Head Bangerz: Concussion Sensor

Final Design

Group 2

Greg Defalco, Sofia Main, Mackenzi NierDuffy, Brennah Satterfield, and Lauren Stark

Prof. Mike Schafer

EE Senior Design

Spring 2023

Table of Contents

1 Introduction

Concussions seriously affect athletes with injuries lasting a lifetime. Despite these long-lasting injuries, concussions still occur frequently. In one year, 1.7 to 3.0 million concussions are recorded in all levels of sports. Most of these concussions occur in sports with athletes wearing helmets, such as football, hockey, and lacrosse. The majority of the athletes that play these sports are students ranging from youth ages 5 to 6 all the way to college students. The human brain does not fully develop until the age of twenty-five, which is years after even the highest-level players end their careers. As a result, student-athletes are more prone to concussions with traumatic brain injuries (TBI). Because coaches, trainers, and doctors do not have numerical data to diagnose a concussion when it happens, serious injuries are often overlooked, allowing athletes to continue to play. An estimated 5 in 10 concussions go unreported due to funds, ignorance of TBIs, and lack of access to medical care. However, this can happen at any level of the sport due to a lack of immediate data on hits that occur during games. For example, Tua Tagovailoa, a professional football player, suffered two serious head injuries in a short time frame. The public saw Tagovailoa possibly seriously injure himself in a game after a hit where he was stumbling off the field and could barely stand. Yet, a week later, Tagovailoa was cleared to play again and was hit in the head again. Instead of stumbling, he started clenching his hands and seizing on the field in response to most likely a brain stem injury that could have been prevented with definitive data on his head injury status the first time. Tagovailoa is one example of the serious injuries that can occur from the lack of data and care of concussions, even when the care, money, and funds are present, such as at a professional football game.

A major issue in the reporting of concussions is that it largely depends on self-reporting from the athlete in question. Often, an athlete will not report symptoms to stay in the game or due to them not understanding the symptoms they are experiencing. When not reported, a concussion will not get the treatment it needs, and it is possible that another concussion could occur. This is known as Second Impact Syndrome and can lead to permanent brain damage or even death. Also, even multiple concussions, even when given time to heal, can lead to long-term health complications like chronic traumatic encephalopathy, otherwise known as CTE–a degenerative brain disease that can cause memory loss, depression, anxiety, and even death. Currently, concussion sensing systems often cost thousands of dollars. This makes these products feasible for college and professional football teams with large budgets but not for youth teams where many of these injuries still occur.

Although it takes serious medical devices to properly diagnose a concussion, such as a PET or a CT scan, detecting hard hits can predict the possibility of serious head injuries extremely well. Concussion tolerance for youth athletes occurred in a head acceleration of 62.5 ± 29.7 G force and rotational head acceleration of 2609 ± 1591 rad/s^2. Adult athletes' concussion tolerance is about double these numbers.[1] The study reports that they could accurately predict concussion occurrences in youth ages 9 to 14 with up to 95% accuracy. Therefore, we are basing our predictions of concussion occurrence on their data.

[1] Campolettano, E.T., Gellner, R.A., Smith, E.P. *et al.* Development of a Concussion Risk Function for a Youth Population Using Head Linear and Rotational Acceleration. *Ann Biomed Eng* 48, 92–103 (2020). https://doi.org/10.1007/s10439-019-02382-2

To combat the serious epidemic of traumatic brain injuries in contact sports, we have designed and built a concussion sensor that comfortably fits inside athletic helmets. The system revolves around a central wireless communication device, microcontroller, accelerometer, and gyroscope. Each device costs around $50 to manufacture in order to reasonably meet the cost requirements of low-budget teams.

There are two main ways to receive a concussion, fast head movements that shake the brain or direct impacts to the head that bruise the brain. The system has both rotational and acceleration sensors. The board is placed at the top of the helmet such that the gyroscope has one plane parallel to the player’s body while standing straight, one plane is parallel to the floor and the shoulders, and the third plane is parallel to the floor and the player's nose. In the event of a rotational impact, the gyroscope reports the angular acceleration of the player’s head and the change in XYZ coordinates of the top of the head. The accelerometer on the board is attached to the helmet such that one of the mounting pins touches the helmet's surface. The pin reacts to the shock of the impact felt by the helmet's surface and reports the impact’s g force. If any significant changes in linear or rotational acceleration occur, an interrupt trigger sends the data via ESP-NOW to a WiFi hub that hosts a website. The website displays the most up-to-date information about the severity level of the impact, the numerical values of the data, and the player number.

Although the system's performance worked as expected, a few details were unanticipated. For example, the gyroscope was more sensitive than the accelerometer. Smaller rotational events triggered the gyroscope as well as linear acceleration events. This resulted in the gyroscope almost always reporting data. Also, the accelerometer was less sensitive than expected, and the amount of force required to trigger the accelerometer only allowed for medium to large impact events to send data. Sometimes, the accelerometer was not triggered at all during forceful impacts. This might be a problem in gameplay because not every movement of the head or hit is significant, nor is every small impact insignificant. Overall, the system needs improvement and has small quirks that come from specificities but works well enough to suggest it has the potential to work flawlessly.

2 Detailed System Requirements

When we were designing our concussion sensor, we developed *5 main requirements for the system:*

Detecting when a helmet/head has been hit:

The system contains two main sensors, the BNO055 gyroscope and ADXL375BCCZ-RL high-G sensor. Each sensor reports information critical to understanding the nature of the head movement or impact. The gyroscope detects rotational impact for whiplash and other fast head movements. The high-G sensor detects impacts and head jerks that could potentially harm the brain. The sensors take 30 samples of the triggering event detected by one of the sensors or both.

Receiving live data from the sensors:

This is the most important part of the system. If no data is received from the sensors, then there is no chance of detecting a possible concussion. A WiFi hub that hosts the website receives live data from the sensors through ESP-NOW. ESP-NOW is a wireless fast communications protocol that sends small data packages such as ones triggered by the sensors.

Differentiating between a regular hit and a concussion hit:

This is the secondary function of the application. Through programming and the concussion-possibility threshold, an alert will be created when the change of position of a head hits an extreme or is moved in a way that has been known to cause concussions.

Differentiating between players:

Each board corresponds with a helmet and a player. In a game, it is critical that the player be identified such that if the coach does not see the hit, they can still tell by the data who got hit and how hard the hit was. Using ESP-NOW, we can detect which ESP is transmitting the data corresponding to the triggered event.

Alerting when a concussion hit has occurred:

The alert is the third most important function of the application. The alert is the easiest and fastest method to notify the user when a potential concussion hit has occurred. The website conveys this by a measure of low, medium, and high severity of the head’s impact or movement, displayed at the top of the page. This would allow the user to pull the athlete out immediately and check up on them.

3 Detailed project Description

3.1 System Theory of Operation

The overall system consists of nodes inside the helmets and a WiFi hub. The system was made to look simple from an outside perspective, but it takes six subsystems for the comprehensive system to work: linear acceleration, angular acceleration, power delivery, communication, website, and casing. The linear and angular acceleration subsystem encompasses the board design and fabrication, including sensors. The power delivery subsection includes the rechargeable batteries. The communication subsystem establishes the connection between the boards and hosts the website to display the sensor data. Lastly, the casing subsystem provides protective cases for the boards while considering the players' comfort and ease of use for coaches.

Each helmet contains a board and rechargeable battery to run the board which is referred to as a node. The board and the battery are placed in separate 3D-printed cases inside the helmet due to the fragility of the board's components. Each board contains an accelerometer and a gyroscope. These allow for the level of rotational and linear acceleration to be sensed by the board. The accelerometer and the gyroscope also have predetermined thresholds set to the minimum level of acceleration associated with possibly causing concussions in youth athletes in helmet sports. When the accelerometer or gyroscope senses a hit or acceleration surpassing these limits, an interrupt is triggered, and data points from the moment of impact and immediately surrounding the impact are sent via ESP-NOW to the website. The website then is updated with the peak value from the set of data points sent, which will theoretically represent the most severe impact that triggered the warning. The numerical value of the impact is then updated on the website following the impact. On the website, a table showcases the most recent contact and the previous four. The table contains information about the player, the severity of the impact, and the numerical value of the force and gyroscope sensor. This serves as an official warning to remove the hit player from play as a result of the hit.

3.2 System Block Diagram

Figure 1 below shows the complete system diagram. The left side of the figure details the linear acceleration, angular acceleration, and power delivery subsystem. This subsystem is located within the helmet on the right side of the figure. The right side of the figure details the communication and website subsystem. The linear and angular acceleration subsystems are located within the helmets, communicating to the WiFi Hub through ESP-NOW. The WiFi Hub receives these communication signals and hosts the website where the data is shown.

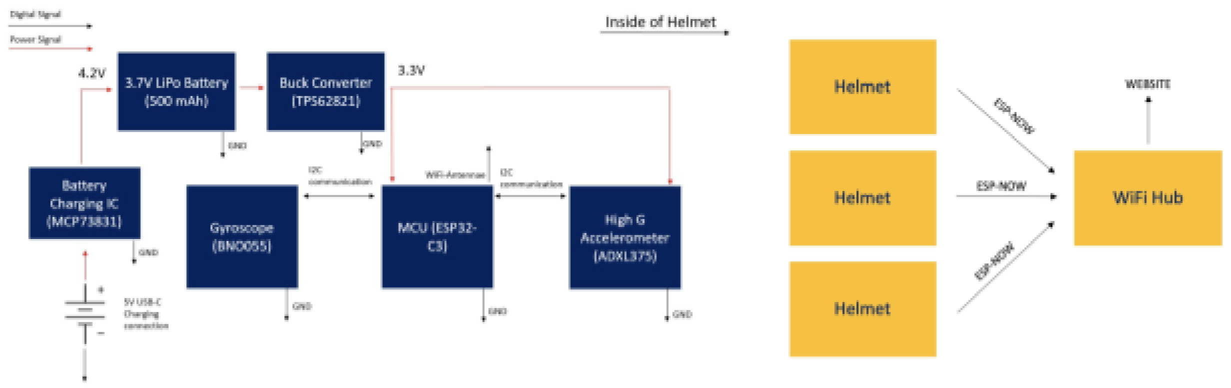


Figure 1. Complete System Diagram

3.3 Linear Acceleration Subsystem

The linear acceleration sensing subsystem has the following requirements:

- Sense linear acceleration in all directions (x y and z) up to 90 Gs of force.
- Interface with the ESP32, and send linear acceleration data when a high G event occurs.
- Withstand shock events of 90 Gs or greater without damage to the sensor.

The sensor which best fit these requirements was the ADXL375 Digital MEMS Accelerometer from Analog Devices. This device can sense linear acceleration up to 200 Gs on all 3 axes (x y and z). It also has options for both a SPI connection or an I2C connection. The device is also rated to survive shock events up to 10000 Gs. The schematic of the subsystem is shown in Figure 2.

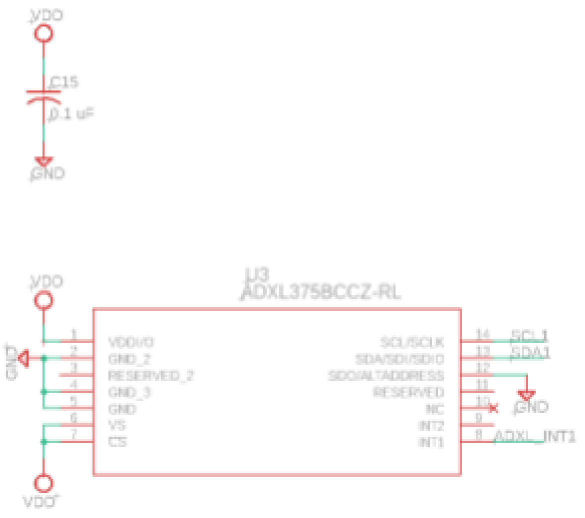


Figure 2. Linear Acceleration Subsystem Schematic f

The subsystem is powered from a 3.3 V main power rail. To interface with the ESP32, an I2C connection was used. This decision was made to fit the overall system requirements. Due to the small size of the board (40 x 40.5 mm) it was important to minimize the number of signals on the board. Therefore, a 2-wire I2C connection was preferable to a 4-wire SPI connection. The 100 kHz data transmission speed over I2C was more than sufficient for our device.

The subsystem is also interrupt-driven: this is preferable because it both ensures that all high-G impact events are captured and allows the overall device to save power by minimizing data transmission. The ADXL_INT1 signal is tied directly to IO6 on the ESP32, which has an internal pull-up resistor that also helps minimize the board area. The ADXL375 has two interrupt options available: SINGLE_SHOCK and DOUBLE_SHOCK. Figure 3 shows an interrupt diagram for both options.

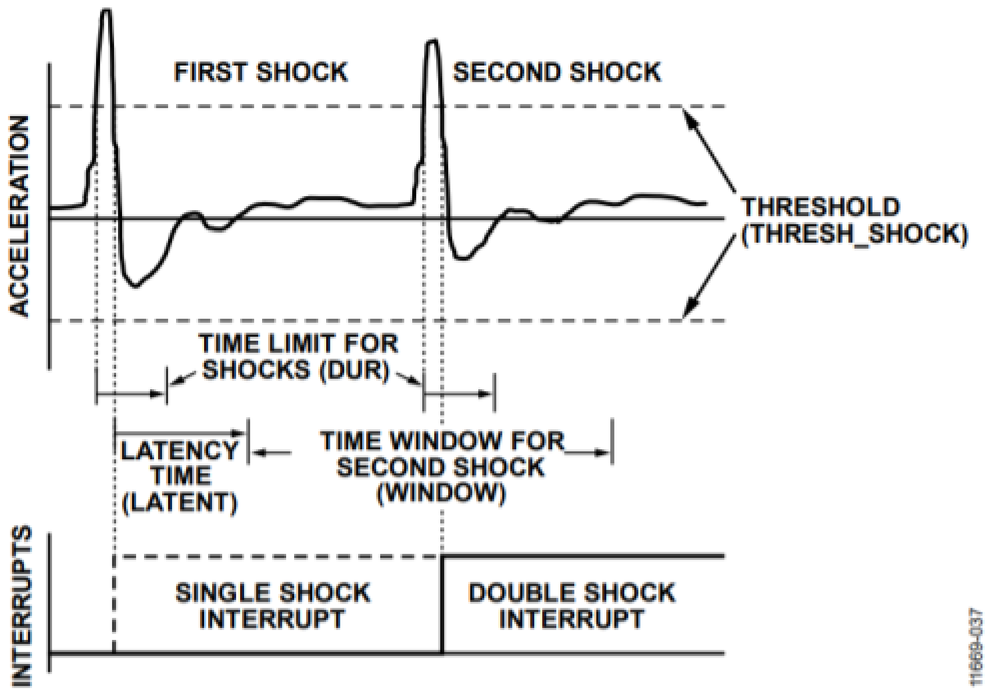


Figure 3. Shock Interrupt Function with Valid Single and Double Shocks

The SINGLE_SHOCK option has two parameters: threshold and time limit for shocks. The threshold is the acceleration value (Gs) which if exceeded will trigger the interrupt. The time limit is the maximum time the acceleration has, once above the threshold, to return under the threshold value. When an acceleration value exceeds the threshold, and then returns under the threshold within the time limit, an interrupt is triggered. The double shock interrupt allows for the monitoring of multiple shocks in a row. However, it does not fit the needs of our project, which is more focused on detecting a singular high-G event. Therefore, the single shock interrupt is used with a threshold value of 30 Gs and is set to 50 μs. This device also has a FIFO memory buffer on-chip which allows for the capture and storing of acceleration data in the case of an interrupt being triggered. The amount of samples taken and stored in this buffer can be varied up to 32 data points for the x, y, and z directions. For this system, we chose to save and recover 10 linear acceleration samples for each direction. When the interrupt occurs, the ESP32 retrieves the buffer data with I2C, and then resets the interrupt.

3.4 Angular Acceleration Subsystem

The angular acceleration subsystem has the following requirements:

- Sense angular acceleration in all directions (x y and z) up to 1500 rad/s^2
- Interface with the ESP32 and send angular acceleration data when a high angular acceleration event occurs
- Withstand up to 90 Gs of shock without damage to the sensor

The sensor which best fit these requirements was the BNO055 from Bosch Sensortec. It is a 3-in-1 sensor device with an integrated accelerometer, gyroscope, and magnetometer. The accelerometer only has a range of 16 Gs, so it was not useful for this project. The magnetometer is also left unused. The gyroscope has a range of up to 2000 °/s of angular velocity, which roughly corresponds to 35 rad/s. It also has an I2C interface which allows for simple communication with the ESP32. Also, it survived testing with 90G impacts and continued to function as expected. The schematic of the subsystem is shown in Figure 4.

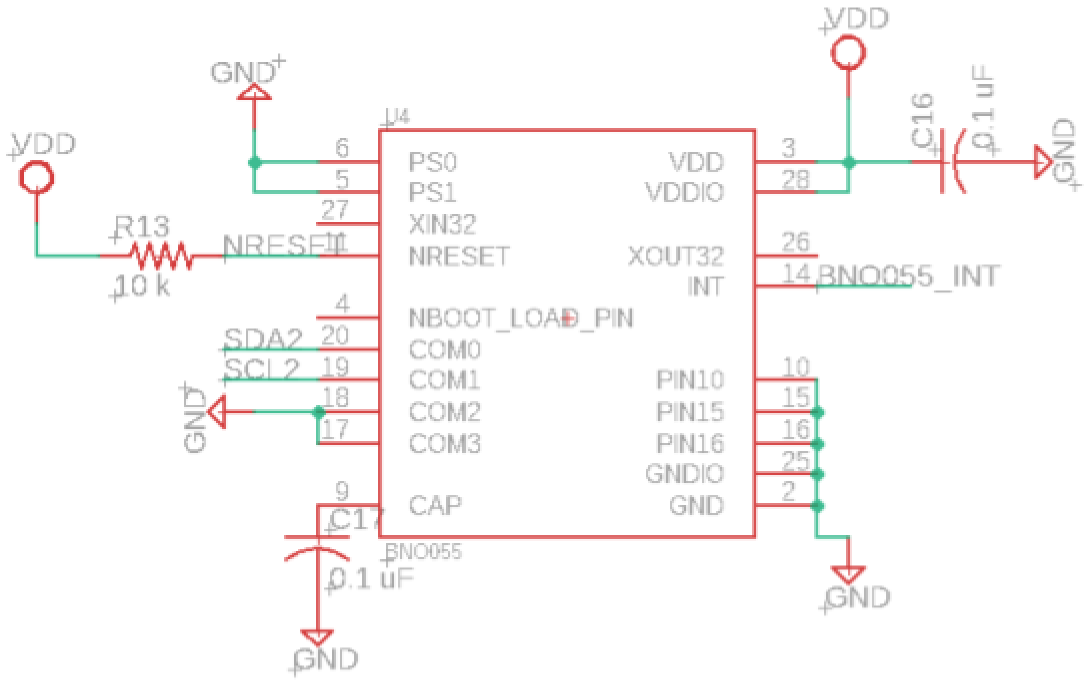


Figure 4. Angular Acceleration Subsystem Schematic

The subsystem is powered by a 3.3 V rail, filtered by a 0.1 uF capacitor. The device is configured to I2C mode by grounding the COM2 and COM3 lines. It is notable that the I2C bus used here (SDA2 and SCL2) is separate from the I2C bus used by the ADXL375 (SDA1 and SCL1). The reason for this is that during testing with the ADXL375 and BNO055 on the same I2C bus, it was found that the BNO055 stretched out the I2C clock signal enough to cause a communication timeout. In order to circumvent this issue, an additional I2C bus was added to service only the BNO055.

The BNO055 is interrupt triggered, and is configured for what is called a Gyroscope Any Motion Interrupt. This interrupt is triggered when the slope of successive angular velocity measurements (so effectively the angular acceleration) exceeds a certain hard-coded threshold in the device's registers for a set amount of samples. An interrupt diagram can be found in Figure 5.

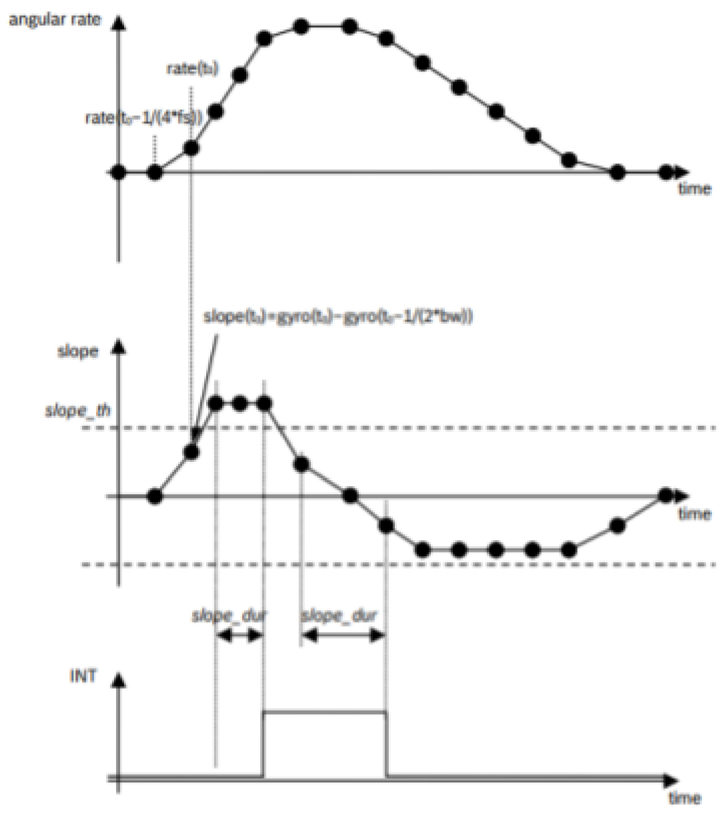


Figure 5. Principle of Any-Motion Interrupt Detection

The registers are set to make the interrupt slope threshold 1000 rad/s^2, and trigger when 8 samples are detected above said threshold at a 400 Hz rate. When the interrupt is triggered, the ESP32 uses the I2C connection to read 10 samples of Gyroscope data for each axis (x, y, and z). The interrupt is then reset.

3.5 *Power Delivery Subsystem*

The power delivery subsystem has the following requirements:

- Provide a constant 3.3V to the ESP32, ADXL375, and BNO055
- Be battery driven so that the device does not need to be plugged in
- Last approximately 3 hours, the upper bound for real time of a football or hockey game
- Be conveniently rechargeable

These requirements were met with a combination of a USB-C charging port, a simple battery charging IC (MCP73831), a 3.7 V 500 mAh Lithium Polymer battery, and a variable Buck converter stepping that voltage down to 3.3 V (TPS62821). Figure 6 is the schematic of the subsystem.

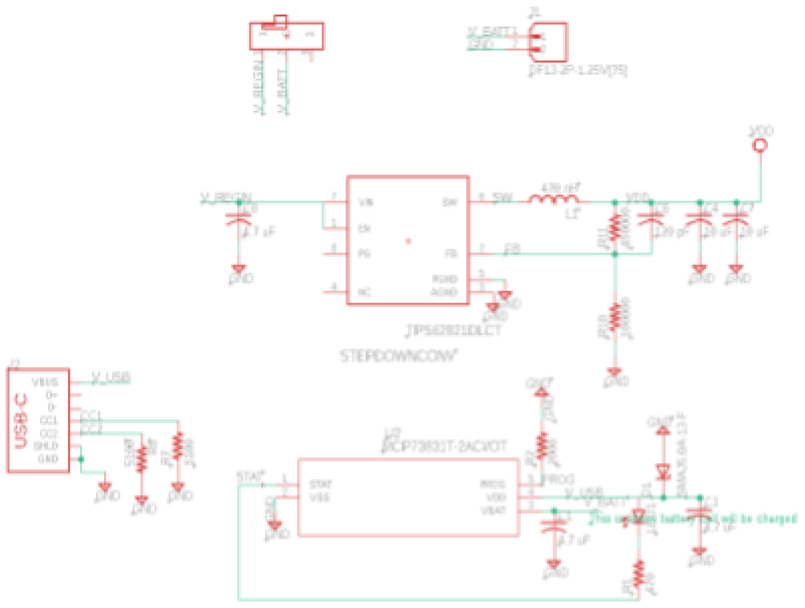


Figure 6. Power Delivery Subsystem Schematic

The USB-C was chosen simply to recharge the battery, and not actually transmit any data. USB-C was chosen because it is new and widely available, but an older and simpler connector like USB-B micro would have possibly been easier to solder and implement. A Zener diode (SMAJ5) was tied from the USB-C power line to ground in order to protect from surge events from the USB-C connection damaging other parts on the board.

The USB-C delivers 5V to the MCP73831 single cell battery charger. This device charges at a constant rate of 500 mA/hour and steps down the voltage to 4.2 V, which is preferable for the 3.7 V battery. A constant 500 mA/hour charging rate is not optimum for long term battery health. However, for the first iteration of this project, it behaved well over about a month of testing. In the future, a slower variable charging rate would be preferred.

The 500 mAh 3.7 V Lithium Polymer battery is connected to the board with a simple JST connector. Before selection, the cumulative current consumption of the board was calculated, with the assumption that the ESP32 would be in power-saving sleep mode for the majority of the operation due to the low frequency of threshold-exceeding hits and the completely interrupt controlled nature of the software. Because of this, 500 mAh was deemed more than enough to operate the device for our requirement of around 3 hours, and this was later confirmed during board bring-up and testing.

The battery voltage of 3.7 V is then fed to the variable buck converter TPS62821. Through the selection of feedback resistors of 100 Kohm and 450 Kohm, an output voltage of 3.3 V was attained, and acted as the main power rail for the ESP32 and sensors on the board. The reason a buck converter was used instead of a simple LDO was to further save power. This specific buck converter also came in an incredibly small package (2 mm x 1.5 mm) which was helpful to further minimize our board size.

3.6 Communication Subsystem

The communication subsystem has the following requirements:

- Send accurate data from the helmet to the WiFi hub in real-time
- Support more than one helmet's communication
- Communicate over a large area, such as a football field's length

The concussion sensor design prioritized ease-of-use, especially when it comes to data. Ideally, when the sensor is being used, the coach would be able to access a website that displays the data and alerts when a player has been hit. The linear acceleration system collects the data, while the communication system transfers the data. There are two main components–all based on ESP32-C3s–and one type of communication. The two components are called the node and WiFi Hub. ESP-NOW is used to transfer the data between these components since it is a form of direct and low-power wireless communication.

The node is a term used for the helmet that contains the linear acceleration system. This means the node is continuously monitoring the gyroscopic and acceleration data as a player competes. Although the node is constantly checking data, it is only storing and transmitting the data when a potential concussion occurs. The saved data–an integer for the player number, a vector for the force data, and a vector for the gyro data–is being sent to the WiFi hub through ESP-NOW. ESP-NOW was chosen because it can handle up to 20 devices within a 220-meter radius. Typically, only eleven players of the same team are allowed to play on the field, and the average football field is 110 meters. These numbers are similar throughout most sports.

Although the node gathers the data, the WiFi hub manipulates and displays the data on the website. Before doing so, the WiFi hub collects the data the node is sending. The node sends one number and two arrays. The number is associated with the player, meaning player one's number should be one. The two arrays are force and gyroscopic (gyro for short), where the measurements are gravitational force and velocity, respectively. The number and measurements are populated in the node when a concussion hit occurs. Once the WiFi hub has the values, it finds the absolute maximum value of both force and gyro. These values determine the severity of the concussion. If the force sensor triggered the concussion, then the severity is high if it's above 65g, low below 45g, and medium in between. In a similar fashion, if the gyro sensor triggered the concussion, then the severity is high if it's above 42 rad/s, low if it's below 26 rad/s, and medium if it's in between. These numbers are based on current research in the field. In the end, the WiFi hub has the player number, max value of force and gyro data, and severity of the concussion.

3.7 Website Subsystem

The website subsystem has the following requirements:

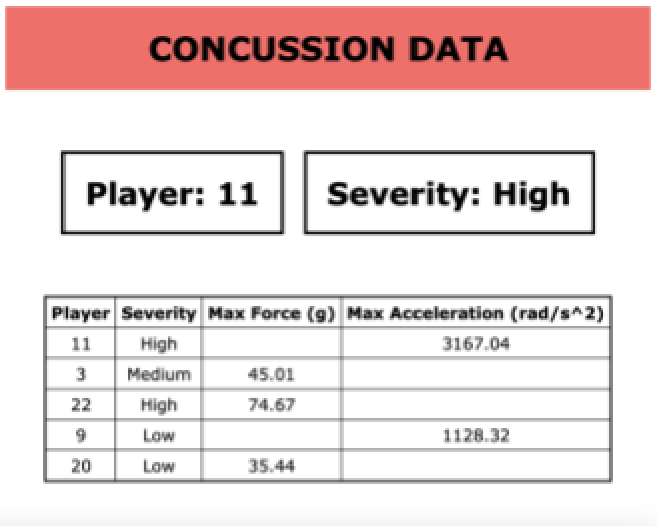
- Accessible to any coach or player
- Showcases the player number, severity, and numerical values of impacts

- Updates the data displayed on the website live

On top of manipulating data, the WiFi hub hosts the website in order to display data. The ESP32-C3 has the built-in capability to host a web server. However, since the main communication between devices is ESP-NOW, the WiFi hub can only host a web server through AsyncWebServer, not WebServer. The website can't be hosted by WebServer because ESP-NOW and WebServer use similar interfaces. This means that both operations would be calling the same interface, causing multiple crashes or data blockage. On the other hand, AsyncWebServer and ESP-NOW do not use the same interface. In fact, AsyncWebServer is built on a completely different networking stack than ESP-NOW. This means that both AsyncWebServer and ESP-NOW can run at the same time. Using AsyncWebServer rather than WebServer does not limit the website. The main difference between WebServer and AsyncWebServer is that AsyncWebServer handles requests asynchronously, meaning it can handle multiple requests simultaneously, while WebServer cannot. This allows the website to be accessed by multiple people, as long as they have the IP address, without crashing.

The website uses JSON and HTML. JSON stands for JavaScript Object Notation and is used to transfer data between web servers and web applications. In particular, JSON is used to populate the number, max value of force and gyro data, and severity of the concussion on the website. Although JSON puts the data on the website, HTML is what formats the website. HTML stands for Hypertext Markup Language and defines the structure and content of the webpage. In other words, the HTML is what makes the website cohesive. Through HTML, the website has a header that says "Concussion

Data" and a data table. The data table is filled with the most recent concussion data through JSON commands. The website is shown in Figure 7.


CONCUSSION DATA

Player: 11 Severity: High

Player	Severity	Max Force (g)	Max Acceleration (rad/s^2)
11	High		3167.04
3	Medium	45.01	
22	High	74.67	
9	Low		1128.32
20	Low	35.44	

Figure 7. Concussion Data Website

3.8 Casing Subsystem

The casing subsystem has the following requirements:

- Protect the node and battery from impacts
- Minimize the weight added to the helmet from the node and battery
- Protect the WiFi hub from the elements

The design subsystem refers to the casing protecting the node and the website subsystem. Both cases are made using a 3D Printer and ABS-M30 material. The material is a high-strength thermoplastic that produces sturdy and lightweight parts. This is an ideal material for the subsystem casing since it is strong enough to withstand intense hits and light enough to not change the helmet's weight.

The node’s casing holds the circuit board and attaches to the inside of the helmet. The circuit board is held to the casing through a snap-in design. If the snap-in were to fail, the board can also be held in using tape. The material holding the board does matter as some tapes, glue, or other sticky materials can potentially conduct and cause the circuit board to fail. On the other side, the casing is attached to the helmet using velcro. This allows the user to pull the case in and out easily to charge the battery. The node’s casing does not hold the battery due to the placement of the node on the helmet. The battery is located in a slightly higher location than the node to prevent any discomfort for the player and is tucked under helmet padding in a similar way that the board is. Thus, a separate casing was designed to protect the battery. Figure 8 shows the design of the node and battery casing.

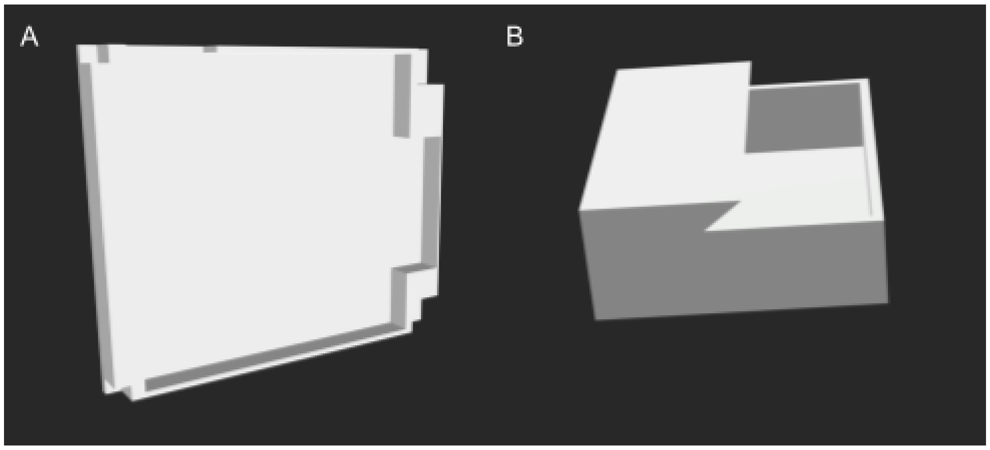


Figure 8. A) Board Casing B) Battery Casing

The protective casing for the website subsystem is much simpler than the node’s casing. The website subsystem should be plugged into a computer at all times. Considering the subsystem is attached to the computer, the casing just has to protect the subsystem from the elements, rather than potential concussion hits. Figure 9 shows the casing for the website subsystem.

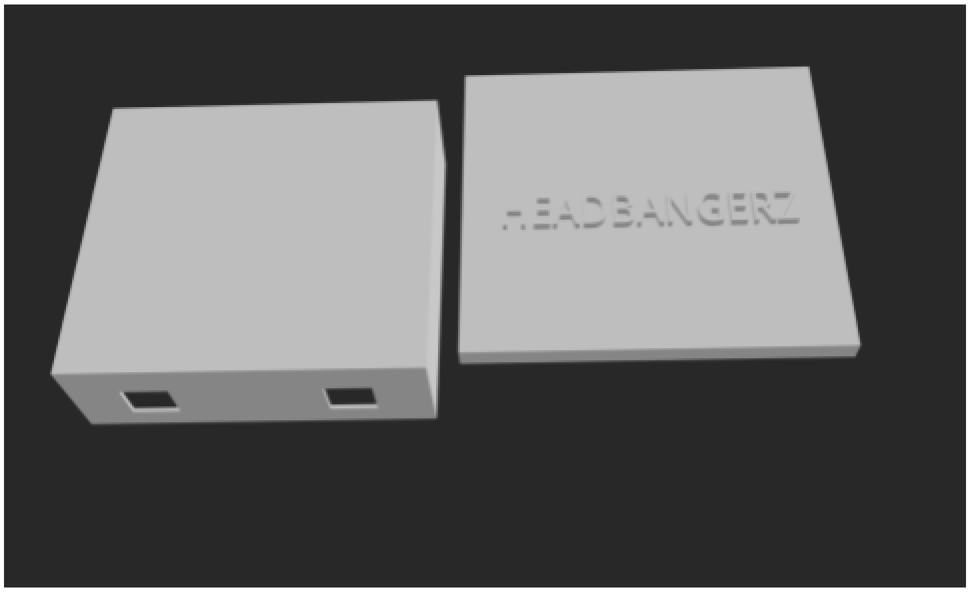


Figure 9. Website Hub Casing

3.9 Interfaces

The casing fits inside of the helmet snugly so that it does not move around a lot. One issue we ran into was that the board was affected by the tape which held it to the casing. Also, there were few places inside the helmet that were small enough for the battery with the casing to fit without bothering the comfort of the athlete, such as under some of the padding between shock absorbers. When the actual placing of the parts inside the helmet came to fruition, the board was placed where we initially intended to at the top of the helmet at the crown of the head, but the battery struggled to find a home for itself, as seen in Figure 10.

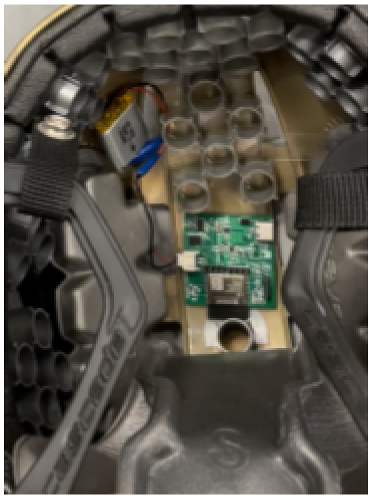


Figure 10. Node Placed Inside the Helmet

The casings were held in place by velcro and tape that ensured their stability inside the helmet during testing. The padding of the helmet also secured the position of the board and the battery.

4 System Integration Testing

4.1 Subsystem Testing

Preliminary testing of the integrated subsystems was done by placing the board and accelerometers in the helmet. The website was then pulled up and the helmet was shaken and moved vigorously in order to simulate a concussion hit passing the threshold. While this was done, the ESP32 was run off of a computer and the XYZ acceleration was displayed on the serial monitor while the code was running. When the acceleration on the serial monitor was seen to surpass the predetermined thresholds, the website successfully updated the data table, reflecting the occurrence of a possible concussion-causing hit. In the preliminary testing, the board was attached to the helmet simply with tape since a kitboard was used. Most preliminary testing did not involve the helmet, and instead was done by simply moving the accelerometers to see the response on the serial monitor and the website.

While preliminary testing was done with ESP32 kitboards, final testing was done using the fabricated PCB with a rechargeable battery. The board was attached to the helmet by the casing and velcro. The helmet was then moved around vigorously and hit to see if thresholds were reached and if the data updated on the website following the hit.

4.2 Design Requirements Testing

Each design requirement was at first individually tested as subsystems, and then tested as the system altogether. All of the testing can be done through the website monitor which confirms if it is running all of our design requirements properly. The first design requirement tested the sensor's ability to read in data by banging the helmet, which sends messages to the serial monitor on the computer running the program or the website, which displays the numbers on the table. The sensors passed by sending the correct estimated size of the packages and numbers associated with the impact, as well as the rotational data.

The second design requirement testing sees if the sensors are talking to the WiFi hub. This can also be checked using the first testing. If the signal goes through and is read on the computer hosting the website and connected to the WiFi hub, then the data is being properly sent and received. To see if the data continuously comes in live, bang the helmet multiple times and make sure multiple packages come through. To test if the device is differentiating between regular and concussive hits, look in the website screen, which displays the severity level and which player is receiving the hit.

All testing can be done without the website, but the website was made for ease of reading the data and alerting when a concussive event occurs. Thus, the most efficient way to check the subsystems is through the website interface.

5 Users Manual/Installation manual

5.1 How to install your product

The product is installed by putting the board in the helmet. The protected board is put in the board holder that is covered and made safe for the player, and is then placed inside the helmet under the padding. The battery is detached and charged when need be. When the battery is connected to the board, it turns the board on inside the helmet, and the product is ready to be used, assuming the website is open. It should now be ready so that it will update the website if a hit surpasses either the threshold set on the accelerometer or the threshold set on the gyroscope.

5.2 How to set up your product

The product is set up by opening the website and attaching the charged battery to the board already in the helmet. It should then be ready for the player to put on and play, assuming the website is open with someone watching it. When a hit surpasses either threshold, the website will be updated and alerted so that whoever is watching the website can pull the player out of the game.

5.3 How the user can tell if the product is working

The user can tell if the product is working by seeing the feedback on the website. When a concussion hit is dealt to the helmet and felt by the accelerometer or gyroscope, it is reported to the website where the linear acceleration and rotational velocity are displayed, along with the severity of the hit. The website is only updated when a hit is dealt that is at or above the predetermined threshold that characterizes a hit that could possibly cause a concussion. The user could test this themselves by vigorously shaking or twisting the helmet quickly and seeing the data relayed on the website.

5.4 How the user can troubleshoot the product

If the user is having issues with the website, they should make sure that they are properly connected to the WiFi and refresh the website. If the website is not working no matter what, they should run the program on PlatformIO and read the serial monitor output IP address and copy that into the search bar. In further iterations of this project, the website would not exist, but rather an app that sends push notifications, so this step would be obsolete.

Finally, if the board and website are not working, the user could remove the battery and check to make sure it is fully charged. If the battery is not working at all, it should be replaced with a new battery that delivers the expected 500mAh 3.7V that is required to run this device.

If the problem persists and is larger than replacing the battery and refreshing the WiFi, then the user should bring the device to us to fix. The user should not be able to further troubleshoot the device unless they have knowledge on how to code the backend or how to solder a circuit board.

6 To-Market Design Changes

While this design is a workable solution and prototype, there are several additional logistical challenges that arise when considering taking this product to market. First of all, this prototype was made to work with a specific type of helmet made by Cascade with removable padding and shock absorbers. Depending on the level of the helmet, the padding is different, and a lot of helmets do not have removable padding. Therefore, the board and sensors could not be placed in the same configuration inside of the helmet as in the prototype. In order to make this a feasible solution for multiple helmets, it would be necessary to find a place to put the boards and sensors that could attach to any type of helmet while still collecting accurate data. In order to ensure this, more testing of different placements would need to be done, as well as more extensive testing with different helmets. There would also need to be a sort of adjustable attachment to fit different sizes of helmets.

In addition to the logistical challenges posed by different types of helmets, it would be necessary to get approval from sports leagues and leadership that the product can be used in games and is safe for surrounding players. This would likely take a lot of time and testing by other authorities to get the approval. It would also likely require a few redesigns to ensure safety for those around as well as the player wearing the helmet in that the board must not in any way damage the protective integrity of the helmet.

The prototype would also need to be made more user-friendly in order to be taken to market. Since a website is more difficult to keep continuously running on a phone and computers are not portable, it would be more helpful if the concussion data was transmitted to an app that would give real-time updates and notifications to the coach or trainer. A phone application or a computer application would make it more accessible to more people who don't have a robust understanding of the system. It would be beneficial to possibly add more feedback details on the website, such as a graph of an impact in order to show the event's duration or the events that occurred over the course of the game.

Furthermore, if it were to transmit to a phone or a computer, a Bluetooth connection to an app would be ideal so that WiFi instability or location does not affect the node's signals. Also, since the people using the product do not have the same electrical engineering expertise as the designers, there would likely need to be a way to turn the board on and off without removing it from its casing. So, adding a switch would be helpful. It would therefore also make sense to find a way to detach the product from the helmet and charge the battery with the casing and the board without removing the entire system. This could be done by inductive charging and making a matching inductive charger to charge the detachable casings that hold the board. This would avoid more possible problems such as the board getting damaged by the elements or parts coming loose or getting damaged as the board is moved around. Therefore, this would make it safer, more accessible, and more durable for the everyday person to use.

7 Conclusions

5 out of every 10 concussions go unreported in the United States [2] and although most concussions are mild traumatic brain injuries, repeated concussive injuries come with life-threatening repercussions and sometimes permanent brain damage. This is especially concerning considering most athletes who play contact sports will receive more than one concussion in their time playing their sport. If this project can detect the severity of a concussive impact and report the data to a coach, trainer, or doctor, then we can reduce the number of unreported and untreated concussions in sports. Although we cannot stop people from getting concussions in sports, we can help relay the right data to allow them to make informed decisions about the state of their brain and the severity of hits received.

As a whole, this project functions as an effective prototype to detect hits of certain levels associated with causing concussions and send an alert when this occurs. The sensor detects movement and impact such that all events surpassing a threshold are recorded and sent to the website for display. In addition, it is possible to recognize which player receives the impact based on which board felt the acceleration past the threshold. The data and severity levels of the impacts is clearly displayed on the website, hosted by the WiFi hub. The board fits inside of the helmet properly without discomfort and has yet to fall out of the helmet. Although there are some issues with the overall design and there are things that could be improved upon, the device works properly and meets all the requirements for concussion detection in a contact sport.

[2] University of Pittsburgh Schools of the Health Sciences. Concussion statistics and Facts.

8 Appendices

Appendix A: Hardware Schematics

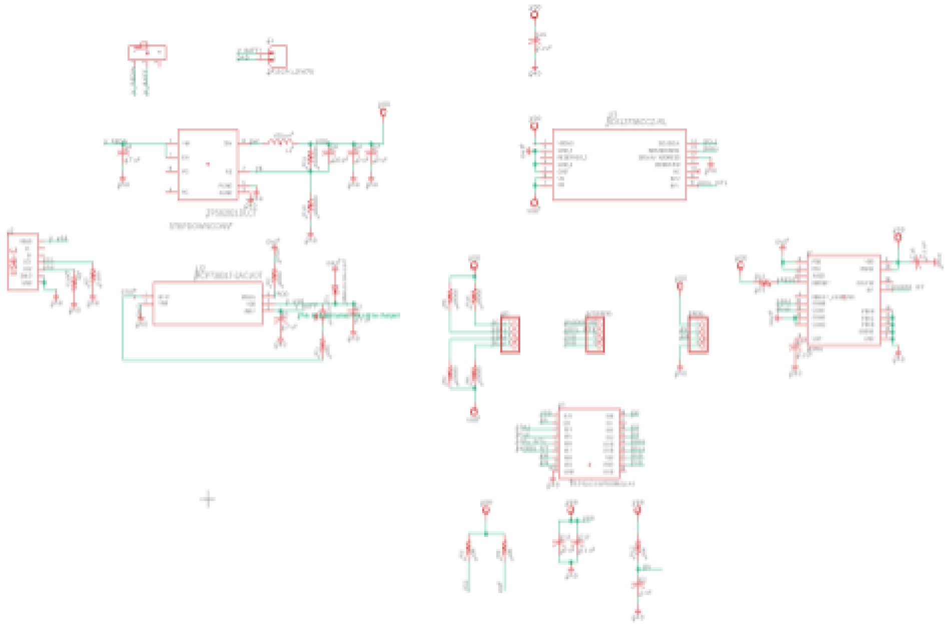


Figure A.1. Complete Hardware Schematic

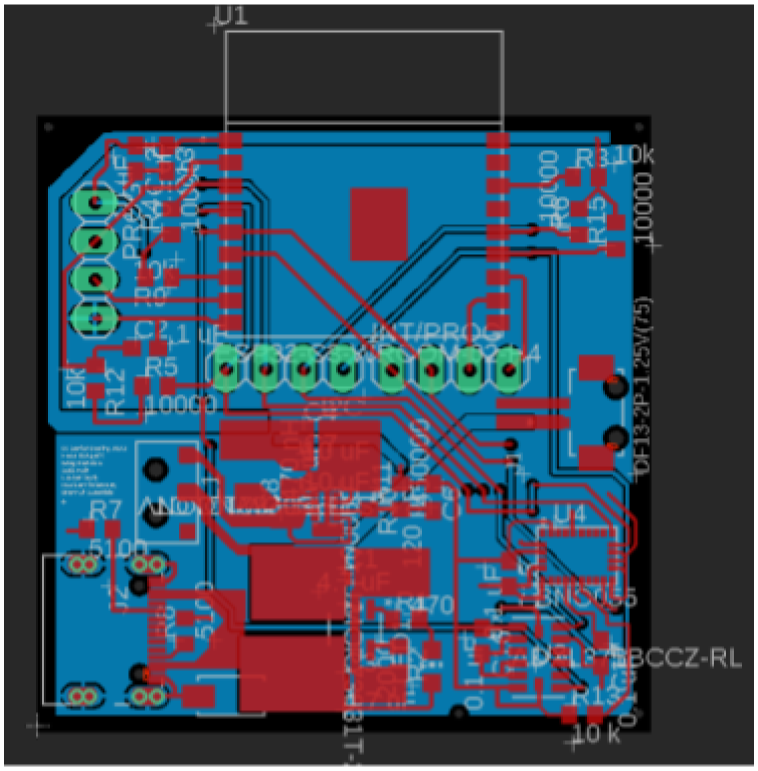


Figure A.2. Board Design

Appendix B: Software Listings

Code B.1. Node.cpp

```
#include <painlessMesh.h>
#include <Adafruit_ADXL375.h>
#include <Adafruit_Sensor.h>
#include <interruptcontrol.h>
#include <Adafruit_BNO055.h>
#include <sstream>
#include <string>
#include <esp_wifi.h>
#include <esp_now.h>


#define SIZE_DATA 15

#define NUMBER 2
#define SAMPLE_TIME 1/(32*4)

vector<double> HIGH_G_DATA;
vector<double> GYRO_DATA;
bool sendtag;
```

```
Adafruit_ADXL375 accel = Adafruit_ADXL375(12345);
Adafruit_BNO055 bno = Adafruit_BNO055(-1, 0x28, &Wire1);
Scheduler taskScheduler;
constexpr char WIFI_SSID[] = "SDNet";

uint8_t broadcastAddress[] = {0x58, 0xCF, 0x79, 0x16, 0x77, 0x7C}; // WiFi Hub
Mac Address

typedef struct struct_message
{
  int number;
  double force[SIZE_DATA];
  double gyro[SIZE_DATA];
} struct_message;

struct_message myData;
esp_now_peer_info_t peerInfo;

void detectConcussion();
void sendMessage();

Task taskDetectConcussion(0, TASK_FOREVER, &detectConcussion);
Task taskSendAlert(0, TASK_FOREVER, &sendMessage);

bool concussionDetected = false;
bool angular = false;
bool impact = false;

void sendMessage()
{
  Serial.printf("Trying to send message");
  if (impact == true)
  {
    // Don't do anything here
    myData.number = NUMBER;
    int count=0;
    for (int i = 14; i < 30; i++)
    {
        if (HIGH_G_DATA[i] <= -120 || HIGH_G_DATA[i] >= 120)
        {
          myData.gyro[count] = 0;
          myData.force[count] = 0;
        }
        else
        {
          myData.gyro[count] = 0;
          myData.force[count] = HIGH_G_DATA[i];
        }
        count++;
    }
  }
  if (angular == true)
  {
    // x,y,z format each time for rotational -- these are velocity in rad/s ...
need in acceleration rad/s^2. so we need to find slope of the values
    // time being 1/400 seconds = variable
    // finding the slope between each one
    // range 15 ~+=10
    myData.number = NUMBER;
    for (int i = 0; i < SIZE_DATA; i++)
    {
          myData.gyro[i] = GYRO_DATA[i];
          myData.force[i] = 0;
    }
  }

    for (int i = 0; i < SIZE_DATA; i++)
    {
        Serial.println(myData.force[i]);
    }
    for (int i = 0; i < SIZE_DATA; i++)
    {
        Serial.println(myData.gyro[i]);
    }

    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&myData,
sizeof(myData));
    Serial.println("just tried to send");
    if (result == ESP_OK)
    {
        Serial.println("Sent with success");
    }
    else
    {
        Serial.println("Error sending the data");
    }

    concussionDetected = false;
    angular = false;
    impact = false;
    HIGH_G_DATA.clear();
    GYRO_DATA.clear();
    taskDetectConcussion.enable(); // Re-enable concussion detection task
    taskSendAlert.disable();
    attachInterrupt(7, ISR_ANGULARDETECT, RISING);
    Reset_BNO055_Interrupts();
  }

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}

int32_t getWiFiChannel(const char *ssid)
{
  if (int32_t n = WiFi.scanNetworks())
  {
    for (uint8_t i = 0; i < n; i++)
    {
      if (!strcmp(ssid, WiFi.SSID(i).c_str()))
      {
        return WiFi.channel(i);
      }
    }
  }
  return 0;
}

void setup()
{
  Serial.begin(115200);

  WiFi.mode(WIFI_STA);

  int32_t channel = getWiFiChannel(WIFI_SSID);
  //Serial.println("Channel: ");
  //Serial.println(channel);

  // WiFi.printDiag(Serial); // Uncomment to verify channel number before
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
  esp_wifi_set_promiscuous(false);

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

  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  esp_now_register_send_cb(OnDataSent);

  taskScheduler.addTask(taskDetectConcussion);
  taskScheduler.addTask(taskSendAlert);
  taskDetectConcussion.enable();

  // Sensor setup
  pinMode(6, INPUT);
  pinMode(7, INPUT);

  Clear_ADXL375_Interrupts();
  Reset_ADXL375_Interrupts();
  ADXL375_Config();

  Clear_BNO055_Interrupts();
  Reset_BNO055_Interrupts();
  BNO055_Config();

}
```

```
void detectConcussion()
{
  // if data > threshold then
  // greg data here <3

  HIGH_G_DATA.clear();
  GYRO_DATA.clear();

  if (shocktag)
  {

    shocktag = 0;

    for (int i = 0; i < 30; i++)
    {
      ReadFIFO(HIGH_G_DATA);
    }

    for (int i = 0; i < 30; i++)
    {
      if (HIGH_G_DATA.at(i) > 30)
      {
        sendtag = true;
      }
    }

    if (sendtag)
    {
      /*for (int j = 0; j < 15; j++)
      {
         Serial.println(HIGH_G_DATA.at(j));
      }*/

      concussionDetected = true;
      impact = true;
      taskDetectConcussion.disable(); // Disable concussion detection task
      taskSendAlert.enable();         // Enable alert task
      sendtag = false;
    }

    Clear_ADXL375_Interrupts();
    Reset_ADXL375_Interrupts();
    attachInterrupt(7, ISR_ANGULARDETECT, RISING);
  }

  if (angulartag)
  {

    // Serial.print("INT TRIGGERED");

    angulartag = 0;

    for (int i = 0; i < 15; i++)
    {
      ReadGyro(GYRO_DATA);
    }
    /*for (int j = 0; j < 15; j++)
    {
       Serial.println(GYRO_DATA.at(j));
    }*/

    concussionDetected = true;
    angular = true;
    taskDetectConcussion.disable(); // Disable concussion detection task
    taskSendAlert.enable();         // Enable alert task

    attachInterrupt(6, ISR_SHOCKDETECT, RISING);
  }
}

void loop()
{
  taskScheduler.execute();
}
```

Code B.2. InterruptControl.h

```
#ifndef _INTERRUPTCONTROL_H
#define _INTERRUPTCONTROL_H

#include <vector>

#define ADXL375_Address 0x53
#define BNO055_Address 0x28

#define SDA_ADXL 19
#define SCL_ADXL 18

#define SDA_BNO055 4
#define SCL_BNO055 5

using namespace std;

void ADXL375_Config();
void BNO055_Config();

void ISR_SHOCKDETECT();
void ISR_ANGULARDETECT();

extern int shocktag;
extern int angulartag;

void ReadFIFO(vector<double>& HIGH_G_DATA);

void Clear_ADXL375_Interrupts();

void Reset_ADXL375_Interrupts();

void Reset_BNO055_Interrupts();
int Clear_BNO055_Interrupts();
```

```
void ReadGyro(vector<double>& GYRO_DATA);


#endif
```

Code B.3. InterruptControl.cpp

```
#include "interruptcontrol.h"
#include <iostream>
#include <stdlib.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <Arduino.h>

int shocktag = 0;
int angulartag = 0;


byte buff[6];

void ADXL375_Config(void){

  Wire.begin(SDA_ADXL, SCL_ADXL);
  pinMode(SDA_BNO055, INPUT);
  pinMode(SCL_BNO055, INPUT);

  /*START Disable Trigger Mode/enable Bypass Mode*/
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x38); //FIFO_CTL Register Address
  Wire.write(0x2A); //Disable Trigger Mode, set samples = 10
  Wire.endTransmission();
  /*END Disable Trigger Mode/enable Bypass Mode*/

  /*START Enable Trigger Mode*/
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x38); //FIFO_CTL Register Address
  Wire.write(0xEA); //Enable Trigger Mode, set samples = 10
  Wire.endTransmission();
  /*END Enable Trigger Mode*/

  //set shock threshold
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x1D);
  Wire.write(0x26);  //30 g (.78 g per LSB)
  Wire.endTransmission();

  //set shock duration
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x21);
  Wire.write(0x05); //10
  Wire.endTransmission();

  //set shock latency
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x22);
  Wire.write(0x00); //disable second shock
  Wire.endTransmission();

  //set shock window
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x23);
  Wire.write(0xF0); //300 ms
  Wire.endTransmission();

  //include all axes in shock detection
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x2A);
  Wire.write(0x07);
  Wire.endTransmission();

  //set data rate
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x2C);
  Wire.write(0x0F); //3200 Hz
  Wire.endTransmission();

  //endable single shock interrupt
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x2E);
  Wire.write(0x40); //single shock
  Wire.endTransmission();

  //set single shock to int1 pin
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x2F);
  Wire.write(0x00); //int 2 pin
  Wire.endTransmission();

  /*START Data Format*/
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x31); //DATA_FORMAT Reg
  Wire.write(0x0B);
  Wire.endTransmission();


  //start the measurement
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x2D);
  Wire.write(0x08); //measurement mode
  Wire.endTransmission();

  Wire.end();

  attachInterrupt(6, ISR_SHOCKDETECT, RISING);
}

void ISR_SHOCKDETECT(){

detachInterrupt(7);

shocktag = 1;
```

```
}

void ReadFIFO(vector<double>& HIGH_G_DATA){

Wire.begin(SDA_ADXL, SCL_ADXL);
pinMode(SCL_BNO055, INPUT);
pinMode(SDA_BNO055, INPUT);

int16_t data_x = 0, data_x_lsb = 0;
int16_t data_y = 0, data_y_lsb = 0;
int16_t data_z = 0, data_z_lsb = 0;

Wire.beginTransmission(ADXL375_Address);
Wire.write(0x32); //read LSB
Wire.endTransmission();

Wire.requestFrom(ADXL375_Address, 6);
while (Wire.available()) {
   data_x_lsb = Wire.read();
   data_x = Wire.read();
   data_y_lsb = Wire.read();
   data_y = Wire.read();
   data_z_lsb = Wire.read();
   data_z = Wire.read();

    data_x = (data_x << 8) | (data_x_lsb);
    data_y = (data_y << 8) | (data_y_lsb);
    data_z = (data_z << 8) | (data_z_lsb);
}

double data_x_1 = (double)data_x*.049;
double data_y_1 = (double)data_y*.049;
double data_z_1 = (double)data_z *.049;

HIGH_G_DATA.push_back(data_x_1);
HIGH_G_DATA.push_back(data_y_1);
HIGH_G_DATA.push_back(data_z_1);


//Serial.print("X: "); Serial.print(data_x_1); Serial.print("  Y: ");
Serial.print(data_y_1);
//Serial.print("  Z: "); Serial.print(data_z_1); Serial.print("  ");
Serial.println("g");

Wire.end();

return;

}

void ReadGyro(vector<double>& GYRO_DATA){

Wire.begin(SDA_BNO055, SCL_BNO055);
pinMode(SDA_ADXL, INPUT);
pinMode(SCL_ADXL, INPUT);

int16_t data_x = 0, data_x_lsb = 0;
int16_t data_y = 0, data_y_lsb = 0;
int16_t data_z = 0, data_z_lsb = 0;

//Register Page 0
Wire.beginTransmission(BNO055_Address);
Wire.write(0x07); //Register Page Sel
Wire.write(0x00); //Register page 0;
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x14);
Wire.endTransmission();

Wire.requestFrom(BNO055_Address, 6);
while (Wire.available()) {
   data_x_lsb = Wire.read();
   data_x = Wire.read();
   data_y_lsb = Wire.read();
   data_y = Wire.read();
   data_z_lsb = Wire.read();
   data_z = Wire.read();

    data_x = (data_x << 8) | (data_x_lsb);
    data_y = (data_y << 8) | (data_y_lsb);
    data_z = (data_z << 8) | (data_z_lsb);
}

double data_x_1 = (double)data_x/900;
double data_y_1 = (double)data_y/900;
double data_z_1 = (double)data_z/900;

GYRO_DATA.push_back(data_x_1);
GYRO_DATA.push_back(data_y_1);
GYRO_DATA.push_back(data_z_1);

//Serial.print("X: "); Serial.print(data_x_1); Serial.print("  Y: ");
Serial.print(data_y_1);
//Serial.print("  Z: "); Serial.print(data_z_1); Serial.print("  ");
Serial.println("rad/s");


Wire.end();

}


void Clear_ADXL375_Interrupts(){

Wire.begin(SDA_ADXL, SCL_ADXL);
pinMode(SCL_BNO055, INPUT);
pinMode(SDA_BNO055, INPUT);


Wire.beginTransmission(ADXL375_Address);
Wire.write(0x30); //read INT_SOURCE REG to clear
Wire.endTransmission();

Wire.requestFrom(ADXL375_Address, 1);
while (Wire.available()) {

  int result = Wire.read();


}

Wire.end();

return;

}

void Reset_ADXL375_Interrupts(){

  Wire.begin(SDA_ADXL, SCL_ADXL);
  pinMode(SDA_BNO055, INPUT);
  pinMode(SCL_BNO055, INPUT);

  /*START Disable Trigger Mode/enable Bypass Mode*/
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x38); //FIFO_CTL Register Address
  Wire.write(0x2A); //Disable Trigger Mode, set samples = 10
  Wire.endTransmission();
  /*END Disable Trigger Mode/enable Bypass Mode*/

  /*START Enable Trigger Mode*/
  Wire.beginTransmission(ADXL375_Address);
  Wire.write(0x38); //FIFO_CTL Register Address
  Wire.write(0xEA); //Enable Trigger Mode, set samples = 10
  Wire.endTransmission();
  /*END Enable Trigger Mode*/

  Wire.end();

  return;

}

void BNO055_Config(){

Wire.begin(SDA_BNO055, SCL_BNO055);
pinMode(SDA_ADXL, INPUT);
pinMode(SCL_ADXL, INPUT);


//Register Page 0
Wire.beginTransmission(BNO055_Address);
Wire.write(0x3D); //OP Mode reg
Wire.write(0x00); //config mode
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3F); //SYS_TRIGGER Register
Wire.write(0x40); //internal clock, reset int output
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3B); //Unit_SEL Register
Wire.write(0x02); //Rad/s for gyroscope data regs
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x07); //Page SEL Register
Wire.write(0x01); //Change to register page 1
Wire.endTransmission();


//Register Page 1

Wire.beginTransmission(BNO055_Address);
Wire.write(0x0F); //INT_MASK Register
Wire.write(0x04); //map AM gyro int to pin
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x10); //INT_EN Register
Wire.write(0x04); //Enable gyroscope any motion interrupt
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x17); //GYR_INT_SETTING Register
Wire.write(0x47); //Unfiltered, x-y-z enabled data
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x1E); //Gyro any-motion threshold calculator
Wire.write(0xF0); //36 deg/s for testing (save 0x24 = 36 deg/s)
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x0A); //GYR_Config Reg
Wire.write(0x38); //2000 deg/s 400 hz for interrupt (00) set at 40 for testing
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x1F); //GYR_AM_SET Reg
Wire.write(0x0B); //8 samples awake (int), 8 samples above the threshold to
trigger
Wire.endTransmission();

//Register Page 0
Wire.beginTransmission(BNO055_Address);
Wire.write(0x07); //Register Page Sel
Wire.write(0x00); //Register page 0;
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3D); //configuration register
Wire.write(0x03); //set to gyro mode
Wire.endTransmission();

attachInterrupt(7, ISR_ANGULARDETECT, RISING);

Wire.end();


return;

}

void Reset_BNO055_Interrupts(){

Wire.begin(SDA_BNO055, SCL_BNO055);
pinMode(SCL_ADXL, INPUT);
pinMode(SDA_ADXL, INPUT);

Wire.beginTransmission(BNO055_Address);
Wire.write(0x07); //Register Page Sel
Wire.write(0x00); //Register page 0;
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3D); //OP Mode reg
Wire.write(0x00); //config mode
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3F); //SYS_TRIGGER Register
Wire.write(0x40); //internal clock, reset int output
Wire.endTransmission();

Wire.beginTransmission(BNO055_Address);
Wire.write(0x3D); //configuration register
Wire.write(0x03); //set to gyro mode
Wire.endTransmission();

Wire.end();

}

int Clear_BNO055_Interrupts(){

  Wire.begin(SDA_BNO055, SCL_BNO055);
  pinMode(SDA_ADXL, INPUT);
  pinMode(SCL_ADXL, INPUT);

  int intstatus;

  Wire.beginTransmission(BNO055_Address);
  Wire.write(0x07); //Page SEL Register
  Wire.write(0x01); //Change to register page 1
  Wire.endTransmission();


  Wire.beginTransmission(BNO055_Address);
  Wire.write(0x37); //int STA register
  Wire.endTransmission();

  Wire.requestFrom(BNO055_Address,1);
  while(Wire.available()){

    intstatus = Wire.read();


  }

  Wire.end();

  return intstatus;

}


void ISR_ANGULARDETECT(){

  detachInterrupt(6);
  detachInterrupt(7);

  angulartag = 1;

  return;

}
```

Code B.4. WiFiHub.cpp

```
#include <esp_now.h>
#include <WiFi.h>
#include <Arduino.h>
#include <ArduinoJson.h>
#include "SPIFFS.h"
#include <esp_wifi.h>
#include <WebServer.h>
#include <vector>
#include "ESPAsyncWebServer.h"
#include <math.h>

//Defines WiFi requirements
#define MAC_ADDRESS "58:CF:79:16:85:CC"
#define SERVER_PORT 80

//Define sizes for arrays
#define SIZE_DATA 15
#define SEND_SIZE_DATA 12
#define ARRAY_LENGTH 5

//Define types for SendMessage
#define FORCE_TRUE 0
#define GYRO_TRUE 1

using namespace std;
```

```
//Data structure, must match the sender structure
typedef struct struct_message {
  int number;
  double force[SIZE_DATA];
  double gyro[SEND_SIZE_DATA];
} struct_message;

const char* ssid = "SDNet";
const char* password = "CapstoneProject";

//Data structur for website
double maxForceValues[ARRAY_LENGTH];
double maxGyroValues[ARRAY_LENGTH];
double numberValues[ARRAY_LENGTH];
String severityValues[ARRAY_LENGTH];
String severity = "Default";

//WebServer server(SERVER_PORT);
AsyncWebServer server(SERVER_PORT);

// Create a struct_message called myData
struct_message myData;

//Finds the absolute max value in the data
double findMax(double arr[], int size, bool type) {
  double max = abs(arr[0]); // Assume first element is the maximum
  for (int i = 1; i < size; i++) { // Loop through the remaining elements
    if (abs(arr[i]) > max) { // If an element is greater than the current max
      max = abs(arr[i]); // Store the new maximum
    }
  }
  if (type == FORCE_TRUE)
  {
    if (max > 65)
    {
      severity = "High";
    }
    else if ((max >= 45) && (max <= 65))
    {
      severity = "Medium";
    }
    else
    {
      severity = "Low";
    }
  }
  if (type == GYRO_TRUE)
  {
    if (max > 42)
    {
      severity = "High";
    }
    else if ((max >= 26) && (max <= 42))
    {
      severity = "Medium";
    }
    else
    {
      severity = "Low";
    }
  }

  return max; // Return the maximum value
}

//Callback function that will be executed when data is received
//Calls findmax() to find maximum value and define severity
//Moves data values into maxforceValues, etc to display on the website
continuously (look at getJSON)
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);

  Serial.print("Vector: ");
  for (int i = 0; i < 5; i++) {
    Serial.printf("%f ", myData.force[i]);
  }

  Serial.printf("Max: %f ", findMax(myData.force, SIZE_DATA, FORCE_TRUE));
  Serial.printf("Number: %d", myData.number);
  Serial.println();

  //Moves data values into website arrays
  for (int i = ARRAY_LENGTH - 1; i >= 1; i--) {
          maxForceValues[i] = maxForceValues[i-1];
          maxGyroValues[i] = maxGyroValues[i-1];
          numberValues[i] = numberValues[i-1];
          severityValues[i] = severityValues[i-1];
    }

    numberValues[0]=myData.number;
    maxForceValues[0]=findMax(myData.force, SIZE_DATA, FORCE_TRUE);
    maxGyroValues[0]=findMax(myData.gyro, SEND_SIZE_DATA, GYRO_TRUE);
    severityValues[0]=severity;

  //event.send ...
}

//Ensures that the Hub and Nodes are on the same WiFi channel
int32_t getWiFiChannel() {
  if (int32_t n = WiFi.scanNetworks()) {
      for (uint8_t i=0; i<n; i++) {
          if (!strcmp(ssid, WiFi.SSID(i).c_str())) {
              return WiFi.channel(i);
          }
      }
  }
  return 0;
}

//Updates the values displayed on the webpage
String getJSON(){
  String json = "{";
```

```
    json += "\"numberA\":" + String(numberValues[0])  + ",";
    json += "\"numberB\":" + String(numberValues[1])  + ",";
    json += "\"numberC\":" + String(numberValues[2])  + ",";
    json += "\"numberD\":" + String(numberValues[3])  + ",";
    json += "\"numberE\":" + String(numberValues[4])  + ",";
    json += "\"severityA\":\"" + severityValues[0]  + "\",";
    json += "\"severityB\":\"" + severityValues[1]  + "\",";
    json += "\"severityC\":\"" + severityValues[2]  + "\",";
     json += "\"severityD\":\"" + severityValues[3]  + "\",";
    json += "\"severityE\":\"" + severityValues[4]  + "\",";
    json += "\"maxForceA\":" + String(maxForceValues[0]) + ",";
    json += "\"maxForceB\":" + String(maxForceValues[1]) + ",";
    json += "\"maxForceC\":" + String(maxForceValues[2]) + ",";
    json += "\"maxForceD\":" + String(maxForceValues[3]) + ",";
    json += "\"maxForceE\":" + String(maxForceValues[4]) + ",";
    json += "\"maxGyroA\":" + String(maxGyroValues[0]) + ",";
    json += "\"maxGyroB\":" + String(maxGyroValues[1]) + ",";
    json += "\"maxGyroC\":" + String(maxGyroValues[2]) + ",";
    json += "\"maxGyroD\":" + String(maxGyroValues[3]) + ",";
    json += "\"maxGyroE\":" + String(maxGyroValues[4]);
    json += "}";
    return json;
    //myData.e.clear();
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_AP_STA);
  Serial.println("Mac Address: ");
  Serial.println(WiFi.macAddress());

  // Connect to Wi-Fi and Display IP Address for Website
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to Wi-Fi...");
  }
  Serial.print("Connected. IP: ");
  Serial.println(WiFi.localIP());
  Serial.print("Wi-Fi Channel: ");
  Serial.println(WiFi.channel());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", "text/html");
  });

  server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request){
    String jsonData = getJSON();
    request->send(200, "application/json", jsonData);
  });

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

  // Once ESPNow is successfully Init, we will register for recv CB to get
packer info
  esp_now_register_recv_cb(OnDataRecv);

  // Initialize and mount SPIFFS (allows us to upload HTML file)
  if (!SPIFFS.begin(true)) {
    Serial.println("An error has occurred while mounting SPIFFS");
    return;
  }

  // Start the server for the website
  server.begin();
}

void loop() {
  //Empty on purpose
}
```

Code B.5. Website.html

```
<!DOCTYPE html>
<html>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script>

<head>

  <style>
    table {
      border-collapse: collapse;
      border: 1px solid black;
      margin: auto;
      /* set margin to auto to center horizontally */
      display: inline-block;
    }

    th,
    td {
      padding: 5px;
      border: 1px solid black;
    }

    .container {
      text-align: center;
      /* center child elements horizontally */
    }

    body {
      background-color: white;
      font-family: verdana;
    }
    .headertext {
      display: flex;
      flex-flow: row;
      justify-content: space-evenly;
      align-items: center;
      text-align: center;
      margin: 20px;
      background: #ff6666;
      font-family: verdana;
    }

    .mainbody {

      display: flex;
      align-items: center;
      text-align: center;
      margin: 30px;
      padding: 20px;
      justify-content: safe center;
    }

    .mainbody>p {

      font-weight: bold;
      border-style: solid;
      justify-content: flex-start;
      margin: 10px;
      padding: 20px;
      font-size: 30px;
      line-break: strict;
    }

    span {
      font-weight: lighter;
      line-break: strict;
      font-size: 40px;
    }

    .chart {
      display: flex;
      align-items: center;
      text-align: center;
      width: 50%;
      /* set width of chart to 50% of container */
      float: right;
    }
  </style>
  <meta charset="UTF-8" />
  <title>ESP-NOW Data</title>
</head>

<body>
  <div class="headertext">
    <h1>CONCUSSION DATA</h>
  </div>
  <div class="mainbody">
```

```
    <p>Player: <span id="numberA0"></span></p>
    <p>Severity: <span id="severityA0"></span></p>
  </div>

  <div class="container">
    <title>Random Table Data</title>

    <table>
      <tr>
        <th>Player Number</th>
        <th>Severity</th>
        <th>Max Force (g)</th>
        <th>Max Velocity (rad/s)</th>
      </tr>
      <tr>
        <td><span id="numberA1"></span></td>
        <td><span id="severityA1"></span></td>
        <td><span id="maxForceA1"></span></td>
        <td><span id="maxGyroA1"></span></td>
      </tr>
      <tr>
        <td><span id="numberB1"></span></td>
        <td><span id="severityB1"></span></td>
        <td><span id="maxForceB1"></span></td>
        <td><span id="maxGyroB1"></span></td>
      </tr>
      <tr>
        <td><span id="numberC1"></span></td>
        <td><span id="severityC1"></span></td>
        <td><span id="maxForceC1"></span></td>
        <td><span id="maxGyroC1"></span></td>
      </tr>
      <tr>
        <td><span id="numberD1"></span></td>
        <td><span id="severityD1"></span></td>
        <td><span id="maxForceD1"></span></td>
        <td><span id="maxGyroD1"></span></td>
      </tr>
      <tr>
        <td><span id="numberE1"></span></td>
        <td><span id="severityE1"></span></td>
        <td><span id="maxForceE1"></span></td>
        <td><span id="maxGyroE1"></span></td>
      </tr>
    </table>
  </div>

  <script>
    function updateData() {
      fetch("/data")
        .then((response) => response.json())
        .then((data) => {
          //A Values
          document.getElementById("numberA0").textContent = data.numberA;
          document.getElementById('severityA0').textContent = data.severityA;
          document.getElementById('numberA1').textContent = data.numberA;
          document.getElementById('severityA1').textContent = data.severityA;
          document.getElementById('maxForceA1').textContent = data.maxForceA;
          document.getElementById('maxGyroA1').textContent = data.maxGyroA;

          //B Values
          document.getElementById('numberB1').textContent = data.numberB;
          document.getElementById('severityB1').textContent = data.severityB;
          document.getElementById('maxForceB1').textContent = data.maxForceB;
          document.getElementById('maxGyroB1').textContent = data.maxGyroB;

          //C Values
          document.getElementById('numberC1').textContent = data.numberC;
          document.getElementById('severityC1').textContent = data.severityC;
          document.getElementById('maxForceC1').textContent = data.maxForceC;
          document.getElementById('maxGyroC1').textContent = data.maxGyroC;

          //D Values
          document.getElementById('numberD1').textContent = data.numberD;
          document.getElementById('severityD1').textContent = data.severityD;
          document.getElementById('maxForceD1').textContent = data.maxForceD;
          document.getElementById('maxGyroD1').textContent = data.maxGyroD;

          //E Values
          document.getElementById('numberE1').textContent = data.numberE;
          document.getElementById('severityE1').textContent = data.severityE;
          document.getElementById('maxForceE1').textContent = data.maxForceE;
          document.getElementById('maxGyroE1').textContent = data.maxGyroE;

        })

        .catch((error) => console.error(error))
    }

    // Update the data on the page every second
    setInterval(updateData, 100);
  </script>
</body>

</html>
```

Appendix C: Parts

Part C.1. Step Down Converter:

https://www.ti.com/product/TPS62821/part-details/TPS62821DLCT

Part C.2. Battery Controller:

https://www.digikey.com/en/products/detail/microchip-technology/MCP73832T-2ACI-MC/1223140

Part C.3. Gyroscope Sensor:

https://www.bosch-sensortec.com/products/smart-sensors/bno055/

Part C.4. ESP32-C3:

https://www.espressif.com/en/products/socs/esp32-c3

Part C.5. Accelerometer Sensor:

https://www.analog.com/en/products/adxl375.html