Self-Activating Fall Alarm

Bradley University Department of Electrical and Computer Engineering

Maisha Talukder, Michael McGrath, William Kelly

Advisors: Dr. Mohammad Imitiaz, Dr. Jing Wang, Dr. In Soo Ahn

ABSTRACT

The self-activating fall alarm is an “invisible” wearable device which clips on to the waist of the user (i.e through belt, waistband, etc.). This device automatically detects falls with the goal of shortening fall response time, which can prevent serious injury. The self-activating fall alarm utilizes a Bluetooth module (Bluetooth Mate Gold) directly connected to an Inertial Measurement Unit (IMU) to transmit sensor data to a laptop. The data from the three axis-accelerometers, -gyroscopes and -magnetometers, built-into the IMU, are analyzed by a fall-detection algorithm to determine if a fall has occurred. Once a fall has been detected, the system runs a fall-response algorithm through C++, which notifies the user’s caregivers and EMT, and reassures the user that help is on the way. The final portion of the algorithm, involves storing the user’s data allowing for physicians to analyze and interpret the fall occurrences to prevent future falls from happening

TABLE OF CONTENTS

TABLE OF FIGURES

I. Introduction

A. Problem Background

The self-activating fall alarm is used to alert medical professionals and caregivers when a family/friend has fallen in attempts to shorten the rescue response time and duration of treatment. Falls are the leading cause of common injuries in hospitals, with 2.8 million elderly people treated each year for a falling episode [1]. One out of five falls causes serious injuries such as broken bones, like wrist, arm, ankle, hip fractures, or head injuries [1]. The elderly population (those aged 65 and older) has increased from 35.9 million to 44.7 million (24.7% increase) between 2003 and 2013. About 28% (12.5 millions) of non-institutionalized older people live alone (8.8 million women, 3.8 million men), and almost half of older women (46%) age 75+ live alone [2]. The device is targeted primarily to these elderlies who are living independently in attempts to give them assurance of when an emergency arises, they will have a reliable, automated device that will contact help.

This report will cover the project specification-such as the objectives and goals of the project; technical specification-the parts used and why they were selected; design approach-how the software was created, the difficulties that were faced, and testing the system; cost analysis- the total amount spent and the projected amount to be spent in the future; and conclusion-the next phase of the project.

B. Problem Statement

The self-activating fall alarm must be a wearable, wireless, device which collects acceleration and orientation data of the user to depicts if a fall has resulted. The device must autonomously send alerts to a series of caregivers when an incident has occurred.

C. Problem Constraints

The self-activating fall alarm constraints are shown on Table 1 below. The IMU must contain an onboard microcontroller which can process the data from the inertial sensors (triple axis -accelerometer, -gyroscope, and -magnetometer) and connect to Bluetooth. The Bluetooth must be able to stay connected to the IMU and transmit serial data at a range of 100 feet. Lastly, the battery Lithium Ion Battery must supply 3.7 V to IMU and be able to recharge.

Constraints	
Inertial Measurement Unit (IMU)	Microcontroller, accelerometer, gyroscope, & magnetometer
Bluetooth Connectivity	At least 100 feet of open air range
Lithium Ion Battery with Charger	Provide 3.7 V and recharging capabilities

Table 1- Project Constraints

D. Related Work

Based on several journals and experiments, it was clear that a triple-axes acceleration and gyroscope sensor would be the major component for our device. Utilizing the acceleration data of an individual is the foundation in which an algorithm can be written to determine a fall.

Multiple 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 [2, 3]. When testing where the ADXL345 collects the most accurate acceleration data, it was found to be the waist [2]. Comparing the acceleration of different movements-, sitting down, standing up, and falling data- was a substantial part of the research. Identifying the changes in acceleration between everyday motions would help distinguish what type of data is generated during a fall versus non-fall. For example, walking had very minimal changes in acceleration as the elderly movement are relatively slow. For sitting down, the change in acceleration would be sudden and drastic at the time of a sit in the Y-axis [2]. During a fall, it was found that there was a sharp peak at the time of the fall.

A fall-detection algorithm must base upon a threshold calculated through the summation of the three acceleration axes (magnitude), $|a|$, and the rotation angle [3]. This threshold can be used to identify a fall. However, to only use the magnitude as the only basis of detecting a fall would be insufficient and give several false falls. Motions such as jumping or sitting can produce high peak values which can reach the threshold, allowing the device to assume it as a fall. Thus, another layer for diagnose a fall should be including the angle calculated based on 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 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." [3]

Once the device starts to collect data, the threshold magnitude is continuously compared to the real-time magnitude. If $|a_{current}| \geq |a_{threshold}|$, a fall may have possibly occurred.

The next step is to analyze the rotation angle. If the rotation angle proves that the user's orientation is toward the ground, the device will send out an SMS for help.

II. Technical Specification

Hardware System Block Diagram

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

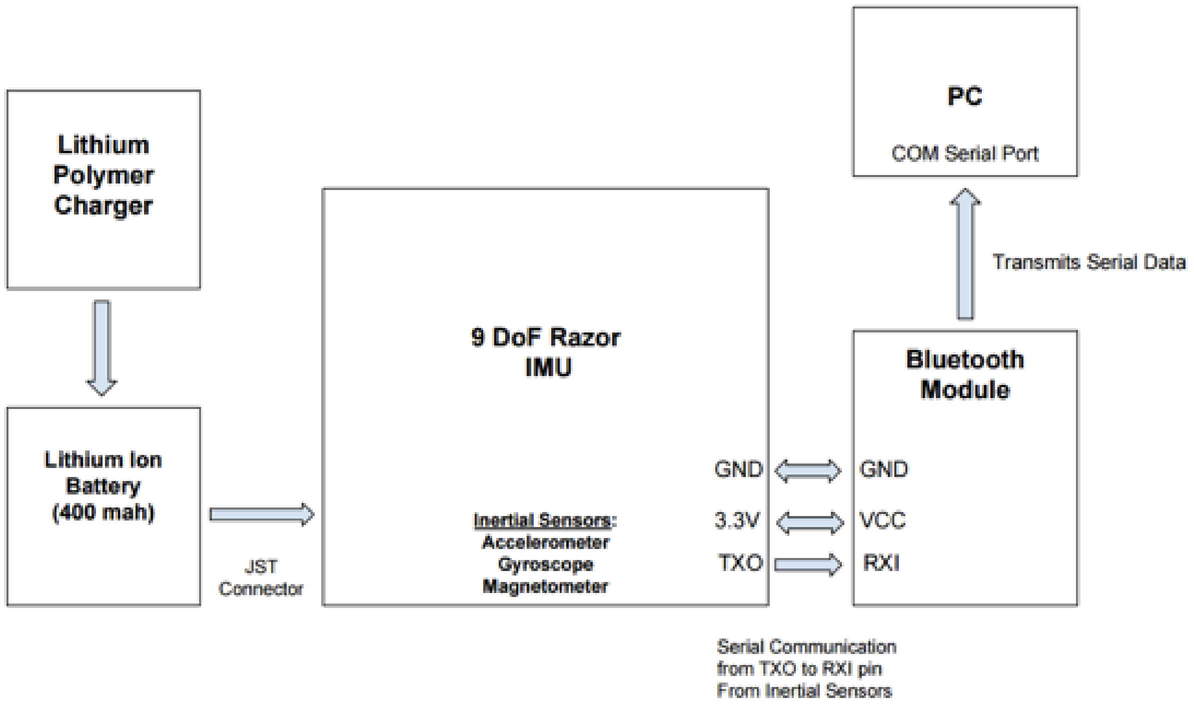


Figure 1 - System Hardware Block Diagram

Software System Block Diagram

Figure 2 shows the software block diagram of the self-activating fall alarm. This block diagram displays how the hardware components are interfaced with software. The system will connect to the Bluetooth module using a C++ program designed to specify the serial port of the PC. This will establish connection from the IMU to the PC to gather inertial sensor data. Once the C++ program starts, it will run continuously gathering the data from the IMU and will transmit the data through the Bluetooth module to the PC using serial communication. Then the program will output the data through a text file which is imported into Matlab. The Matlab Software is used to plot the data from the IMU displaying the data obtained from the accelerometer, gyroscope and magnetometer sensors.

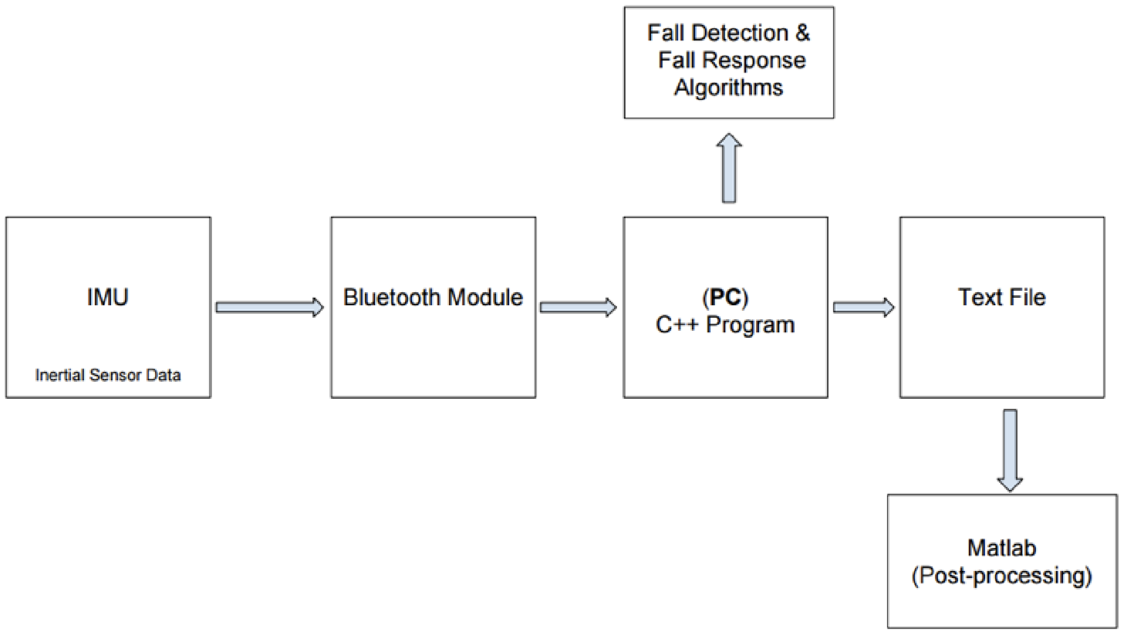


Figure 2 - System Software Block Diagram

High-Level System Flowchart

The high-level system flow chart show in Figure 3, is a visual representation of the logic process of the self-activating fall alarm. Once the IMU is powered up, the Bluetooth will create a connection with PC to transmits the senor data to display on the serial monitor. The next portion describes the algorithm portion of the system. The device will continuously monitor the acceleration to depict a fall. Once a fall a fall has been interpreted, it will initiate the fall-response algorithm.

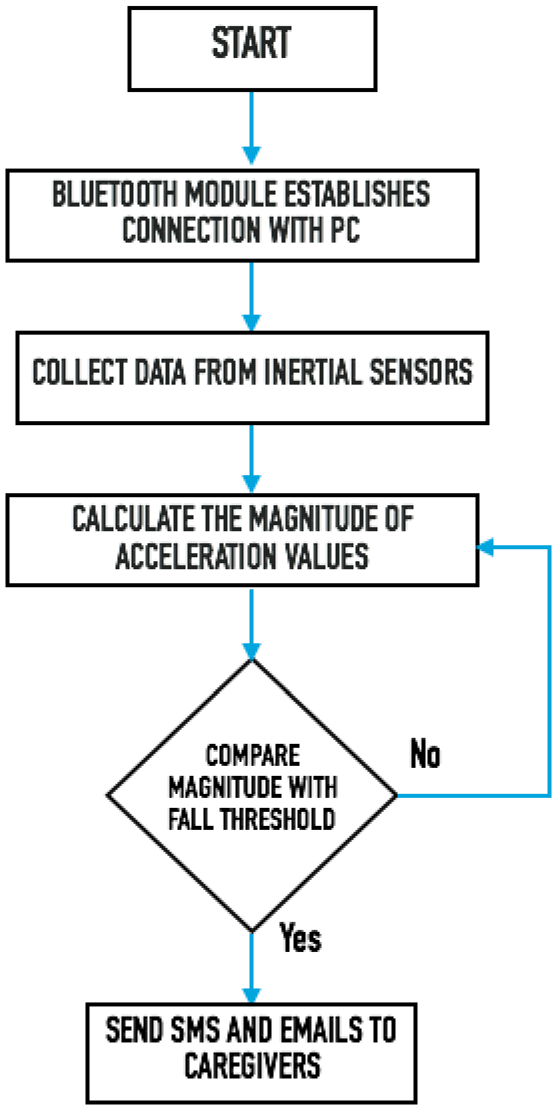

START
BLUETOOTH MODULE ESTABLISHES CONNECTION WITH PC
COLLECT DATA FROM INERTIAL SENSORS
CALCULATE THE MAGNITUDE OF ACCELERATION VALUES
COMPARE MAGNITUDE WITH FALL THRESHOLD
No
Yes
SEND SMS AND EMAILS TO CAREGIVERS

Figure 3 - System Flowchart

III. Hardware Components

Inertial Measurement Sensors

The 9DoF Razor IMU incorporates three sensors, an ITG-3200 (MEMS triple-axis gyro), ADXL345 (triple-axis accelerometer), and HMC5883L (triple-axis magnetometer) - to yield nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328 microcontroller and output over a serial interface. The system output describes the user's orientation and position in real-time.

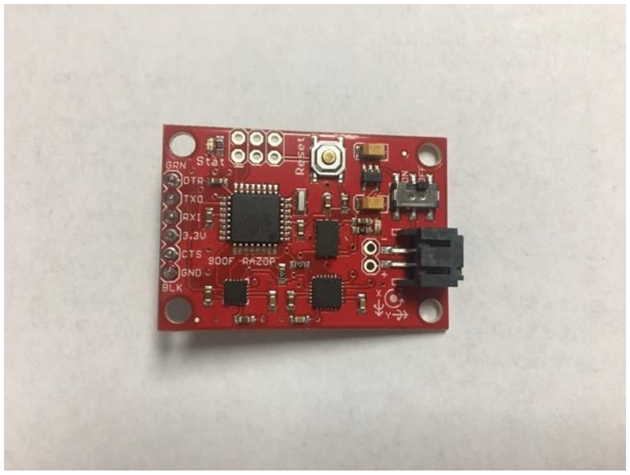


Figure 4 - 9 Degrees of Freedom Inertial Measurement Unit

Bluetooth Mate Gold

The RN41 Bluetooth Mate Gold, has a 2.4 GHZ frequency, is 13.4mm x 25.88 mm x .2m, and has a baud rate of 1200bps up to 921Kbps (but non-standard baud rates can be programmed). When connected, the LED blinks at various speeds to imply the state of connections [5]. The Bluetooth Mate Gold transmits the data from the IMU sensor to PC through wireless serial communication. The Bluetooth Module connects to the Rx/Tx pins of the IMU and connects to the PC's COM serial port to link the IMU to the PC.

Figure 5 - Bluetooth Mate Gold

Lithium Ion Battery and Charger

The system contains a Lithium Ion Battery that will allow the system to be wireless and charged by a LiPo battery charger. This very small, lightweight battery based on Lithium Ion chemistry, with the highest energy density. Each cell outputs a nominal 3.7V at 400mAh [6]. The battery contains terminated standard 2-pin JST-PH connector with 2mm spacing between pins. The LiPo charger is a basic charging circuit that allows you to charge 3.7V LiPo cells at a rate of 500mA or 100mA. It is designed to charge single-cell Li-Ion or Li-Polymer batteries [6].

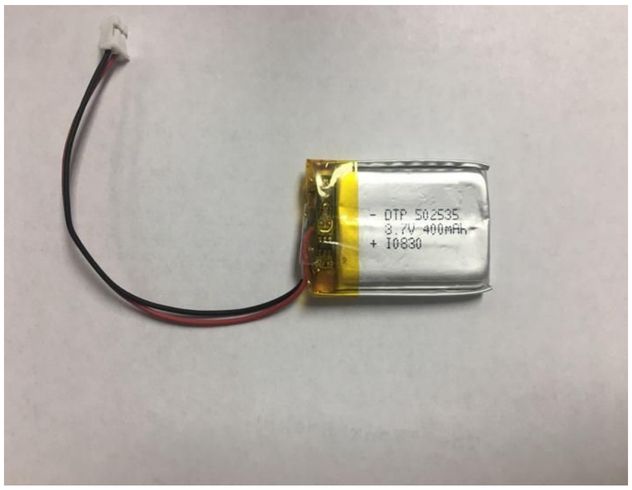


Figure 6 - Lithium Ion Battery

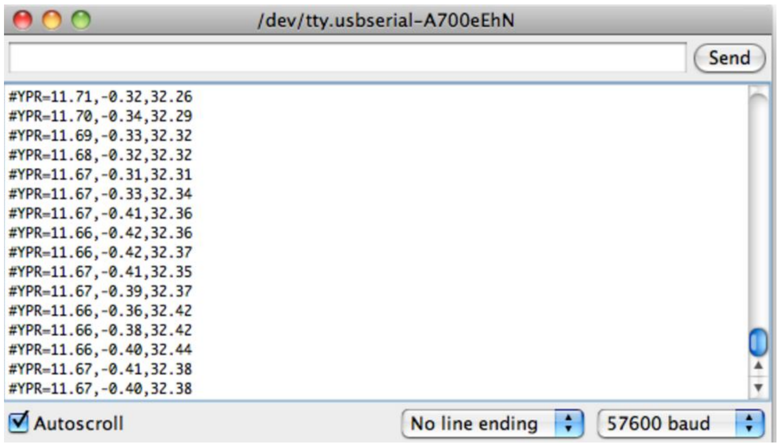
Figure 7- LiPo Charger

IV. Method of Solution

A. Software Design

The first task in designing the project was uploading the firmware to the Inertial Measurement Unit, which was done by connecting the IMU to the PC using an FTDI cable. The Razor AHRS (Attitude Heading and Reference System) Firmware, found on Github, is an Arduino based software that programs the on-board inertial sensors within the IMU.

Once the firmware was uploaded using Arduino, the serial monitor within the Arduino Console Window was opened to display the serial output coming out of the IMU; the displayed data shows output of the three inertial sensors as shown in Figure 8 below.



```
#YPR=11.71,-0.32,32.26
#YPR=11.70,-0.34,32.29
#YPR=11.69,-0.33,32.32
#YPR=11.68,-0.32,32.32
#YPR=11.67,-0.31,32.31
#YPR=11.67,-0.33,32.34
#YPR=11.67,-0.41,32.36
#YPR=11.66,-0.42,32.36
#YPR=11.66,-0.42,32.37
#YPR=11.67,-0.41,32.35
#YPR=11.67,-0.39,32.37
#YPR=11.66,-0.36,32.42
#YPR=11.66,-0.38,32.42
#YPR=11.66,-0.40,32.44
#YPR=11.67,-0.41,32.38
#YPR=11.67,-0.40,32.38
```

Figure 8 - Serial Monitor displaying Raw Serial Data; Yaw, Pitch, Roll Values

Second task is calibrating the Inertial Measurement Unit through the Razor AHRS firmware. The Arduino AHRS code contains a section called "USER_SETUP_AREA"/"SENSOR_CALIBRATION" in which the user can change the default values corresponding to the three inertial sensors.

Once the first two steps were accomplished, we transferred to Matlab to design a Matlab

Script that will read the sensor data from the IMU. Then we plotted the output sensor data in order to visualize the inertial sensor data to find the peak acceleration values.

C++ Code:

The C++ program initializes the Bluetooth connection between the IMU and the laptop, then collects and organizes the data into a matrix which we can then analyze. The algorithm then detects if a fall has occurred based on a threshold. Once a fall has been detected, the program sends a text message to caretakers to let them know the user needs help.

Bluetooth:

The Bluetooth is connected via serial connection (virtual COM port) to a laptop. The connections between our Bluetooth module and the IMU are VCC, ground, RXI, and TXO. Two connections, DTR and CTS, are not used as they are not needed for our application.

Algorithm:

Previous research has stated that falls can be detected by the magnitude of the three axes of acceleration. Equation 1 is used to calculate the magnitudes of the acceleration axes. When an actual fall happens, the human body and ground will produce obvious peak acceleration values where a_x, a_y, and a_z present accelerometer measurements of three axes.

To determine what magnitude is reached during a fall, a free-fall test was performed. The device was held at shoulder height and dropped on a table, the output of the test is displayed below. From this, it was established during a fall, the magnitude of the acceleration was well over 500 256/g. The C++ code below, once receiving the sensor data, will take the X, Y, and Z data of the accelerometer and compute the magnitude. It will then compare that the result with the threshold 600. If it is greater than or equal to 600 256/g, a code will flag this as a fall and the fall protocol is initiated.

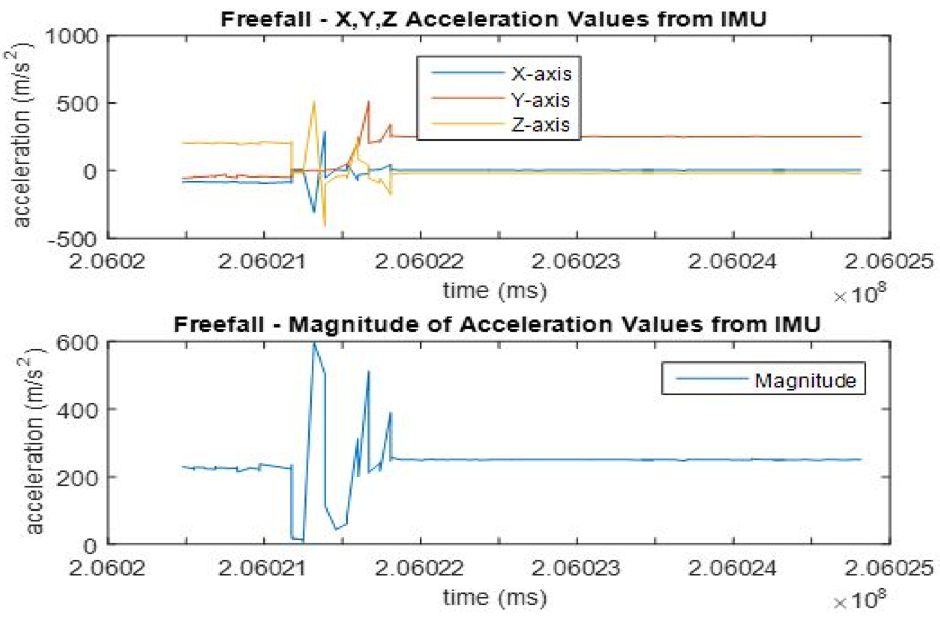
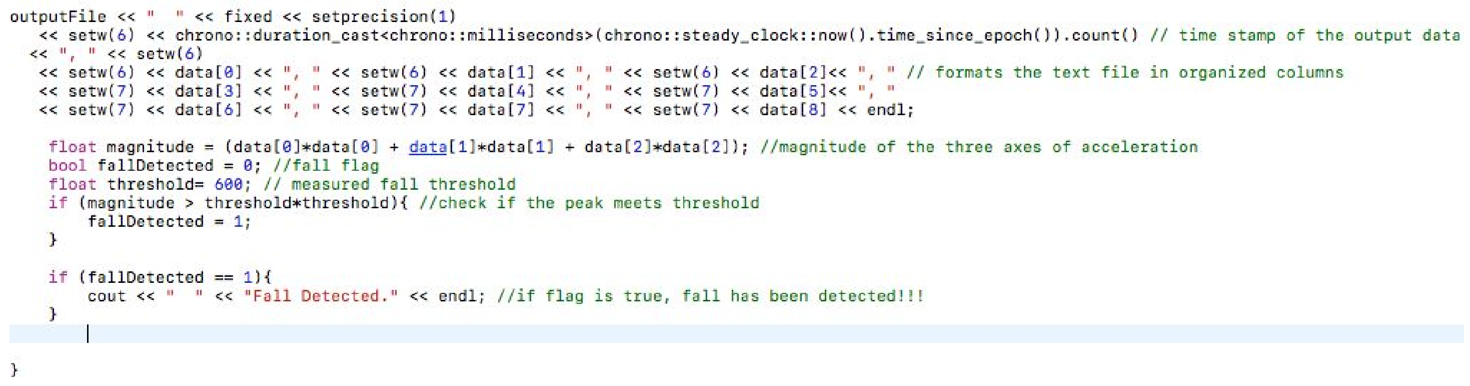


Figure 9 – Free-fall Data



```
outputFile << "  " << fixed << setprecision(1)
   << setw(6) << chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now().time_since_epoch()).count() // time stamp of the output data
  << ", " << setw(6)
   << setw(6) << data[0] << ", " << setw(6) << data[1] << ", " << setw(6) << data[2]<< ", " // formats the text file in organized columns
   << setw(7) << data[3] << ", " << setw(7) << data[4] << ", " << setw(7) << data[5]<< ", "
   << setw(7) << data[6] << ", " << setw(7) << data[7] << ", " << setw(7) << data[8] << endl;

    float magnitude = (data[0]*data[0] + data[1]*data[1] + data[2]*data[2]); //magnitude of the three axes of acceleration
    bool fallDetected = 0; //fall flag
    float threshold= 600; // measured fall threshold
    if (magnitude > threshold*threshold){ //check if the peak meets threshold
        fallDetected = 1;
    }

    if (fallDetected == 1){
        cout << "  " << "Fall Detected." << endl; //if flag is true, fall has been detected!!!
    }

}
```

Detection Algorithm Syntax

$$|\mathbf{a}| = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2},$$

Equation 1 - Magnitude of X, Y, Z acceleration

Post-Fall Processing:

Using postfix to send email messages, which can then be converted to SMS through email-to-text services provided by phone carriers. Once the fall flag is high, an email addressed to a phone number connected to a carrier's email, which automatically converts the message to SMS and sends to the recipient's phone, is sent. Initially, multiple fall alerts were sent for a single fall due to falls giving more than one frame of data with acceleration greater than the threshold. To fix this, a time threshold where the device can only send one fall alert per threshold was introduced. The time threshold is set at 30 seconds. This is reasonable because it eliminates all the multiple alert problems we encountered.

B. Software Difficulties

The first difficulty involved choosing a suitable software platform. We originally used Matlab to create the fall detection algorithm, but Matlab does not support real-time data analysis. Due to this, we used C++ programming to allow for continuous processing so that the device can continually update the PC with inertial data. Matlab was solely used for plotting data for testing purposes.

Our next obstacle we came across involved the Bluetooth connectivity. The Bluetooth module would disconnect from the PC due to the DTR and CTS pins. We resolved this issue by using a four-pin connector cable to remove the two pin connections.

Our largest and most time-consuming complication was distinguishing between a ‘false-positive’ and an ‘actual’ fall in our algorithm. To fix this problem, we created different movement test scenarios and saved the collected data. This allowed us to differentiate the outputted result from each case, as well as, recognize the patterns of the data when a fall is indicated. We found through the plots the associated peak value of acceleration indicating when the device fell. By collecting multiple sets of data, we could find a specific threshold.

V. Results

The IMU was tested in several different motions which include walking, sitting down, lying down and falling to obtain multiple sets of “test scenarios” which are shown below.

The scenarios were plotted using Matlab with the top plot showing the X,Y,Z values of acceleration and the bottom plot corresponding to the magnitude of acceleration values. These plots were then evaluated based on their ‘peak acceleration’ to distinguish the type of motion the user is experiencing. The peak values decide whether the movement is an actual fall or normal motion.

While performing these movements, we contained all pieces of hardware in a secured box with the measurements of 3.25”. x 1.4”. x 2” (smaller than a $20 bill). The picture below is a comparison of the box with a $20 bill. To ensure the components are not damaged, thin cushions were used to sandwich the parts.

This box was held at waist level, or put in a pocket, while performing most of the simulations. The first set of data, shown in Figure 10, was collected by holding the box at waist level and walking for 15 seconds. The acceleration magnitude of walking did not trigger the alarm; the magnitude was well below the fall threshold. Figure 11 was gathered by starting off as the user standing position. After 5 seconds, the user sat down at normal pace and the back to standing. The whole experiment was recorded for about 10 seconds. Again, during this assessment, the acceleration magnitude is much lower than the required fall threshold. From the plot, standing and sitting are mirror images. The third plot is of a falling movement. As the plot in Figure 12 demonstrates, the magnitude peaked above the fall threshold, generating a fall alert to be sent. Lastly, Figure 12 is an experiment where the user was running with the device. This test did activate a fall alert due to the magnitude reaching the threshold; the system assumed that a fall was occurring. This trial is an example of a 'false' positive fall, which must be further looked in to eliminate. However, this device is targeted to the elderly, who will not run at the same pace as a 22-year-old male. Thus, this 'false' positive is not considered a huge threat to the success of the device.

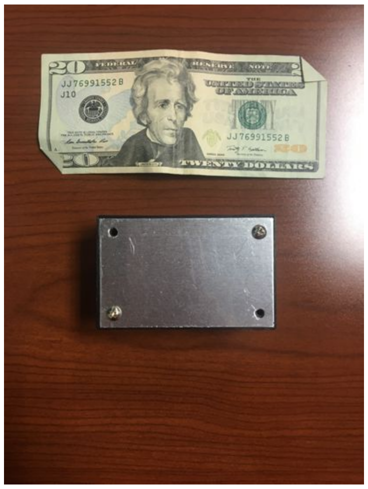


Figure 9- Comparison of Device with a $20 Bill

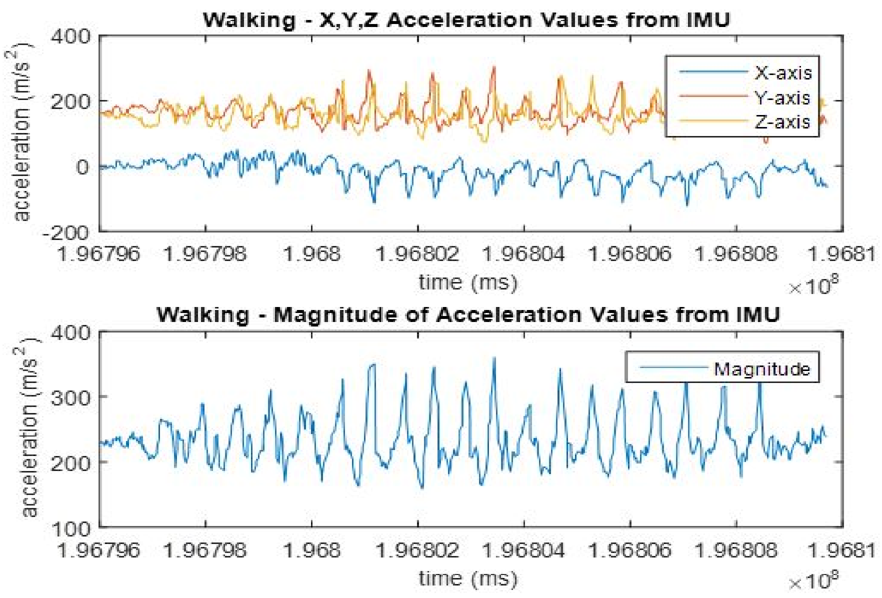


Figure 10 – Walking Acceleration Data

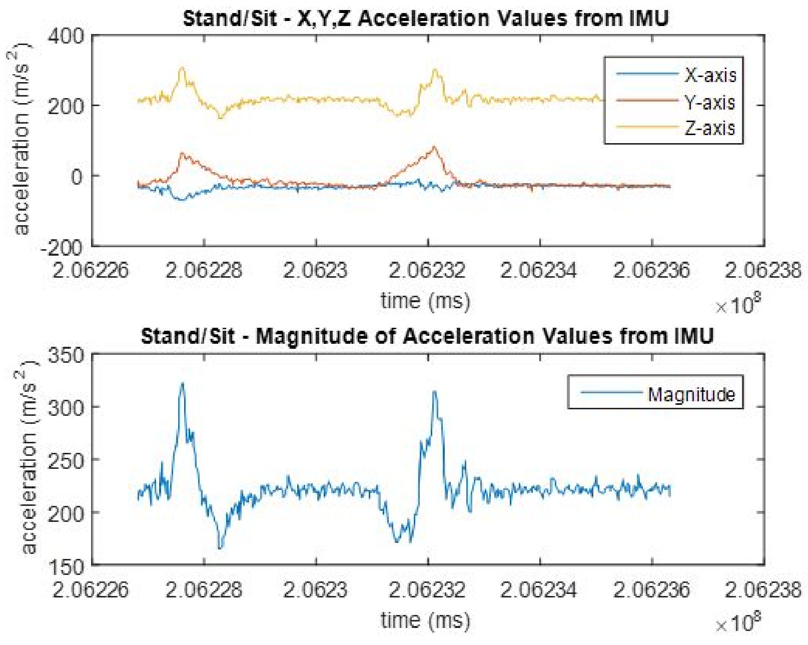


Figure 11 – Stand/Sit Acceleration Data

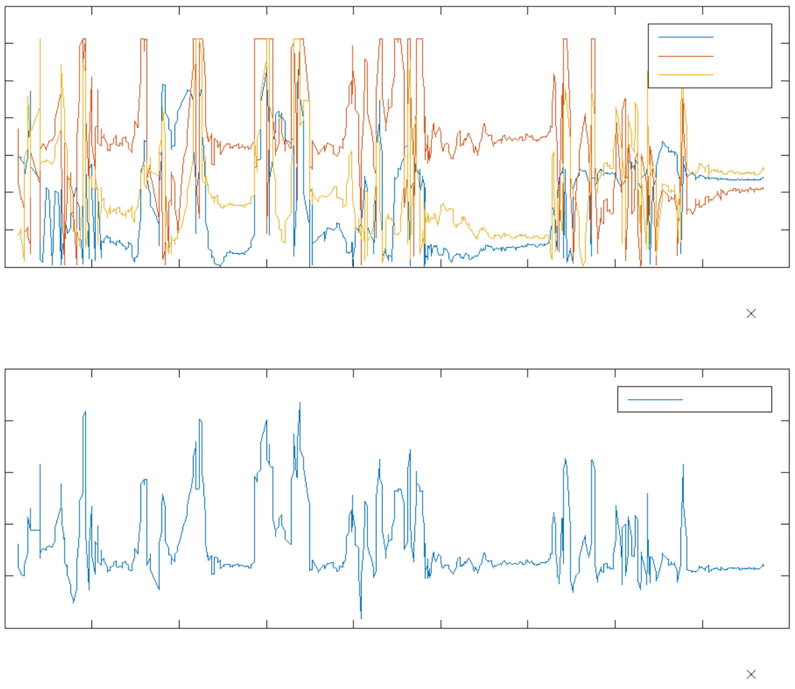


Figure 12- Normal Fall Data

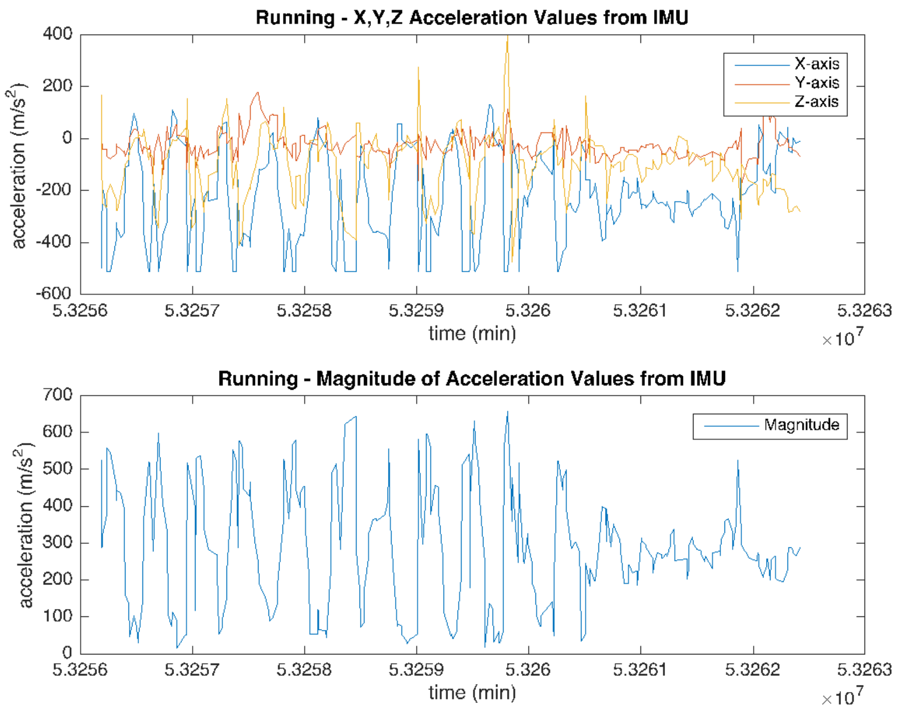


Figure 13- Running Data

VI. Discussion

From the testing results, it can be concluded that a fall generates acceleration values that are much higher than normal motions such as walking and sitting; therefore, we can design an algorithm based around a peak threshold value of 600 $\frac{m}{s^2}$. Normal motions resulted in values between 300-400 $\frac{m}{s^2}$ while falls generate as much as 1000 $\frac{m}{s^2}$ or even higher. The peak threshold was chosen to assure that we catch every fall, minimizing 'false' positives.

In addition, the fall detection algorithm does not have the capability catch slower falls due to the threshold acceleration value. Since it is limited to only acceleration value, in the future it is important to utilize on-board gyroscope to detect the user's orientation so that the device can have another basis to detect a fall based on user's position.

From the study, another observation that was made was the placement of the device on the user. It was found that when the user fell on the side that the device was positioned (i.e. right side of the waist), the acceleration was much more skewed. The reasoning for this is because the device was not placed at the user's center of gravity.

VII. Conclusion

The self-activating fall alarm has been successfully developed using a 9DOF Razor IMU which incorporates three sensors, a triple-axis - Gyroscope, Accelerometer and Magnetometer - to yield nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328 and output over a serial interface to a PC using a Bluetooth module. Then a C++ program on the PC stores the serial data within a text file and analyzes the data using a fall detection algorithm. When the fall detection algorithm recognizes a peak acceleration value that exceeds the predefined fall threshold, the program will send an email and SMS to the caregiver to help the user

The self-activating fall alarm meets several of the objectives stated earlier in the report. This device has completely taken the user out of the picture and is self-activating. Additionally, the device can send multiple alerts to a handful of caregivers to notify when user has fallen within 10 seconds (depending on Wifi strength). If the user is inactive for a long period, the system also recognizes this as problematic and will also alert caregivers. The algorithm save the pre-fall data for physicians to analyze which actives are associated with the most serious falls and which falls result into the most traumatic injuries.

The next phase of this project should be aimed in erasing ‘false-positive’ falls. To accomplish this, the fall algorithm should integrate the gyroscope data as the second basis for detecting false. This will greatly reduce ‘false-positive’ falls, requiring the device to consider the orientation of the user before declaring a fall. Incorporating a GPS module would be beneficial for this device. When an SMS is sent to caregivers, the geographic data of the user can also be attached in the test. Other improvements include switching to Wife to increase range of connectivity and replacing PC with a Raspberry Pi; having the device connect to a PC is not practical to market this product and will improve efficiency of the product. Lastly, the outer appearance of the overall device should be targeted. Design a customize box which is compact, lightweight system is very crucial to the overall success of the device.

VIII. Reference

[1] Reyes. Angels "Slip and Fall Facts — Dallas Auto Accident Attorneys Blog — October 27, 2016." Dallas Auto Accident Attorneys Blog. 27 Oct. 2016. Web. 6 Mar. 2017.

[2] "Administration on Aging (AoA)." AoA. U.S Department of Health and Human Service Administration for Community Living. Web. 21 Oct. 2016.

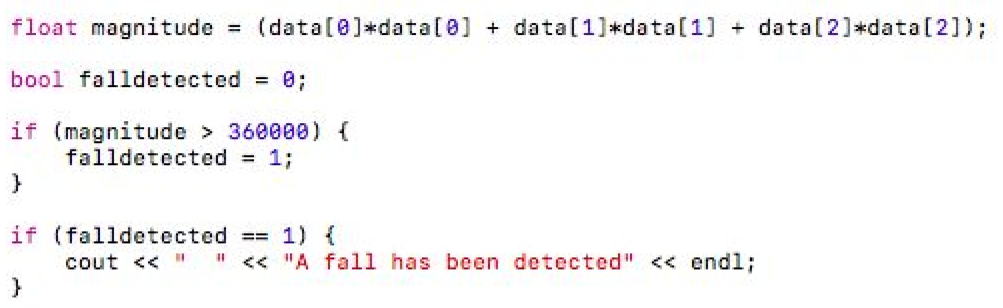
[3] Jia, Ning. "Detecting Human Falls with a 3-Axis Digital Accelerometer." Analog Dialogue, July 2009. Web. 30 Aug. 2016.

[4] 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

[5] Firmware provided by Sparkfun

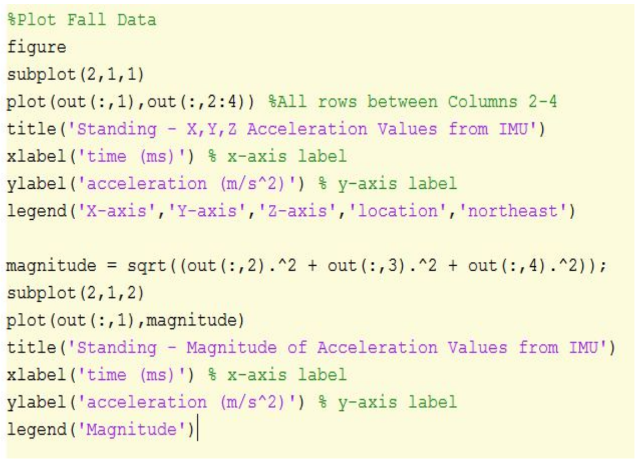
Appendix A:

Fall-Detection Algorithm:



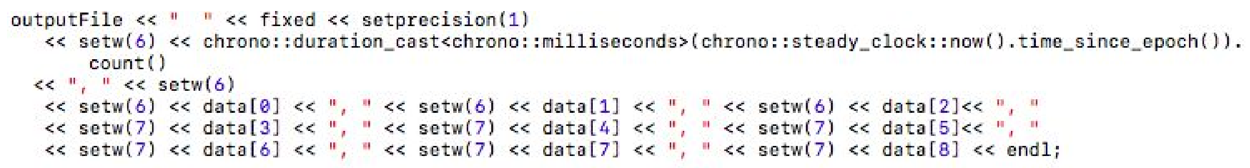
```
float magnitude = (data[0]*data[0] + data[1]*data[1] + data[2]*data[2]);

bool falldetected = 0;

if (magnitude > 360000) {
    falldetected = 1;
}

if (falldetected == 1) {
    cout << "  " << "A fall has been detected" << endl;
}
```

Plotting IMU Fall Data using Matlab:



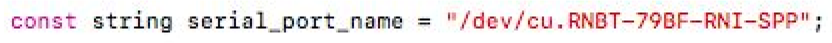
```
%Plot Fall Data
figure
subplot(2,1,1)
plot(out(:,1),out(:,2:4)) %All rows between Columns 2-4
title('Standing - X,Y,Z Acceleration Values from IMU')
xlabel('time (ms)') % x-axis label
ylabel('acceleration (m/s^2)') % y-axis label
legend('X-axis','Y-axis','Z-axis','location','northeast')

magnitude = sqrt((out(:,2).^2 + out(:,3).^2 + out(:,4).^2));
subplot(2,1,2)
plot(out(:,1),magnitude)
title('Standing - Magnitude of Acceleration Values from IMU')
xlabel('time (ms)') % x-axis label
ylabel('acceleration (m/s^2)') % y-axis label
legend('Magnitude')
```

Creating Fall Data Text File:



```
outputFile << "  " << fixed << setprecision(1)
   << setw(6) << chrono::duration_cast<chrono::milliseconds>(chrono::steady_clock::now().time_since_epoch()).
       count()
  << ", " << setw(6)
   << setw(6) << data[0] << ", " << setw(6) << data[1] << ", " << setw(6) << data[2]<< ", "
   << setw(7) << data[3] << ", " << setw(7) << data[4] << ", " << setw(7) << data[5]<< ", "
   << setw(7) << data[6] << ", " << setw(7) << data[7] << ", " << setw(7) << data[8] << endl;
```

Bluetooth Connection:



```
const string serial_port_name = "/dev/cu.RNBT-79BF-RNI-SPP";
```