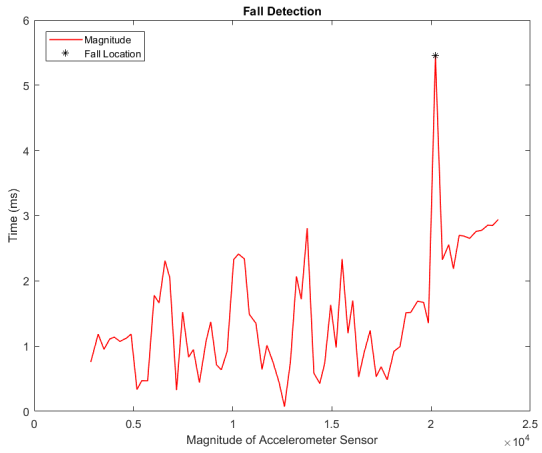


Figure 10 - Fall Detection based on Accelerometer Data

4.1.2 Python Code

Using a built-in Python library, numpy, all imported data gets stored into an array. The next step is to calculate the magnitude of the accelerometer sensor data and compare it to a threshold. For a second comparison, the algorithm uses the relative error of the acceleration compared to a second threshold. The relative error is based on the previous and current acceleration data point from the dataset. To determine and optimize the thresholds the script runs a differential evolution algorithm. This allows for the fall detection error to be minimized. It is important to note that the small amount of variables being optimized will not cause overfitting of the data and will not result in any issues. During the process of developing this algorithm, it was found that the magnetometer, and the accelerometer are both capable of producing accurate results. However, at this point only the magnetometer is being utilized.

4.1.3 Sending SMS Alerts

Using a SMTP server for sending email messages that are converted into a SMS message. This is triggered by the fall flag being high. For this to work, an email addressed to a phone number connected to

a carrier's email, which will automatically convert the message to a SMS that is sent to the recipient's phone. The number of messages that get sent out is based on the number of emergency contacts that are specified at the beginning of the script.

4.1.4 Comparison of Previous and Current Detection Algorithm

The current algorithm utilizes two optimal thresholds. These optimizations were originally performed on the single threshold algorithm that was previously being used. During the process of optimizing the number of false positives that occurred, it was found that the previous detection algorithm had the capability to correctly identify whether a fall had occurred in 23 of the 30 SD card datasets. Based on these findings, it was determined that the success rate was 76.67%. The current algorithm has the capability of correctly identifying whether a fall has occurred in 27 of the 30 SD card datasets. This algorithm currently has a success rate of 90%. Both of these algorithms were using the accelerometer readings prior to observing that the magnetometer had better accuracy. Using the data provided by the magnetometer the current algorithm was about to correctly identify 26 of the 30 datasets. Based on these findings using the magnetometer and the current algorithm the success rate was 86.67%. Even though this success rate is lower than the success rate when using the accelerometer it is still a significant improvement comparatively to the old algorithm.

After further testing of the optimized thresholds, it was determined the higher accuracy was needed. Due to this determination a neural network has been designed. The designed neural network was a Deep Neural Network(DNN) with three hidden layers and a feedback node. To test the network, the data was partitioned. One dataset from each directory were allocated for testing and the remaining two were used for training. The inputs of the neural network were the magnitude of both the magnetometer and accelerometer output readings. At this time the network is not successful due to not having enough datasets for testing and training of the neural network. The neural network at this time only works approximately 40% of the time on the datasets.

4.2 Software Difficulties

The software difficulties that were faced are getting the device up and running, data collection using Bluetooth, logging to the microSD card, and setting up how the data is saved. To solve the issue of getting the device up and running was to upload the current firmware for the device onto the device. For collecting data while using the Bluetooth, it was not solved until the baud rate was changed to the proper option. Currently, the Bluetooth communication still does not accurately work. The serial monitor just displays random symbols. Further researching and adjustments are needed to solve this option. Logging to the microSD card was difficult to work due to the lack of experience with data logging on microSD cards. From doing research and making proper adjustments to the firmware. After re-uploading the firmware to the IMU, the logging to the microSD was completed. The setting up of the data was simply a change in the firmware. Additionally the optimization algorithms had to be smoothly written, so that run-time was not excessive when selecting parameters. There was also found to be insufficient data to properly train a network.

4.3 Hardware Design

The hardware design was just a choice of use the old device or to upgrade to the newest version of the IMU. It was a simple decision of go with the newest version. One item that was needed to be considered was if the Bluetooth module would still be compatible.

4.3.1 Bluetooth

The Bluetooth connection is established via a serial connection (virtual COM port) of a laptop. The connections between the IMU and the Bluetooth module are VCC, GND, RXI, and TXO.

4.4 Hardware Difficulties

Most of the hardware difficulties consisted with connecting the Bluetooth module and the new device. It was discovered that there were two VCC pins on the new IMU; this means that it needed to be determined which was the proper pin to use. All other connections were the same as before.

5. Results

In the following sections, the results of everything that was worked on is discussed and determined whether it was fully functioning by the end of the semester.

5.1 Overall Results

For data collection, the test cases that were determined to be useful included walking, running, sitting down, lying down, standing, standing up and falling for all appropriate cases. A portion of the data collections from post processing are shown in appendix B.

The test cases were plotted using MATLAB. The X-axis of each of the plots are Time. The Y-axis is dependent on what plot, so it is either the raw data of the sensor, magnitude of the sensor data, and normalized data for each sensor. The plots were then analyzed based on the ‘peak acceleration’ to distinguish the type of motion that the user has performed. Based on the peak values it can be determined whether there was a fall or not.

During the data collection process, the device system was contained in an attachable 3D printed box. The box has the measurements of 3.5” x 2.5” x 3”. The reason for the box to be 3D printed is to help minimize the Bluetooth interference that was inexperienced with the old metal box. It is also able to be attached with a belt for the wearer of the device. Since it was attachable to the waist, the data was able to be consistently measured at the same position. In the figure on page 19, the device and attachable case can be reviewed.

Figure 11 - Fall Detection System and Attachable Case

5.2 Bluetooth Device Streaming

The Bluetooth data streaming was a large challenge to overcome. It was not fully overcome by the end of the semester. When initially trying to get the device to connect via Bluetooth to a laptop, it wasn't wanting to stay connected. To fix this issue that was reoccurring, it required less intermittent connections from the IMU to the Bluetooth module. The next step was to get the transmitted data to display on the output serial monitor. For this to happen, the baud rate needed to be set to the default value of 115200 bits per second instead of the 9600 bits per second.

5.3 Real-Time Fall Detection

The real-time fall detection is like the post analysis fall detection algorithm. It is still based on the same threshold and comparison of magnitudes. It takes the streaming data and compares to the threshold continuously. It is still currently in testing and optimization phases.

5.4 Fall Alert Messages

The fall alert messages are SMS based messages that get sent from a SMTP server using an email and the recipient's phone carrier. For this portion of the code to work the flag for if a fall detection must be set high. Once the flag is set high, the code runs through a for loop of contacts, verifying the length of the number is valid. Once this is true a message gets sent to the server and then the server will convert it from an email format to an SMS format and send that message to the recipient. With having a for loop that runs through any number of contacts, the system can contact as many emergency contacts that are programmed.

5.5 Data Logging - Circular Buffer

For data logging, all the data gets saved to the microSD card that is inserted into the IMU. The microSD card size can be of any size, during this project a 32 gigabyte was used. However once that space is filled,

it can cause a lot of problems, especially if a fall were to happen after the microSD card was filled. To handle this issue a circular buffer was designed to delete the first log that does not have a fall.

5.6 Power Management

Power management was a great accomplishment that was needed to be completed. For the first step of this process was to find what causing the drainage of the battery. While doing this it was also found that the gyroscope was not needed. So, the ability of not collecting data using the gyroscope gave some needed time. Another way to help gain more time is to use a less power dependent form of data transmission. There was an improvement from 6 hours to 17 hours worth of battery life.

5.7 Neural Network

To solve the problem of false positives for fall detection, it was determined that an artificial neural network would be an effective solution and classifier for the falls. Since fall detection is a time-based problem a recurrent neural network would be the best option. Another portion of the neural network solution was to have a deep neural network. The training for the neural network was performed by classifying every sample point within a dataset. If any of the samples contained a fall, then the dataset would be identified as a fall dataset. This identification would then be cross referenced with other datasets that contained falls, this allows for the error to be calculated. Using the Scientific Python function, backpropagation was used to minimize and potentially reduce the error. The following figure shows a general three hidden layout, which is how the neural network was designed.

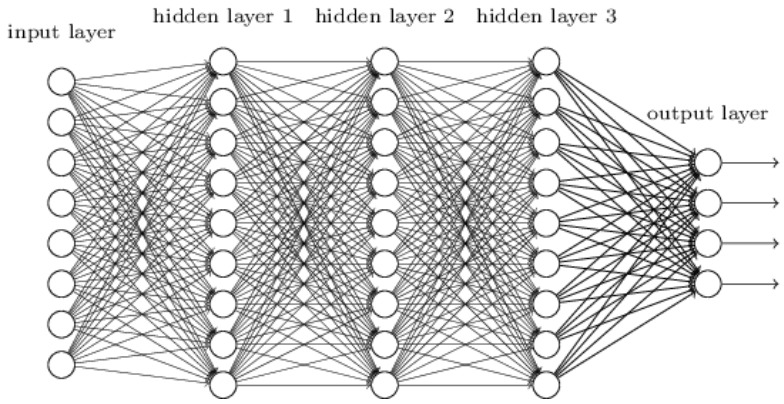


Figure 12 - General Layout of a Three Hidden Layer Neural Network

5.7.1 Deep Neural Networks

A Deep Neural Network functions by having multiple hidden layers between the input layer, and the output layer. These hidden layers provide significantly more parameters for optimization allowing the network to accomplish more difficult tasks. The network designed for this project had 3 hidden layers with 12 nodes each. Additionally, every layer possessed one bias node.

5.7.2 Recurrent Neural Networks

Recurrent Neural Networks(RNNs) are networks which consider not only the current input, but previous outputs as well. A RNN would be useful for sequences that are a function of time, such as the one designed for this project. This particular network possessed three feedback nodes which each

corresponded to one of the previous three outputs of the network. These nodes were then each assigned their own weights which were also optimized during backpropagation. The following figure shows what a RNN would potentially look like.

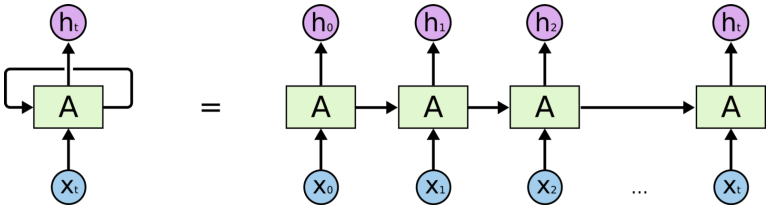


Figure 13 - Example of a Recurrent Neural Network

6. Discussion

Through the observation of the testing results, it can be concluded that a fall will generate an acceleration value between 4 m/s^2 and 6 m/s^2. These values are greater than the 1.5 m/s^2 acceleration that was observed for normal motions, such as walking, sitting, and running. A normal acceleration for the common motions of movement were from 1.5 m/s^2 to 3 m/s^2. This allowed for the peak threshold to be 4 m/s^2. This threshold was chosen to eliminate false positives.

It was discovered through the testing process that the magnetometer readings were useful for finding a where a fall has occurred. Once this was discovered the fall detection algorithm was updated to use these values. In the future, a better fall detection algorithm will include a threshold for both the acceleration and magnetometer data.

Another observation that was found was that the placement of the device on the user could give skewed results depending on how the user falls. So, if the user were to fall on the same side as the device then the data would have a higher magnitude because there was not a dampening factor of a human body. A better way to handle this situation would to have the device at the center of mass of the user. The issue with this is that depending on different movements the center of mass is in a different location on a human body. That is why for testing purposes we determined the waistline be the best fit at this time.

7. Conclusion

Over the past two semesters of working on this project, the objectives and goals were successfully accomplished. The objectives of this project were to minimize the size of the device system, maximize the battery life, eliminate false positives of the fall detection algorithm, and provide data for future corrective measures. There were also a few additions that were made that were not originally in the plan of action. These additions consisted of an attachable case, microSD card data logging, and circular buffer system for data saving.

To minimize the device, we ended up using a single board system, where the battery could directly plug into the device instead of needing a second board regulate the battery voltage and charge the battery. The new IMU board was also smaller in size. As discussed previously the battery life was maximized from 6 hours to 17 hours of constant usage. The increase came from not having to power two boards and not having to collect data for more than the necessary sensor. The eliminate of false positives came from the redoing of the previous fall detection algorithm and using the magnetometer and accelerometer data

outputs. To accomplish the last objective was rather easy, since it was just collecting a bunch of data that can be used in the future.

In the future of this project, it would be recommended that a different less power intensive data transmission board be used, another form of data transmission, such as WI-FI, the addition of a central data storage station, the ability to manually use the device, and detection of the device being dropped. With the change of data transmission applications, the battery can potentially last longer, the data could be more accurate, and there would be less data loss during the transmission. If there was a dedicated data storage station, anyone would be able to see the data and it would not be easier for data collection. The ability to have a manual option would help for the occasions of when a fall has happened, but the user does not need any help or if the user needs help from a fall that does not get correctly detected. It would also be useful to be able to tell the difference between a fall or the user fumbling with the device and potentially dropping it. This is significant so that the emergency contacts are not contacted for a non-fall.

8. Reference

[1] “Important Facts about Falls | Home and Recreational Safety | CDC Injury Center.” [Online]. Available: https://www.cdc.gov/homeandrecreationalsafety/falls/adultfalls.html. [Accessed: 16-Nov-2017].
[2] “Latest Alzheimer’s Facts and Figures,” *Latest Facts & Figures Report | Alzheimer’s Association*, 17-Sep-2013. [Online]. Available: //www.alz.org/facts/overview.asp. [Accessed: 16-Nov-2017].
[3] “9DoF Razor IMU M0 Hookup Guide - learn.sparkfun.com.” [Online]. Available: https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide. [Accessed: 27-Apr-2018].
[4] Reyes, Angels, “Slip and Fall Facts - Dallas Auto Accident Attorneys Blog - October 27, 2016.” Dallas Auto Accident Attorneys Blog. 27 Oct. 2016. Web. 6 Oct. 2017.
[5] “Administration on Aging (AoA).” AoA. U.S. Department of Health and Human Service Administration for Community Living. Web. 21 Oct. 2017.
[6] Jia, Ning. “Detecting Human Falls with a 3-Axis Digital Accelerometer.” Analog Dialogue, July 2009. Web. 16 Oct. 2017.
[7] Falin Wu, Hengyang Zhao, Yan Zhao, and Haibo Zhong, “ Development of a Wearable-Sensor-Based Fall Detection System.” International Journal of Telemedicine and Applications, vol. 2015, 30 December 2014.
[8] FallAlert2017, *Contribute to Self-Activating-Fall-Alert development by creating an account on GitHub*. 2017.
[9]P. Radhakrishnan, “Introduction to Recurrent Neural Network,” *Towards Data Science*, 20-Aug-2017. [Online]. Available: https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3. [Accessed: 16-May-2018].
[10]M. A. Nielsen, “Neural Networks and Deep Learning,” 2015.

9. Appendices

The following appendices include the results of the data collection and post analysis, as well as the code that is used during the project. All the code in Appendix A is just snippets of the full code.

Appendix A:

Data Logging to microSD Card

```
// SD Library manages file and hardware control
#include <SD.h>
bool enableSDLogging = ENABLE_SD_LOGGING;
bool enableTimeLog = ENABLE_TIME_LOG;
bool enableAccel = ENABLE_ACCEL_LOG;
bool enableGyro = ENABLE_GYRO_LOG;
bool enableCompass = ENABLE_MAG_LOG;
bool enableHeading = ENABLE_HEADING_LOG;
//////////////////////
// SD Card Globals //
//////////////////////
bool sdCardPresent = false; // Keeps track of if SD card is plugged in
String logFileName; // Active logging file
String logFileBuffer; // Buffer for logged data. Max is set in config
  // Check for the presence of an SD card, and initialize it:
  if ( initSD() ){
    sdCardPresent = true;
    // Get the next, available log file name
    logFileName = nextLogFile(); }
                Serial1.begin(115200);
                void loop()
  // The loop constantly checks for new Serial1 input:
  if ( LOG_PORT1.available() ){
    // If new input is available on Serial1 port
    parseSerial1Input(LOG_PORT1.read()); // parse it}
  // Then check IMU for new data, and log it
  if ( !imu.fifoAvailable() ) // If no new data is available
    return;                   // return to the top of the loop
  // Read from the digital motion processor's FIFO
  if ( imu.dmpUpdateFifo() != INV_SUCCESS )
    return; // If that fails (uh, oh), return to top
  // If enabled, read from the compass.
  if ( (enableCompass || enableHeading) && (imu.updateCompass() != INV_SUCCESS) )
    return; // If compass read fails (uh, oh) return to top
  // If logging (to either UART and SD card) is enabled
  if ( enableSerial1Logging && enableSDLogging)
    logIMUData(); // Log new data
                // If SD card logging is enabled & a card is plugged in
  if ( sdCardPresent && enableSDLogging){
    // If adding this log line will put us over the buffer length:
    if (imuLog.length() + logFileBuffer.length() >=
        SD_LOG_WRITE_BUFFER_SIZE){
      sdLogString(logFileBuffer); // Log SD buffer
      logFileBuffer = ""; // Clear SD log buffer
      blinkLED(); // Blink LED every time a new buffer is logged to SD}
    // Add new line to SD log buffer
    logFileBuffer += imuLog;}
bool initSD(void){
  // SD.begin should return true if a valid SD card is present
  if ( !SD.begin(SD_CHIP_SELECT_PIN) ){
    return false;}
  return true;}
```

```
// Log a string to the SD card
bool sdLogString(String toLog){
  // Open the current file name:
  File logFile = SD.open(logFileName, FILE_WRITE);
  // If the file will get too big with this new string, create
  // a new one, and open it.
  if (logFile.size() > (SD_MAX_FILE_SIZE - toLog.length())){
    logFileName = nextLogFile();
    logFile = SD.open(logFileName, FILE_WRITE);}
  // If the log file opened properly, add the string to it.
  if (logFile){
    logFile.print(toLog);
    logFile.close();
    return true; // Return success}
  return false; // Return fail}
// Find the next available log file. Or return a null string
// if we've reached the maximum file limit.
String nextLogFile(void){
  String filename;
  int logIndex = 0;
  for (int i = 0; i < LOG_FILE_INDEX_MAX; i++){
    // Construct a file with PREFIX[Index].SUFFIX
    filename = String(LOG_FILE_PREFIX);
    filename += String(logIndex);
    filename += ".";
    filename += String(LOG_FILE_SUFFIX);
    // If the file name doesn't exist, return it
    if (!SD.exists(filename)){
      return filename;}
    // Otherwise increment the index, and try again
    logIndex++;}
```

Importing of Data

```
filename = 'DATA_4.10.18/WalkingFall/LOG7.txt';
file = 'LOG7';
delimiter = ',';
header = 0;
dataArray = importdata(filename, delimiter, header);
newDataArray = Import_Data(dataArray);
dataLength = length(dataArray);
if dataLength <= 250
    noOfPoints = 75;
    reductionFactor = fix(dataLength / noOfPoints);
elseif dataLength <= 500
    noOfPoints = 425;
    reductionFactor = fix(dataLength / noOfPoints);
elseif dataLength <= 750
    noOfPoints = 550;
    reductionFactor = fix(dataLength / noOfPoints);
elseif dataLength <= 1000
    noOfPoints = 675;
    reductionFactor = fix(dataLength / noOfPoints);
else
    noOfPoints = 1000;
    reductionFactor = fix(dataLength / noOfPoints);
end
if reductionFactor < 1
    newDataSize = dataLength;
    reductionFactor = 1;
else
    newDataSize = fix(dataLength/reductionFactor);
end
newDataArray = zeros(newDataSize,10);
for index = 1:1:length(newDataArray)
```

```
    newDataArray(index,:) = dataArray(index*reductionFactor,:);
end
```

Post Processing Fall Detection Algorithm

```
MAGACC = sqrt(NormACCXaxis.^2 + NormACCYaxis.^2 + NormACCZaxis.^2);
fallDetected = Fall_Detection(MAGACC,TIME);
function fall = Fall_Detection(MAGACC,TIME)
data = MAGACC;
fall = 0;
y = diff(data);
m = max(data);
z = min(data);
leny = length(y);
maxMag = round(m);
switch maxMag
    case 0
        threshold = m + z;
    case 1
        threshold = m + z;
    case 2
        threshold = m + z;
    case 3
        threshold = (m-z)/(2*m);
    case 4
        threshold = 2*((m-z)/(2*m));
    case 5
        threshold = 5*((m-z)/(2*m));
    case 6
        threshold = m + z;
    case 8
        threshold = (m-z)/(2*m);
    otherwise
        threshold = m + z;
end
smallestSlope = 0;
for i = 1:1:leny
    if y(i) < 0
        if y(i) < smallestSlope
            smallestSlope = y(i);
        end
    end
end
if abs(smallestSlope) > threshold
    fall = 1;
    location = find(y == smallestSlope);
    figure('Name', 'Location');
    plot(TIME,data, 'r', 'Linewidth', 1);
    hold on;
    plot(TIME(location),data(location),'*k', 'Linewidth',0.75);
    title('Fall Detection');
    xlabel('Magnitude of Accelerometer Sensor');
    ylabel('Time (ms)');
    legend('Magnitude', 'Fall Location', 'Location', 'northwest');
end
if fall == 1
    fall = 1;
end
```

Phone Alert

```
Contact1 = "##########";
Contact2 = "##########";
Contact3 = string(missing);
Carrier1 = "carriername";
```

```
Carrier2 = "carriername";
Carrier3 = string(missing);
number = [Contact1, Contact2, Contact3];
carrier = [Carrier1, Carrier2, Carrier3];
if fallDetected == 1
   lenN = length(number);
   for i = 1:1:lenN
      numLength = strlength(number(i));
      if (numLength == 10)
       Phone_Alert(number(i),carrier(i),'ALERT','Testing, A fall has occurred')
       disp('A fall has been detected!')
      end
   end
end
% Ke Feng, Sept. 2007
% Please send comments to: jnfengke@gmail.com
% $Revision: 1.0.0.0 $  $Date: 2007/09/28 16:23:26 $
% ============================================================================
% YOU NEED TO TYPE IN YOUR OWN EMAIL AND PASSWORDS:
mail = 'MyEmailAddress';    %Your GMail email address
password = 'MyPassword';          %Your GMail password
% ============================================================================
if nargin == 3
   message = subject;
   subject = '';
end
number = strrep(number, '-', '');
if length(number) == 11 && number(1) == '1';
   number = number(2:11);
end
switch strrep(strrep(lower(carrier),'-',''),'&','')
   case 'alltel';    emailto = strcat(number,'@message.alltel.com');
   case 'att';       emailto = strcat(number,'@txt.att.net');
   case 'boost';     emailto = strcat(number,'@myboostmobile.com');
   case 'cingular';  emailto = strcat(number,'@cingularme.com');
   case 'cingular2'; emailto = strcat(number,'@mobile.mycingular.com');
   case 'nextel';    emailto = strcat(number,'@messaging.nextel.com');
   case 'sprint';    emailto = strcat(number,'@messaging.sprintpcs.com');
   case 'tmobile';   emailto = strcat(number,'@tmomail.net');
   case 'verizon';   emailto = strcat(number,'@vtext.com');
   case 'virgin';    emailto = strcat(number,'@vmobl.com');
   case 'fi';        emailto = strcat(number,'@msg.fi.google.com');
end
setpref('Internet','SMTP_Server','smtp.gmail.com');
setpref('Internet','SMTP_Username',mail);
setpref('Internet','SMTP_Password',password);
setpref('Internet','E_mail','MyEmailAddress');
props = java.lang.System.getProperties;
props.setProperty('mail.smtp.auth','true');
props.setProperty('mail.smtp.socketFactory.class', 'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port','465');
props.setProperty('mail.smtp.starttls.enable','true');
sendmail(emailto,subject,message)
```

Plotting IMU Sensor Data

```
figure('Name','Raw of Accelerometer');
plot(TIME, ACCX, 'r', TIME, ACCY, 'g', TIME, ACCZ, 'b', 'Linewidth',0.75);
title('Raw Accelerometer Sensor')
xlabel('Time (ms)')
ylabel('Acceleration (m/s^2) of all Axes')
legend('X-Axis', 'Y-Axis', 'Z-Axis');
figure('Name','Mag of Accelerometer');
plot(TIME, MAGACC, 'r', 'Linewidth',0.75);
title('Magnitude of Accelerometer Sensor')
```

```
xlabel('Time (ms)')
ylabel('Acceleration (m/s^2) of all Axes')
legend('Magnitude');
figure('Name','Raw of Magnetometer');
plot(TIME, MAGX, 'r', TIME, MAGY, 'g', TIME, MAGZ, 'b', 'Linewidth',0.75);
title('Raw Magnetometer Sensor')
xlabel('Time (ms)')
ylabel('Magnetic field (T) of all Axes')
legend('X-Axis', 'Y-Axis', 'Z-Axis');
figure('Name', 'Mag of Magnetometer');
plot(TIME, MAGMAG, 'r', 'Linewidth',0.75);
title('Magnitude of Magnetometer Sensor')
xlabel('Time (ms)')
ylabel('Magnetic field (T) of all Axes')
legend('Magnitude');
```

Plotting of Post Processing Fall Detection

```
figure('Name', 'Location');
 plot(TIME,data, 'r', 'Linewidth', 1);
hold on;
plot(TIME(location),data(location),'*k', 'Linewidth',0.75);
title('Fall Detection');
xlabel('Magnitude of Accelerometer Sensor');
ylabel('Time (ms)');
legend('Magnitude', 'Fall Location', 'Location', 'northwest');
```

Appendix B:

Walking

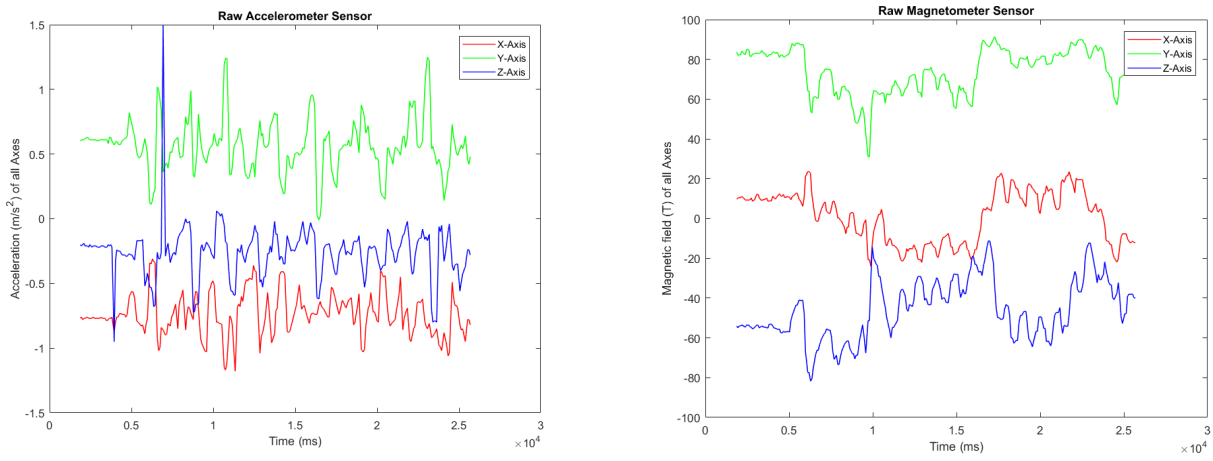
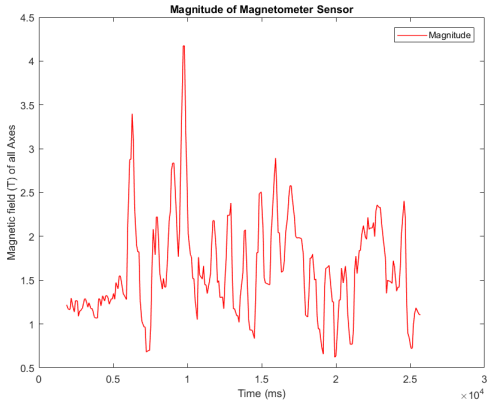


Figure 14 and Figure 15 - Raw Accelerometer and Raw Magnetometer Sensor Data



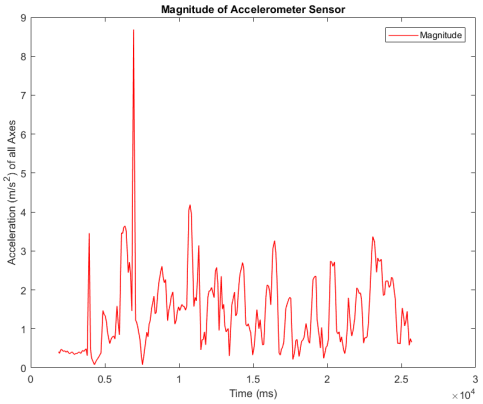
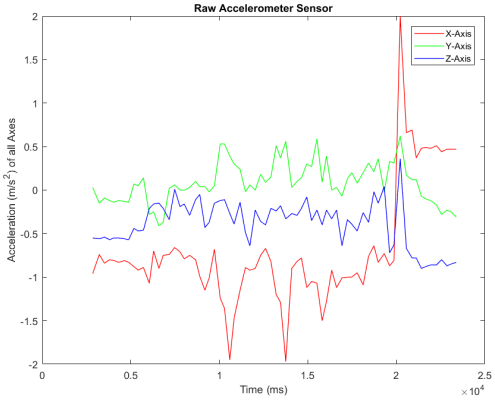


Figure 16 and Figure 17 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data

Walking with a Fall



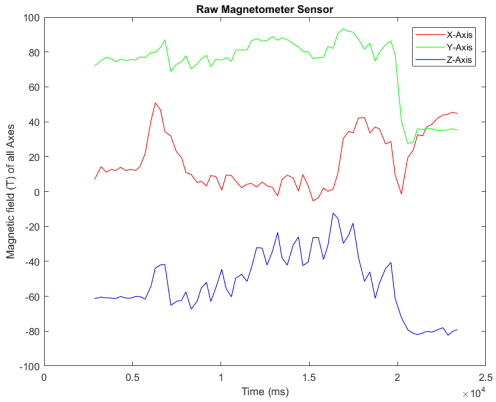
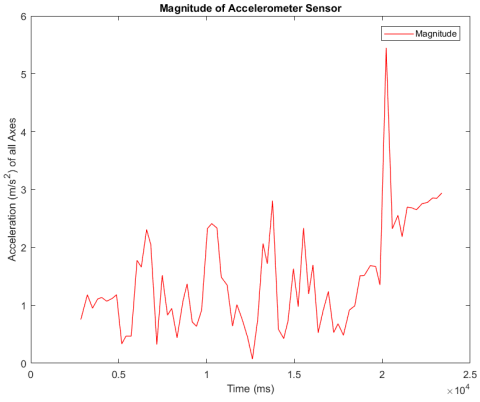


Figure 18 and Figure 19 - Raw Accelerometer and Raw Magnetometer Sensor Data



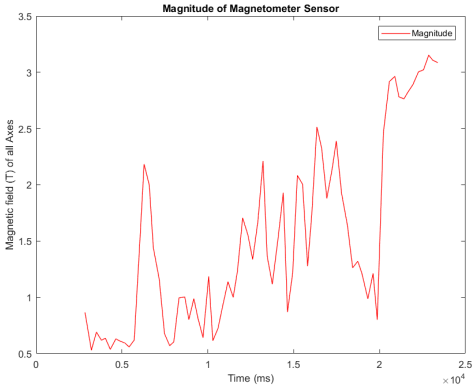


Figure 20 and Figure 21 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data

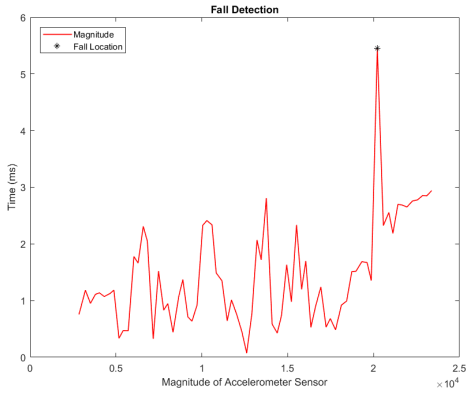
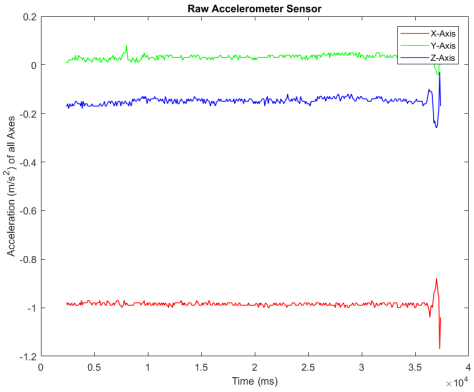


Figure 22 - Fall Detection

Standing



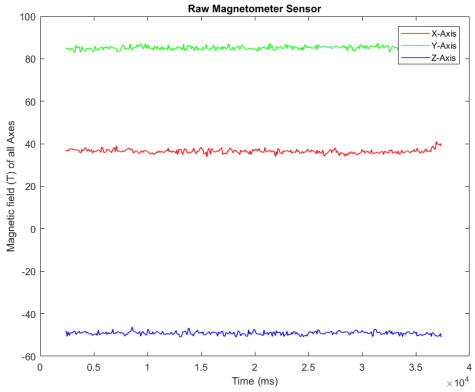
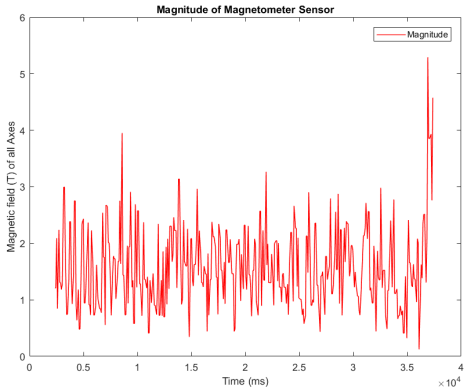


Figure 23 and Figure 24 - Raw Accelerometer and Raw Magnetometer Sensor Data



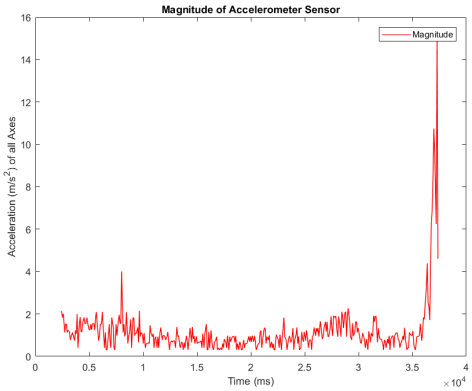
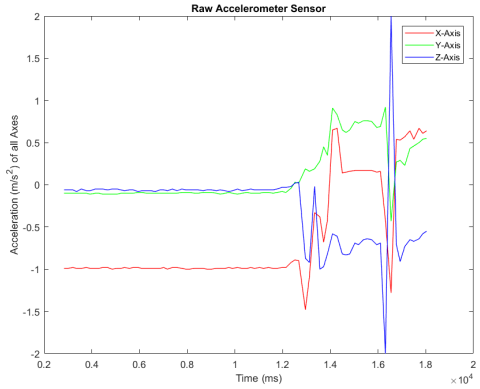


Figure 25 and Figure 26 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data

Standing with a Fall



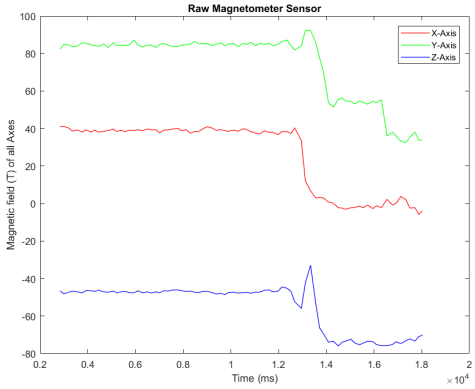
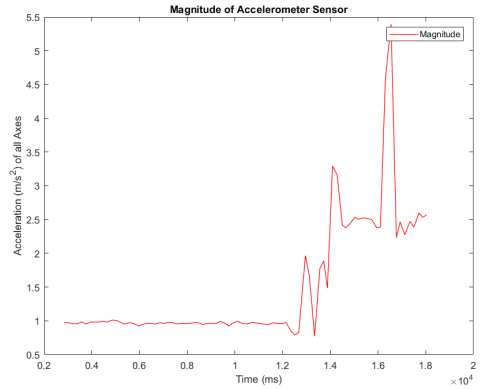


Figure 27 and Figure 28 - Raw Accelerometer and Raw Magnetometer Sensor Data



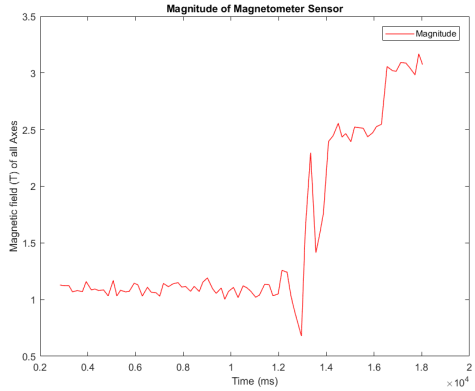


Figure 29 and Figure 30 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data

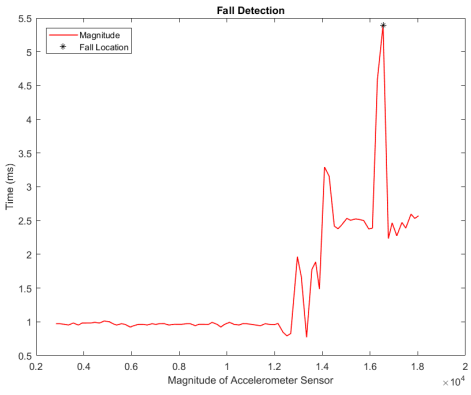
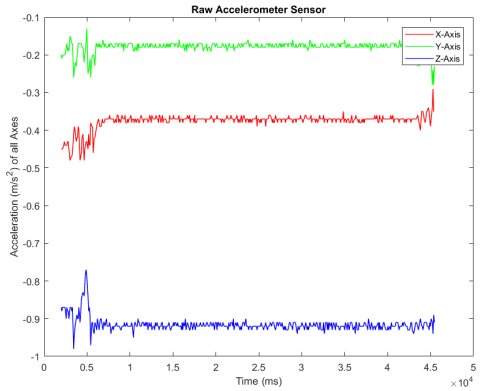


Figure 31 - Fall Detection

Sitting



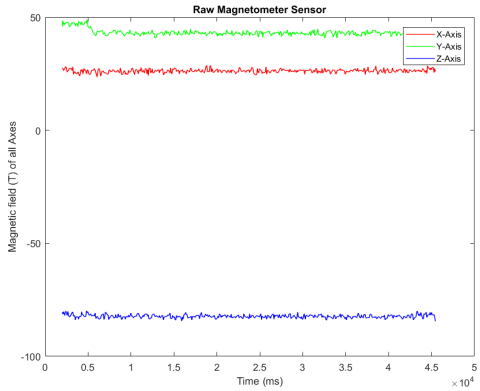
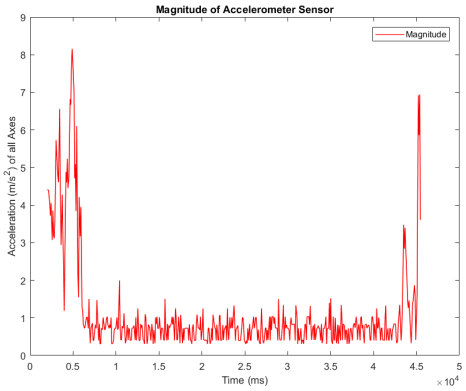


Figure 32 and Figure 33 - Raw Accelerometer and Raw Magnetometer Sensor Data



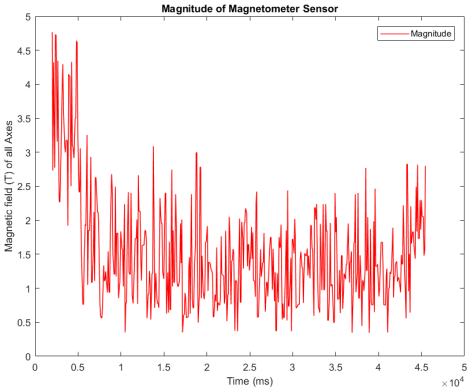
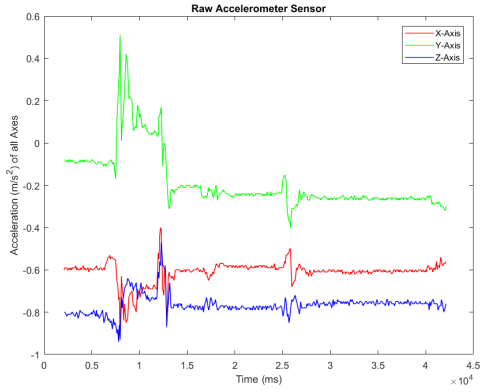


Figure 34 and Figure 35 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data

Sitting with Minimal Movement



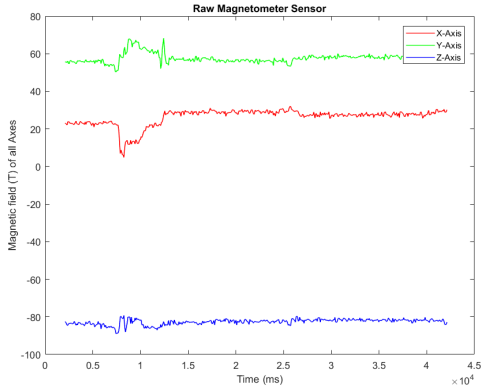
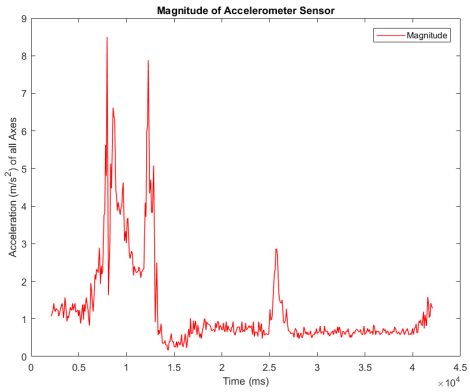


Figure 36 and Figure 37 - Raw Accelerometer and Raw Magnetometer Sensor Data



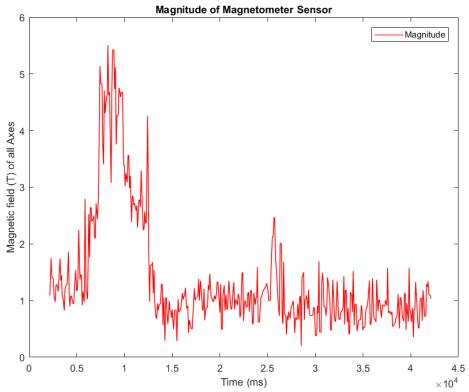


Figure 38 and Figure 39 - Magnitude of Accelerometer and Magnitude of Magnetometer Sensor Data