Directional Impact Sensing Helmet (DISH)

ECE 445 Final Report

Team 5: Patrick Sear, Saathvik Narra, Ryan Josephson
Professor: Viktor Gruev
TA: Ugur Akcal
Spring 2023

Abstract

The purpose of this document is to highlight the progress, successes, and failures of the Directional Impact Sensing Helmet (DISH). Within, discussions on design processes, design details, control sequences, and procedures can be found.

Contents

1 Introduction

The introduction provides the problem statement, describes the solution, displays a visual diagram about how our solution will work, and explains the system from a high level.

1.1 Problem

In the NFL, many athletes suffer from concussions or other conditions as a result of repetitive, severe head trauma. These events can lead to long-term effects on the athlete's health and significantly contribute to the reduced lifespan of professional football players. The average professional football player dies younger than age 60 [1], no doubt accelerated by frequent, intense head trauma. This problem can be helped by making accurate collision data immediately available to medical personnel for making game-time decisions and helmet manufacturers so that they can make informed design choices based on real, in-game data.

1.2 Solution

The Directional Impact Sensing Helmet (DISH) is an electronic system that can be attached inside any existing football helmet. It can determine where on an athlete's head a collision occurs, as well as how hard the hit is. The data collected will be quickly available to personnel on the sidelines so that they can make real-time decisions to ensure the safety of their players. The information will be visualized for ease of use of the team medical staff so that they can best inspect the player.

The data will also be useful for designing the next generation of helmets. Over the course of a season, all hits can be tracked and analyzed so that newer models can be tailored precisely to protect from hits that occur in a real, game-time environment. The helmets could even be designed differently for each position, prioritizing the most common hits each role receives.

For the DISH to work properly, we can group our project into three modules: power, control, and data reception and visualization.

1.3 Visual Aid

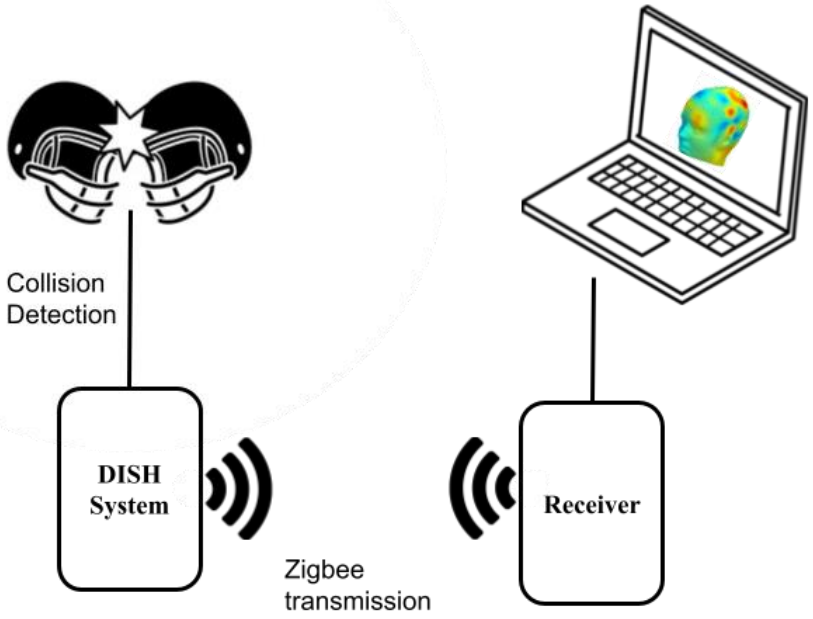

Figure 1: High-Level Overview of DISH System

Upon detecting a collision through signal output from the force sensitive resistor array, the onboard microcontroller begins transmitting data through the Zigbee module. This data will be received and passed to the personal computer, in which it will be analyzed and displayed for the user.

1.4 High-Level Requirements

- The helmet must track the location and severity of each collision within <20% error.
- The receiver must know that a dangerous collision has occurred within 5 seconds of the collision at least 90% of the time.
- The in-helmet system must be able to comfortably fit within a standard football helmet.

2 Design

2.1 Block Diagram

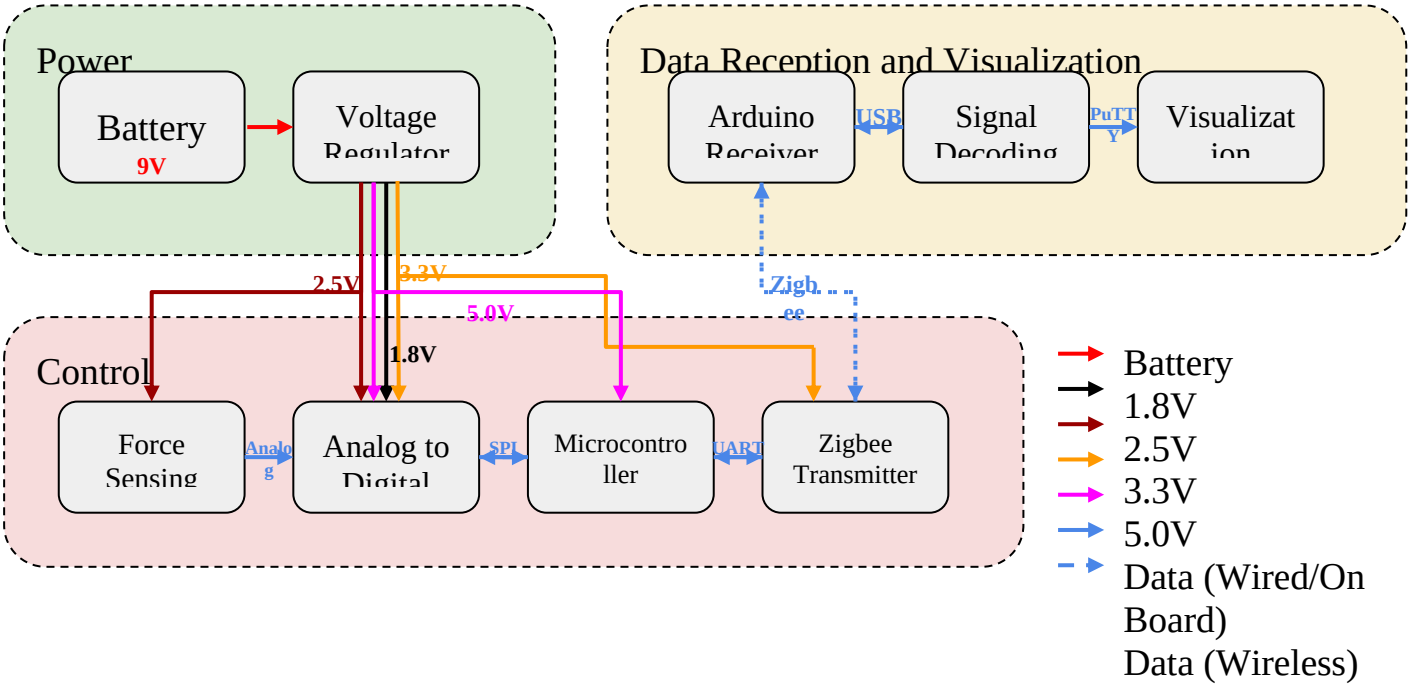

Figure 2: Block diagram

2.2 Power Subsystem

2.2.1 Overview

The purpose of the power subsystem is to provide power to the other subsystems. Four different voltage levels are required in total: 1.8V, 2.5V, 3.3V, and 5.0V. Within the power subsystem is the battery and the voltage regulators. The power subsystem's schematic can be seen in *Figure 3*.

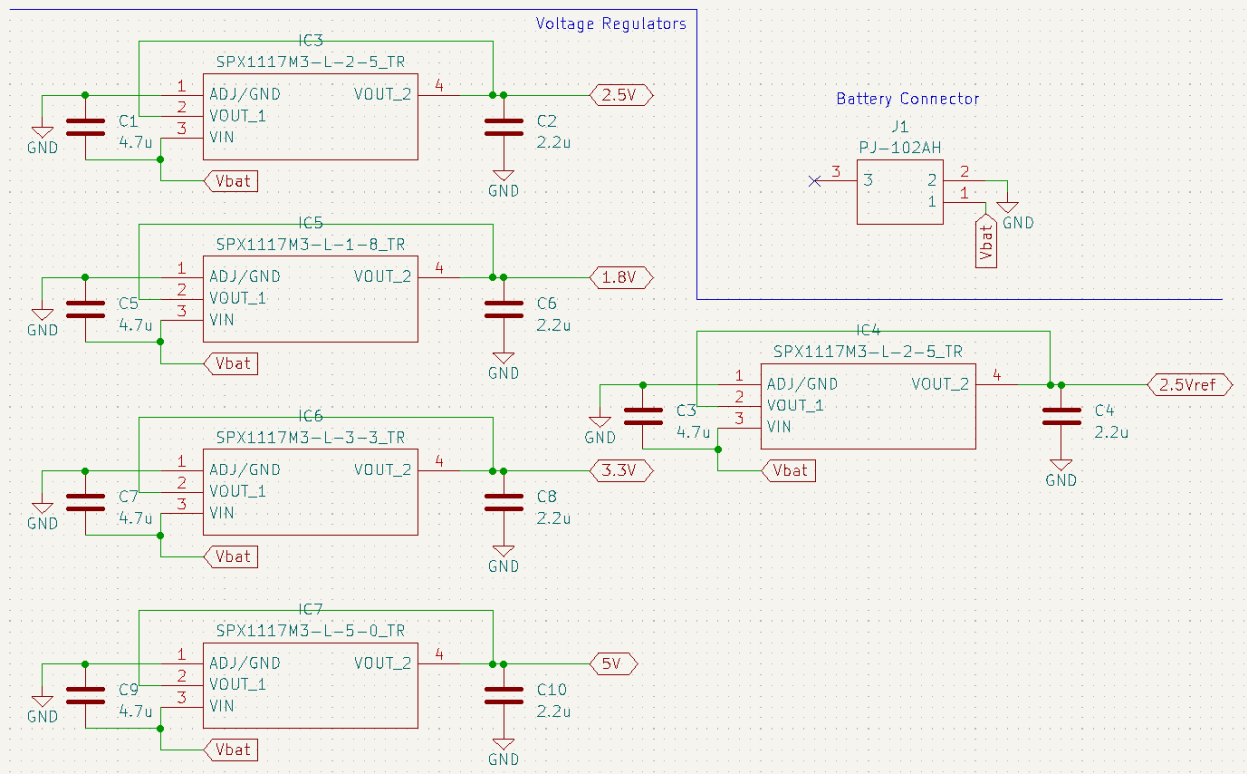

Figure 3: Power subsystem schematic

2.2.2 Design Procedure

In designing the power subsystem, we knew we needed to maintain a few voltage rails to supply power for the other subsystems' components. So, we first selected the voltage regulators. We selected SPX1117 voltage regulators, since they all have the same packaging, and we had some prior experience working with them. Also, according to the datasheet [2], the voltage regulators have an overlapping input voltage range of 6.4V-10V. For both this reason, as well as the fact that we need a battery small enough to fit inside a football helmet, a 9V battery was selected as the main power source, which is to be fed into the voltage regulators. To physically connect the battery to the PCB, we opted for a PJ-102AH connector, which is a very standard connector that we have previously used with 9V batteries.

2.2.3 Design Details

As seen in *Figure 3*, each of the voltage regulators have a 4.7µF capacitor tied from the input voltage to ground, and a 2.2µF capacitor tied from the output voltage to ground. This is as required per the datasheet [2]. Also, the battery is connected in such a way that the positive and negative leads of the battery correspond to the connector in the correct fashion, where the voltage across Vbat and ground is the battery voltage.

2.2.4 Requirements and Verification

The power subsystem must satisfy the following requirements:

1. The battery must supply a constant voltage between 6.4V and 10V and supports up to 350mA of current draw.
2. The 1.8V rail must maintain a voltage between 1.65V and 1.95V at a current of at least 17mA.
3. The 2.5V rail must maintain a voltage between 2.475V and 2.525V at a current of at least 20mA.
4. The 3.3V rail must maintain a voltage between 2.7V and 3.6V at a current of at least 220.5mA.
5. The 5V rail must maintain a voltage between 4.75V and 5.25V at a current of at least 91mA.

For the requirements and verification table and the results of the tests, refer to Appendix A. As shown in Appendix A, all five of the power subsystem requirements passed. The battery and the voltage regulators consistently output a voltage in the desired voltage range under the worst-case current draw.

2.3 Control Subsystem

2.3.1 Overview

The control subsystem oversees taking all the force sensing resistor (FSR) data and seeing if a collision has occurred. The voltage read across the FSRs in a voltage divider circuit is converted to digital data using an analog to digital converter (ADC) and is then fed into the microcontroller (MCU). The microcontroller then, using an algorithm we built, will detect whether a collision has occurred. If a collision has occurred, the eight FSR data values are sent at the highest collision moment over the Zigbee to the data reception and visualization subsystem. The schematics for the MCU and the ADC can be seen in *Figure 4* and *Figure 5*, respectively.

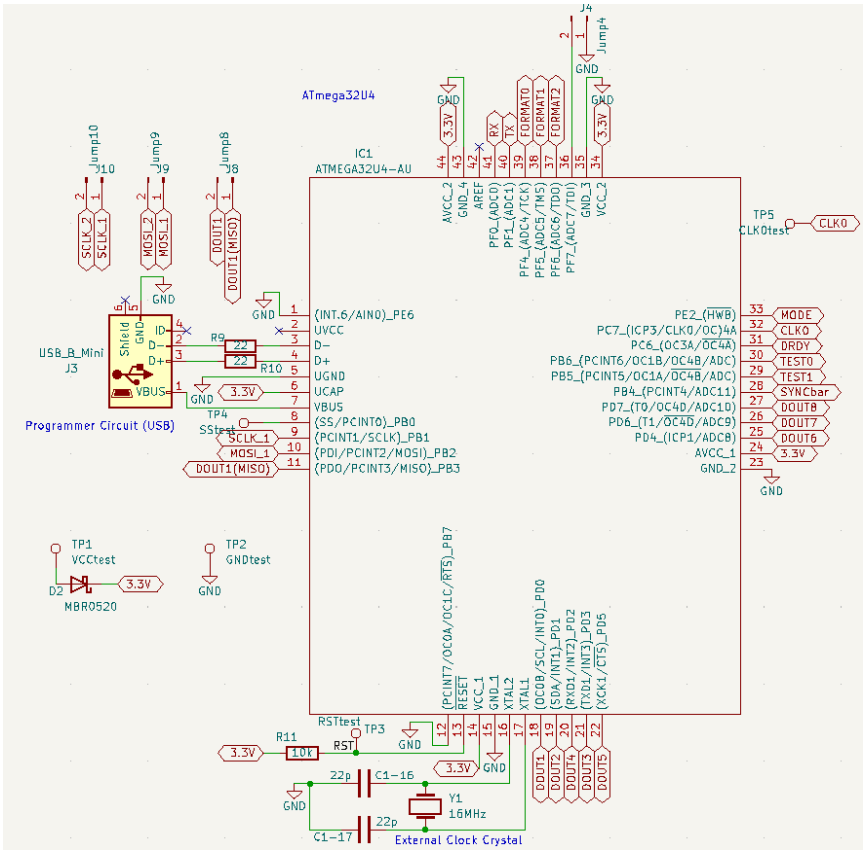

Figure 4: ATmega32u4 Schematic

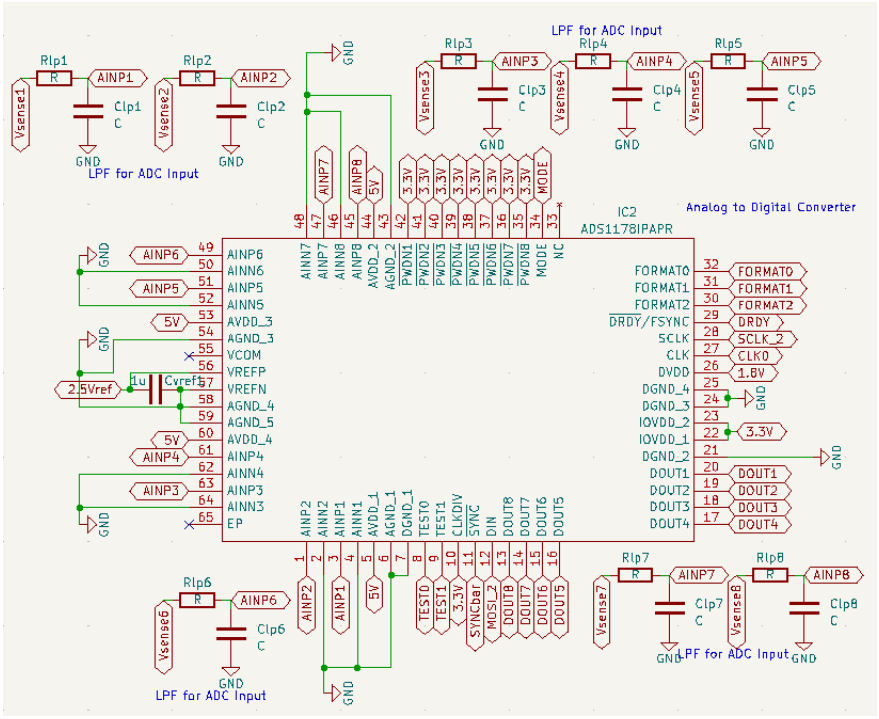

Figure 5: Analog to Digital Converter Schematic

2.3.2 Design Procedure

In designing the control subsystem's hardware, we knew we needed a variety of functionalities within this subsystem. Firstly, we need to have a way to sense the force of collisions. For this reason, we opted for force sensing resistors. We can use a voltage divider circuit, which can be an effective way to read changes in the force applied, since a larger force applied corresponds to a lower resistance. We have a detachable connector between the FSRs and the connections to the ADC such that the main module can be removed from the helmet. Since we are trying to collect as much data as possible through as many sensors as possible, we knew that we had to select a chip that has as many I/O pins as possible. For this reason, using the ATmega32u4 seems like the perfect option. To read the data generated by the force sensing resistors and send it wirelessly, we need to convert it from analog to digital data. However, according to the ATmega32u4's datasheet [5], the ATmega32u4 collects its eight analog inputs at distinct times through a multiplexer rather than reading the values simultaneously. To improve accuracy in the data collected by the force sensing resistors and ensure the data is all from the same exact moment, we decided that using a simultaneous sampling analog to digital converter is optimal. The specific part, the ADS1178IPAPR chip, has eight analog to digital converters, according to its datasheet [3]. Also, it can communicate via SPI, so communicating with the microcontroller is a straightforward task. Thirdly, we needed a way to communicate with the wireless receiver. We chose to use Zigbee because the amount of data needed to be sent wirelessly wasn't too large and our rate of transmission is relatively low. We also needed something that had a range greater than forty meters so we could send a signal from anywhere on much of the football field, which Zigbee can easily accomplish per the datasheet [4]. Zigbee is designed for internet-of-things applications and is well suited to applications where one central controller (the receiver module) interacts with many accessory nodes (each helmet on the field).

Once we had designed the hardware, we had to design the software that would go on the MCU. This software would be responsible for reading in the sensor data as produced by the ADC via SPI and determining if the data indicated that a collision had occurred. Once the MCU detects a collision, it will begin to record all data from the ADC until it detects that the collision has ended. At this point, the data is processed, packaged, and sent through the Zigbee transmitter to the receiver. Following the data transmission, the MCU baseline readings for each sensor must be recalibrated, as the resting reading for each sensor may shift significantly following each collision.

2.3.3 Design Details

Looking at *Figure* 4, the ATMega32u4 is powered at 3.3V. The USB interface circuit for serial communication and testing is derived from the datasheet, namely the typical self-powered application with 3.0V to 3.6V I/O circuit [5]. Also, there is an external clock crystal connected to the chip. This allows the chip to successfully operate at the 8MHz operating frequency. The RX and TX signals are connected from the MCU directly to the Zigbee module. The only required pins for use with the Zigbee module are the power pin of 3.3V, the TX and RX pins to communicate

with the microcontroller via UART, and the ground pin. These connections should allow for proper communication via Zigbee.

When looking at *Figure 5*, there are many things to observe. First, the ADC chip is fed in each of the four power rails. The chip itself needs 5V for power. 2.5V is used as a voltage reference for the analog inputs. The 3.3V rail is used for the I/O, so 3.3V will be the voltage level used to communicate with the microcontroller. And lastly, the DVDD pin is set to 1.8V to power the internal digital components of the circuit [3]. The analog inputs are taken from the output of a voltage divider circuit consisting of one fixed resistor and one force sensing resistor. There are eight AINP and AINN pins on the ADC, corresponding to the eight positive and negative differential analog inputs. The voltage divider output is passed through a low-pass filter before being input to the ADC in order to remove some of the random noise. The filter consists of the eight Clp and Rlp components shown in *Figure 5*. The CLK0 pin outputs the MCU's internal clock, and the ADC runs off the same clock. All other pins are connected directly to I/O pins of the microcontroller to be controlled using software.

The microcontroller code control flow can be seen in *Figure 6*. When the board is powered on, the MCU is first initialized, where settings for the different protocols are defined, as well as setting baud rates and creating variables to reduce the amount of machine cycles later code would take to perform. After the MCU is initialized (and after any Zigbee transmit), the sensors are calibrated, and we find the resting baseline for the sensors. Sensor data is read via SPI communication between the ADC and microcontroller. All eight digital outputs are transmitted sequentially to the designated SPI MISO pin on the MCU. We then take the sensor readings and see if the resting threshold of a sensor is ever passed by a certain amount, which indicates that a collision has occurred. If this is not the case, we simply keep reading in sensor values until we detect a collision. We classify the collision as the time when the sensors are above the resting threshold. Once we come back down to the resting threshold, the collision is finished, and we store this data and send the highest detected force over the Zigbee. We then go back to the calibration phase, and we repeat the process.

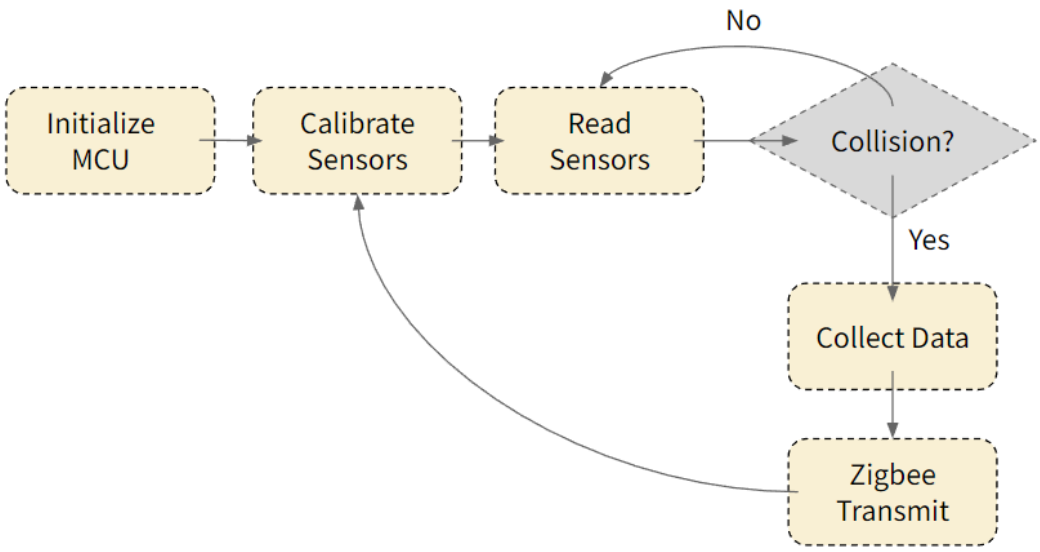

Figure 6: Microcontroller Control Flow

The ADC digital I/O runs on 3.3V, so we also need to run the MCU on 3.3V such that the I/O levels are matched. Running the MCU at a higher voltage runs the risk of not properly recognizing a 3.3V input signal as a logic high. A problem we ran into when running on 3.3V power is that we occasionally triggered the built-in brown-out protection. Brown-out protection is triggered when the MCU is powered at too low of a voltage, typically below 2.7V, and is used to prevent damage to internal components. However, triggering brown-out runs the risk of corrupting the MCU fuses and potentially bricking the entire system. When using the programmer, we had a Schottky diode that dropped the voltage power rail to 3V. Due to having fluctuations in voltage, we frequently fell below 2.7V and triggered the brown-out protection. After enough infractions, the controller fuses became corrupted, and it was entirely unprogrammable, requiring the use of a new chip.

2.3.4 Requirements and Verification

The control subsystem must satisfy the following requirements:

1. The microcontroller must be able to comprehend data from the analog to digital converter via SPI over 90% of the time.
2. The microcontroller must be able to correctly identify when a collision occurs with more than 90% accuracy.
3. The microcontroller must be able to send and receive signals to and from the remote receiver via the Zigbee transmitter more than 90% of the time.
4. The off-chip simultaneous sampling ADC must be able to measure all signals from the force sensing resistor array concurrently more than 90% of the time.
5. The Zigbee transmitter must be able to properly send a signal up to 40 meters away.
6. The signal must be able to be sent through the helmet and through adverse conditions with an accuracy of over 90%.

For the requirements and verification table and the results of the tests, refer to Appendix B. As shown in Appendix B, all six of the control subsystem requirements passed.

2.4 Data Reception and Visualization Subsystem

2.4.1 Overview

The reception and visualization subsystem is critical to the accessibility of the project. The DISH system is designed to maximize the ease of use and plug-and-play aspect of data collection. By presenting the data received to the end user in an easily digestible format, the widespread appeal of the system is increased. A diagram depicting the general operation of the subsystem can be seen in *Figure 7*. The reception and visualization subsystem begins at the Arduino receiver. After the microcontroller receives and decodes the Zigbee packet, it passes the relevant information through to the Python script via serial communication. The script processes the eight sensor readings into three-dimensional force data. Finally, the displayed 3-D mesh is updated to display the calculated force readings.

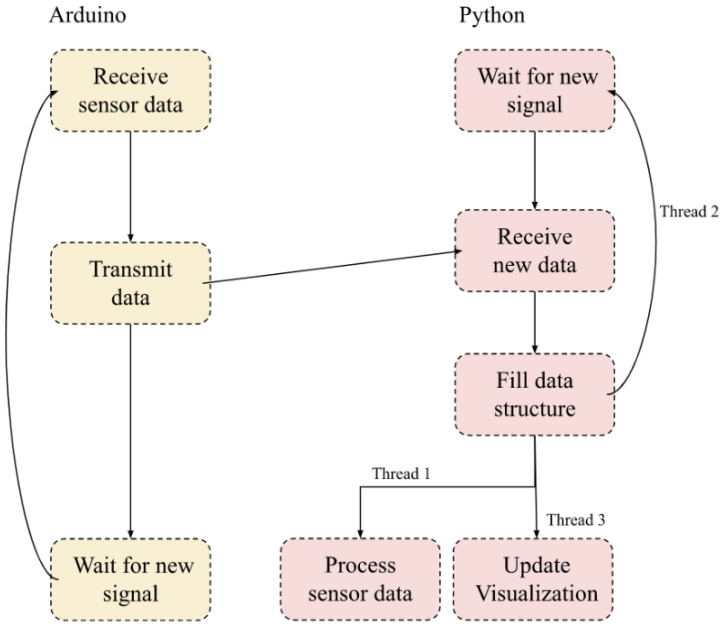

Figure 7: General operation of Data Reception and Visualization Subsystem

2.4.2 Design Procedure

Earlier on, we considered using a custom PCB as a receiver. This was replaced by an off-the-shelf SparkFun RedBoard (ATMega328P, Arduino Uno equivalent). Building a custom receiver would have ultimately been an inefficient use of time and money to perform the trivial tasks of Zigbee reception and serial transmission. Initial designs also made use of PuTTY, a serial monitor, as a middleman between the serial communication and Python script. This would have enabled

immediate logging of serial data to text files for close record keeping. The serial monitor was also eliminated in favor of direct communication between the script and serial port. While text file generation is slower, the script can create post-processing data files rather than saving raw data only. Direct communication also allows the packet to reach the script faster, therefore alerting the user faster such that immediate substitutions can be made in game-time.

2.4.3 Design Details

Packet integrity is internally verified by the Zigbee module, so the only purpose of the RedBoard is to strip packet metadata and output the seventeen bytes of sensor data to the serial port. One byte is reserved for team and player data, and sixteen bytes correspond to data, where two bytes of data comes from each of eight sensors. Upon reception by the script, the user is immediately notified of the collision. At all times, a model head is being displayed for the purpose of data visualization. At the user's prompt, the model will be updated for the newly received data with a heatmap, the center of which indicates the predicted location of the collision. The collision location is predicted by a weighted average of the three most significant data points in each dimension, as seen in *Equation 1*.

$$x_f = \sum_{i=1}^{3} \frac{f_i}{\sum_{j=1}^{3} f_j} * x_i \tag{1}$$

In *Equation 1*, x_f represents the position in one dimension of the predicted average point, so x_1, x_2, and x_3 represent the positions in the same dimension of each sensor we use as data, and f_1, f_2, and f_3 represent the force at each of these sensors. This is repeated in all three dimensions to find an average point. Taking the normal vector of the plane defined by the three sensor points, we move the average point along the normal until it coincides with a position on the surface of the head. This final point is where we determine to be the center of the collision. While imperfect, we have determined this point to be well within the margin for error as defined by our requirements.

2.4.4 Requirements and Verification

The reception and visualization subsystem must satisfy the following requirements:

1. The receiver must receive packets sent with a maximum error rate of 10%.
2. The signal received by the Arduino must be delivered to the serial input exactly as it is seen by the hardware.
3. The visualizer must be able to identify the impact within 20% of where it occurred.
4. The data must be visualized within 30 seconds of packet reception.

For the requirements and verification table and the results of the tests, refer to Appendix C. As shown in Appendix C, all six of the data reception and visualization subsystem requirements passed. The receiver module easily receives and transmits all data without error. Data is visualized within 15 seconds of packet reception through the script, and the located result is exactly as the data predicts.

2.5 Physical Design

The physical design of the DISH can be seen in *Figure 8*, which shows the fully assembled helmet.

Figure 8: Assembled DISH

2.5.1 PCB Encasement

As seen in *Figure 8*, the helmet, when fully assembled, has an encasement fit snugly inside. This is the encasement for where the PCB resides. With help from the machine shop and by using a case that they had on hand, it was made possible to create the ideal encasement to house the PCB. The case, as can be seen in *Figure 8*, fits where there is no padding by the left ear. The case can be removed from the FSR array via a connector that connects directly to the board within.

2.5.2 Helmet Assembly

As seen in *Figure 8*, the force sensing resistors aren't always visible. They are placed in the space between the outer shell of the helmet and the internal padding, as seen in *Figure 9*. This is done so as much force can be applied as possible to the FSRs. Additionally, on each FSR, there is a spacer that closes the gap between the pad and the outer shell of the helmet. This is done to ensure the FSRs are always contacting and will pick up on a force when applied.

Figure 9: FSR placement within helmet with spacers

3 Cost & Schedule

3.1 Cost Analysis

3.1.1 Parts Cost Analysis

The total cost for parts, as seen in *Table D4* in Appendix D, is $209.31. Notably, more parts were ordered than required. This is done in case parts are damaged during testing or to obtain better deals.

3.1.2 Hours of development and Labor Costs

Our group consists of two electrical engineers and one computer engineer. The average pay for electrical engineers with a bachelor's degree is $80,000 and is $100,000 for computer engineers according to the Grainger College of Engineering [6, 7].

Category	**Estimated Hours**		
	Saathvik	Ryan	Patrick
Circuit Design	0	20	5
Board Layout and Components Check	10	25	20
Full Stack System Monitor	40	4	30
Soldering	6	15	8
Prototype And Debug	60	60	60
Documentation and Logistic	25	30	25
Total Hours	**141**	**154**	**148**

Table 1: Hours of Development

Labor Cost:

$$\text{Saathvik: } \$50\ (hourly\ rate) \cdot 2.5 \cdot 141\ (total\ hours) = \$17625 \quad (2)$$

$$\text{Ryan: } \$40\ (hourly\ rate) \cdot 2.5 \cdot 154\ (total\ hours) = \$1540 0 \quad (3)$$

Patrick: $\$40\ (hourly\ rate) \cdot 2.5 \cdot 148\ (total\ hours) = \$14800$ (4)

Total Labor Cost = $47825

3.1.3 External Resources

- Machine Shop: The machine shop was used to modify a physical enclosure for our project to encase the electronics. The generic enclosure we got was modified in the workshop to have holes for power and a hole for connectors to connect force sensors with PCB. It was estimated modifications to our enclosure took three hours.
- Senior Design Lab Workshop: We needed the lab in order to do soldering and testing using the oscilloscope, a digital multimeter, DC power supply, and electronic load.

3.1.4 Total Costs

Category	Estimated Cost
Parts	$260.31
Labor	$47825.00
Total	$48085.31

Table 2: Total Costs

3.2 Schedule

The full schedule can be seen in Appendix D in *Table D2*. This schedule was used as a baseline for planning our work for each week, but modifications were made as appropriate depending on the progress of the previous week. Work was divided up equally, with each team member taking a greater share of the work pertaining to their own previous experiences. We also frequently worked cross-functionally so that development of each subsystem could benefit from an extra outside perspective and the whole team would be familiar with all parts of the project.

4 Conclusion

4.1 Accomplishments and Uncertainties

From the work that we have done, we have found great success in creating the solution to the problem presented. All our subsystem requirements are met as proven by Appendix A, Appendix B, and Appendix C. When combined, the subsystem requirements help meet the high-level requirements. The final project is fully functional and met our goals.

One area of uncertainty is the existence of false positives. On occasion, the system registers a collision when obviously none occurred. These false positives are certainly better than experiencing false negatives, as in the application of the DISH, it may be obvious if a collision on the football field happened or not, so the false positive can be ignored. Also, false positives are beneficial in the idea that it is better to be safe than sorry; pulling an athlete out of play over concerns that a collision may have occurred is better than not pulling out the athlete due to a false negative.

4.2 Future Work and Alternatives

If we were to continue work on this project, we would consider many design alternatives. For the power subsystem, one possible change is the removal of one of the 2.5V regulators. The main reason we had two 2.5V regulators was to isolate the analog to digital converter's voltage reference from the regulator supplying power to the voltage divider circuit, since a change in the resistance of the voltage divider circuit could change the current draw of the regulator, causing a fluctuating voltage, which would also make the reference voltage fluctuate. However, we found out that the voltage fluctuation was rather minimal. Also, it may have been better to have the fluctuating voltage on the same rail as the reference, as the fluctuating voltage of the resistor's rail would be compared to a mimicked fluctuating reference, rather than comparing the fluctuating voltage of the resistor's rail to a constant 2.5V rail.

One design alternative with the control subsystem would be to operate the MCU at 5V. This would allow the clock of the MCU to operate at 16MHz, which would allow faster machine cycles and faster commands. Also, we wouldn't need to change some of the clock division in the software. One issue with operating the MCU at 5V is that the I/O of the ADC is set at 3.3V. For this reason, we would need some voltage shifting logic between the chips to allow interfacing of 5V and 3.3V logic. Also, it is important to note that, if this design alternative were to be implemented, the MCU would have a larger current draw. Another design alternative we could have done is we could've used an STM chip instead of the ATmega32u4. STM chips have more example code and have more versatility compared to the ATmega32u4. Also, the ATmega32u4 chip operates at 3.3V, so the level shifting logic wouldn't need to be present to interface with the selected ADC.

The primary improvement to make in the data reception and visualization subsystem is in collision localization. With a more extensive set of real-world test data, a superior model could be constructed. During the semester, we considered training a machine learning model on known collision data, but the generation of precisely located, reproducible collisions of fixed magnitude proved too big of an obstacle to this approach. Another improvement to the visualization subsystem that can be made is a more advanced localization strategy. With more data, we could construct a spherical interpolation across the (approximately) spherical helmet which should yield better results within a smaller error margin when identifying collision points.

4.3 Ethics and Safety

Given that this device is designed to be used in a high-impact environment, it is critical that it is designed with only the highest standard of safety in mind. The IEEE Code of Ethics states that it is our responsibility "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design" [8]. This is especially relevant since our device will be located within a helmet, very close to the head of the user.

It is important that the physical footprint of the device is as small as possible so that it will not pose any extra risk to the wearer. Existing in-helmet devices like quarterback transceiver modules can serve as a model of safe physical design choices. A small, plastic encasement with additional padding will help to mitigate any risk presented by the device.

Another area for concern is the network of force sensing resistors placed within the helmet. We must ensure the addition of our device does not reduce the safety the football helmet already provides to the user. For this reason, we must ensure all padding is maintained within the helmet. With this, we must also make sure all cables placed within the helmet pose no safety hazards, including extra pressure felt on the head and possibly loose cables.

We also must consider the inherent risk of a battery system. We must consider both battery placement, as well as the durability of the battery. There is a risk of placing a battery so close to the head and in a heavy contact environment. The batteries must be well encased and have no possibility of failing in a dangerous way due to a collision. Per OSHA guidelines [9], potential hazards to lithium batteries include "dropping, crushing, and puncturing," each of which are possible within a football game. The device will also be in the bottom corner of the helmet, somewhere collisions are less frequent, so the circuit and battery will be under reduced stress. We must also ensure there is minimal risk of our battery exploding, as certain lithium batteries have this possibility. According to the OSHA [9], to reduce risk to the user, the batteries must be inspected "for signs of damage, such as bulging/cracking, hissing, leaking, rising temperature, and smoking before use" in case thermal runoff is imminent.

For this reason, we have chosen to use a rechargeable 9V 1300mAh Lithium-Polymer battery. While not as robust as a LiPO4 or NiMH battery, the Lithium-polymer battery does offer some advantages over a traditional Lithium-ion battery. The primary difference is that Lithium-polymer electrolytes are contained in a solid or gel polymer rather than liquid carbonate electrolytes as in a Lithium-ion battery. This comes with many advantages such as low flammability, mechanical stability, and zero electrolyte leakage [10]. To expand further upon our design, we would ideally use a more durable battery chemistry. However, for the scope of this prototype, the safety features of the Lithium-polymer battery should be more than sufficient.

Due to the nature of working with lithium-based battery systems, we have investigated the proper safety precautions for maintenance and use of lithium batteries in an electrical system. For more passive safety precautions, we will be sure to store the battery in dry, cool locations when not in use. It is equally important that it is maintained in good condition even when not in use. Since it is a rechargeable battery, we will ensure to remove it from the power source when it is fully charged. As previously mentioned, the battery has built-in overcharge protection, which should prove to be a safety bonus as well. Before and after running any experiments in which the battery will be used to power the DISH circuit, it will be carefully inspected to ensure that there is no cracking, burn marks or any other sign of external damage. When testing the full system, due to the nature of the product, it will be essential to not place the battery in any location that may put it at greater risk than if the helmet system is being used as intended.

Additionally, the testing process for the DISH helmet has potential risks if not executed properly. Whenever possible, the helmet will be tested with a dummy or model head, not a human head. While the helmets, of course, are designed to protect from collision, there always exists a risk of injury. This is especially the case if the wearer is not properly trained or acclimated to the impacts.

These safety precautions will allow for the rapid and safe development of a system with the potential to create a safer environment for many more athletes in the future.

5 Citation

[1] V. T. Nguyen et al., “Mortality Among Professional American-Style Football Players and Professional American Baseball Players,” JAMA Network Open, vol. 2, no. 5, p. e194223, May 2019, doi: https://doi.org/10.1001/jamanetworkopen.2019.4223.

[2] MaxLinear, “800mA Low Dropout Voltage Regulator,” SPX1117 datasheet, Sept. 2018, https://assets.maxlinear.com/web/documents/sipex/datasheets/spx1117.pdf.

[3] Texas Instruments, “Quad/Octal, Simultaneous Sampling, 16-Bit Analog-to-Digital Converters,” ADS1174/ADS1178 Datasheet, Sept. 2008, https://www.ti.com/lit/ds/symlink/ads1178.pdf.

[4] Digi, “Digi XBee S2C 802.15.4 RF Modules,” XBee S2C Datasheet, https://www.digi.com/resources/library/data-sheets/ds_xbee-s2c-802-15-4.

[5] Atmel, “8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller,” ATmega16U4/ATmega32U4 Datasheet, Apr. 2016, https://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf.

[6] “Electrical Engineering,” grainger.illinois.edu. https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/electrical-engineering.

[7] “Computer Engineering,” grainger.illinois.edu. https://grainger.illinois.edu/academics/undergraduate/majors-and-minors/computer-engineering.

[8] “IEEE Code of Ethics,” IEEE.org, 2023, https://www.ieee.org/about/corporate/governance/p7-8.html.

[9] Occupational Safety and Health Administration, “Preventing Fire and/or Explosion Injury from Small and Wearable Lithium Battery Powered Devices,” June 2019, https://www.osha.gov/sites/default/files/publications/shib011819.pdf.

[10] Z. Chen *et al.*, “4-V flexible all-solid-state lithium polymer batteries,” *Nano Energy*, vol. 64, p. 103986, Oct. 2019, doi: https://doi.org/10.1016/j.nanoen.2019.103986.

Appendix A: Power RV

The power subsystem RV table can be seen in *Table A1*.

Requirements	Verification
• The battery must supply a constant voltage between 6.4V and 10V and supports up to 350mA of current draw.	• Connect wires to the cathode and anode of the battery. Do not solder the wires onto the battery; use a connector. • Connect the wires to an electronic load. • Connect a digital multimeter across the battery in voltage-read mode. • Set the electronic load to operate as a load of 350mA. • Verify that the voltage read across the battery is between 6.4V and 10V while supplying at least 350mA to the electronic load.
• The 1.8V rail must maintain a voltage between 1.65V and 1.95V at a current of at least 17mA.	• Solder a wire on each of the pins of the 1.8V SPX1117 chip. Connect the proper operating capacitors on the input and output nodes. • Connect a power supply across the input voltage and the ground pins. • Connect a digital load across the output voltage and the ground pins, set to a resistance of 115Ω or less to draw at least 17mA. • Connect a digital multimeter in voltage-read mode across the output voltage and the ground pins. • Set the power supply to 9V and power it on. • Verify that the voltage read across the resistive load falls between 1.65V and 1.95V and that the current supplied by the power source is greater than or equal to 17mA.
• The 2.5V rail must maintain a voltage between 2.475V and 2.525V at a current of at least 20mA.	• Solder a wire on each of the pins of the 2.5V SPX1117 chip. Connect the proper operating capacitors on the input and output nodes. • Connect a power supply across the input voltage and the ground pins. • Connect a digital load across the output voltage and the ground pins, set to a resistance of 127Ω or less to draw at least 20mA. • Connect a digital multimeter in voltage-read mode across the output voltage and the ground pins. • Set the power supply to 9V and power it on.

	• Verify that the voltage read across the resistive load falls between 2.475V and 2.525V and that the current supplied by the power source is greater than or equal to 20mA.
• The 3.3V rail must maintain a voltage between 2.7V and 3.6V at a current of at least 220.5mA.	• Solder the 3.3V SPX1117 chip to one of the extra PCBs. Solder the proper operating capacitors on the input and output nodes as well. • Solder a wire on each of the pins of the 3.3V SPX1117 chip. • Connect a power supply across the input voltage and the ground pins. • Connect a digital load across the output voltage and the ground pins, set to a resistance of 16.5Ω or less to draw at least 220.5mA. • Connect a digital multimeter in voltage-read mode across the output voltage and the ground pins. • Set the power supply to 9V and power it on. • Verify that the voltage read across the resistive load falls between 2.7V and 3.6V and that the current supplied by the power source is greater than or equal to 220.5mA.
• The 5V rail must maintain a voltage between 4.75V and 5.25V at a current of at least 91mA.	• Solder a wire on each of the pins of the 5V SPX1117 chip. Connect the proper operating capacitors on the input and output nodes. • Connect a power supply across the input voltage and the ground pins. • Connect a digital load across the output voltage and the ground pins, set to a resistance of 57Ω or more to draw at least 91mA. • Connect a digital multimeter in voltage-read mode across the output voltage and the ground pins. • Set the power supply to 9V and power it on. • Verify that the voltage read across the resistive load falls between 4.75V and 5.25V and that the current supplied by the power source is greater than or equal to 91mA.

Table A1: Power Subsystem RV Table

The remainder of Appendix A denotes the testing procedures performed to verify each test, as well as the direct results of each test.

For each test done with an LDO (except for the 3.3V regulator, more on that later), the required components had leads soldered to wires such that testing can be done more easily. The required

capacitors were also applied at the proper locations. The breadboarded setup can be seen in *Figure A1*.

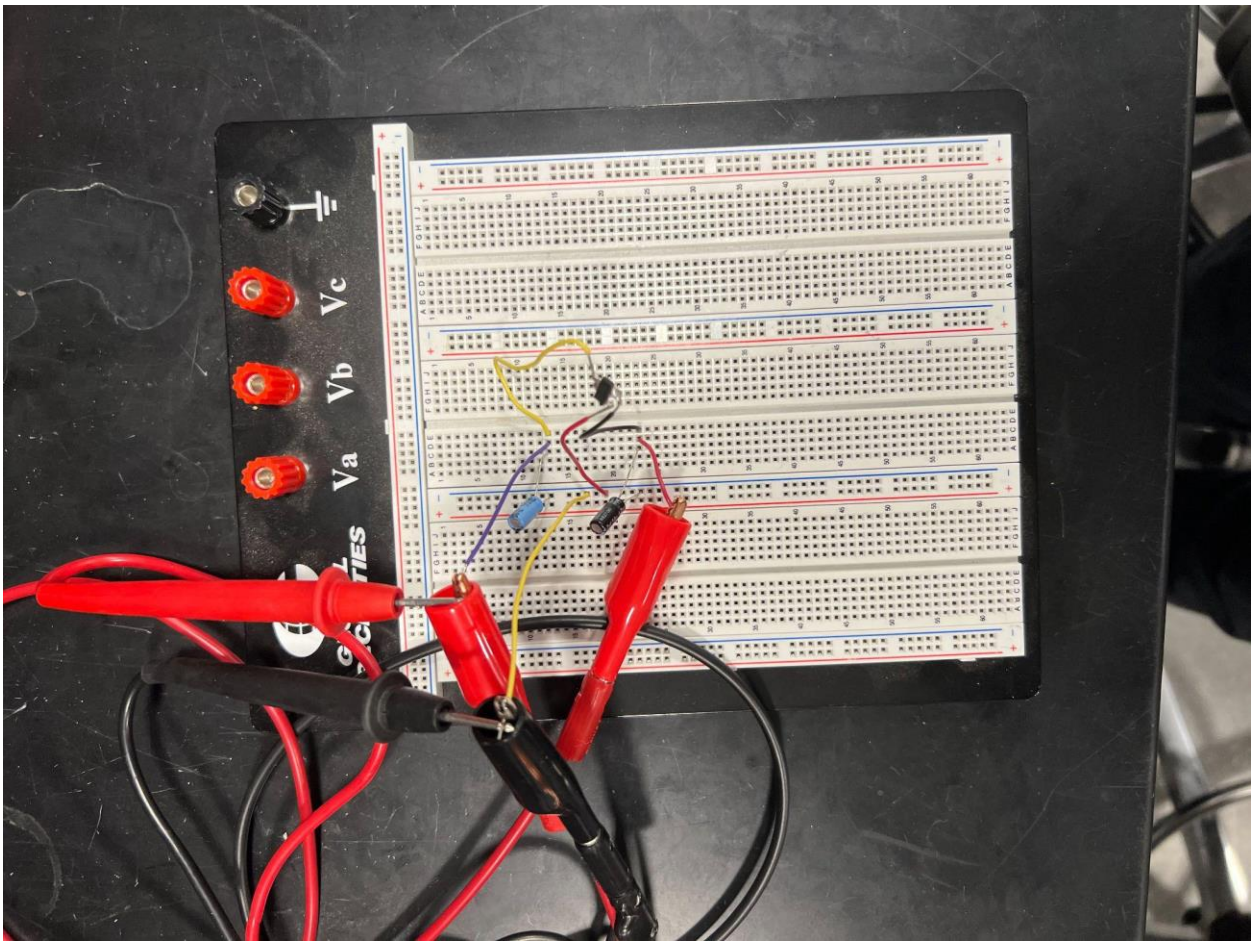

Figure A1: Breadboarded setup of LDO voltage regulator testing.

Requirement: The battery must supply a constant voltage between 6.4V and 10V and supports up to 350mA of current draw.
Result: PASS

First, the battery was charged up until the green light turned on, indicating it was fully charged. Next, the 9V battery was connected to a connector that allows breadboard or alligator clip connections. The electronic load was then turned on such that it drew 350mA. Next, the battery was connected to the electronic load, and a multimeter was connected across the battery. The electronic load was then turned on. To get the right current, constant resistance mode was used to play it safe, and slowly increase the current. This setup can be seen in *Figure A2*.

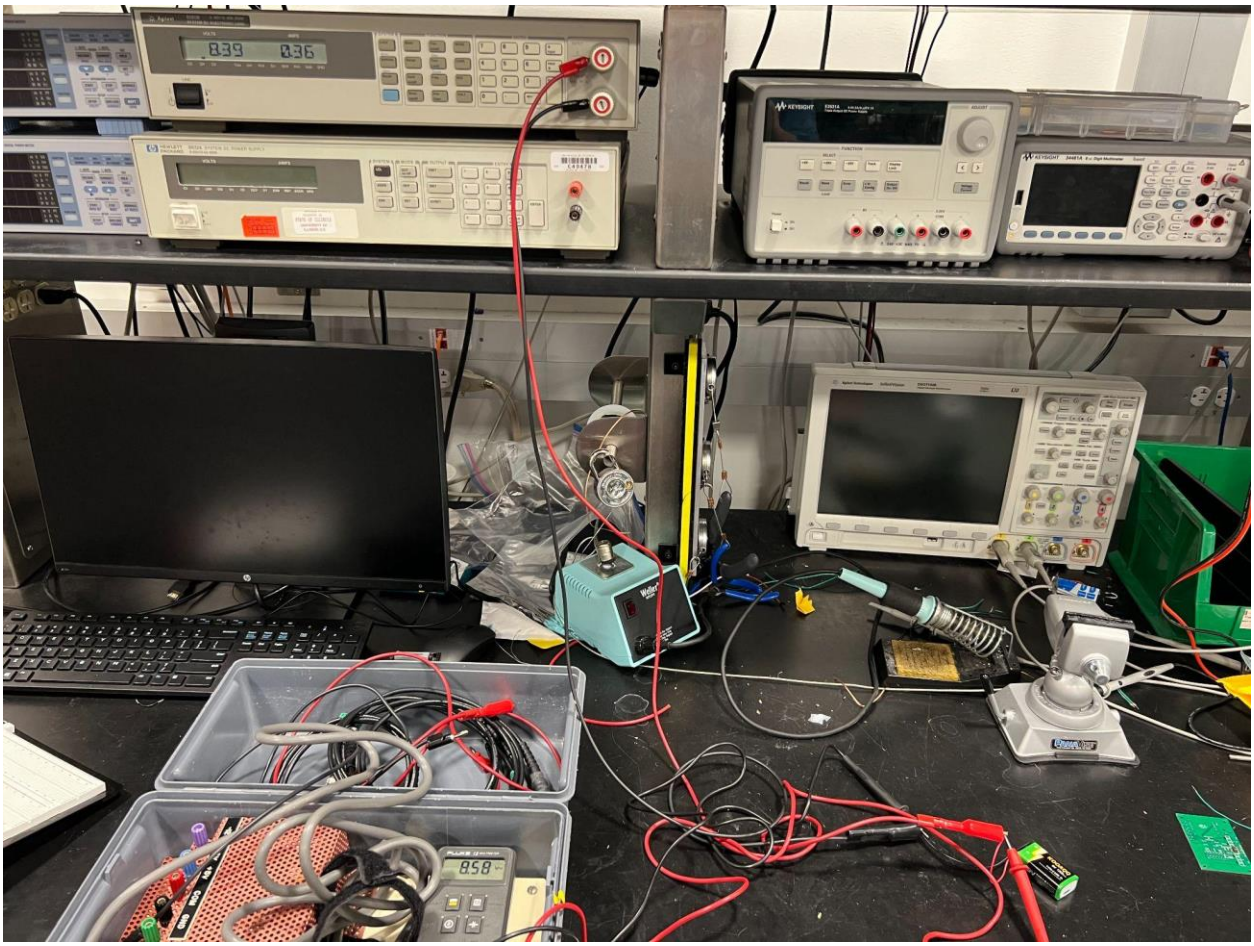

Figure A2: Setup for the battery test.

The results of this test can be seen in *Figure A3*.

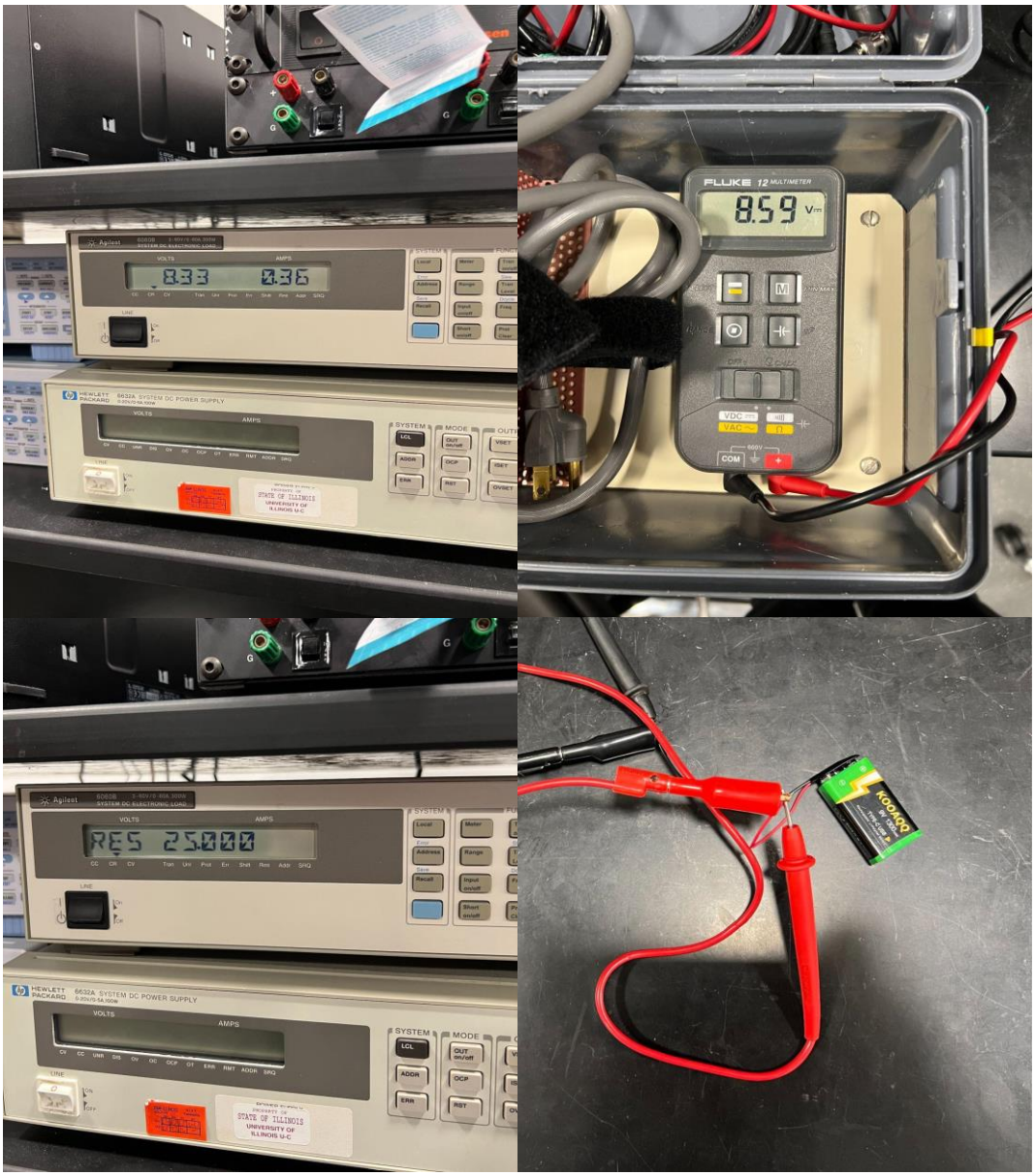

Figure A3: Battery test readings. Top left is electronic load readings. Top right is the battery voltage. Bottom left is electronic load resistance. Bottom right is the testing setup.

As seen in *Figure A3*, this setup worked for a load of 25Ω, which drew 360mA. This test appears to be a success. As a further test, we allowed the battery to remain at this power draw for a few minutes, not measuring the time. We ensured it works for at least a few minutes at this current draw. It was operational for about 5 minutes and remained around 8.58V with no fluctuation. This is well within the allowed voltage range, and the battery drew more than the expected maximum current. One thing of note is that the battery voltage was around 8.6V rather than advertised 9V. This is likely due to the large power draw; however, it is probably fine. With the results shown in *Figure A3*, this test passed.

Requirement: The 1.8V rail must maintain a voltage between 1.65V and 1.95V at a current of at least 17mA.
Result: PASS

The digital load was first turned on to constant resistance mode with 115Ω. The 1.8V regulator was then connected to the power supply set to 9V, which was then turned on. The overall setup can be seen in *Figure A1*. The results for the 1.8V regulator can be seen in *Figure A4*.

Figure A4: 1.8V regulator testing readings. The top left is the electronic load readings. The top right is the power source readings. The bottom left is the electronic load resistance. The bottom right is the voltage read across the Vout and ground pins of the SPX1117.

As seen in *Figure A4*, all levels were within the desired ranges. It supported 20mA, which means a lower current of 17mA can also be supported. This signifies a success, and the result of this test is a pass.

Requirement: The 2.5V rail must maintain a voltage between 2.475V and 2.525V at a current of at least 20mA.
Result: PASS

The digital load was first turned on to constant resistance mode with 127Ω. The 2.5V regulator was then connected to the power supply set to 9V, which was then turned on. The overall setup can be seen in *Figure A1*. The results of the 2.5V regulator test can be seen in *Figure A5*.

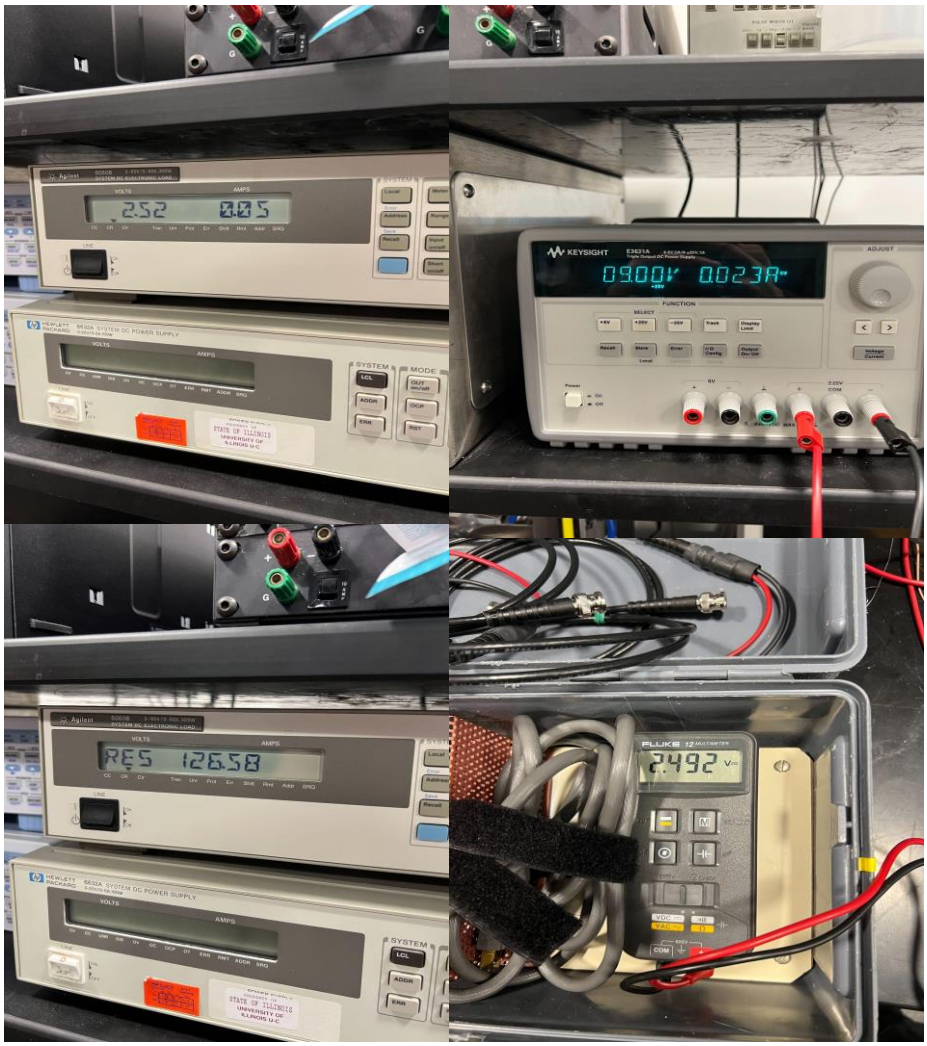

Figure A5: 2.5V regulator readings. The top left is the electronic load readings. The top right is the power source readings. The bottom left is the electronic load resistance. The bottom right is the voltage read across the Vout and ground pins of the SPX1117.

As seen in *Figure A5*, all the values were within the required ranges. The system supported 23mA, which is greater than the 20mA requirement. This indicates a successful test, and the result of this test is a pass.

Requirement: The 3.3V rail must maintain a voltage between 2.7V and 3.6V, at a current of at least 220.5mA.
Result: PASS

First, the electronic load was turned on to 16.5Ω. The 3.3V regulator was then connected to the power supply set to 9V, which was then turned on. The setup used for this test was initially the same one as shown in *Figure A1*. It successfully supplied 0.202A from the power supply, and 0.22A from the digital load. However, one of these had too low of current. So, the resistance was decreased to 15Ω to increase the current draw. This worked for a second, but then the current started dropping rapidly. The 3.3V LDO chip began to overheat. This indicates a failure in the initial testing. The system was operational at lower currents, at around say, 100mA, but failed at higher currents. The most likely reason as to why this test failed was because the LDO was suspended in the air, and the 46degC/W is likely calculated when the regulator is soldered to pads on a PCB.

So, this test was retried by soldering the 3.3V SPX1117 chip to one of the extra PCBs, along with SMD versions of the capacitors. A wire was connected to each pin on the board directly as well; this allowed for better thermal dispersion in the metal of the board compared to the air.

Attached to the board, with the electronic load set to 14.9Ω (to get 0.221A from power supply), the test is reperformed. This new setup can be seen in *Figure A6*.

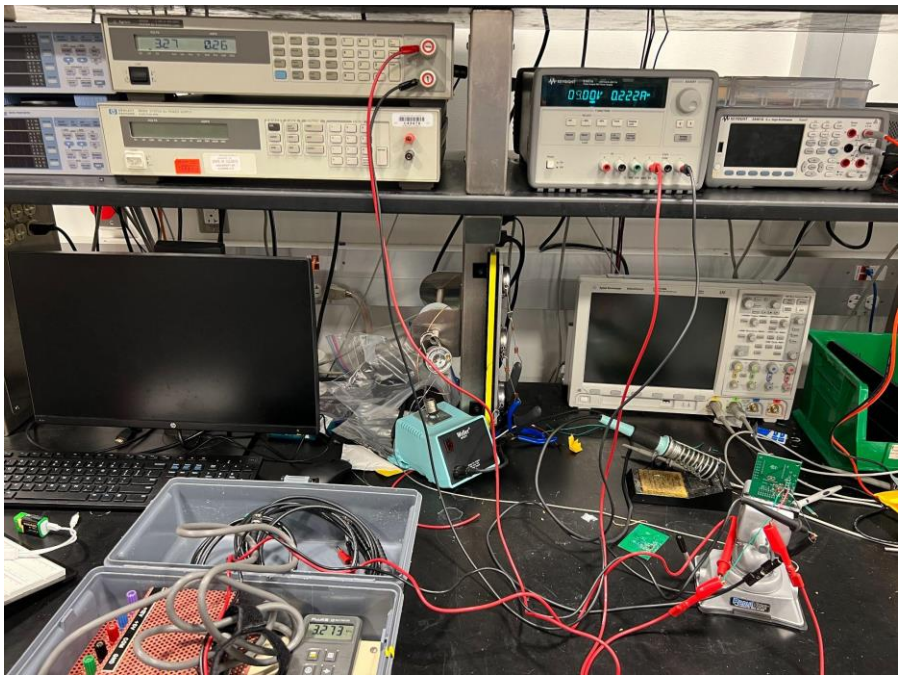

Figure A6: The renovated 3.3V regulator testing setup.

The results of this test can be seen in *Figure A7*.

Figure A7: 3.3V regulator readings. The top left is the electronic load readings. The top right is the power source readings. The bottom left is the electronic load resistance. The bottom right is the voltage read across the Vout and ground pins of the SPX1117.

As seen in *Figure A7*, the voltages and currents were within their desired ranges. Also, the regulator was able to maintain this power draw for a sustained period. This testing was successful, and the result is a pass.

Requirement: The 5V rail must maintain a voltage between 4.75V and 5.25V at a current of at least 91mA.
Result: PASS

The digital load was first turned on to constant resistance mode with 57Ω. The 5V regulator was then connected to the power supply set to 9V, which was then turned on. The overall setup can be seen in *Figure A1*. The results from this test can be seen in *Figure A8*.

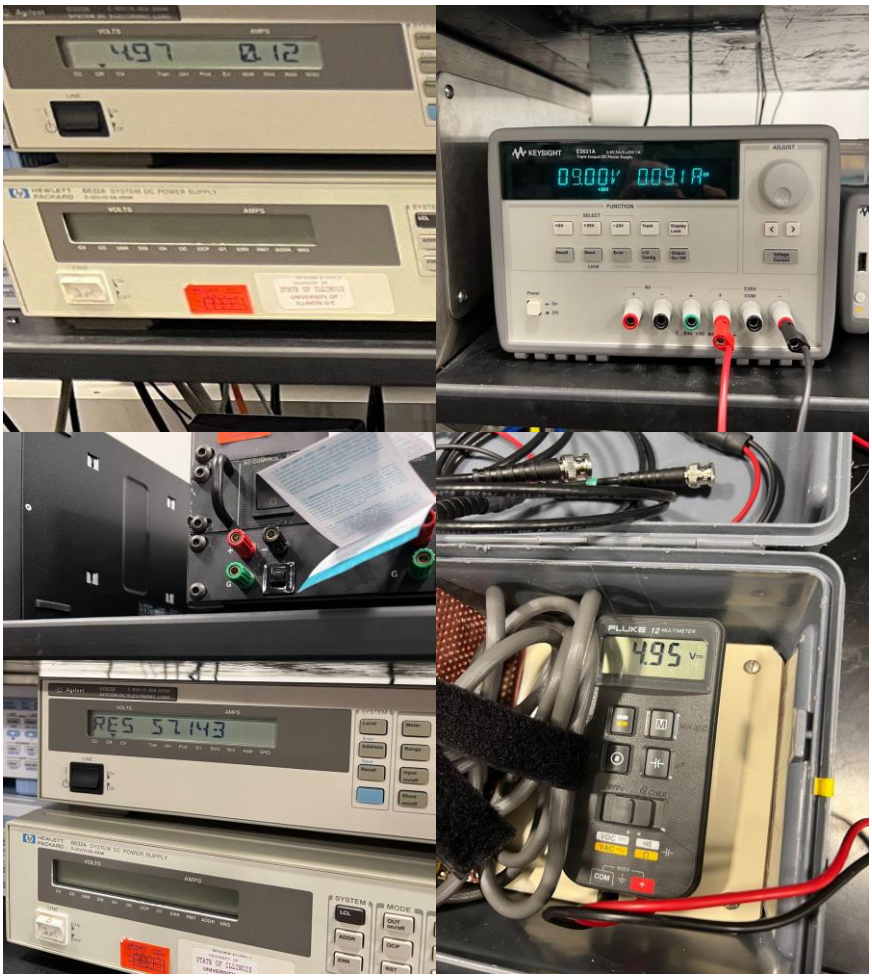

Figure A8: 5V regulator readings. The top left is the electronic load values. The top right is the power source. The bottom left is the resistance the electronic load was set to. The bottom right is the reading across the Vout and ground pins of the SPX1117.

As seen in *Figure A8*, the voltage measured in the digital load was 4.97V and the current measured in the power supply was 91mA. The digital multimeter measured an output of 4.95V. The digital load measured a load of 120mA. These are all within the desired ranges, signifying a success. The result of this testing is a pass.

Appendix B: Control RV

The control subsystem RV table can be seen in *Table B1*.

Requirements	Verification
• The microcontroller must be able to comprehend data from the analog to digital converter via SPI over 90% of the time.	• Construct a completed DISH without putting the circuit inside the helmet or plugging in the force sensing resistor array connector. • Program the microcontroller to communicate with the ADC via SPI to read the data from all eight data channels, and to print the data to the serial monitor. • Connect the board to a power supply using the connector. Plug the USB connector into the board and to a computer with the Arduino IDE opened and the serial monitor opened. • Turn on the power supply to 9V and set the port in the Arduino IDE to the port being used by the DISH. • Verify that data is being shown in the serial monitor. The digital values represented in each channel should be around hex 7FFF, or decimal 32768. Verify the data is shown at a frequency greater than 90% of the time.
• The microcontroller must be able to correctly identify when a collision occurs with more than 90% accuracy.	• With a fully constructed DISH helmet, connect a power supply to the board using the connector. Connect a USB cable to the DISH and to a computer with the Arduino IDE opened and the serial monitor opened. • Program the microcontroller to operate as a completed DISH system, but instead of sending the data via Zigbee, to return the data in the serial monitor with a notification that a collision occurs. • Ensure something is inside the helmet to apply a force on the other side of the FSRs. With a hammer, tap where one of the force sensing resistors are on the outside of the helmet. • Referring to the serial monitor, verify that when the helmet is hit, 90% of the time, the collision is detected, and a notification is shown in the serial monitor.

• The microcontroller must be able to send and receive signals to and from the remote receiver via the Zigbee transmitter more than 90% of the time.	• Fully construct a DISH helmet. There is no need to connect the force sensing resistor array for this testing. • Connect the other Zigbee module to an Arduino and a laptop. • Program the microcontroller to send a known-test signal periodically via Zigbee. Program the Zigbee-Arduino setup to receive the Zigbee signal and print it to the Arduino console. • Send a test signal from the DISH to the Zigbee-Arduino receiver. • Verify that the intended Zigbee signal matches the data shown in the Arduino console.
• The off-chip simultaneous sampling ADC must be able to measure all signals from the force sensing resistor array concurrently more than 90% of the time.	• Fully construct a DISH helmet. Do not connect the force sensing resistor array for this testing. • Program the microcontroller to read the data from the analog to digital converter normally, except have it print all data to the serial monitor. • Connect a power supply to the DISH using the connector. Connect the DISH to a computer with the Arduino IDE opened with the serial monitor opened. • Power on the power supply to 9V. Select the port being used by the DISH. • Verify that data is being shown in the serial monitor, and that the data shown is visible to be from each of the eight channels. With this test, it is expected that the digital values represented in each channel should be around hex 7FFF, or decimal 32768. Verify the data is shown from all eight channels at a frequency greater than 90% of the time.
• The Zigbee transmitter must be able to properly send a signal up to 40 meters away.	• Connect each of the Zigbee modules to an Arduino and a laptop. • Create Arduino code for one Zigbee-Arduino setup to act as a transmitter and the other to act as a receiver. • Send a test signal from the transmitter to the receiver at a close range to ensure proper operation of the transmission. • Move the Zigbee-Arduino setups such that they are exactly 40m away. Measure this distance using a tape measure, or a similar tool. • Test the test signal once more, and verify that the

	transmitted signal is received.
• The signal must be able to be sent through the helmet and through adverse conditions with an accuracy of over 90%.	• Connect each of the Zigbee modules to an Arduino and a laptop. • Create Arduino code for one Zigbee-Arduino setup to act as a transmitter and the other to act as a receiver. • Send a test signal from the transmitter to the receiver at a close range in open air to ensure proper operation of the transmission. • Place a water bottle divider between the Zigbee-Arduino setups to simulate rain. • Test the signal again and verify that the transmitted signal is received. • Move one Zigbee-Arduino setup behind a wall. • Test the signal once more and verify that the transmitted signal is received.

Table B1: Control Subsystem RV Table

The remainder of Appendix B denotes the testing procedures performed to verify each test, as well as the direct results of each test.

Throughout testing requirements #5 and #6, the hexadecimal test signal to be transmitted is:

7E016111111064697368656C6D123456789ABE

The payload of this packet is:

0x64 0x69 0x73 0x68 0x65 0x6C 0x6D 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A

The first byte is for player number and team identification, and then every two bytes after represents one sensor values' data, starting from sensor 1 to sensor 8.

Requirement: The microcontroller must be able to comprehend data from the analog to digital converter via SPI over 90% of the time.
Result: PASS

For this test, the DISH was programmed to read the data from the ADC, and then after each read, it would print the data to the Arduino's serial monitor. Data channels 2-8 were set floating, while a 10kΩ potentiometer was placed at data channel 1. The setup can be seen in *Figure B1*.

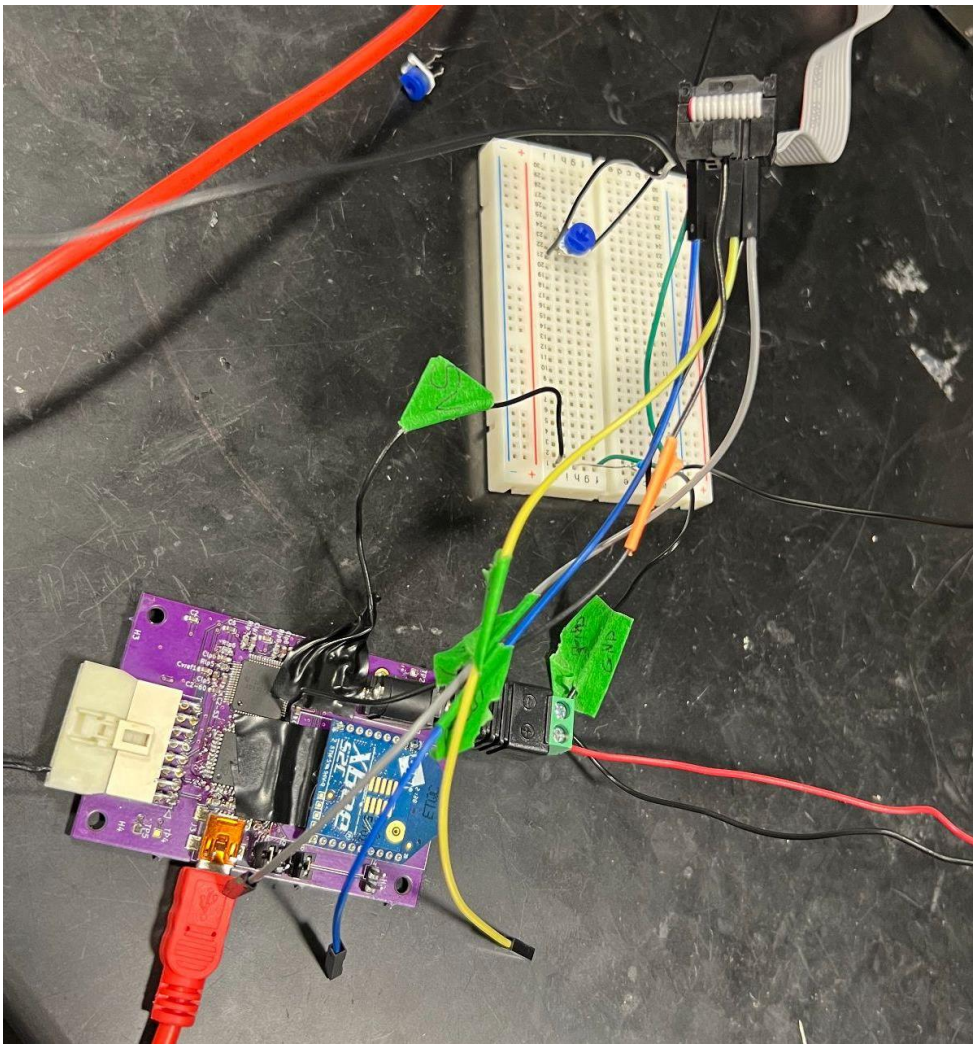

Figure B1: ADC and SPI communication test setup.

As seen in *Figure B2*, the ADC was able to read changing resistances from channel 1, while channels 2-8 were held constant for the most part.

```
22:38:16.991 -> ADC Output: [118, 32341, 32340, 32339, 32346, 32338, 32355, 32346, ]
22:38:16.991 -> ADC Output: [119, 32341, 32340, 32339, 32346, 32338, 32356, 32346, ]
22:38:16.991 -> ADC Output: [119, 32341, 32340, 32339, 32345, 32338, 32356, 32346, ]
22:38:16.991 -> ADC Output: [119, 32341, 32340, 32339, 32256, 30590, 21886, 21374, ]
22:38:16.991 -> ADC Output: [119, 32341, 32341, 32339, 32346, 32338, 32356, 32346, ]
22:38:16.991 -> ADC Output: [119, 32342, 32340, 32339, 32346, 32338, 32356, 32345, ]
22:38:16.991 -> ADC Output: [119, 32341, 32340, 32339, 32346, 32338, 32355, 32345, ]
22:38:16.991 -> ADC Output: [119, 32340, 32339, 32338, 32345, 32338, 32356, 32346, ]
22:38:17.022 -> ADC Output: [119, 32341, 32340, 32339, 32346, 32338, 32355, 32345, ]
22:38:17.022 -> ADC Output: [119, 32341, 32256, 30590, 21886, 21630, 21374, 23166, ]
22:38:17.022 -> ADC Output: [119, 32341, 32339, 32339, 32346, 32338, 32355, 32346, ]
22:38:17.022 -> ADC Output: [119, 32341, 32340, 32339, 32346, 32338, 32355, 32345, ]
22:38:17.022 -> ADC Output: [119, 32342, 32341, 32339, 32346, 32338, 32355, 32345, ]
22:38:17.022 -> ADC Output: [119, 32341, 32340, 32339, 32346, 32338, 32356, 32345, ]
```

```
22:38:11.262 -> ADC Output: [27305, 32342, 32341, 32340, 32347, 32338, 32356, 32347, ]
22:38:11.262 -> ADC Output: [27306, 32342, 32341, 32340, 32347, 32338, 32356, 32346, ]
22:38:11.262 -> ADC Output: [27306, 32342, 32341, 32340, 32347, 32339, 32355, 32346, ]
22:38:11.262 -> ADC Output: [27306, 32342, 32341, 32340, 32347, 32339, 32355, 32345, ]
22:38:11.262 -> ADC Output: [27306, 32342, 32341, 32340, 32346, 32339, 32355, 32346, ]
22:38:11.262 -> ADC Output: [27306, 32343, 32341, 32341, 32347, 32339, 32356, 32346, ]
22:38:11.262 -> ADC Output: [27306, 32342, 32340, 32340, 32347, 32339, 32356, 32347, ]
22:38:11.262 -> ADC Output: [27307, 32342, 32341, 32340, 32347, 32339, 32356, 32345, ]
22:38:11.262 -> ADC Output: [27307, 32343, 32341, 32340, 32347, 32339, 32356, 32345, ]
22:38:11.291 -> ADC Output: [27306, 32342, 32341, 32340, 32347, 32339, 32355, 32345, ]
22:38:11.291 -> ADC Output: [27306, 32342, 32341, 32339, 32346, 32339, 32354, 32344, ]
22:38:11.291 -> ADC Output: [27306, 32343, 32340, 32340, 32347, 32339, 32356, 32347, ]
22:38:11.291 -> ADC Output: [27306, 32342, 32341, 32341, 32346, 32339, 32354, 32345, ]
22:38:11.291 -> ADC Output: [27307, 32343, 32341, 32340, 32347, 32339, 32355, 32345, ]
22:38:11.291 -> ADC Output: [27306, 32342, 32341, 32339, 32346, 32338, 32356, 32346, ]
22:38:11.291 -> ADC Output: [27306, 32343, 32341, 32340, 32347, 32339, 32356, 32346, ]
22:38:11.291 -> ADC Output: [27306, 32341, 32340, 32340, 32347, 32339, 32356, 32346, ]
22:38:11.291 -> ADC Output: [27307, 32343, 32342, 32341, 32348, 32340, 32356, 32346, ]
22:38:11.316 -> ADC Output: [27306, 32343, 32341, 32340, 32347, 32340, 32356, 32346, ]
```

Figure B2: The top image is when the potentiometer is set to a low resistance. The bottom image is when the potentiometer is set to a high resistance.

Because the data shown in *Figure B2* is variable as the potentiometer is changed, and the rate of successful reading is 100%, the result of this test is a pass.

Requirement: The microcontroller must be able to correctly identify when a collision occurs with more than 90% accuracy.
Result: PASS

A video of proper operation can be seen at the following link:

https://www.youtube.com/embed/zYbCbe7iI3s

As proven by the fact that the DISH is fully operational, the result of this subsystem requirement is a pass.

Requirement: The microcontroller must be able to send and receive signals to and from the remote receiver via the Zigbee transmitter more than 90% of the time.
Result: PASS

For this test, the DISH was set up and connected to the force sensing resistor array. This testing setup can be seen in *Figure B3*.

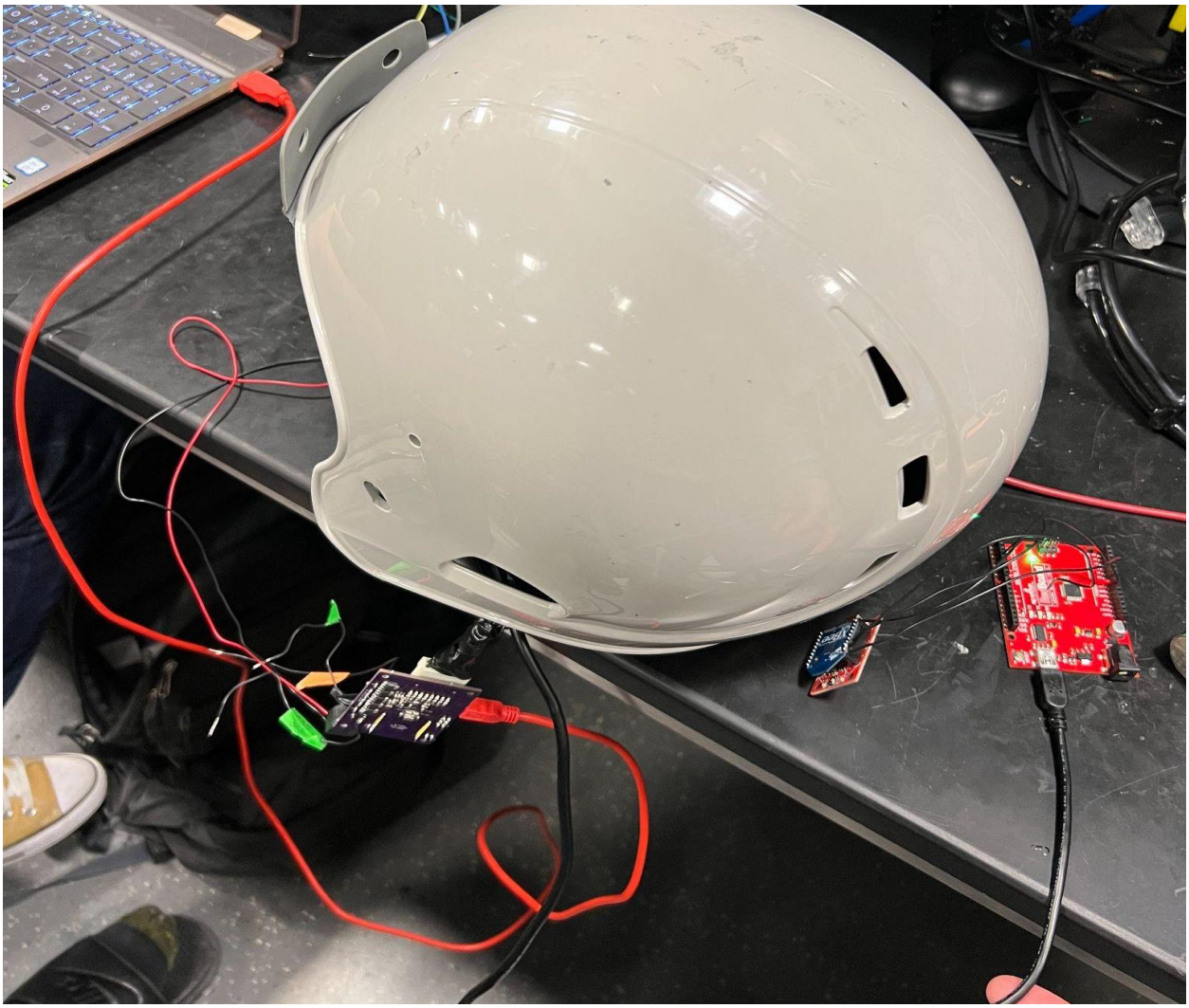

Figure B3: Zigbee transmission capabilities setup.

The data both sent and received is visible in *Figure B4*.

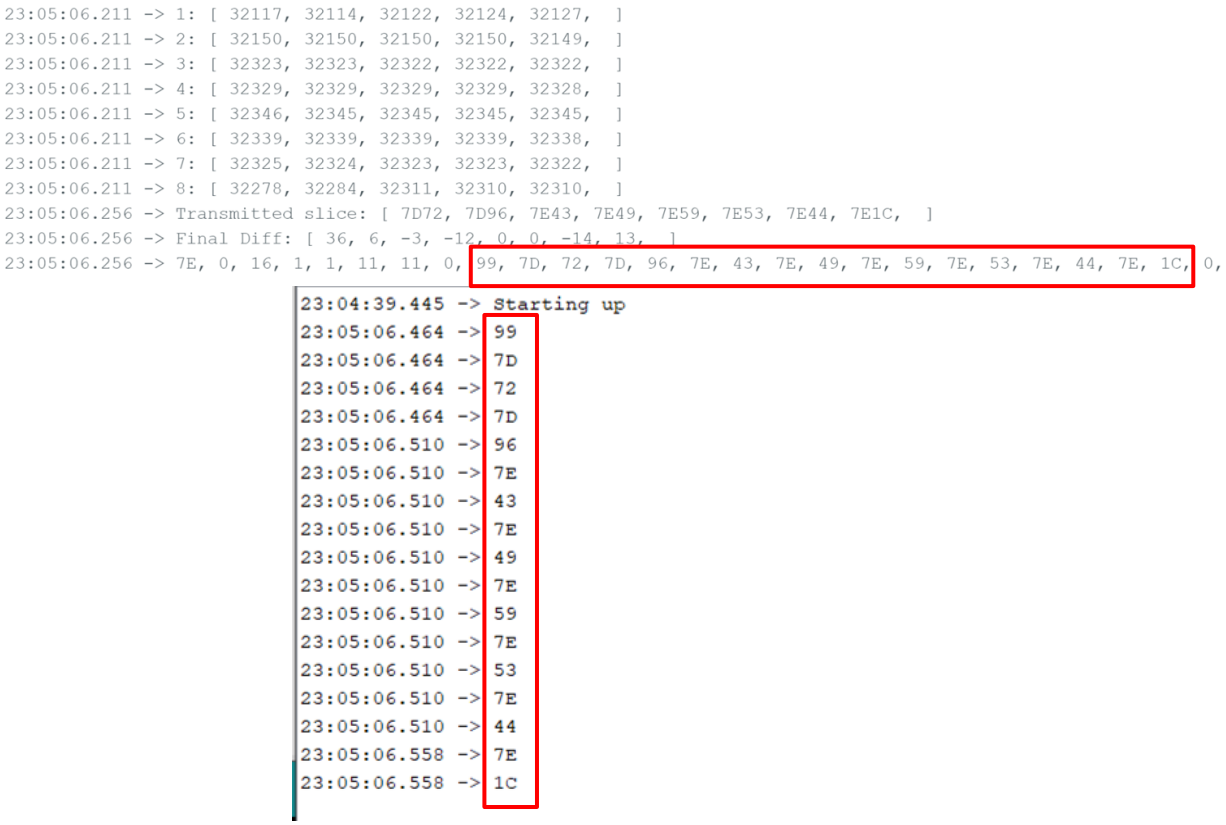

Figure B4: The top image is the data transmitted from the transmitter. The bottom image is the data received by the receiver.

As seen in *Figure B4*, the data is properly transmitted, as the packets received by the receiver line up with those sent from the transmitter. The result of this test is a pass.

Requirement: The off-chip simultaneous sampling ADC must be able to measure all signals from the force sensing resistor array concurrently more than 90% of the time.
Result: PASS

For this test, the PCB was programmed with the proper code to operate as intended. On collision, the processor was programmed to print out the data to be selected where the real, transmitted data is found. From the data seen in *Figure B5*, it can be seen that all of the force sensing resistors in the array are measured concurrently.

```
01:03:42.590 -> Baseline: [31977, 32342, 32321, 32317, 32344, 32338, 32299, 32325, ]
01:03:42.590 -> Calibration Complete
01:03:45.353 -> flag2 triggered
01:03:45.353 -> Primary Sensor: 3
01:03:45.353 -> min_Val: 32285
01:03:45.353 -> 1: [ 32048, 32051, 32050, 32049, 32048, 32053, 32056,  ]
01:03:45.353 -> 2: [ 32342, 32343, 32343, 32343, 32342, 32342, 32342,  ]
01:03:45.384 -> 3: [ 32296, 32285, 32285, 32285, 32287, 32297, 32306,  ]
01:03:45.384 -> 4: [ 32308, 32298, 32297, 32296, 32295, 32296, 32298,  ]
01:03:45.384 -> 5: [ 32342, 32343, 32343, 32342, 32343, 32340, 32335,  ]
01:03:45.384 -> 6: [ 32336, 32340, 32340, 32340, 32340, 32338, 32337,  ]
01:03:45.384 -> 7: [ 32315, 32320, 32321, 32321, 32321, 32322, 32321,  ]
01:03:45.384 -> 8: [ 32329, 32297, 32298, 32301, 32303, 32309, 32316,  ]
01:03:45.384 -> Transmitted slice: [ 7D33, 7E57, 7E1D, 7E2A, 7E57, 7E54, 7E40, 7E29,  ]
```

Figure B5: Simultaneous data collected from the FSR array using the ADC.

As seen in *Figure B5*, the data is measured as intended. Because of this, the result of this test is a pass.

Requirement: The Zigbee transmitter must be able to properly send a signal up to 40 meters away.
Result: PASS

For this test, the sender and receiver were placed a significant distance apart to test for accuracy at large distances. For this test, the long corridor in the ECEB was used, namely, the corridor comprising corridors C2000, C2001, and C2002. Each Arduino-Zigbee setup was placed on either end. As seen in *Figure B6*, the setups are separated by a large distance. This distance is measured to be greater than 40 meters, so if this test passes, the 40 meter requirement will also pass, as any distance shorter than the tested distance must also pass as well.

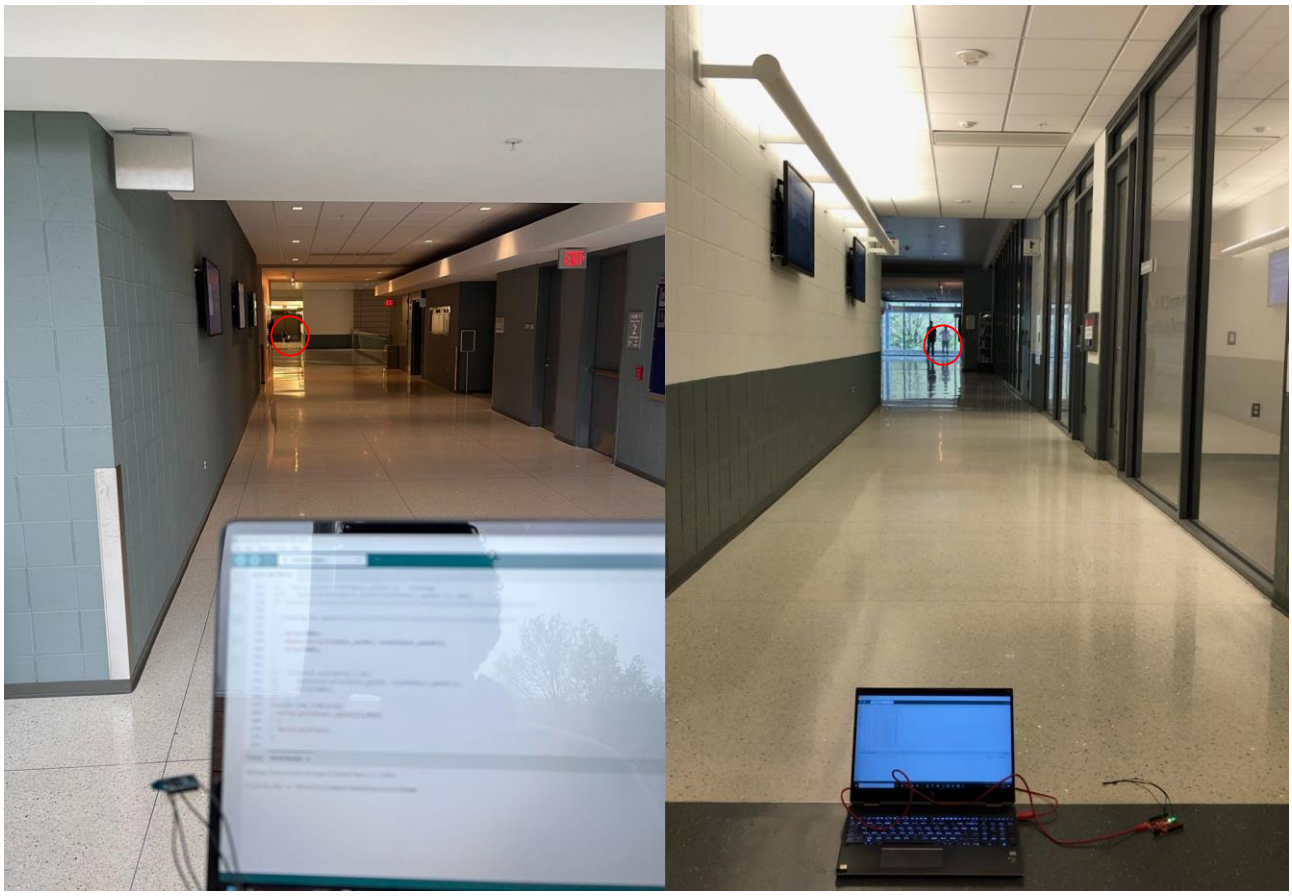

Figure B6: The image on the left is from the transmitter's point of view, where the red circle shows where the receiver is. The image on the right is from the receiver's point of view, where the red circle shows where the transmitter is.

Figure B7 shows the data transmitted and received during the 40 meter distance test.

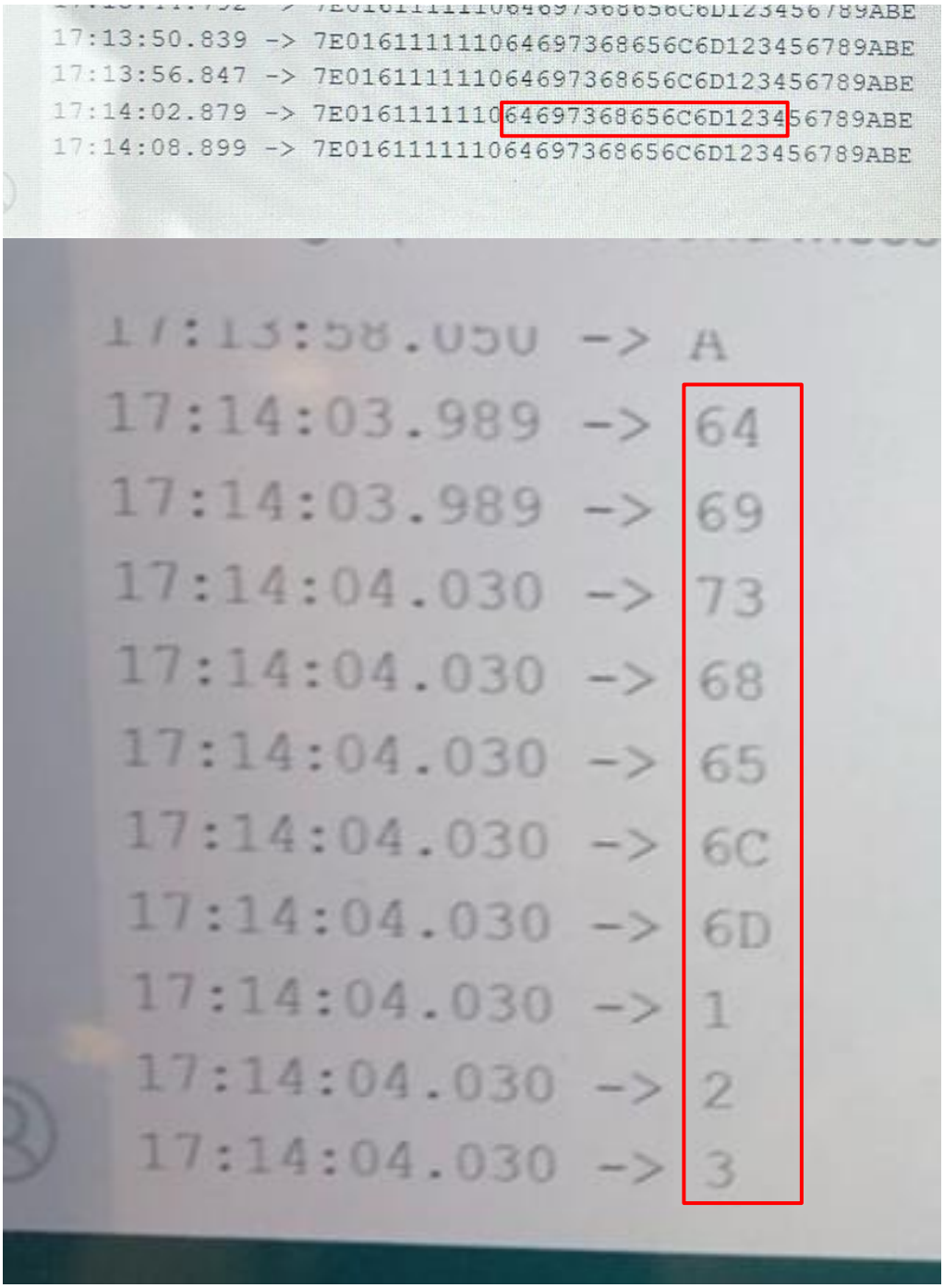

Figure B7: 40m test data. The top image shows the transmitter's serial monitor, showing what is sent. The bottom image shows what is received by the receiver. It can be seen that the data packets are correctly transmitted, as the packets in the boxes match.

As seen in *Figure B7*, the data is properly sent over this distance larger than 40 meters. This says that the setup is operational at this 40 meter benchmark, signifying a successful test. The result of this test is a pass.

Requirement: The signal must be able to be sent through the helmet and through adverse conditions with an accuracy of over 90%.
Result: PASS

The first test that was performed was a test through the wall. For this, one Zigbee-Arduino combo was set up as a sender, the other was set up as a receiver. The sender was in ECEB room 2070 at workstation N, and the receiver was in ECEB room 2072, as seen in *Figure B8*.

Figure B8: The left image shows the transmitter set up in room 2072. The right image shows the receiver set up in room 2070. This is the overall setup of the wall test.

When both setups are turned on, the correct test signal is both sent and received, as seen in *Figure B9*.

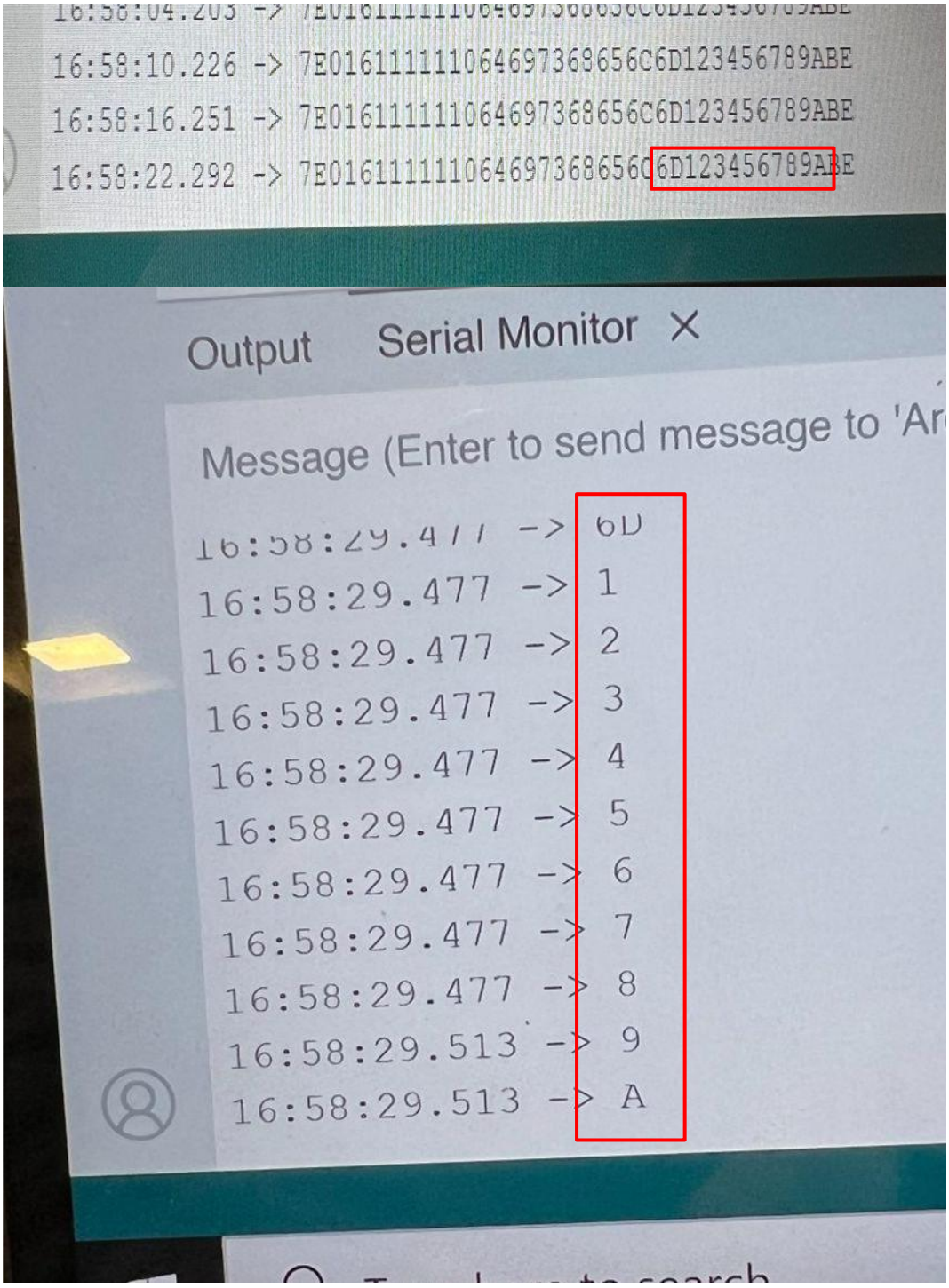

Figure B9: The top image shows the transmitter, which is printing the packets to be sent in the wall test. The bottom image shows the receiver, which is printing the packets that are received in the wall test. The packets line up, as the packets in the boxes match.

As seen in *Figure B9*, there are no errors in the packets transmitted, so this portion of the test passes.

Next, the water bottle test is performed. This setup is shown in *Figure B10*, where the sender and receiver are placed in close proximity with a wall of water bottles acting as a separator.

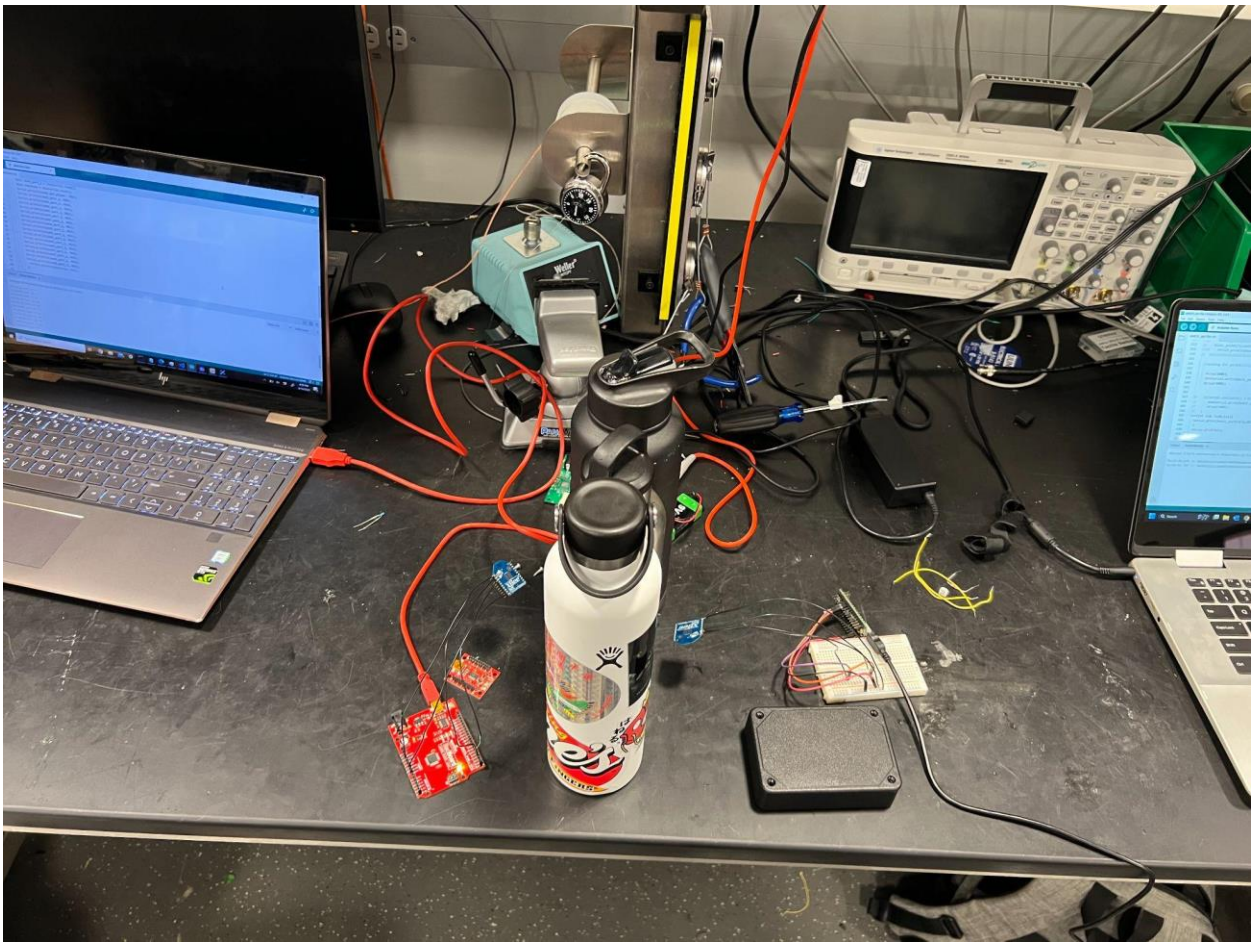

Figure B10: Water bottle test setup.

The results from this test are shown in *Figure B11*.

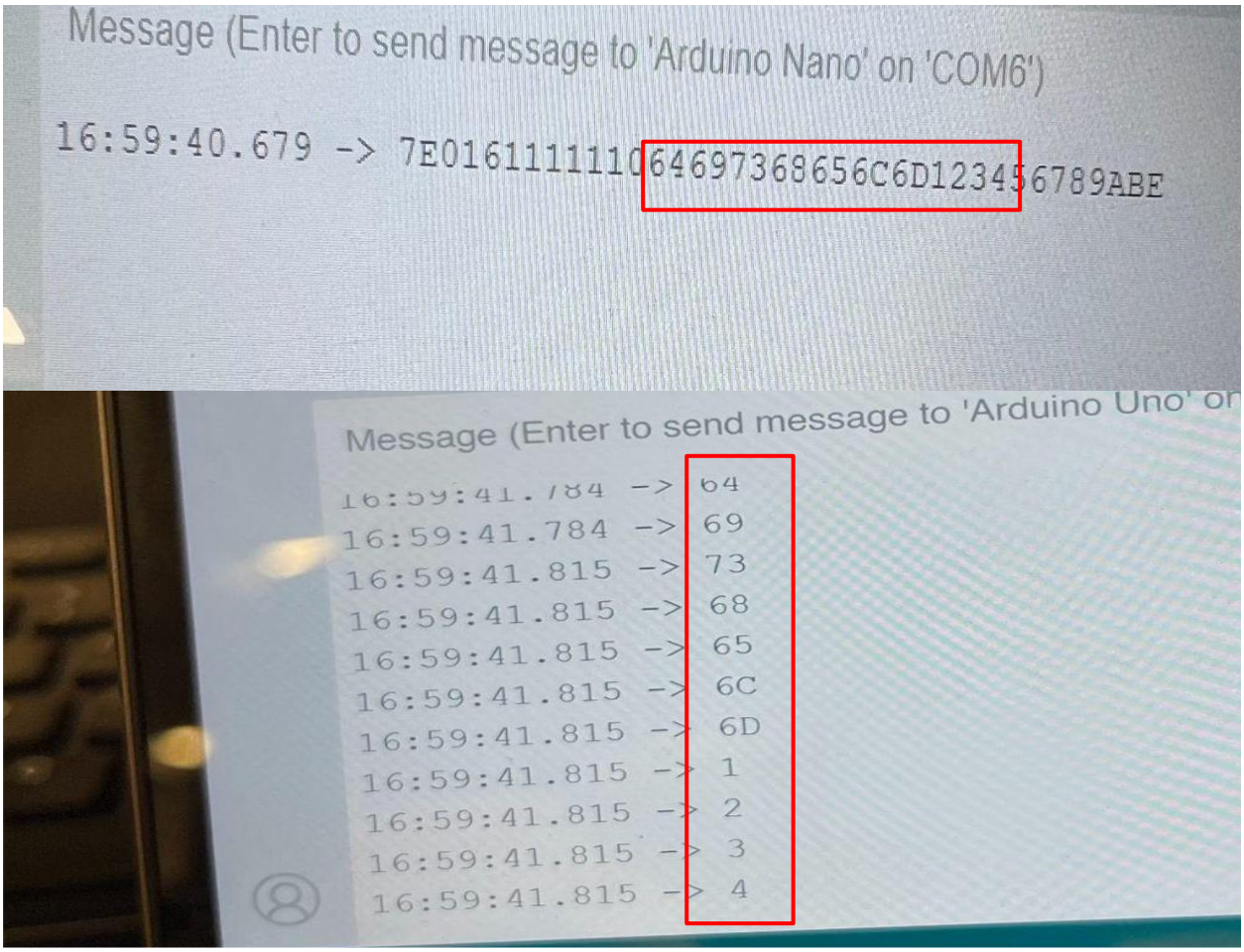

Figure B11: The top image shows the packets to be sent from the transmitter in the water bottle test. The bottom image shows the packets received by the receiver in the water bottle test. It can be seen that these line up correctly, as the packets in the boxes match.

As seen in *Figure B11*, the packets are properly transmitted through the wall of water bottles. This portion of the test passes as well.

Because both the wall test and the water bottle test were successful, this requirement can be considered verified. The result of this test is a pass.

Appendix C: Data Reception and Visualization RV

The data reception and visualization subsystem RV table can be seen in *Table C1*.

Requirements	Verification
• The receiver must receive packets sent with a maximum error rate of 10%.	• Connect each of the Zigbee modules to an Arduino and a laptop. • Create Arduino code for one Zigbee-Arduino setup to act as a transmitter and the other to act as a receiver. • Program the transmitter to send a fixed series of packets in the standard Zigbee format, and the receiver to print the received packets to the serial monitor. • Compare the transmitted and received signals, verifying that the error rate is less than 10%.
• The signal received by the Arduino must be delivered to the serial input exactly as it is seen by the hardware.	• Connect an Arduino to a computer. Program it to send fixed packets to the computer over the serial port. • Create a Python script that takes the data from the Arduino and displays it. • Read the data saved by the Python script.. • Verify that the data is exactly the same as the data that was programmed into the Arduino.
• The visualizer must be able to identify the impact within 20% of where it occurred.	• Set up the visualizer code to operate off of an internal input rather than the Arduino receiver. • Give the visualizer simulated force data that would return a recognizable output, namely between three sensors. • Begin the visualizing software and have it visualize the simulated data. • Ensure that the calculated impact location is within 20% of the expected location.
• The data must be visualized within 30 seconds of packet reception.	• Set up the Arduino receiver to send fixed packets to the visualization code. Program it to turn on an LED when it sends the data to the COM port. • Plug in the Arduino. Start a timer when the LED turns on. • Run the visualization software when the data has been transmitted. • Ensure that all times between data reception and displaying on the screen are less than 30s.

Table C1: Reception and Visualization Subsystem RV Table

The remainder of Appendix C denotes the testing procedures performed to verify each test, as well as the direct results of each test.

Requirement: The receiver must receive packets sent with a maximum error rate of 10%.
Result: PASS

Other tests indicate there are no errors in packet reception. Additionally, the completed product demonstrates no packet losses. Hence, the result of this requirement is a pass.

Requirement: The signal received by the Arduino must be delivered to the serial input exactly as it is seen by the hardware.
Result: PASS

For this test, an Arduino was plugged into a computer, and a script in Python, which is the language being used to port data from the COM port connected to the Arduino with the visualizing software, was written. The Arduino is programmed to continually send data to the serial monitor, with the following code written in the loop() function:

Serial.println(i);
i++;
delay(500);

The purpose of this code is to print to the COM port a number, which increases by one, every half-second. The Python script contains the following code to read the data from the COM port:

```
ser = serial.Serial('COM5', 9600)
for i in range(20):
    id = int(ser.readline().decode().rstrip())
    print(id)
ser.close()
```

The purpose of this code is to read the data in the COM port, and print out the first twenty data points received. As seen in *Figure C1*, the data is properly received.

Figure C1: The left image is the data being sent from the Arduino. The right image is the data as received by the COM port in the Python script.

As seen in *Figure C1*, the data from the Arduino, which is the same hardware as the receiver, can successfully send its data to the computer's serial input and makes its way to the Python script via the COM port. The result of this test is a pass.

Requirement: The visualizer must be able to identify the impact within 20% of where it occurred.
Result: PASS

For this test, the visualization software was opened and run. Simulated data was then given to the visualization software. In the case of the data given to generate *Figure C2*, the data was equal force between three sensors.

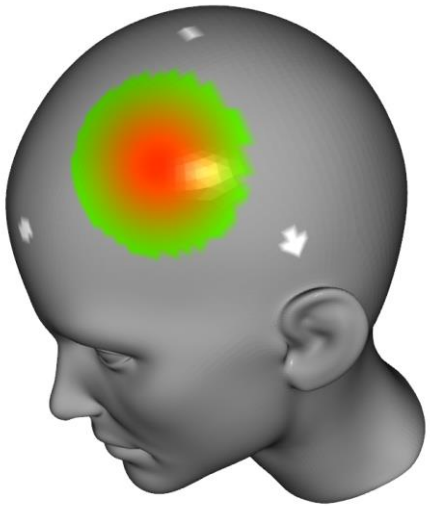

Figure C2: Visualization of hypothetical sensor data. For an equal force reading on sensors 1, 3, and 7, the collision is predicted to be in the center of the three white points.

As seen in *Figure C2*, the visualizer recognized that the packet has equal forces at three of the sensors, which would correspond to a hit in the center of the arrangement. Because the visualizer can properly visualize data it is given, the result of this test is a pass.

Requirement: The data must be visualized within 30 seconds of packet reception.
Result: PASS

For this test, an Arduino was connected to the computer with the visualization software. It was programmed to turn on an LED, and then send the data to the visualization software on the computer via serial communication. The visualization software then rendered the collision heat map, as seen in *Figure C3*. This entire process was captured with a video.

Figure C3: *Visualized data from received packets.*

As seen in *Figure C3*, the packets visualized the heat map successfully. Also, as seen in *Figure C4*, the video recording the LED turning on and then the rendering of the heat map was 15 seconds, which is less than 30 seconds.

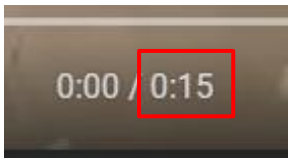

Figure C4: *Time elapsed for rendering the data was 15 seconds.*

Because the time elapsed for rendering the visualization was less than 30 seconds, the result of this test is a pass.

Appendix D: Cost and Schedule Tables

The parts purchased and the costs associated with those are seen in *Table D1*. The week-to-week schedule can be seen in *Table D2*.

Description	Part Number	Unit Price	Quantity	Cost for Quantity
ADC	ADS1178	$18.95	2	$37.90
Microcontroller	ATmega32U4	$5.68	2	$11.36
XBee Transmitter	Zigbee Modules - 802.15.4 XBee, S2C, 2.4GHz Through-hole,PCB ant	$22.95	2	$66.20
1.8V regulator	SPX1117M3-L-1-8/TR	$0.53	3	$1.59
2.5V regulator	SPX1117M3-L-2-5/TR	$0.53	5	$2.65
3.3V regulator	SPX1117M3-L-3-3/TR	$0.53	3	$1.59
5.0V regulator	SPX1117M3-L-5-0/TR	$0.53	3	$1.59
Force Sensing Resistors	CN1501541594	$24.44	1	$24.44
Male Conn	WM15179-ND	$4.68	2	$9.36
Female Conn	WM15032-ND	$1.03	2	$2.32
9V battery	n/a	$16.99	1	$16.99
22Ω Resistor	WR06X220 JTL	$0.01	10	$0.09
2kΩ Resistor	RR0816P-202-D	$0.11	20	$2.20
10kΩ Resistor	RR0816P-103-D	$0.13	3	$0.39
1uF Capacitor	06033C105KAT2A	$0.154	40	$6.16
2.2uF Capacitor	0603ZD225KAT2A	$0.11	10	$1.10
4.7uF Capacitor	06033D475KAT2A	$0.234	10	$2.34
22pF Capacitor	06035A220JAT2A	$0.038	10	$0.38
USB-mini-B	65100516121	$1.41	4	$5.64
16MHz Crystal	ECS-160-20-3X-TR	$0.35	3	$1.05
Schottky Diode	MBR0520	$0.16	5	$0.80
Battery Connector	PJ-102AH	$0.82	3	$2.46

Battery Cable	1927-1053-ND	$1.17	2	$2.28
Wire Crimps	538-105300-2400	$0.351	24	$8.43
Additional PCBs	n/a	$51.00	1	$51.00
Total				$260.31

Table D1: Parts Lists Cost

Week	**Task**	**Person**
2/20	Start circuit schematic design, component selection	Ryan
	Research XBee/Arduino interface research, impact localization algorithm from sensor data	Saathvik
	Visualization software research, initial stage development	Patrick
	Design Document, Team Contract	**All**
2/27	Complete schematic design, begin PCB layout, finalize component selection and ordering	Ryan
	Begin writing receiver Arduino code for XBee interface,	Saathvik
	Expand existing software visualization, full heatmap functionality, streamlined file IO	Patrick
	Design review	**All**
3/6	Finalize PCB layout, confirm communication/plans with machine shop	Ryan
	Complete Arduino code, begin testing with XBee modules, PCB verification	Saathvik
	Log serial communication to txt file, define mappings from raw data to meaningful info, PCB verification	Patrick
	1st round PCB orders	**All**

3/13	Purchased component validation	Ryan
	Confirm Zigbee communications	Saathvik
	Streamline serial → Python → visualization process	Patrick
3/20	Assemble PCB, validate on-board functionality	Ryan
	Integrate XBee module with microcontroller on board, ensure communication	Saathvik
	Validate timing, error requirements of software, PCB validation	Patrick
3/27	PCB revisions, component re-selection	Ryan
	PCB revisions, verify wireless connectivity/range requirements	Saathvik
	PCB revisions, demo full information flow helmet → software	Patrick
	Second round PCB orders	**All**
4/3	Full PCB/encasement/helmet assembly, modify as needed	Ryan
	Impact localization algorithm testing/modification	Saathvik
	Impact localization algorithm implementation with display	Patrick
4/10	Secondary PCB validation, enclosure modifications	Ryan
	Secondary PCB validation, verification of high-level requirements	Saathvik
	Secondary PCB validation, complete integration of software	Patrick
	Team Contract Fulfillment	**All**
4/17	Prepare demo, hardware troubleshooting	Ryan

	Prepare demo, communications troubleshooting	Saathvik
	Prepare demo, software troubleshooting	Patrick
	Mock Demo	**All**
4/24	Prepare demo, presentation, report	Ryan
	Prepare demo, presentation, report	Saathvik
	Prepare demo, presentation, report	Patrick
	Final Demo, Mock Presentation	**All**
5/1	Prepare presentation, final report	Ryan
	Prepare presentation, final report	Saathvik
	Prepare presentation, final report	Patrick
	Final Presentation	**All**

Table D2: Schedule