Fall Detection and Prevention Device for the Elderly

By:
Martti Muzyka
&
Obatosin Obat-Olowu

Electrical Engineering Technology Project (EELE-2939-WA)
Department of Electrical Engineering
Lakehead University
Thunder Bay, Ontario, Canada

Abstract

Falling is a serious health problem that plagues the world's growing elderly population. Falls among the elderly can lead to serious injuries, such as bone fractures, bone dislocations, cerebral injuries and in some cases even death. After an elderly individual experiences a fall, it is important that they seek immediate medical attention. However, this may not be possible if the person is living alone and sustains injuries that make it difficult to right themselves, or if they are knocked unconscious. A fall detector is a device that solves these issues, if a fall occurs it will detect it, and automatically notify emergency services. Here lies a problem, as current commercial fall detectors can be quite expensive, charging large monthly fees to service and operate the fall detector. The objective of this project is to design and create a fall detection device that is effective and affordable. It will be able to distinguish between falls and non-falls, and will update a linked web page accordingly. It will operate using an algorithm based around a threshold(s), and if this threshold(s) is surpassed will provide a status update indicating that a fall has occurred and that immediate help is needed.

Acknowledgment

If it were not for the dedication and helpfulness of the Electrical Engineering lab technologists at Lakehead University, this project would not have been made possible. Their depth of knowledge and ability to effectively support us with the issues faced during the project, made for the investigation to be a very pleasant experience. We would like to especially thank Daniel Vasiliu, our supervising technologist, for all his insight and passion to assist us. There was never an instance where Mr. Vasiliu was not available to talk to or too busy to help. Without him this project could not have been finished in such a timely manner and with such high quality.

Table of Contents

Introduction (1)

- Objective (1.1)
- Background Knowledge and Theory (1.2)
- Features (1.3)

Objective (1.1)

The main objective of this technology project is to design a fall detector using a microcontroller with wireless capabilities and an accelerometer and/or gyroscope Arduino module. A case will be designed as to house the various components and give the wearer the ability to secure it looped through a belt at hip level. The microcontroller will act as a web server, capable of responding to clients (web browsers) with a web page displaying the status of the fall detector. In the event that a fall occurs, a predetermined threshold(s) will be surpassed and the status of the fall detector will change, indicating that a fall has occurred and the wearer could possibly be hurt.

Background Knowledge and Theory (1.2)

The elderly population across the world has risen immensely over the past decade, and it is predicted that this trend will only increase into the future. According to population projection scenarios published by Statistics Canada, seniors are expected to comprise between 23%-25% of Canada's population by 2036, in 1960 seniors comprised only 8% of the Canadian population. With this aging population, there comes a high demand for technology aimed towards helping elderly people and keeping them safe. This technology can be very expensive, however; and it is not always accessible to individuals who cannot afford it. Of the many ailments that plague this aging population, falling is one of the most prominent issues faced by the elderly. Falling can lead to a variety of injuries including bone fractures, intracranial injuries, and even loss of life. According to the U.S. Centers for Disease Control and Prevention, “Falls are the leading cause of fatal injury and the most common cause of nonfatal trauma-related hospital admissions among older adults.” This statement reveals the true severity of falling among the elderly and the need for more accessible prevention methods.

This project aims to investigate the uses of a microcontroller and an Arduino module capable of measuring 3D acceleration and position, and how both components can be used together to create an affordable fall detector. The creation of such a device would allow for fall detectors to be more easily accessible to families incapable of paying the large monthly fees, that companies currently selling fall detectors charge to service. It also allows family members to more easily check the status of a loved one (i.e. grandmother, or grandfather) and whether or not they might need help, without having to go through a third party company. The flow chart depicted in Figure (1) below, depicts the theory behind the operation of the device and the various steps that go into detecting a fall.

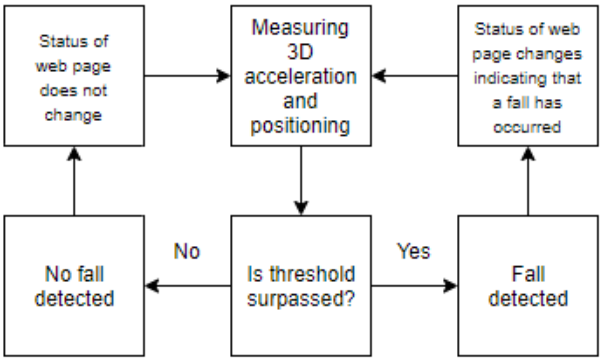


Figure (1) - Flow chart describing operation of the fall detector.

As can be seen in figure (1) above, the operation of this fall detector will be centered around a threshold(s) capable of distinguishing between falls, and the various other movements someone might perform throughout their day (i.e. walking, sitting, bending over, etc.). As this fall detector will possess an accelerometer and a gyroscope, the magnitude of total acceleration (|a|) and the magnitude of total angular velocity ($|v_\theta|$), are the values that the thresholds will test, as to determine if a fall has occurred. An accelerometer is a device capable of measuring the direction and magnitude of an acceleration along a certain axis. Using the acceleration measured along each of the axes in a 3D system, the magnitude of the total acceleration (|a|) can be calculated using equation (1) below. This value is measured in g's, where 1g is equivalent to the force of gravity at sea-level ($9.81 m/s^2$). Likewise, a gyroscope is a device that measures the angular velocity of an object along a certain axis. Similarly to the magnitude of the total acceleration (|a|), the magnitude of the total angular velocity ($|v_\theta|$) can be determined by using equation (2) below. The magnitude of the total angular velocity is measured in degrees per second (˚/s). These values will be used as thresholds as they react rather predictably when a fall occurs. For instance, if someone were to fall, their impact upon hitting the ground would result in a large |a| value, and the $|v_\theta|$ would change due to the 90˚ shift in their position.

$$|a| = \sqrt{(ax^2 + ay^2 + az^2)} \quad (1)$$

$$|v_\theta| = \sqrt{(v_\theta x^2 + v_\theta y^2 + v_\theta z^2)} \quad (2)$$

Features (1.3)

This technology project will include the following features:

1. A microcontroller with wireless capabilities (i.e. ESP32 microcontroller, ESP8266 microcontroller):
    - The microcontroller is the main focus of this project, it will act as a web server and will be responsible for detecting when a fall occurs and accordingly updating the status of the web page it responds to clients with.
2. A 3D printed case:
    - This case will house and protect the necessary components (i.e. microcontroller, battery, Arduino module, etc.), it will be made from a material capable of withstanding minor impacts, and will possess a place where a belt can pass through, securing the fall detector to the wearer at hip level.
3. An accelerometer and/or gyroscope Arduino module:
    - The accelerometer and/or gyroscope Arduino module will measure accelerations and positional changes in 3D space, this data will then be used by the microcontroller to

determine whether a predetermined threshold(s) has been surpassed, and if a fall has occurred.

4. An HTML document (web page):
    - The microcontroller will be acting as a web server, and when it receives a request from a client (web browser) will respond with an HTML document, which will display as a web page. This web page will show the status of the microcontroller as well as other necessary information, and will update every few seconds.

Design (2)

- Design and Operation (2.1)
- Component Price Analysis (2.2)
- Economic and Societal Concerns (2.3)
- Potential Improvements (2.4)

Design and Operation (2.1)

The design and operation for each of the core components/parts of the fall detector can be found below:

1. Fall Detector Case/Cover and Power Supply:
    - To design the case/cover of the fall detector, the dimensions of the components to be contained in the case were carefully measured. It was decided that the ESP32 microcontroller and MPU 6050 Arduino module would be connected to a 7x5 (cm) PCB. Four wires would then be soldered from the 3.3V, GND, SDA and SCL pins on the ESP32 microcontroller to the VCC, GND, SDA and SCL pins of the MPU 6050 Arduino module. To power the ESP32 microcontroller, the input of a DC 7V-24V to DC 5V 3A USB Output Step Down Converter was going to be connected to a 9V Energizer battery and the output, to the 5V and GND pins on the ESP32 microcontroller. This was done as to not use the potentially faulty onboard voltage regulators of the ESP32 microcontroller. To conserve the battery life, instead of powering the MPU 6050 Arduino module from the opposite 5V pin on the ESP32 microcontroller, the 3.3V pin was used instead. The most space efficient configuration was then determined. This meant having the DC-DC converter underneath the PCB connected to the ESP32 microcontroller and the MPU 6050 Arduino module, the 9V battery would then be placed along the side. It was also decided that the device would be worn at the hip, looped through a belt, a hole protruding from the back of the case was modelled to make this possible. With these decisions in mind, the case and cover for the fall detector were modelled on FreeCAD and printed with a Qidi X-One2 3D printer, it took 11.5 hours to print both the cover and the case. The cover was printed the color orange and the case black, they were both made from Polylactic Acid. For images of the 3D modelling done on FreeCAD and images of the case post printing, refer to Appendix D (8).
2. Web Page:
    - The web page returned by the web server to the client, upon request, is an HTML document. There were some unique design decisions that went into the design and layout of this web page. For instance, the status of the fall detector is printed in bright red, 75px text, as to attract the viewers attention straight to the most important aspect of the web page. The red text contrasts nicely with the teal background and the fact that it is a

different color than the other text indicates its importance. Below this are two hyperlinks, the first brings the user to a website with information on how to contact Emergency Medical Services in Thunder Bay. The second hyperlink brings the user to a website with CPR instructions, in the event that the victim of the fall is gravely hurt and/or injured. Lastly, the web page provides instructions on what to do in case the fall detector wrongly detects a fall. See Appendix G (11) for figures depicting the web page.

3. Fall Detection Program:
    - The fall detection portion of the program utilizes the ESP32 microcontroller to obtain and process data from the gyroscope and accelerometer. This program uses the serial data connection established between the ESP32 microcontroller and the GY-521 breakout board of the MPU-6050 module. The x, y, and z axes component of acceleration are read first from registers 59 to 64 of the GY-521, as the magnitude of total acceleration (|a|) was the primary trigger to detect a fall. Each axis data is contained in two registers. |a| is calculated using equation (1) and tested against the acceleration threshold. Once the threshold is surpassed, the x, y, and z components of the angular velocity are read from registers 67 to 72. The magnitude of total angular velocity ($|v_\theta|$), is then calculated using equation (2), and tested against the angular velocity threshold. Once the threshold is surpassed, a fall will be detected and a variable will be set to 1, prompting the web server program to update the fall detection status on the web page to “FALL DETECTED”. The Wire.h library is also used in this program to allow for the ESP32 microcontroller to communicate with I2C devices. This program, along with comments explaining its function and operation, can be found in Appendix C (7).
4. Web Server Program:
    - The web server portion of the overall fall detection program, initializes the ESP32 microcontroller to act as a web server. This program uses the following libraries:
        1. WiFi.h - This library allows the ESP32 microcontroller to connect to the internet.
        2. WiFiServer.h - This library enables the ESP32 microcontroller to act as a server and listen for potential clients.
        3. WiFiClient.h - This library allows a client to request data from the ESP32 microcontroller.
        4. String.h - This library enables various string operations.
        5. SPI.h - This library gives the ESP32 microcontroller the ability to communicate with SPI devices.
    - This program should initially connect the ESP32 microcontroller to a designated wireless network and provide a unique IP address. If this IP address is searched in web browser (client), like Google, an HTTP request will be sent to the ESP32 microcontroller (web server). The ESP32 microcontroller then receives the request, reads it and stores it in a string. Then once it receives a carriage return and a line feed, indicating the end of the HTTP request, it calls upon a function storing an HTML document as a string and sends this data to the client. The client receives this HTML document and displays it as a web page. This webpage will then refresh every few seconds, updating the status of the fall detector. If a fall is detected, a separate function used to test the value of the variable set to 1 in the occurrence of a fall, will return “FALL DETECTED”. This string will then be displayed on the web page indicating that a fall has occurred and help is needed. This status will remain the same until the “RST” button on the ESP32 microcontroller is pressed, restarting the fall detection program and returning the status to “NO FALL

DETECTED". To view this program refer to Appendix C (7). The program in Appendix C (7) contains comments explaining the reasoning and operation for each line of code.

Component Price Analysis (2.2)

Refer to table (1) below for a tabular comparison between the components used to build the fall detector and their corresponding prices. Potentially cheaper substitutes for the components used are also included for the sake of comparison.

Components Used (Name and Manufacturer Part/Model Number)	Unit Price ($ CAD)	Potential Component Substitutes (Name and Manufacturer Part/Model Number)	Unit Price ($ CAD)
Heltec Automation ESP32 Development Board Nodemcu Chip With 0.96inch OLED Display Bluetooth WIFI Kit32 Arduino Compatible CP2012 USB To Serial Chip (HTIT-WB32)	37.00 (Amazon.ca)	NodeMCU LUA WiFi Module Internet Based on ESP8266 Development Board New Version (ESP8266)	11.99 (Amazon.ca)
Aukru MPU 6050 Module 3 Axis Gyroscope + 3 Axis Accelerometer Module for Arduino (MPU-6050-Gyro-mould, GY-521)	3.30 (Amazon.ca)	N/A	N/A
Energizer 522BP Max Alkaline Battery, 9V, 1 Pack (522BP)	3.99 (Amazon.ca)	BlueDot Trading Heavy Duty 9 Volt Battery, Single battery (B00MGLBWZI)	2.78 (Amazon.ca)
DC 7V-24V To DC 5V 3A USB Output Converter Step Down Module KIS-3R33S (5559182658)	10.72 (Amazon.com)	DC-DC Converter 5V 6A output, 9-36V input, Screw Terminals and USB Port (4260474031511)	5.99 (Amazon.ca)
Keystone Battery Connector, Snap 9V 1 Cell Wire Leads - 4"/101.6mm (232)	0.68 (DigiKey.ca)	N/A	N/A

Table (1) - Comparison between the prices of the components used to assemble the fall detector, and potential substitute components.

Refer to table (2) below for information regarding the pricing of the materials and resources that went into producing the case and cover of the fall detector.

Section of the Case	Cost of PLA Filament Used ($ CAD)	Cost of Power Consumed by 3D Printers ($ CAD)	Total Price ($ CAD)
Cover	0.81	0.04	0.85
Case	1.99	0.16	2.15
3mm Socket Button Screws (4)	N/A	N/A	0.04
7x5cm PCB Blank Circuit Board Prototype Paper Solder Circuit Panel	N/A	N/A	1.00

Table (2) - Case and cover development costs.

Refer to table (3) below for the total cost of the project, includes the prices of the components and the 3D printed case.

Components + 3D Printed Case	$59.73 (CAD)

Table (3) - Total cost of fall detector.

Economic and Societal Concerns (2.3)

One of the major goals of this technology project is to design and build, an affordable fall detector that could be made accessible to individuals without a lot of money. Current commercial fall detectors usually charge monthly fees of over $30 a month to operate and service the fall detector. In contrast, the fall detector built for this technology project, costs only $59.73. Therefore, instead of paying $30/month to operate the fall detector, an individual could spend a flat $60 fee (could be even less). In one year they would see savings of over $300. This flat $60 fee makes fall detectors more accessible to individuals who arguably need them more. For instance, someone capable of comfortably paying monthly $30 fees is likely to be wealthier and therefore can afford to pay for other expensive medical services as well.

Initially, the intended demographic for this technology project seemed to be exclusively elderly individuals without a lot of money, as they can sustain grave injuries when they fall. However, it was later realized that this project could also benefit individuals of varying ages who live with debilitating diseases like hemophilia and osteogenesis imperfecta (OI). Hemophilia as defined by the National Hemophilia Foundation is, “a genetic disorder caused by missing or defective factor VIII, a clotting protein.” Hemophilia leads to intense bleeding, either internally or externally, from very little trauma. This can be especially deadly in children unaware of their condition and its consequences. Parents of children with hemophilia could monitor their child’s status, regarding falls, throughout the day, and if one occurs seek medical aid. Osteogenesis imperfecta (OI) on the other hand is a condition known as brittle bone disease, which is most problematic in children and infants. People with OI can sustain significant injuries (i.e. bone fractures) from very little trauma, especially when falling. If a child with OI was wearing a fall detector, their parents could easily monitor them and prevent further injury if they were to fall.

The case of the fall detector is made from Polylactic Acid (PLA), in an article published by Bioplastics News, PLA is described as, “a thermoplastic aliphatic polyester derived from renewable resources.” The benefit of using this material to construct the case is the fact that it is biodegradable and thus will not have as significant of an impact on the environment as the plastic typically used in commercial fall detectors. For reference, a bottle made from PLA would degrade in approximately 24 months if left in the ocean, while it would take a plastic bottle upwards of 450 years.

Potential Improvements (2.4)

The following list provides potential improvements that could be made to the overall design of the fall detector given more time and resources:

- ➔ A sleep mode, if the microcontroller does not detect movement after a certain amount of time, it would shut off to conserve battery life.
- ➔ A redesign of the case, currently the edges of the case are quite sharp and when worn for a significant amount of time can begin to aggravate the wearer. In future iterations the corners could be rounded off to maximize the wearers comfort and enhance the ergonomics.
- ➔ A locking mechanism securing the cover of the case to the base, currently it is held on just by the friction created by the two materials touching which works well, but this change would improve the overall reliability of the device.
- ➔ A physical button that the wearer could push to call for help. This would be useful in the event that the fall detection algorithm does not recognize a fall that occurs, but yet the wearer still needs help.

Testing (3)

- Initial Testing (3.1)
- Failure Analysis (3.2)
- Redesigns (3.3)
- Experimental Testing (3.4)
- Results (3.5)
- Safety Procedures (3.6)

Initial Testing (3.1)

1. Web Server Program:
    - When initially testing the web server portion of the overall program, there were a few difficulties. Initial compilations would let the ESP32 microcontroller connect to the designated wireless network and would return an IP address. When this IP address was searched for in a web browser the web page monitoring the fall detector status would be brought up. This web page would not refresh, however; and the serial monitor would show that the client was still connected to the web server when that connection should have been closed earlier. The only time the client would disconnect would be when a new client connected (i.e. the IP address was searched for on a different device and/or browser).
2. Power Supply:
    - To power the fall detector it was determined that a 9V battery would be input into a buck converter (KIS-3R33S). This buck converter would then step down the voltage to 5V and this 5V output would be connected to the 5V and GND pins on the ESP32 microcontroller. Tests were then performed to confirm that this would in fact work. Initially, as the output of the KIS-3R33S is a

USB port, wires had to be soldered onto the output connections found underneath the device. Next, a program used to read the values from the accelerometer and gyroscope located on the MPU 6050 and display them on the serial monitor, was uploaded onto the ESP32 microcontroller. As there were no 9V batteries on hand, a variable DC power supply was set to 9V and input into the KIS-3R33S. The output of the KIS-3R33S was then connected to the 5V and GND pins on the ESP32 microcontroller which was already connected to the MPU 6050 Arduino module. Initial impressions indicated that all three components were operating as desired, both the lights on the KIS-3R33S and ESP32 microcontroller lit up. A micro-usb cable was then connected to a computer and the ESP32 microcontroller and the serial monitor was opened. This was done as to ensure that the MPU 6050 Arduino module was receiving power and still displaying the measurements from its accelerometer and gyroscope. These initials tests revealed that the power system operated well and did not require any further modifications or redesigns.

3. Fall Detection Program:
    - To begin the initial testing of the fall detection program, a self test was run on the GY-521 breakout board of the MPU 6050 module. This self test ensures that all the sensors are still calibrated as to the factory settings. The procedure for the self-test was followed as indicated in the MPU 6050 specification sheet. The self test program displays the change from factory trim on the serial monitor. When the program was run, the change from factory trim was shown to be within the acceptable range of ±14%. The initial fall detection program was then run as to prove whether a fall would be detected by the the microcontroller. The program was uploaded onto the ESP32 microcontroller and falls were then simulated by dropping the ESP32 microcontroller connected with the MPU 6050 sensor chip. The corresponding data was observed through the serial monitor. “Fall Detected”, would be displayed on the serial monitor if the initial threshold values were surpassed. Once good |a| and $|v_\theta|$ thresholds had been determined, that could accurately detect and register falls when the actual ESP32 microcontroller and MPU 6050 Arduino module were dropped, more elaborate tests were started. After all the components had been added to the fall detector casing, the fall detector could then be tested with real world falls. The developed fall detection algorithm was added to the web server program and initial tests seemed successful. Even though the actual thresholds had to be adjusted to allow for more accurate fall detections, when a fall was detected, it seemed to be updating the web page properly. It was noticed, however; that after certain fall simulations, the web page would take a long time to refresh and once done would not show an updated status on the web page (i.e. FALL DETECTED). This was initially contributed to a slow internet connection, but it began occurring so frequently that it was deemed an issue with the actual program.
4. Fall Detector Case/Cover
    - Once the case and cover had finished 3D printing the components were loosely arranged within the case to ensure that the everything fit as desired. This revealed certain issues with the design. Firstly, the PCB did not sit flush on the posts with the pre-drilled holes as it should have, instead it rested on top of the KIS-3R33S. Secondly, it was realized that there would be no way to access the micro-usb port on the ESP32 microcontroller once all the components were added. This would make it difficult to upload new iterations of the program while testing. Lastly, the cover of the fall detector was obstructed by both the 9V battery and ESP32 microcontroller, preventing it from fully closing.

Failure Analysis (3.2)

1. Web Server Program:

- The initial tests of the web server program revealed that the client would remain connected to the web server indefinitely. To understand why this was occurring, the web server program was read over numerous time, but this revealed no actual errors within the program itself. However, by thinking logically about what was occurring, it was deducted that the issues were stemming from the void loop() portion of the program. Specifically, that the program was getting stuck in the while loop setup to monitor the incoming HTTP request from the client. Theoretically, this while loop should receive the incoming HTTP request and read it up until a carriage return and a line feed are received, thus indicating the end of the HTTP request. As this is happening the HTTP request is stored in a string and printed out on the serial monitor. The program also checks the length of the last received line, to ensure that the carriage return and line feed are the only characters present. If this is true, the program sends the HTML document to the client, breaks out of the while loop, waits half a second, and then disconnects from the client. It was discovered through testing that '\r\n', is considered to be a single character by the Arduino IDE, in other languages it is considered 2. The if statement checking for the length of the final line was initially set to look for a string with a length of 2, not 1. This was causing the program to get stuck in the while loop not letting the program initiate a disconnect from the client.

2. Fall Detection Program:
    - To solve this issue, the program was thoroughly analyzed and the operation of each aspect of it was looked at individually. During these initial tests, the actual fall detection, and corresponding web page status update, revolved around the value of this variable labelled "ht_bit". If both the |a| and |v| thresholds were surpassed, indicating that a fall had occurred, this variable would be set to 1. Otherwise, every time the program read values from the accelerometer and tested the |a| threshold, this variable would reset to 0. If a fall were to occur and the ht_bit was set to 1, the looped web server portion of the program would check this value, and if equal to 1, would stop the web page from refreshing for 10 seconds. This would give the person monitoring the fall detector web page enough time to view the updated status and call for help. If this component were not in place, the status would quickly revert back to "NO FALL DETECTED". It was concluded though, that it was this convoluted method of updating the fall detector status, that was causing the issues regarding the web page refresh time. If a fall were not detected at the perfect time, right before the web page refreshes, then the status of the web page would not change. Either the ht_bit would be reset to 0, or the ht_bit would be set to 1 as the client was disconnecting from the server. The latter is what led to the long refresh times. The program would detect that a fall has occurred and would set the ht_bit to 1, however; this would occur after the HTML document displaying the fall detector status was sent to the client. Therefore, the program would read that the ht_bit is equal to 1 and stop the web page from refreshing for 10 seconds, but the status would not have updated. It was purely by coincidence that earlier tests had proven successful.
3. Fall Detector Case/Cover:
    - To understand why the issues with the fall detector case/cover were occurring, each of the components were looked at individually and re-measured. By doing so, it was observed that the initial measurements of the KIS-3R33S converter had not taken into account the soldering points on the bottom of the component. These soldering points added approximately 1-2 mm to the height of the KIS-3R33S, which caused it to sit above the channel that was designed to contain it. This added height was what was stopping the PCB from lying flush with the pre-drilled holes. It was also determined that the 9V battery dimensions found online, that the case was designed to house, were outdated. The dimensions found online were for 9V batteries produced in 2007, between then and 2019 these dimensions had changed. Finally, the lack of any way to access the micro-usb port on the ESP32 microcontroller was purely a design flaw, and something that had not been taken into consideration during the initial design of the fall detector case/cover.

Redesigns (3.3)

1. Web Server Program:
    - Once the issues affecting the web server program had been effectively analyzed and understood, improvements were made to the code. By changing the HTTPrequest.length() if statement condition from 2 to 1, the program was able to properly recognize the last line of an HTTPrequest and break out of the while loop. Once out of the while loop the program would then disconnect from the client and the web server began working as intended.
2. Fall Detection Program:
    - To resolve the issues plaguing the fall detection program, adjustments had to be made regarding how the program would update the status of the fall detector in the occurrence of a fall. It was decided that instead of always resetting the ht_bit to 0 when the accelerometer values were being read, this variable would be initialized to 0 at the very start of the program. The 10 second delay in refreshing the web page after seeing if the ht_bit was equal to 1 was also removed. Now, with these adjustments in place, if the ht_bit were set to 1 after a fall, the web page status would update correctly, and indefinitely display “FALL DETECTED”, even after refreshing. Furthermore, to reset the ht_bit back to 0 and change the fall detector status back to “NO FALL DETECTED”, the RST button on the ESP32 microcontroller could be pressed.
3. Fall Detector Case/Cover:
    - To solve the lack of access to the micro-usb port, a hole was melted through the case with enough room for a micro-usb cable to pass through and plug into the ESP32 microcontroller port. This was done using an old soldering iron, not part of Lakehead University's equipment. Once completed, it was noticed that melted plastic had solidified on the inside of the case and was obstructing the cover from fully closing. A dremel was used to smoothen this surface and allow the cover to effectively seal the case. To alleviate the issue with the PCB lying on top of the KIS-3R33S, pieces of double sided adhesive tape were positioned along the rectangular posts close to the pre-drilled holes. This tape provided added clearance between the PCB and the KIS-3R33, the screws could then be used to secure the the PCB to the case without bending the PCB. Finally, regarding the issues with the height of the battery, a 2007 model 9V battery was located. When placed inside the case, the cover was no longer obstructed and could almost fully seal. It was decided that this battery would be used for the presentation of the project.

Experimental Testing (3.4)

Once the fall detection program was completed and the fall detector was fully assembled, experimental tests began. These tests were used to determine the optimal threshold levels for both the magnitude of total acceleration (|a|), and the magnitude of total angular velocity ($|v_\theta|$). Refer to equations (1) and (2) respectively, in section (1.2) of the report. An optimal threshold would be one that can distinguish between falls and the various actions an elderly and/or ill, individual might perform throughout their day. Taking into consideration the intended demographic for this fall detector, elderly and/or ill individuals, a series of movements were devised. The actions to be tested were: sitting down (from standing), standing up (from sitting), walking up stairs, walking down stairs, walking on a flat surface, bending over, and falling. Each of the previous movements would then be performed three times. For reference, each test was performed by a 6.0’ tall, 196lb male. The fall detector was worn looped through his belt, approximately 41” off the ground. The belt was also tightened considerably, as to be prevent the fall detector from bouncing excessively and providing inaccurate readings. Standards regarding how each of the test movements were performed can be found listed below:

- Sitting Down (from standing): The individual wearing the fall detector was positioned to stand with the back of their knees approximately 1” from the seat of the chair. The individual then sat down in one controlled and fluid motion. The seat of the chair was measured to be approximately 18” off the ground.
- Standing Up (from sitting): The individual wearing the fall detector began seated in a chair with their back straight and their feet firmly planted on the ground. The individual then stood up in one controlled and fluid motion. The seat of the chair was measured to be approximately 18” off the ground.
- Walking Up Stairs: The individual wearing the fall detector walked up 21 stairs at a constant speed. The height of each stair was measured to be approximately 7.5”.
- Walking Down Stairs: The individual wearing the fall detector walked down 21 stairs at a constant speed. The height of each stair was measured to be approximately 7.5”.
- Walking on a Flat Surface: The individual wearing the fall detector walked 5 paces at a constant speed.
- Bending Over: From standing, the individual wearing the fall detector bent over as if to pick up an object lying on the floor. There was a minimal bend in the legs of the individual as they performed this movement. The angle between the wearers torso and legs was measured to be approximately 90˚ when fully bent over.
- Falling: To avoid injury, but still ensure accuracy, the individual wearing the fall detector was positioned on their knees with the fall detector on their right side. The individual would then fall onto their left side, trying not to absorb any of the impact with their hands.

To test each of the movements listed above, the program depicted in Appendix E (9) was uploaded onto the ESP32 microcontroller. This program was designed to compute the magnitude of the total acceleration (|a|) and angular velocity ($|v_\theta|$) of the device, and display both values on the Arduino IDE serial monitor. A 3m long micro-usb cable was then connected to the fall detector and the computer running the Arduino IDE. Next, the serial plotter option was selected on the Arduino IDE as to provide a visual representation of |a| and $|v_\theta|$ in relation to time. Once setup, the seven actions outlined above were each performed to the described standards. Before performing any of the actions listed above, the individual wearing the fall detector would stand as still as possible so that the graphs displaying |a| and $|v_\theta|$ would level off. Then once the actions were performed, the corresponding changes in |a| and $|v_\theta|$ were easily distinguishable. Once a movement was performed, a screenshot was taken of the |a| and $|v_\theta|$ graphs displayed on the serial monitor. The focus of these graphs was to observe the maximum and minimum |a| and $|v_\theta|$ values that were reached while each of the movements were performed. These values could then be compared to the maximum and minimum |a| and $|v_\theta|$ values seen in the event of a fall. From this comparison an effective threshold for both |a| and $|v_\theta|$ could be determined.

Once effective thresholds for both |a| and $|v_\theta|$ had been determined, to test their accuracy, the fall detector was attached to a chair which was then tipped over multiple times. The chair toppling over onto its side showed similar |a| and $|v_\theta|$ graphs to those of a human falling onto their side, when compared. Also, the use of the chair allowed for many repeated tests without worrying about the health and safety of the individual wearing the fall detector. Refer to Appendix E (9) for data collected during this portion of the project.

Results (3.5)

The graphs collected during the experimental testing described in section (3.4) of the report above, were used to determine the values at which the |a| and $|v_\theta|$ thresholds would be set to. From these tests it became apparent that for most movements, the only forces acting on the device were the forces of gravity, giving a magnitude of total acceleration (|a|) between 0 and 1 g’s. The two exceptions to this, were falling and walking down stairs, the falling tests showed that the fall detector experienced a |a| between 3 and 5 g’s, and the walking down stairs tests showed a |a| between 2 and 3g’s. Furthermore, when observing the graphs showing the magnitude of total angular velocity ($|v_\theta|$) with respect to time, it was noticeable that falling was the only action that resulted in a large negative value (i.e.

$|v_\theta|$ = -150 ˚/s). This negative magnitude was contributed to the fact that the values achieved during the fall tests regarding the $|v_\theta|$, were too large for the variables in which they were stored. This leads to an overflow, that results in the sign bit getting set to 1, thus giving a negative value. While this could be solved by storing $|v_\theta|$ in a variable with more space (i.e. double, long, etc.), due to the predictable results, that $|v_\theta|$ will go negative in the occurrence of a fall, it was kept this way. From both the previous observations it was concluded that the threshold for |a| would be set to 3g (3*4096=12288) and that the threshold for $|v_\theta|$ would be set to -100 ˚/s (-100*131=-13100). The program would then test to see if the measured |a| is greater than the set |a| threshold value, and if the measured $|v_\theta|$ is equal to, or less than the set $|v_\theta|$ threshold value. This would allow the fall detection program to effectively distinguish between falls and the various other tested movements. When the fall detector was attached to a chair and tipped over, using these thresholds, it recognized 5/5 falls, and could distinguish between the other tested movements.

Safety Procedures (3.6)

Refer to the list below for the various safety measures/procedures employed during the testing and design of the fall detector:

- Any safety precautions/measures in place were effectively understood and followed.
- The power source was disconnected before any adjustments were made to the circuit.
- Before attempting to do any soldering, a lesson/test was booked with the lab technologist supervising the project.
- When the circuit was powered, the exposed connections and components were not handled.
- The circuit was connected and disconnected properly and with great care.
- While soldering the proper safety procedures were followed (i.e. wearing gloves while touching lead solder, filtering fans turned on to avoid inhalation of toxic fumes, etc.).
- Great care was taken while soldering to avoid burning the components, oneself and/or individuals in close proximity.
- While testing to see at which threshold the fall detector would detect falls, safety measures were employed to avoid injury (i.e. falling onto a soft surface, clearing the area of potential hazards, falling from knee height, etc.).
- Power was always removed from the circuit when leaving the room.
- No food or drinks were allowed in close proximity with the laboratory equipment.
- Sensitive components were handled with great care as to avoid damage and potentially dangerous malfunctions.
- There were no compromises made regarding safety measures/procedures during the testing and design of the technology project.

Conclusion (4)

To conclude, the main objective of this project was successfully achieved. A program was designed that could detect and distinguish falls from the various other movements an elderly and/or ill individual might perform throughout their day. Furthermore, if a fall was detected, this program would then provide a status update to a web page indicating that a fall has occurred and that help is needed. Regarding the physical components of the fall detector, this project was completed using an ESP32 microcontroller and a MPU 6050 Arduino module. To power both components a 9V battery was input into a KIS-3R33S converter that stepped the voltage down to 5V and then output this into the 5V and GND pins on the ESP32 microcontroller. To house these components a case was designed and 3D printed using FreeCAD.

Appendices

- Parts List (5)
- KIS-3R33S Schematic Diagram and ESP32 Microcontroller Pinout Diagram (6)
- Fall Detection Program (7)
- FreeCAD Diagrams and Fall Detector Layout (8)
- Experimental Results (9)
- Contingency Planning (10.1), Group Contributions (10.2), and Log Book and Gantt Chart (10.3)
- Web Page Layout (11)
- References (12)

Appendix A (5) - Parts List

Refer to table (4) below for a complete list of the components used to construct the fall detector and their corresponding role in the project.

Components Used (Name and Manufacturer Part/Model Number)	Role in Fall Detector
Heltec Automation ESP32 Development Board Nodemcu Chip With 0.96inch OLED Display Bluetooth WIFI Kit32 Arduino Compatible CP2012 USB To Serial Chip (HTIT-WB32)	Used to read data from the MPU 6050 Arduino module and detect the occurrence of a fall. Also acts as a web server that responds to clients with a web page displaying the status of the fall detector.
Aukru MPU 6050 Module 3 Axis Gyroscope + 3 Axis Accelerometer Module for Arduino (MPU-6050-Gyro-mould GY-521)	Used to measure the 3D acceleration and positioning of the fall detector, this data was read by the ESP32 microcontroller to detect falls.
Energizer 522BP Max Alkaline Battery, 9V, 1 Pack (522BP)	Used to power the ESP32 microcontroller and MPU 6050 Arduino module.
DC 7V-24V To DC 5V 3A USB Output Converter Step Down Module KIS-3R33S (5559182658, similar model)	Used to step down the 9V input from the Energizer battery to a constant 5V, this 5V was then supplied to the ESP32 microcontroller via the 5V and GND pins.
Battery Connector, Snap 9V 1 Cell Wire Leads - 4”/101.6mm (232)	Instead of directly soldering wires to the 9V Energizer battery, this connector was used to make it easier to switch out the battery.
4 x 3mm Socket Button Screws (SSB-M2-3-A4)	Used to secure the 7x5cm PCB to the interior of the 3D printed case.
7x5cm PCB Blank Circuit Board Prototype Paper Solder Circuit Panel (A0053)	The ESP32 microcontroller and MPU 6050 Arduino module were connected to this PCB, wires were then soldered between the necessary pins.

Table (4) - Complete list of the components used to construct the fall detector and their corresponding role.

Appendix B (6) - KIS-3R33 Schematic Diagram and ESP32 Microcontroller Pinout Diagram

Refer to figure (2) below for the schematic diagram for the KIS-3R33S converter used in the fall detector power supply circuit.

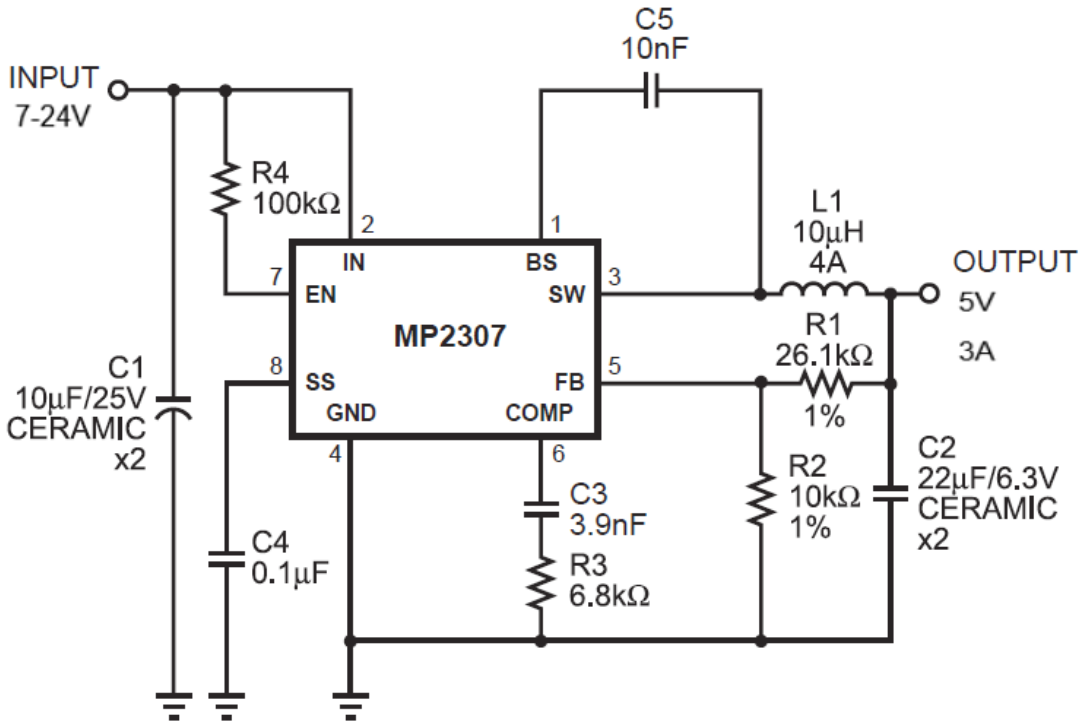


Figure (2) - KIS-3R33S schematic diagram.

Refer to figure (3) below for the pinout diagram for the ESP32 microcontroller used in the fall detector.

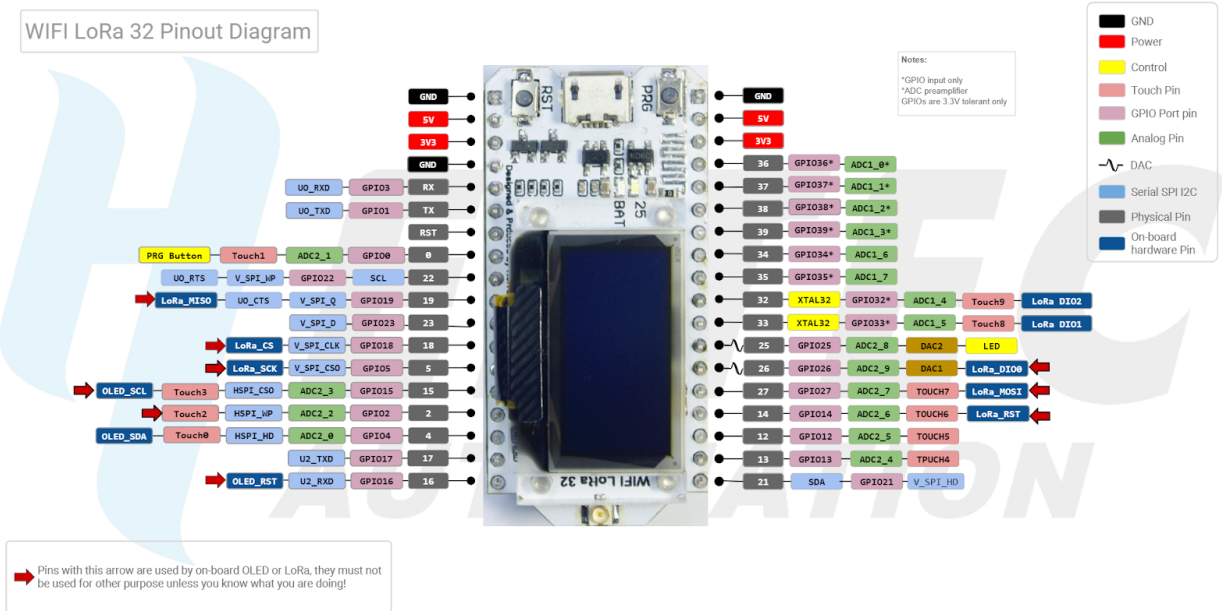


Figure (3) - ESP32 microcontroller pinout diagram.

Appendix C (7) - Fall Detection Program

```
#include <WiFi.h> //gives the ESP32 microcontroller the ability to connect to
a wireless network
#include <WiFiServer.h> //enables the ESP32 microcontroller to act as a web
server and listen for potential clients
#include <WiFiClient.h> //allows a client to request data from the ESP32
microcontroller
#include <String.h> //enables various string operations
#include <SPI.h> //gives the ESP32 microcontroller the ability to communicate
with SPI devices
#include <Wire.h> //gives the ESP32 microcontroller the ability to
communicate with I2C devices (i.e. MPU 6050)

const char* network_name = "ATAC4020"; //name of wireless network that ESP32
microcontroller will be connecting to
const char* network_password = "laketamblyn"; //password for the wireless
network
WiFiServer server(80); //the ESP32 board (server) will respond to the clients
(web browsers) on port 80 which is standard

//registers used to configure the MPU 6050 gyroscope and accelerometer
#define sample_rate 0x19
#define configure 0x1A
#define gyro_config 0x1B
#define accl_config 0x1C
#define accl_data 0x3B
#define gyro_data 0x43
#define sensor_add 0x68
#define user_control 0x6A
#define pwr_mngt 0x6B

//parameters for fall detection algorithm
#define accl_thres 12288 //magnitude of total acceleration threshold,
|a|=12288/4096=3g
#define gyro_thres -13100 //magnitude of total angular velocity threshold,
|v|=-13100/131=-100 degrees/second
int16_t AcX, AcY, AcZ, GyX, GyY, GyZ; //variables that will store readings
from the MPU 6050 sensor registers
int16_t as, gs; //variables that will store the magnitude of total
acceleration (as) and angular velocity (gs)
int ht_bit = 0; //this variable will be set to 1 in the occurrence of a fall,
initialized to 0

void setup() { //initial setup code, runs once
```

```
  //initialization of MPU 6050 sensors
  Wire.begin(); //initiates the wire.h library and joins the I2C bus
  Wire.beginTransmission(sensor_add); //begins transmission to the I2C slave
device (sensor_add)
  Wire.write(pwr_mngt); //sends pwr_mngt byte to device
  Wire.write(0);   //wakes device up from sleep mode
  Wire.endTransmission(true); //ends transmission
  Wire.beginTransmission(sensor_add);
  Wire.write(configure); //sends configure byte to device
  Wire.write(1);        //configures the gyroscope output to 1kHz, and the
accelerometer output to 1kHz
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(sample_rate); //sends sample_rate byte to device
  Wire.write(9);   //initializes the devices sample rate to 100Hz
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(gyro_config); //sends gyro_config byte to device
  Wire.write(0);  //configures the gyroscope to operate at full scale range
(250 degrees/second), sensitivity of 131
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(accl_config); //sends accl_config byte to device
  Wire.write(16);   //configures the accelerometer to operate at full scale
range (8g), sensitivity of 4096
  Wire.endTransmission(true);

  //initialization of the web server
  Serial.begin(115200); //sets the serial data transmission rate to 115200
bits per second
  WiFi.begin(network_name, network_password); //starts the connection to the
designated wireless network
  while (WiFi.status() != WL_CONNECTED) { //loop repeats until the ESP32
microcontroller is connected to the wireless network
    Serial.print("Attempting to connect to network: ");
    Serial.println(network_name); //prints the name of the wireless network
being connected to
    delay(10000); //delays for 10 seconds while the ESP32 microcontroller
attempts to connect to the wireless network
  }
  Serial.print("Successfully connected to network: ");
  Serial.println(network_name); //prints the name of the connected wireless
network
  server.begin(); //the ESP32 microcontroller (server) can now begin
listening for potential clients
  Serial.print("Successfully started web server, search for the following IP
address in a web browser to observe status of the fall detector: ");
```

```
  Serial.println(WiFi.localIP()); //prints the IP address of the ESP32
microcontroller, the HTML document it returns shows the status of the fall
detector
  delay(10000); //waits 10 seconds to all for the user to read and search the
IP address
}

void loop() { //section of the program that will run repeatedly

  //fall detection algorithm
  Wire.beginTransmission(sensor_add); //begins transmission to the I2C slave
device (sensor_add)
  Wire.write(accl_data); //queues accelerometer data for transmission
  Wire.endTransmission(false); //the connection is kept active
  Wire.requestFrom(sensor_add,6,true); //ESP32 microcontroller requests
accelerometer readings from MPU 6050

  //reads x, y, and z axis accelerations from registers
  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();
  as=sqrt(sq(AcX) + sq(AcY) + sq(AcZ)); //the magnitude of total acceleration
is calculated using the read accelerations
  if(as > accl_thres){ //tests the calculated magnitude of total acceleration
against the set threshold values
    Wire.beginTransmission(sensor_add); //begins transmission to the I2C
slave device (sensor_add)
    Wire.write(gyro_data); //queues gyroscope data for transmission
    Wire.endTransmission(false); //the connection is kept active
    Wire.requestFrom(sensor_add,6,true); //ESP32 microcontroller requests
gyroscope readings from MPU 6050

      //reads x, y, and z axis angular velocities from registers
      GyX = Wire.read() << 8 | Wire.read();
      GyY = Wire.read() << 8 | Wire.read();
      GyZ = Wire.read() << 8 | Wire.read();
      gs = sqrt(sq(GyX)+ sq(GyY)+ sq(GyZ)); //the magnitude of total angular
velocity is calculated using the read angular velocities
      if(gs <= gyro_thres){ //tests the calculated magnitude of total angular
velocity against the set threshold value
      ht_bit = 1; //sets this variable to 1, indicating that a fall has
occurred
      }
      }

  //web server portion of program
  WiFiClient client = server.available(); //allows the program to read data
from clients connected over the same wireless network
```

```
  if (client){ //waits for a client to connect, this happens once the servers
IP address is searched for in a browser
    Serial.println("\nNew client has connected.");
      while (client.connected()){ //this while loop will continue to repeat
as long as the client is connected
      if (client.available()){ //this if statement is true if the web server
receives an HTTP request

      //an HTTP request ends with a carriage return and a line feed (\r\n)
      //the incoming HTTP request is stored in a string and read until '\r\n'
is received
      //'\r\n' has a length of 1, so the program checks that the length of
the string is 1, to ensure that these are the last characters received
      //the program then responds to the client with an HTML document, and
breaks out of the while loop
      String HTTPrequest = client.readStringUntil('\r\n');
      if (HTTPrequest.length() == 1){
          client.println(HTMLWebPage());
            break;
            }
      }
      }
  delay(500); //this 0.5 second delay gives the client time to receive the
requested data
  client.stop(); //disconnects the client from the web server
  Serial.println("Client has disconnected.");
  }
}

String HTMLWebPage(){ //function that will return a string containing the
HTML document that the server responds to clients with
  String WebPage = String("HTTP/1.1 200 OK\r\n") + //status code that
indicates that the request from the client has succeeded
  "Content-Type: text/html\r\n" + //header that lets the client know the type
of content it will be receiving
  "Connection: close\r\n" + //the connection between the web server and
client will be closed after the server sends a response
  "Refresh: 1\r\n" + //the client will send requests to the web server every
second
  "\r\n" +
  "<!DOCTYPE HTML>" + //means that the following document is HTML5
  "<html>" + //tag that indicates the start of the HTML document
  "<style>" + //used to define style information for an HTML document
      "body {background-color: teal;}" + //sets the background color of the
webpage to teal
      "p2 {color: red;}" + //sets the color of the text in p2 to red
  "</style>" + //end of style tag
  "<head>" + //the head tag contains meta information about the document
```

```
    "<title>Fall Detector Status</title>" + //this title will appear in the
browser window
  "</head>" + //end of head tag
  "<body> <center>" + //the body is where visible page content is held, this
content will be centered
  "<div id=\"header\">" + //div tag is used to group elements together
     "<b> <h1 style='font-size:100px'>Fall Detector Status</h1> </b>" + //h1
text is bolded and has a size of 100px
     "<p1 style='font-size:80px'>Status: </p1> <br />" + //p1 text will have
a size of 80px
     "<br /> <b> <p2 style='font-size:75px'>" + //fall detector status font
will be 75px and the color red
     String(FallDetection()) + //FallDetection function returns the status of
the fall detector
     "</p2> </b>" +//end of p2 and bolded text
  "<div>" + //new group
      "<p style='font-size:40px'>If a fall has been detected, refer to the
hyperlinks posted below:" + //paragraph with a font size of 40px
      "<br /> <a style='font-size:25px',
href='https://www.thunderbay.ca/en/superior-north-emergency-medical-services.a
spx'>-Click here to contact Emergency Medical Services!</a> <!hyperlink>" +
//paragraph containing hyperlink with a font size of 25px
      "<br /> <a style='font-size:25px',
href='https://www.redcross.org/take-a-class/cpr/performing-cpr/cpr-steps'>-Cli
ck here for directions on how to perform CPR!</a> <!hyperlink>" + //paragraph
containing hyperlink with a font size of 25px
    "</p>" + //end of paragraph tag
      "<p style='font-size:20px'>In the event that the fall detector detects
a fall, or wrongly detects a fall, remove the orange cover from the case and
press the button labelled 'RST' to restart the fall detection program. </p>"
+ //paragraph with a font size of 20px
    "</div>" + //end of grouped elements
    "</center>" + //signals the end of the center tag
    "</html>" + //indicates the end of the HTML document
    "\r\n";
      return WebPage; //returns the HTML document as a string when the
function is called upon
}

//this function is used to update the status of the fall detector, displayed
on the web page
//the fall detector status depends on the value of the variable, ht_bit
//ht_bit is set to 1 once a fall has occurred
//if this variable is set to 0, the function returns "NO FALL DETECTED"
//if this variable is set to 1, the function returns "FALL DETECTED"
String FallDetection(){
  if(ht_bit == 1){
      String FallDetected = String("FALL DETECTED");
```

```
        return FallDetected;
    }
    else{
        String NoFallDetected = String("NO FALL DETECTED");
        return NoFallDetected;
    }
}
```

Appendix D (8) - FreeCAD Diagrams and Fall Detector Layout

Refer to figures (4) to (7) below to view the fall detector case, from different angles, as modelled on FreeCAD.

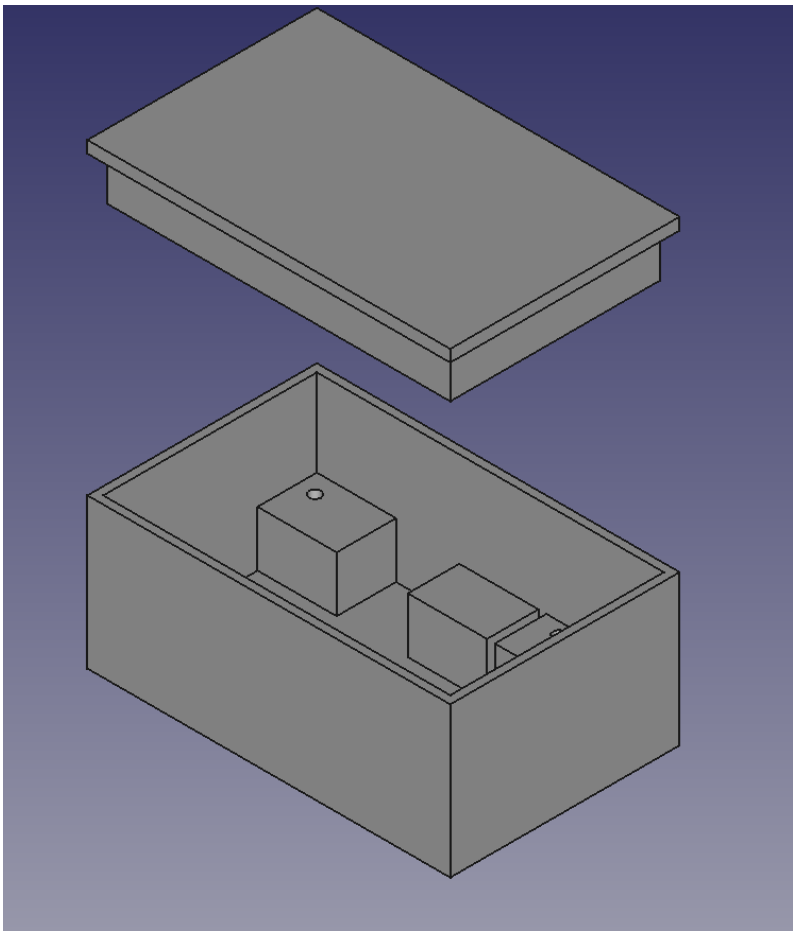


Figure (4) - Axonometric view of case and cover.

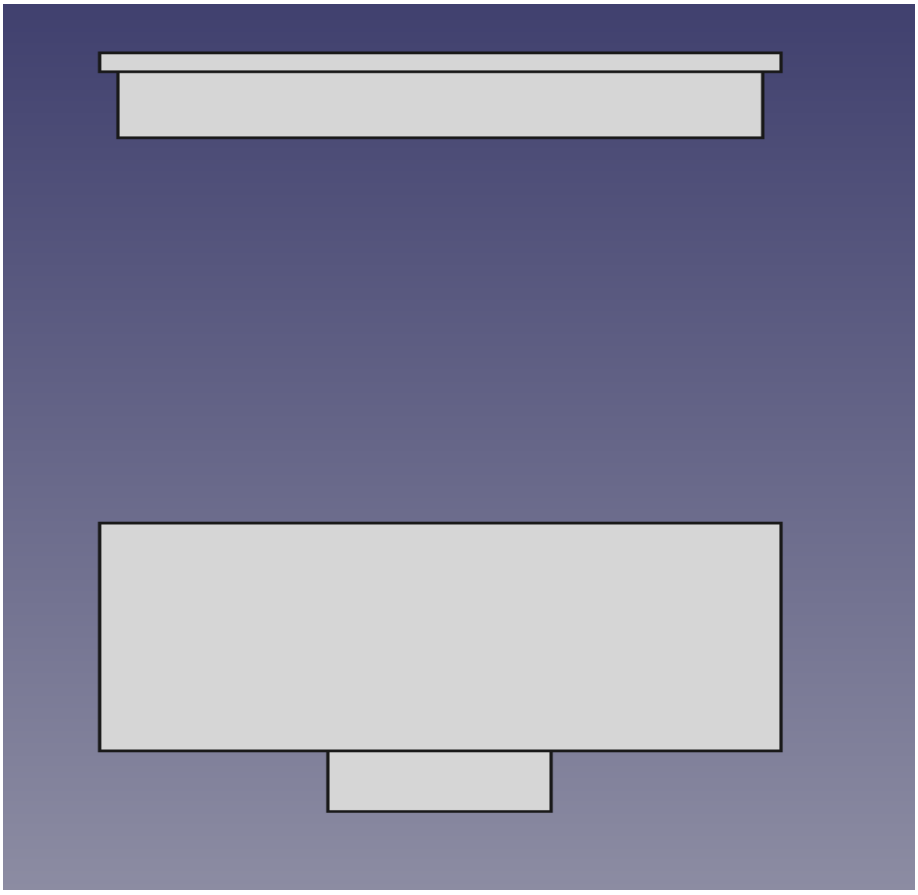


Figure (5) - Front view of case and cover.

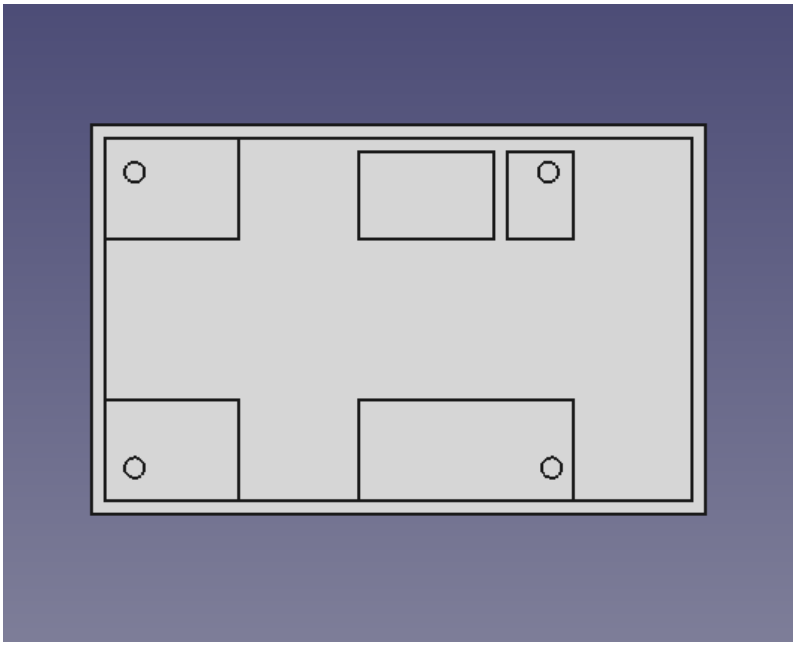


Figure (6) - Top view of case with cover hidden.

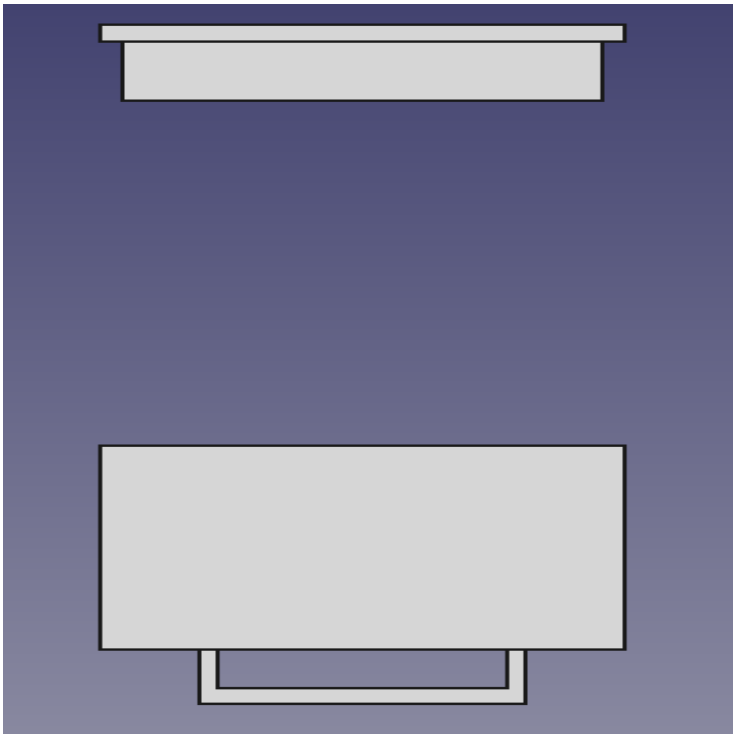


Figure (7) - Right side view of case and cover.

Refer to figures (8) and (9) below for images showing how the components were laid out inside the fall detector case.

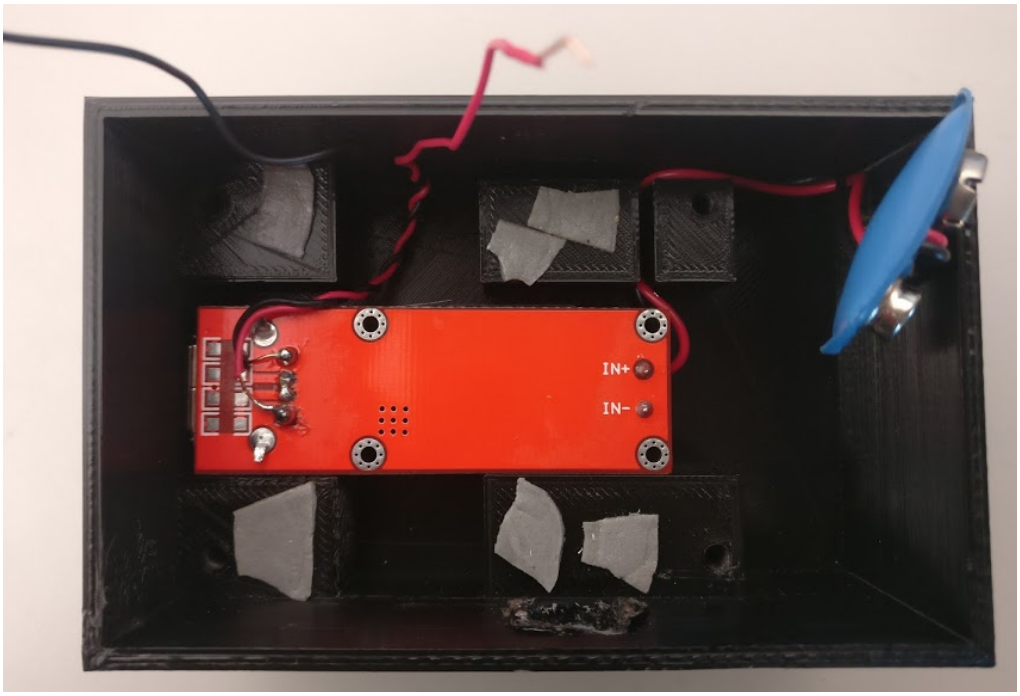


Figure (8) - Fall detector casing with only the KIS-3R33S converter inside.

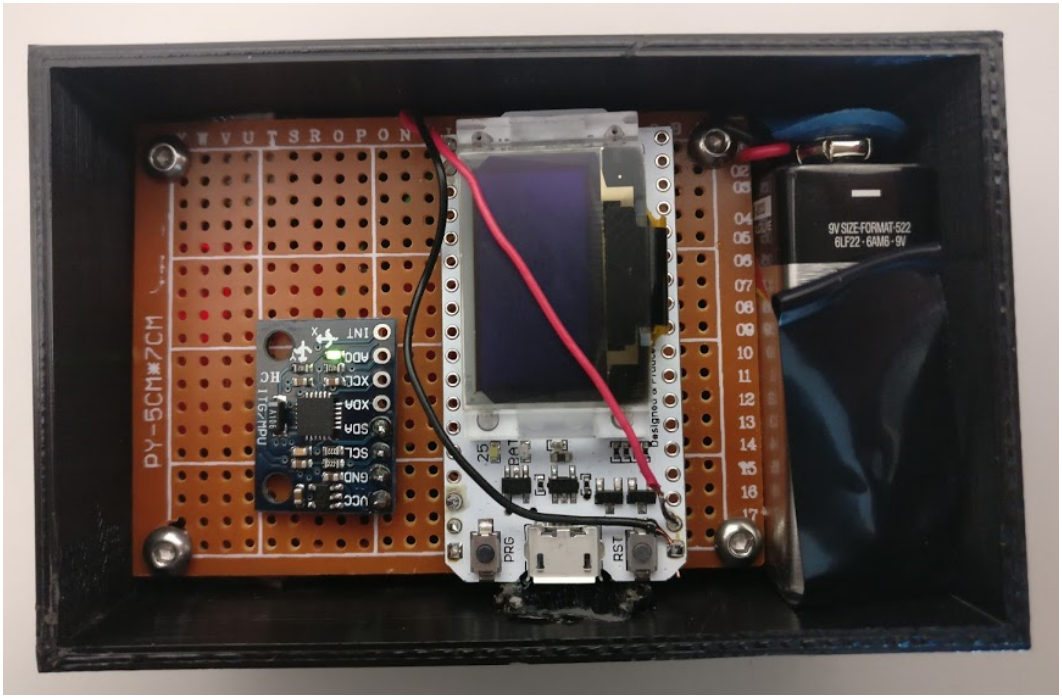
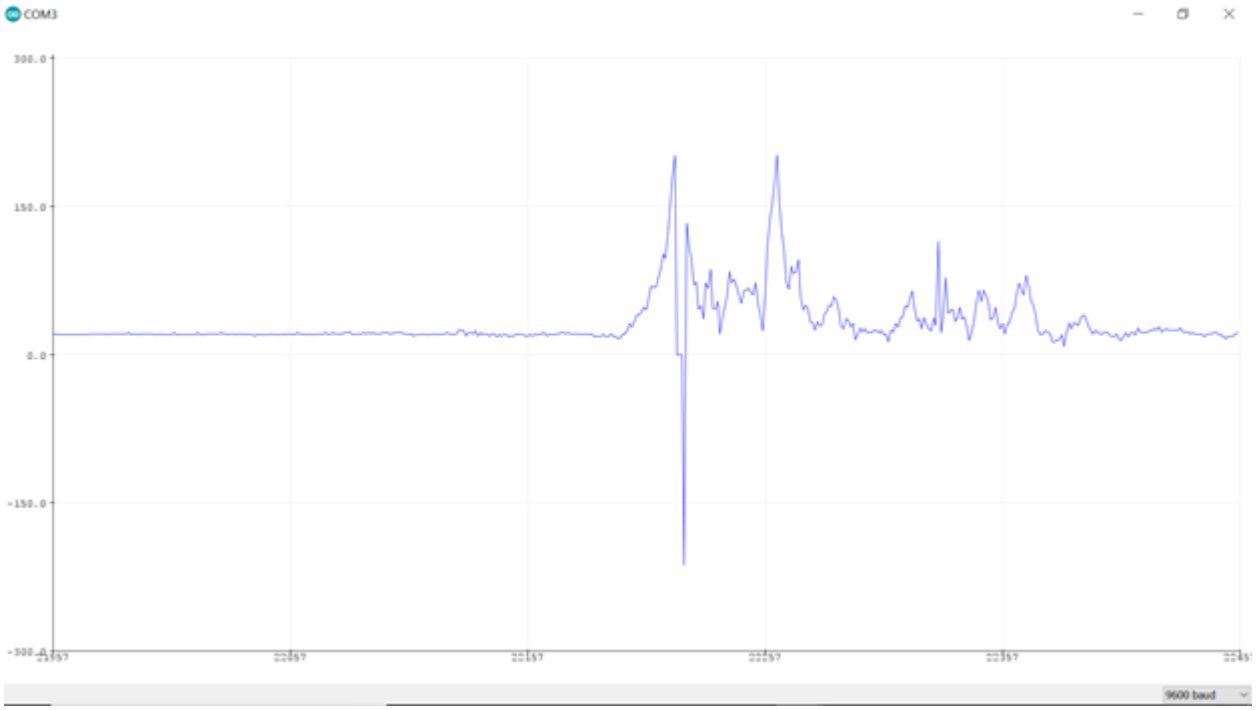


Figure (9) - Fall detector casing once all components had been added.

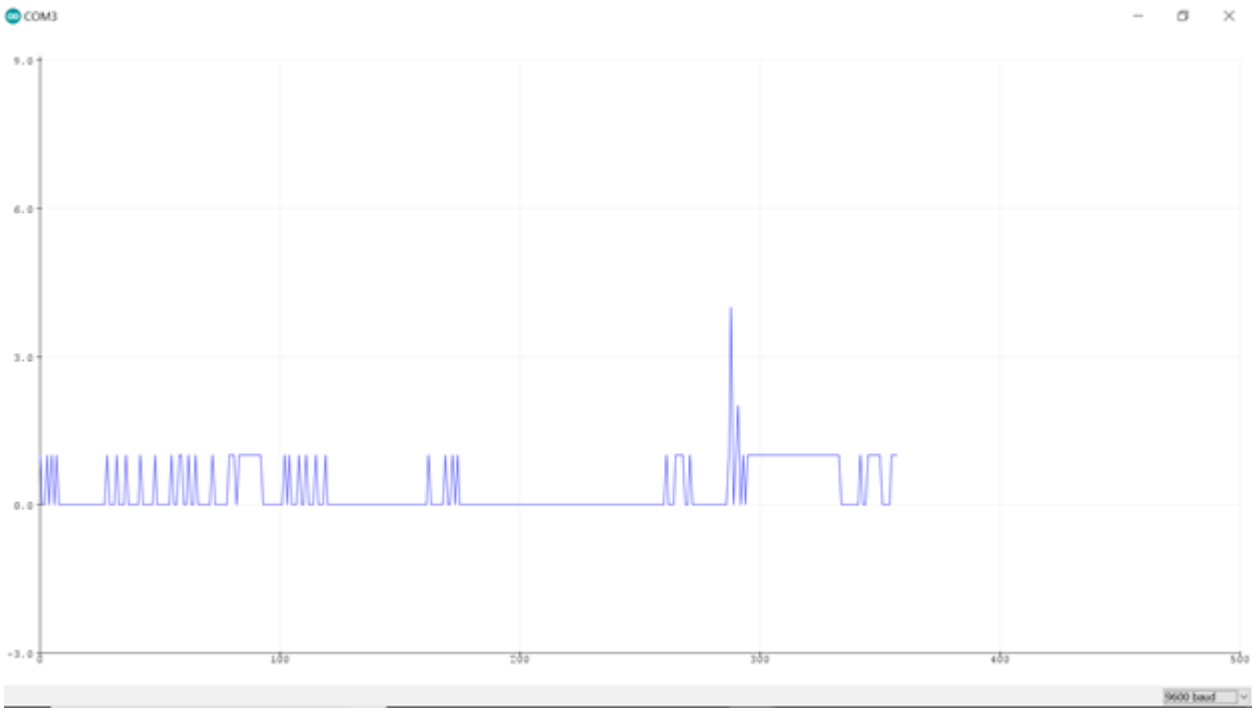
Appendix E (9) - Experimental Results

Refer to graphs (1) to (13) below for the experimental results gathered while testing for the most effective threshold values. These graphs show the magnitude of total acceleration (|a|) and/or the magnitude of total angular

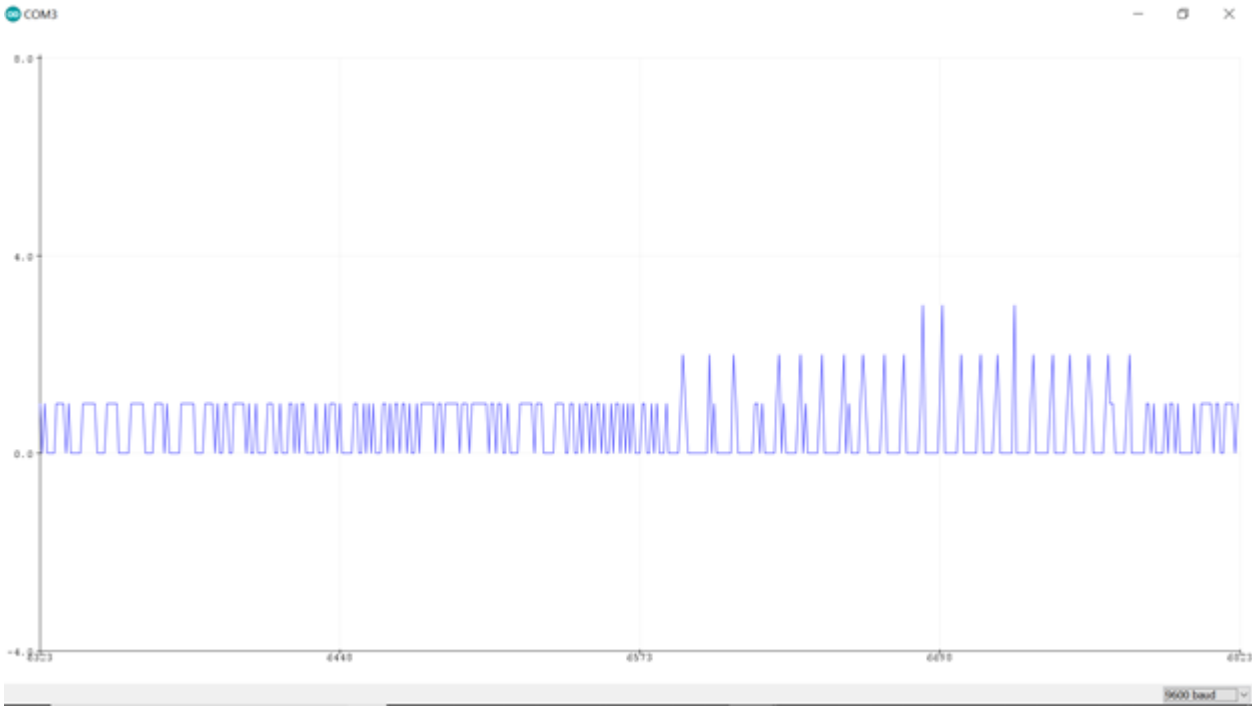
velocity ($|v_\theta|$), with respect to time. Time is shown on the x-axis while both |a| and $|v_\theta|$ are measured along the y-axis. If both |a| and $|v_\theta|$ are shown on the same axis, then |a| has been multiplied by a factor of 50, as to provide legible results.



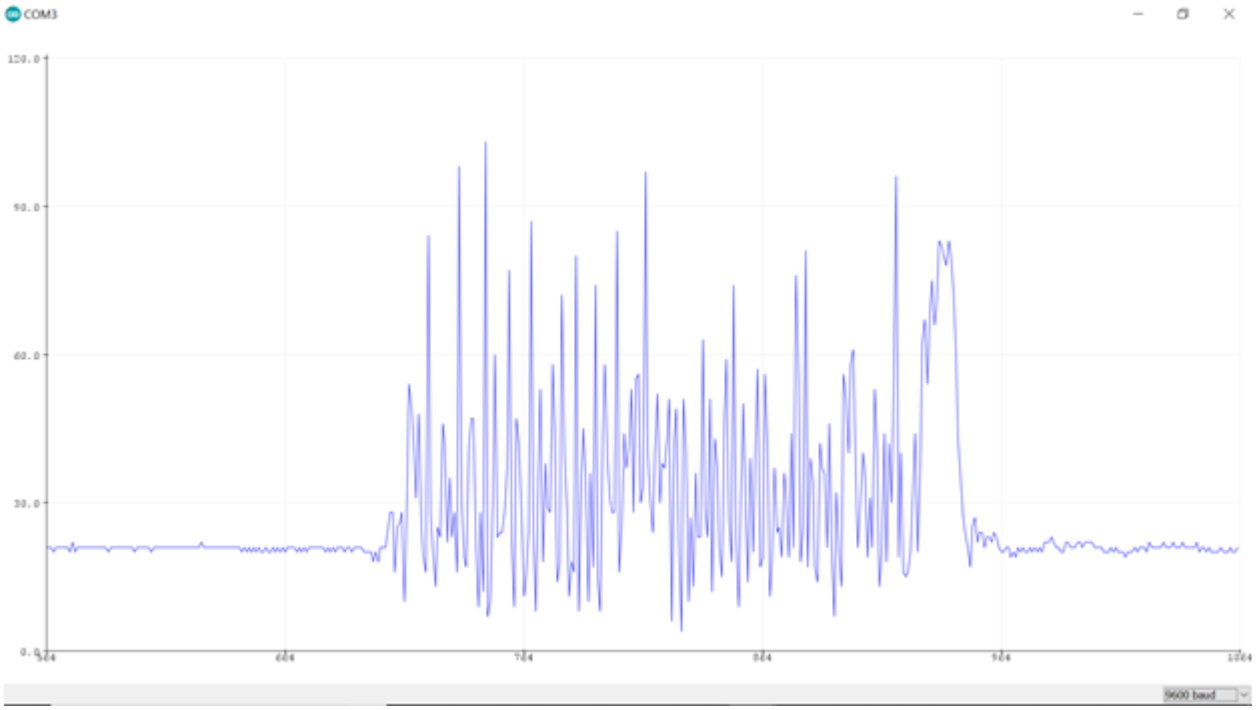
Graph (1) - $|v_\theta|$ vs. time during fall test.



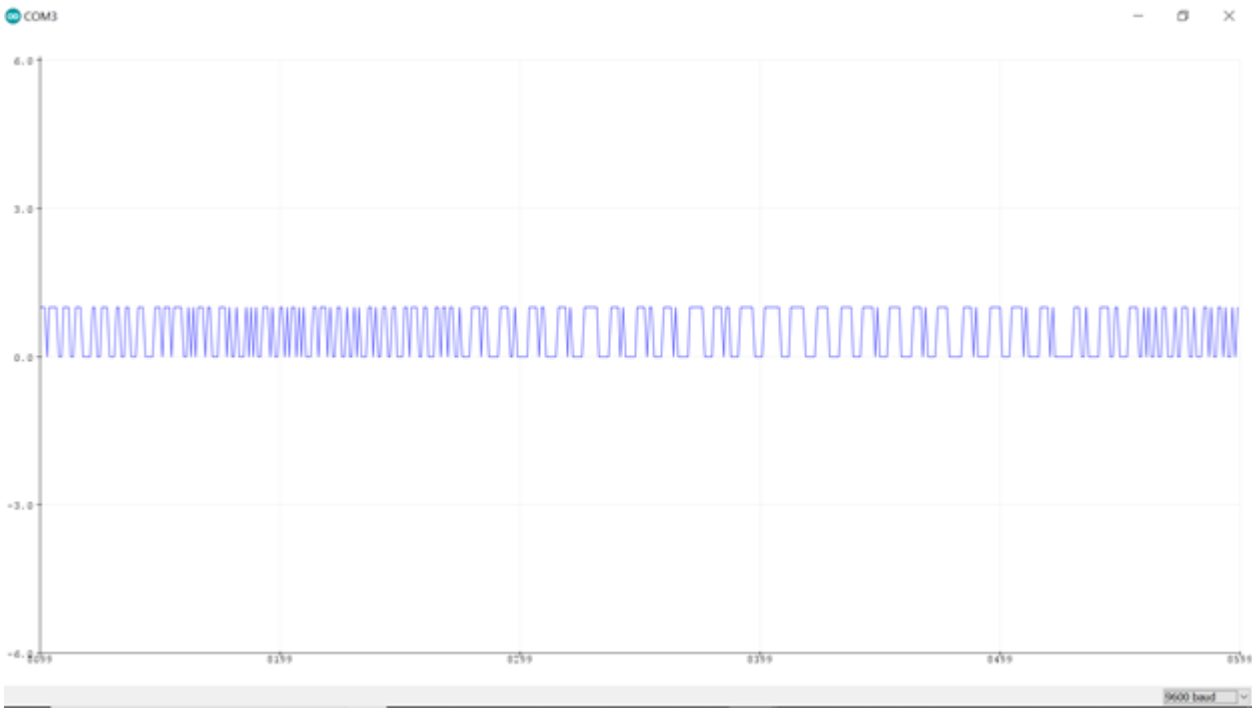
Graph (2) - |a| vs. time while falling.



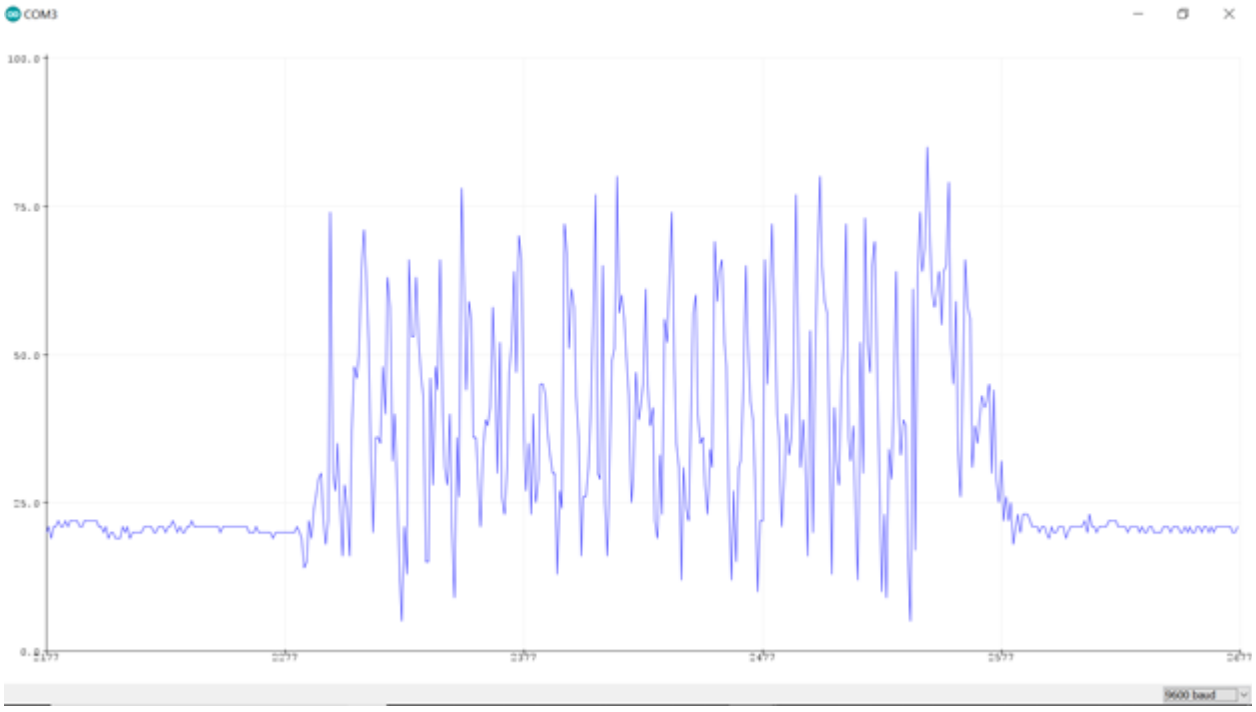
Graph (3) - |a| vs. time while walking down stairs.



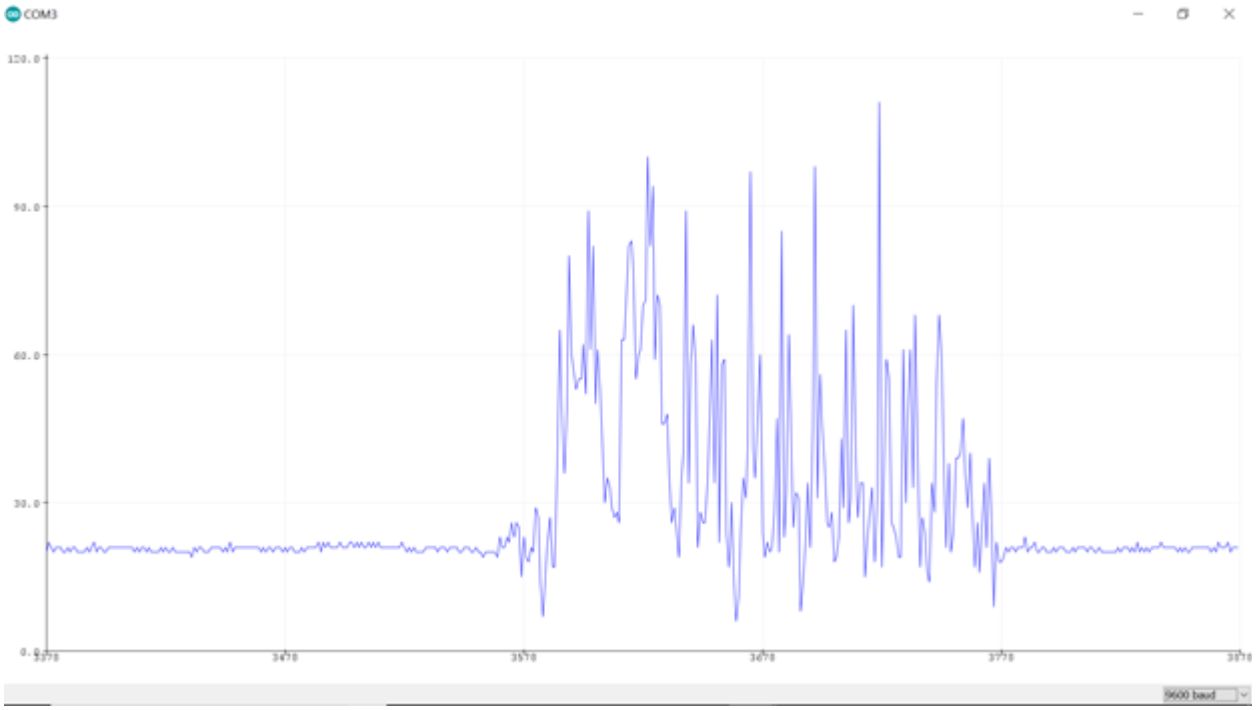
Graph (4) - $|v_\theta|$ vs. time while walking down stairs.



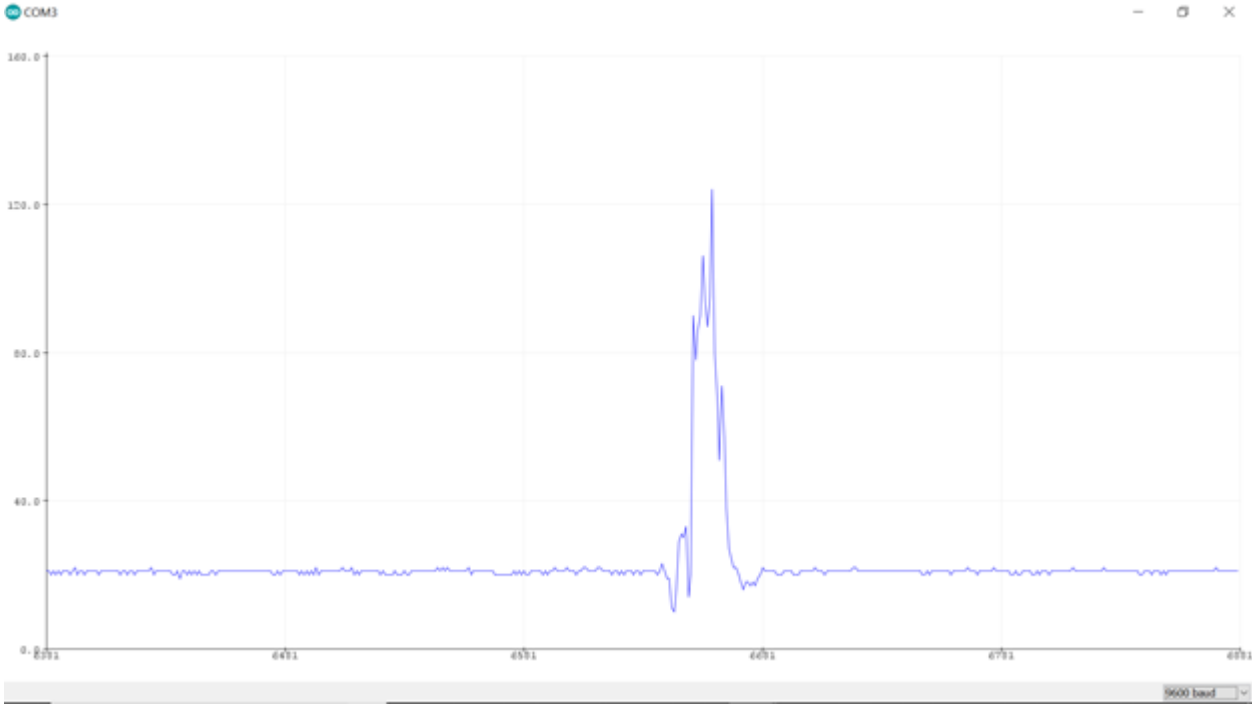
Graph (5) - |a| vs. time while walking up stairs.



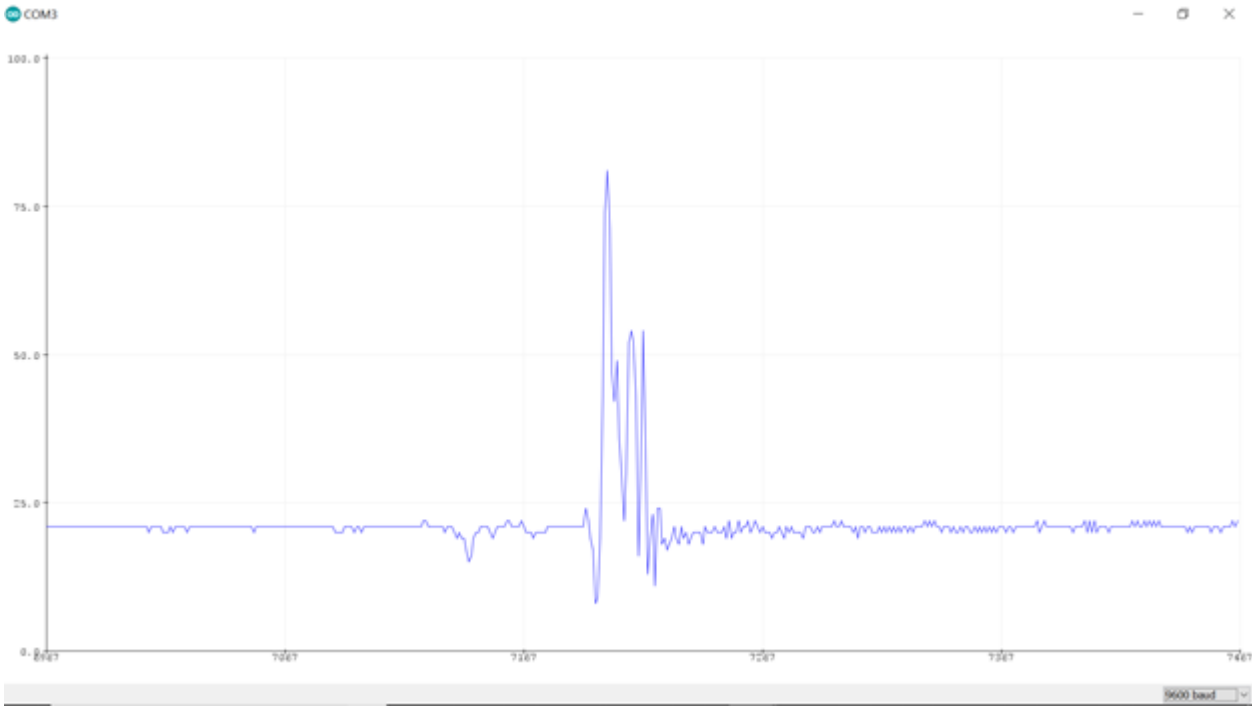
Graph (6) - $|v_\theta|$ vs. time while walking up stairs.



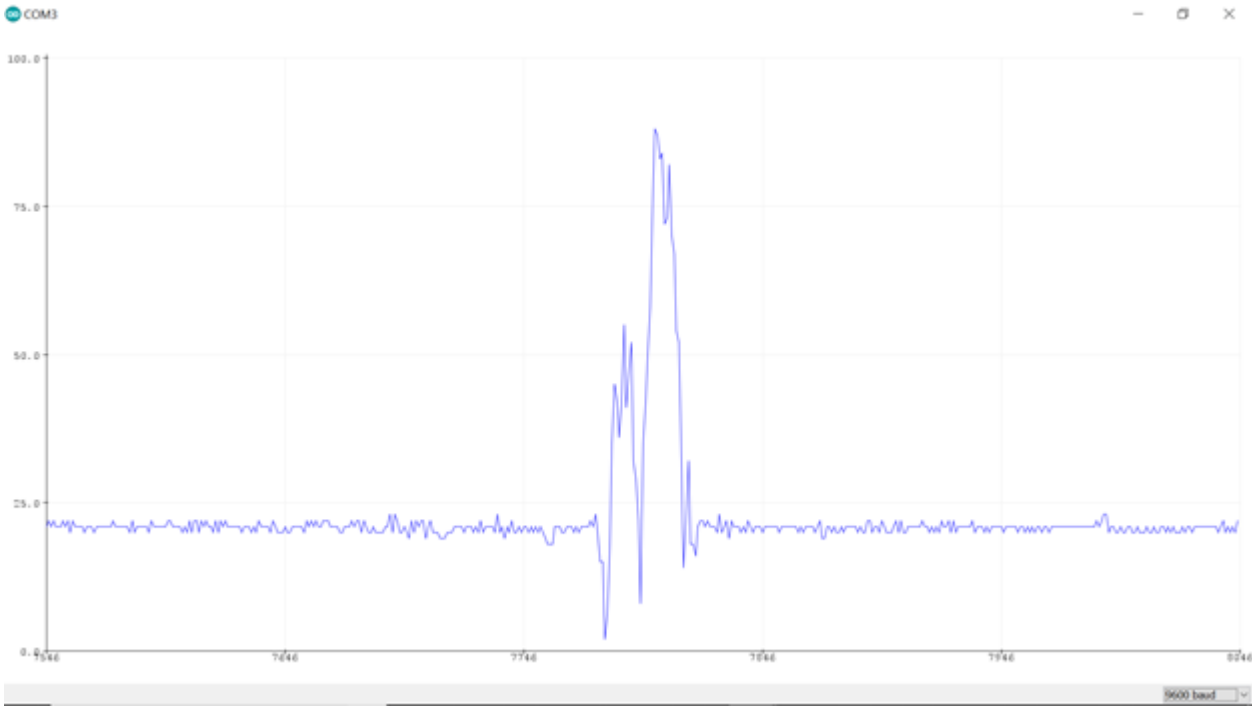
Graph (7) - $|v_\theta|$ vs. time while walking on a flat surface.



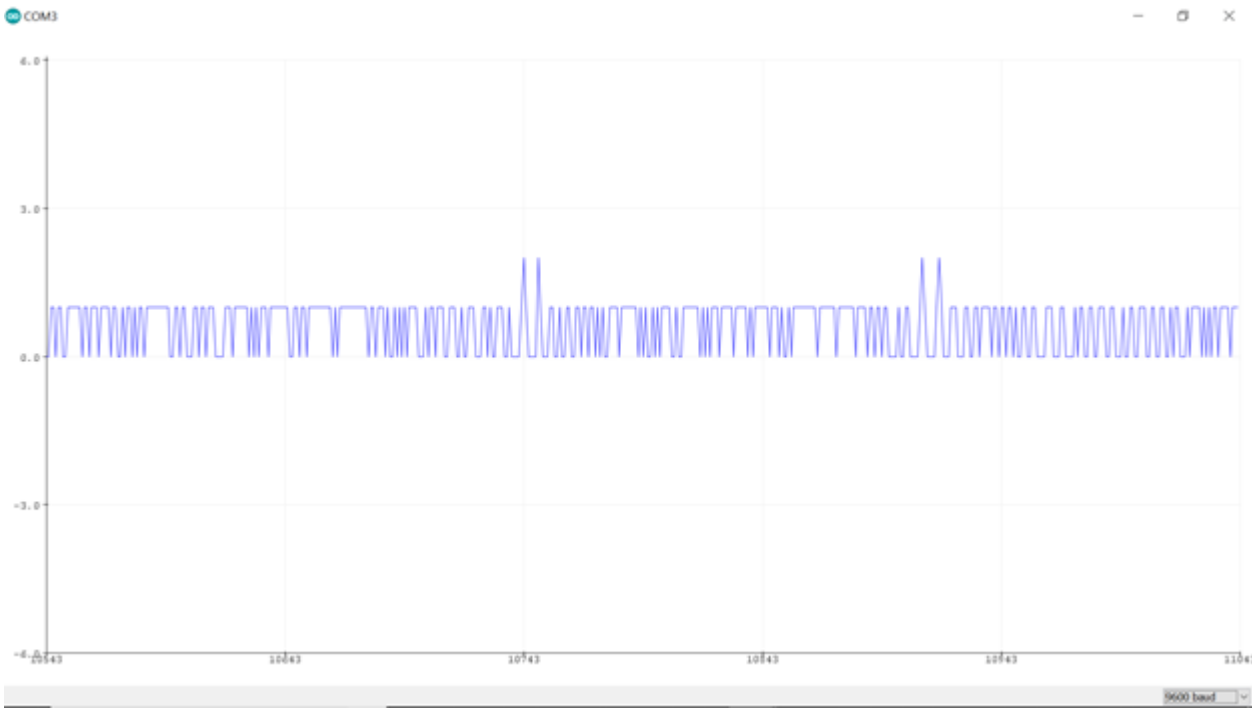
Graph (8) - $|v_\theta|$ vs. time while sitting down (from standing).



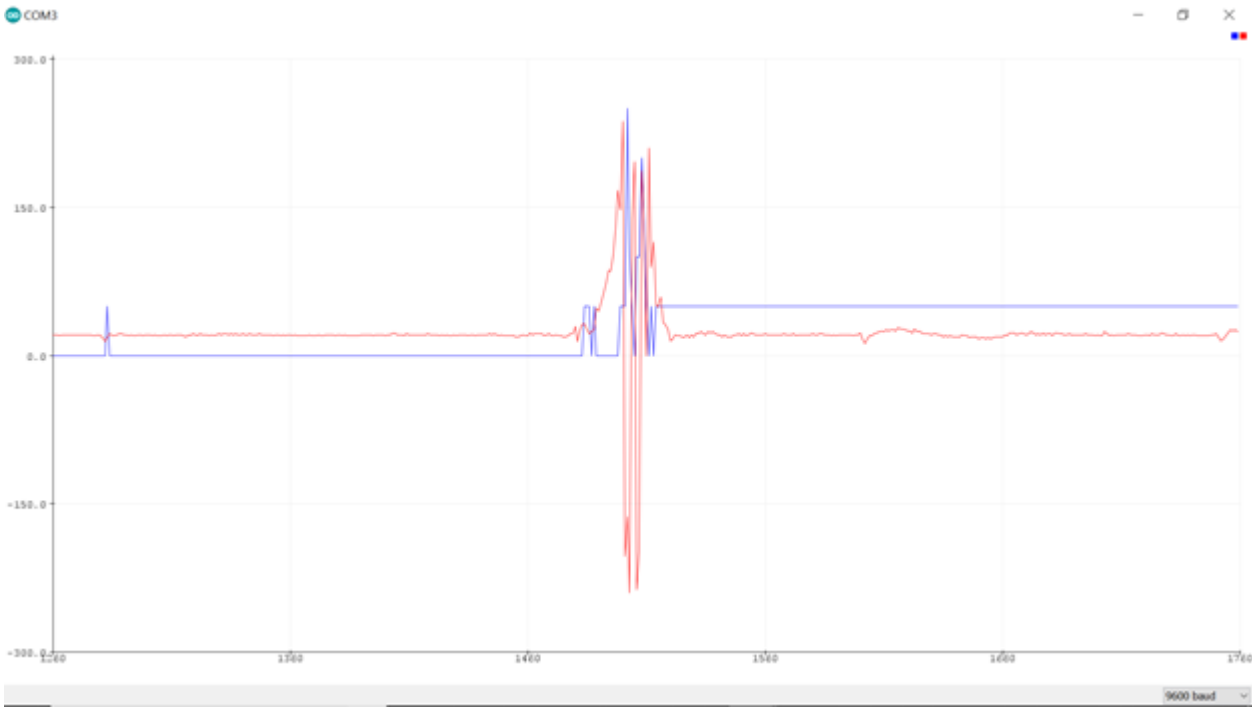
Graph (9) - $|v_\theta|$ vs. time while standing up (from chair).



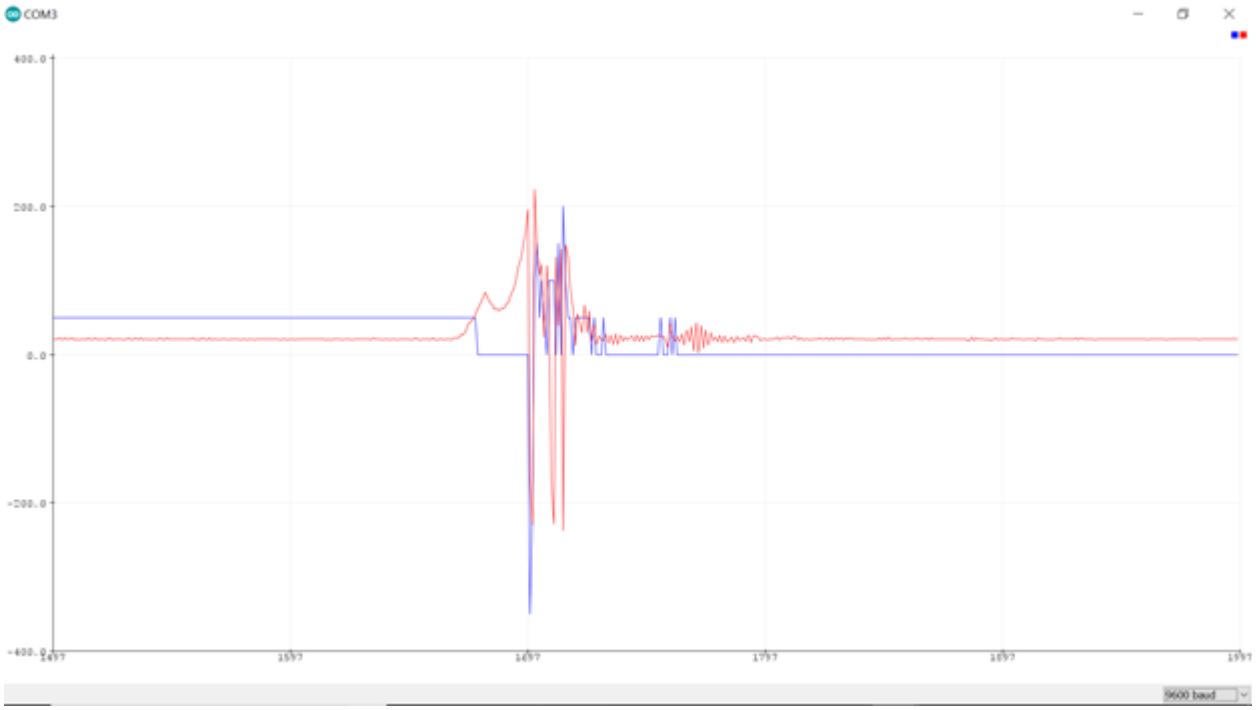
Graph (10) - $|v_\theta|$ vs. time while bending over.



Graph (11) - |a| vs. time approximation for bending over, sitting down (from standing), standing up (from sitting), and walking on a flat surface.



Graph (12) - |a| and $|v_\theta|$ vs. time while falling.



Graph (13) - |a| and $|v_\theta|$ vs. time when the fall detector was attached to a chair and tipped over.

Refer to table (5) below for the approximate maximum and minimum, |a| and $|v_\theta|$ values measured for each of the tested movements.

Actions	Min. |a| (g)	Max. |a| (g)	Min. $\|v_\theta\|$ (˚/s)	Max. $\|v_\theta\|$ (˚/s)
Sitting Down (from standing)	0	1	10	125
Standing Up (from sitting)	0	1	10	80
Walking Up Stairs	0	1	5	90
Walking Down Stairs	0	3	5	110
Walking on a Flat Surface	0	1	5	112
Bending Over	0	1	3	88
Falling	2	4	-200	200
Falling (chair)	-4	4	-225	225

Table (5) - Experimental results.

The program shown below was used for the experimental tests detailed in section 3.4 of the report.

```
#include<Wire.h> //necessary library

// Registers used to configure the gyro and accel
#define sample_rate 0x19
#define configure 0x1A
#define gyro_config 0x1B
#define accl_config 0x1C
#define accl_data 0x3B
#define gyro_data 0x43
#define sensor_add 0x68
#define user_control 0x6A
#define pwr_mngt 0x6B

int16_t AcX, AcY, AcZ, GyX, GyY, GyZ; //variables to read sensor registers
int16_t as, gs; //magnitude of total acceleration and angular velocity
int16_t ag, gd; //magnitude of total acceleration and angular velocity when
divided by proper sensitivity

void setup() {
  Wire.begin();
  Wire.beginTransmission(sensor_add);
  Wire.write(pwr_mngt);
  Wire.write(0);   //wakes device up from sleep mode
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(configure);
  Wire.write(1);        //gyro output= 1kHz, accl output= 1kHz, approx. 2ms
delay
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(sample_rate);
  Wire.write(9);   //sample rate = 100Hz
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(gyro_config);
  Wire.write(0);  //full scale range= 250 degrees/s
  Wire.endTransmission(true);
  Wire.beginTransmission(sensor_add);
  Wire.write(accl_config);
  Wire.write(16);   //full scale range= 8g
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop() {
  //Read Acceleration from registers
  Wire.beginTransmission(sensor_add);
```

```
  Wire.write(accl_data);
  Wire.endTransmission(false);
  Wire.requestFrom(sensor_add,6,true);
  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();
  Wire.beginTransmission(sensor_add);
  Wire.write(gyro_data);
  Wire.endTransmission(false);
  Wire.requestFrom(sensor_add,6,true);
  GyX = Wire.read() << 8 | Wire.read();
  GyY = Wire.read() << 8 | Wire.read();
  GyZ = Wire.read() << 8 | Wire.read();

  as=sqrt(sq(AcX) + sq(AcY) + sq(AcZ)); //calculates the magnitude of total
acceleration
  gs = sqrt(sq(GyX)+ sq(GyY)+ sq(GyZ)); //calculates the magnitude of total
angular velocity

  ag = as / 4096; //dividing value by proper sensitivity
  gd = gs / 131; //dividing value by proper sensitivity

  Serial.print("accel in g = "); Serial.print(50*ag); //multiplies value by
factor of 50 so that the graph is legible
  Serial.print(" gyro in degrees/s = "); Serial.println(gd);
 }
```

Appendix F (10)

- Contingency Planning (10.1)
- Group Contributions (10.2)
- Log Book and Gantt Chart (10.3)

Contingency Planning (10.1)

Refer to table (6) below for the potential risks of this project and the respective management actions.

Risks	Probability of Occurrence	Risk Management Actions
Damage to fall detector components while performing experimental tests (i.e. falling).	High	-Remove the components from the fall detector case and perform a visual inspection to find the damaged component(s). If not visible, use a handheld DMM to determine where the fault lies by checking which devices are and are not receiving power. Once the damaged component(s) has been

		determined, talk with a lab technologist about a possible replacement.
Damage to soldering points while performing experimental tests (i.e. falling).	High	-Remove the components from the fall detector and visually inspect the soldering points to see if any wires have become disconnected and/or the connections have been damaged. Once the faulty soldering point has been determined, resolder the various connections.
Improper soldering causing poor connections (before circuit has been secured in case).	Medium	-Review the circuit and look for faulty soldering connections (i.e. bubbled solder, misplaced solder causing a short circuit). Once the issue has been determined, resolder the connection and test the circuit.
Components malfunctioning (before circuit has been secured in case).	Medium	-Use a handheld DMM to determine if certain components are not receiving power, if so review schematics and possible sources of error. -Check soldering points to see if the connections look faulty, if so resolder. -If none of the above, talk to a lab technologist.
ESP32 microcontroller not being able to connect to designated wireless network.	Medium	-Check the security settings of the designated wireless network, depending on its configuration the ESP32 microcontroller might not be able to connect. If this is the case, setup a router that the ESP32 microcontroller can connect to.
No lab technologists available.	Low	-Note the issues being faced and then focus on other aspects of the project. Once a lab technologist is available, talk to them.
Group is unable to meet.	Low	-Work independently on delegated tasks and decide on another time to meet.

Table (6) - Tabulated contingency plans.

Group Contributions (10.2)

Refer to table (7) below for the tabulated contributions and responsibilities of each group member.

Name of Group Member	Responsibilities and Contributions
Martti Muzyka	Major: ● Web server aspect of overall program ● 3D modelling and design of the fall detector case ● Web page design and creation ● Design of ESP32 microcontroller power supply ● Troubleshooting ● Initial testing ● Project report writing (i.e. introduction, design, appendices, acknowledgment, table of contents, abstract, conclusion, results, and testing) ● Assembling case ● Fall detection aspect of overall program Minor: ● Experimental testing ● Soldering components
Obatosin Obat-Olowu	Major: ● Fall detection aspect of overall program ● Troubleshooting ● Experimental and initial testing ● Design of ESP32 microcontroller power supply ● Project report writing (design, appendices, and testing) ● Soldering components ● Assembling case Minor: ● Web server aspect of overall program ● 3D modelling and design of the fall detector case ● Web page design and creation

Table (7) - Contributions and responsibilities of each group member.

Log Book and Gantt Chart (10.3)

Refer to table (8) below for the project logs formatted as a table.

Date (Day/Month/Year)	Time Spent Working on Project	Details (i.e. accomplishments, events, meetings, etc.)
14/01/2019	20 minutes	-Group met with supervising lab technologist (Daniel Vasiliu) to discuss potential project ideas.
18/01/2019	10 minutes	-Group members agreed on a

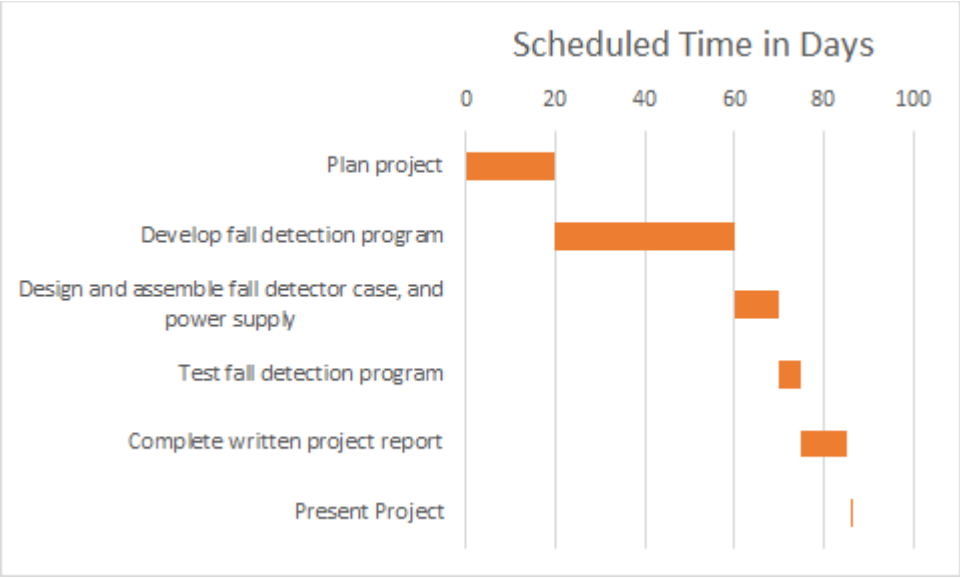
		project idea (fall detector), this was relayed to Mr. Vasiliu via email.
22/01/2019	20 minutes	-Group met with Mr. Vasiliu to discuss specifics of projects (i.e. which components to order/use).
01/02/2019	N/A	-The desired components were decided on and ordered.
05/02/2019	20 minutes	-Group members met up to delegate initial tasks.
11/02/2019	10 minutes	-Components arrived, could now begin programming and running initial tests.
13/02/2019	2 hours	-Group members met up to formulate pseudocode that would help develop the actual program.
15/02/2019	2 hours	-Further planning and research performed on the ESP32 microcontroller and MPU 6050 Arduino module (i.e. libraries, functions, sample programs, etc.).
16/02/2019-24/02/2019	10 hours	-Work done independently on the previously delegated tasks (i.e. web server and fall detection aspects of the overall program).
25/02/2019-05/03/2019	3 hours	-Minimal work was done during this time period due to midterm studying and preparation. Some independent programming was done.
07/03/2019	30 minutes	-Group members met up to discuss the progress of the project and issues with their respective programs.
12/03/2019	3 hours	-Completed soldering lesson/test.
15/03/2019	6 hours	-Web server portion of overall program was completed, tests were run to ensure client requests were effectively received and responded to. -Due to issues connecting the MPU 6050 Arduino module to the ESP32 microcontroller some soldering had

		to be done to the pins to ensure that there was a good connection.
19/03/2019	1 hour	-Group met with Mr. Vasiliu to discuss methods of powering the fall detector, decided on using a 9V battery, stepped down to 5V by a buck converter, the output of the converter would then go to the 5V and GND pins on the ESP32 microcontroller.
21/03/2019	6 hours	-Group members met with Mr. Vasiliu to receive tutorial on FreeCAD 3D modeling. -Afterwards, 3D models of the fall detector case and cover were completed using FreeCAD. -A test was run to ensure that the ESP32 microcontroller could be powered via the 5V and GND pins. -The case and cover of the fall detector were printed overnight.
22/03/2019	3 hours	-The completed fall detector case and cover were picked up, larger holes were drilled into the PCB to allow for the 3mm screws to secure it to the bottom of the case. -The components were loosely arranged in the case to ensure that everything fit as desired. -Changes were made to the fall detection aspect of the overall program. -Colors and helpful information/hyperlinks were added to the fall detector web page (HTML document).
23/03/2019	8 hours	-Organization of written report was decided upon and introduction portion was completed. -More modifications were made to the fall detection aspect of the overall program.
24/03/2019	8 hours	-Design section of written report, as well as various appendices, were completed. -More modifications were made to the fall detection aspect of the overall program.

25/03/2019	5 hours	-Tests were run as to determine a good threshold that would be surpassed in the occurrence of a fall. -The appendices and testing sections of the written project report were worked on.
26/03/2019	4 hours	-Written project report was worked on. -Fall detection algorithm was changed.
27/03/2019	8 hours	-All the components were soldered and connected together, the fall detector was then fully assembled and tests were run to ensure that everything was operating properly. -Made further progress on written report.
28/03/2019	8 hours	-Many experimental tests were run as to find a good threshold at which the fall detector would detect a fall. Unfortunately, the results obtained from these tests were invalid and did not provide useful information regarding the threshold values.
29/03/2019	6 hours	-A new method of experimental testing was devised, the |a| and $\|v_\theta\|$ would be graphed on the serial plotter while each of the tested motions were performed. From this data optimal threshold values for both the |a| and $\|v_\theta\|$ were determined.
30/03/2019	4 hours	-Fall detection program was cleaned up and further progress was made on written report.
01/04/2019	8 hours	-Fall detection program was deemed fully operational and added to written report. -Final tests were performed to ensure accuracy of fall detection. -Written report was completed.
02/04/2019	1 hour	-Written report was read over to check for spelling and/or grammatical errors.

Table (8) - Project logs.

Refer to graph (14) below for a Gantt Chart depicting how the various parts/steps of this project were scheduled. Note that this was the intended scheduling for the various parts of the project, refer to the project logs in table (8) above to see when each step was actually completed.



Graph (14) - Gantt Chart.

Appendix G (11) - Web Page Layout

Refer to figures (10) and (11) below, for images of the web page used to monitor the status of the fall detector.

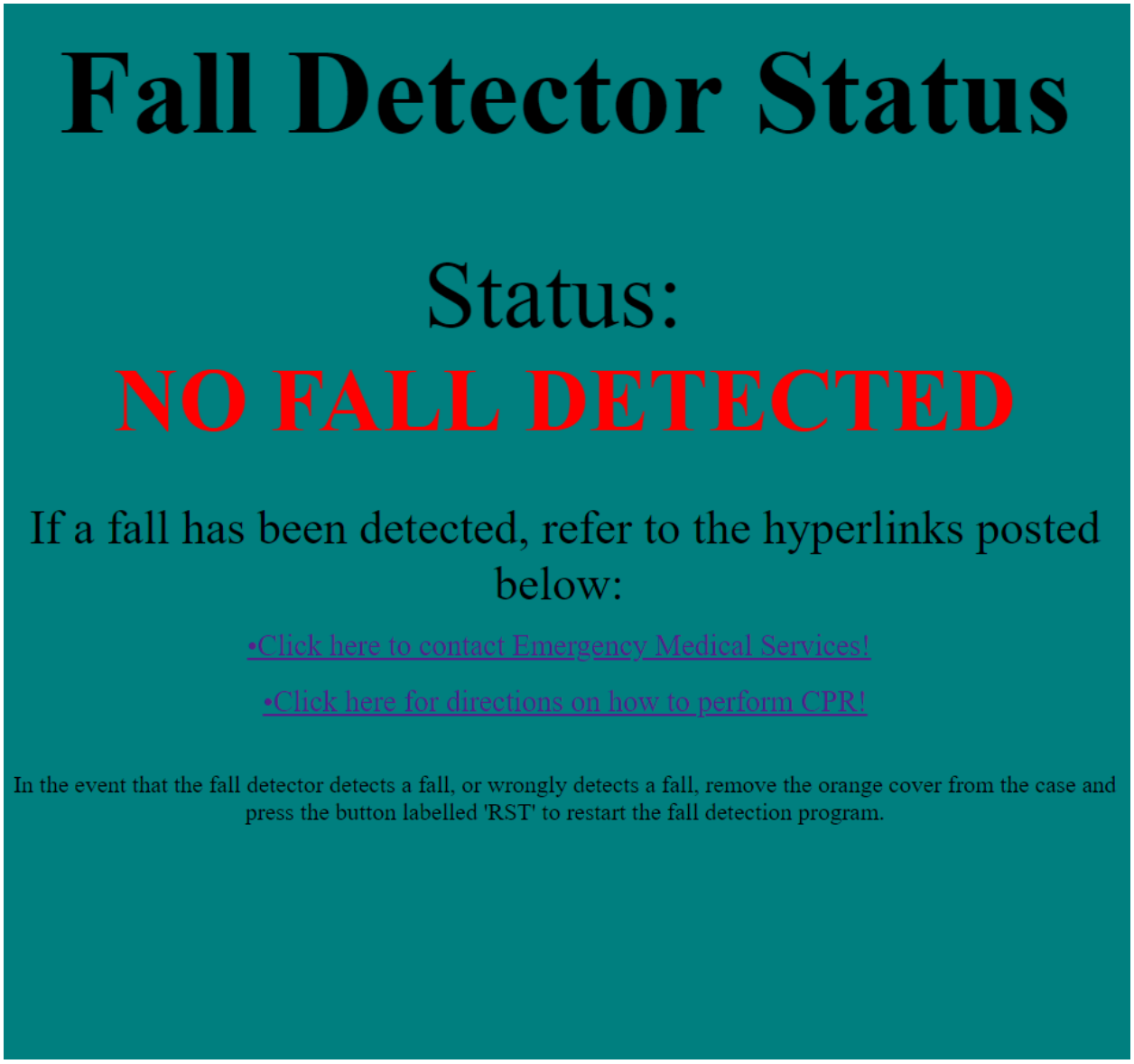


Figure (10) - Web page when no fall has been detected.

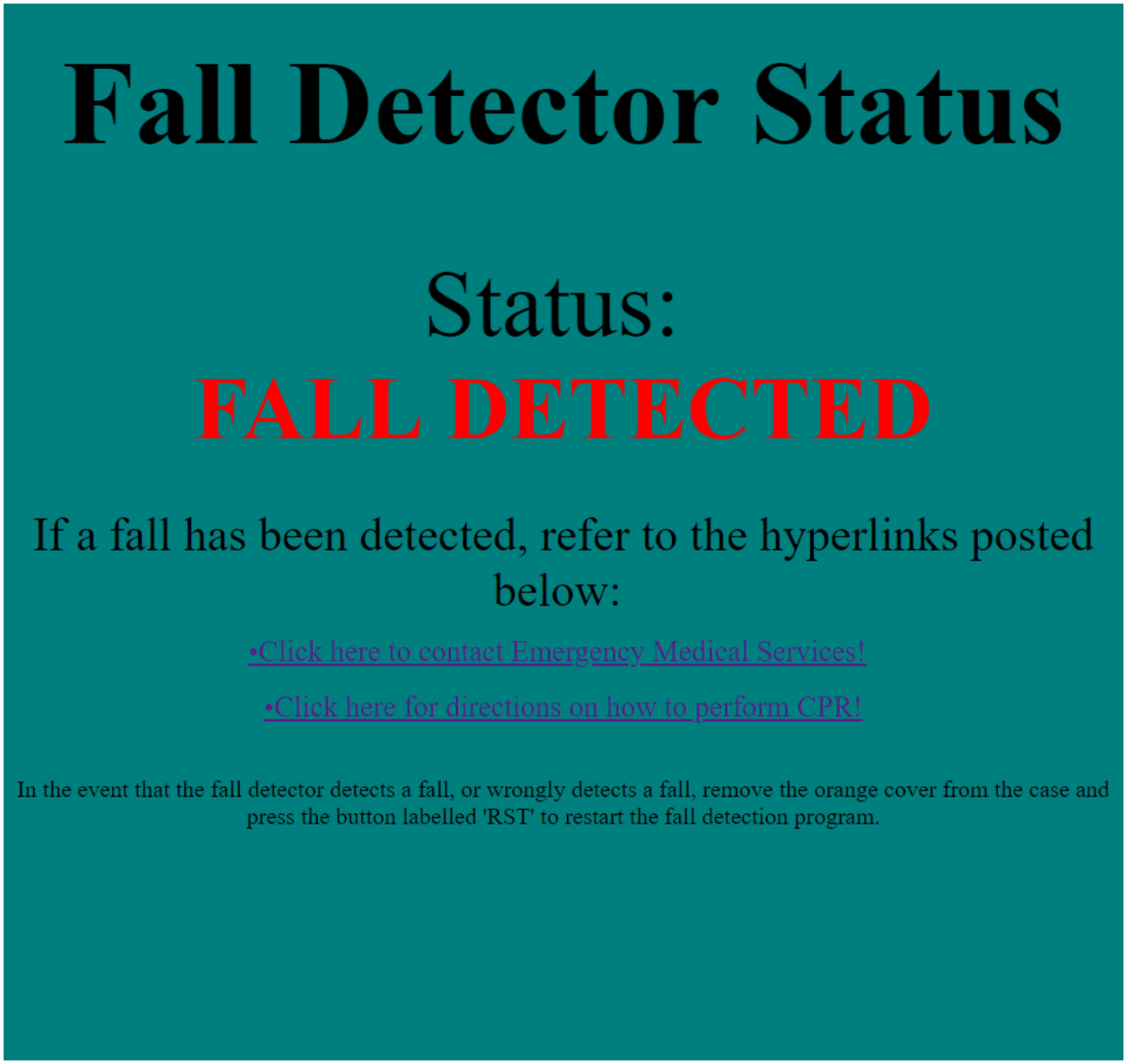


Figure (11) - Web page once a fall has been detected.

Appendix H (12) - References

[1] "An Aging Population." *Statistics Canada: Canada's National Statistical Agency / Statistique Canada : Organisme Statistique National Du Canada*, 7 Oct. 2016, www150.statcan.gc.ca/n1/pub/11-402-x/2010000/chap/pop/pop02-eng.htm.

[2] "Falls Prevention Facts." *NCOA*, 4 June 2018, www.ncoa.org/news/resources-for-reporters/get-the-facts/falls-prevention-facts/.

[3] "Subscribe to TheBreakthrough Newsletter OrE-News." *About OI - Osteogenesis Imperfecta Foundation | OIF.org*, www.oif.org/site/PageNavigator/AOI_Facts.html.

[4] "Hemophilia A." *National Hemophilia Foundation*, 15 July 2015, www.hemophilia.org/Bleeding-Disorders/Types-of-Bleeding-Disorders/Hemophilia-A.

[5] "Polylactic Acid or Polylactide (PLA)." *Bioplastics News*, 15 Sept. 2018, bioplasticsnews.com/polylactic-acid-or-polylactide-pla/.

[6] Rogers, Tony. "Everything You Need To Know About Polylactic Acid (PLA)." *Everything You Need To Know About Polylactic Acid (PLA)*, www.creativemechanisms.com/blog/learn-about-polylactic-acid-pla-prototypes.

[7] "MPU-6000 / MPU-6050 Product Specification Revision 3.4". *InvenSense*, 19 August 2013. https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf

[8] "The Embedded Self-test Feature in MEMS Inertial Sensors". *EDN Network*, 22 July 2012. https://www.edn.com/design/analog/4390843/The-embedded-self-test-feature-in-MEMS-inertial-sensors

[9] "Development of a Wearable-Sensor-Based Fall Detection System". *International Journal of Telemedicine and Applications,* Volume 2015. https://www.hindawi.com/journals/ijta/2015/576364/

[10] "A Threshold-Based Fall-Detection Algorithm using a Bi-axial Gyroscope Sensor". *Medical Engineering and Physics, Volume 30.* January 2008. https://www.sciencedirect.com/science/article/pii/S1350453306002657

[11] "Evaluation of a Threshold-Based Tri-axial Accelerometer Fall Detection Algorithm". *Gait and Posture, Volume 26,* July 2007. https://www.sciencedirect.com/science/article/pii/S0966636206001895

[12] "HTTP." *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/HTTP.

[13] "MPU-6050 | TDK." *InvenSense*, www.invensense.com/products/motion-tracking/6-axis/mpu-6050/.