Team 49-Head Trauma Data Transmission and Logging

ECE 445 Spring 2017

FINAL REPORT

May 3, 2017

Authored by: Phil Meyer, Conner Cecott, Fahim Sheikh
TA: John Capozzo

Team 49-Head Trauma Data Transmission and Logging

ECE 445 Spring 2017

Abstract

The Head Trauma Data Transmission and Logging system is a device that t has the capability of collecting accelerometer data, inferring linear and rotational acceleration, and transmitting the data to a sideline application via Bluetooth. The device's purpose is to be used as a data collection device to collect accurate data which can be used in traumatic head injury research.

Table Of Contents

1 Introduction

1.1 Objective

Research suggests that there are as many as 3.8 million sports-related concussions in the United States every year and this number shows no signs of decreasing [1]. From high school athletics to the NFL, concussion rates are actually increasing annually. NFL athletes have remarked that if concussions don't happen on every play, then they occur on every other or every third play. Consequently, athletes end up playing through concussions and not getting immediate care.

The main reason no system exists to better monitor concussions is because there are numerous factors that play a role, including individual genetics. In addition, there simply is not enough data to enable research in this area. Our project hopes to address a few of these obstacles.

In conjunction with TEAM 55, the goal is to design a wearable device for athletes that has the capability of collecting heart rate and accelerometer data, wirelessly transmitting the data, performing particular algorithms, and storing the data for future analytics in the long term. Our team will be responsible for the latter half of the project. Specifically, our team plans to generate data on a Raspberry Pi, transmit the data wirelessly to our PCB, run our algorithm to gain insights, and wirelessly transmit the data to an Android application that can save the data for long-term storage. We hope that this device will provide insightful information that can be used in research for years to come. Additionally, we hope to address the feasibility of designing such a device and what tradeoffs it presents.

1.2 Background

To date, a few systems do exist that try to address the media's concern around increasing concussions in athletics. For example, the Head Impact Telemetry System (HITS) was designed by Simbex in an effort to measure and record head impact exposure [1]. However, HITS, as well as other devices, only collects accelerometer data. In addition, some of the devices try to predict concussions and alert coaches and/or trainers on the sidelines when a high-impact hit was made. Lastly, some devices are built into helmets, which might compromise the integrity of a helmet and would not be easily adopted by the NFL, for example. To summarize, current devices are limited by pure accelerometer data and provide no additional insights into the increasing amount of concussions.

Our design will be the first to simultaneously collect heart rate data and accelerometer data. However, it is important to note that we will not try to diagnose concussions, as this is not possible to do at the time of impact and will require detailed analysis by qualified medical researchers and professionals. We will simply analyze our data to infer impact locations and rotation of the rigid body to help enable research, as well as time lock this data to the player's heart rate. This data will then be available for coaches and/or trainers on the sideline before the start of the next play.

1.3 High Level Requirements

- Data resolution: In order to collect accurate accelerometer data, we will sample at 1000 Hz [11].
- Algorithm accuracy: In order to make use of the collected data, the algorithm should determine acceleration in six degrees of freedom for a rigid body with accuracy allowing +/- 10% error. Accuracy will be calculated based on simulated data.

- Scalability: The design must be capable of handling a varying number of accelerometers, with an upper limit set by the largest acceptable time delay before seeing data appear on the application. Beyond this, scaling requires an upgrade in processing power.
- Wearability: All hardware components, i.e. Bluetooth module and microcontroller, must fit within a volume of 101 x 42 x 20 mm. This is to ensure that our design can be worn by a player without significantly infringing on their normal movement and uniforms/protective wear.

2 Design

2.1 Block Diagram

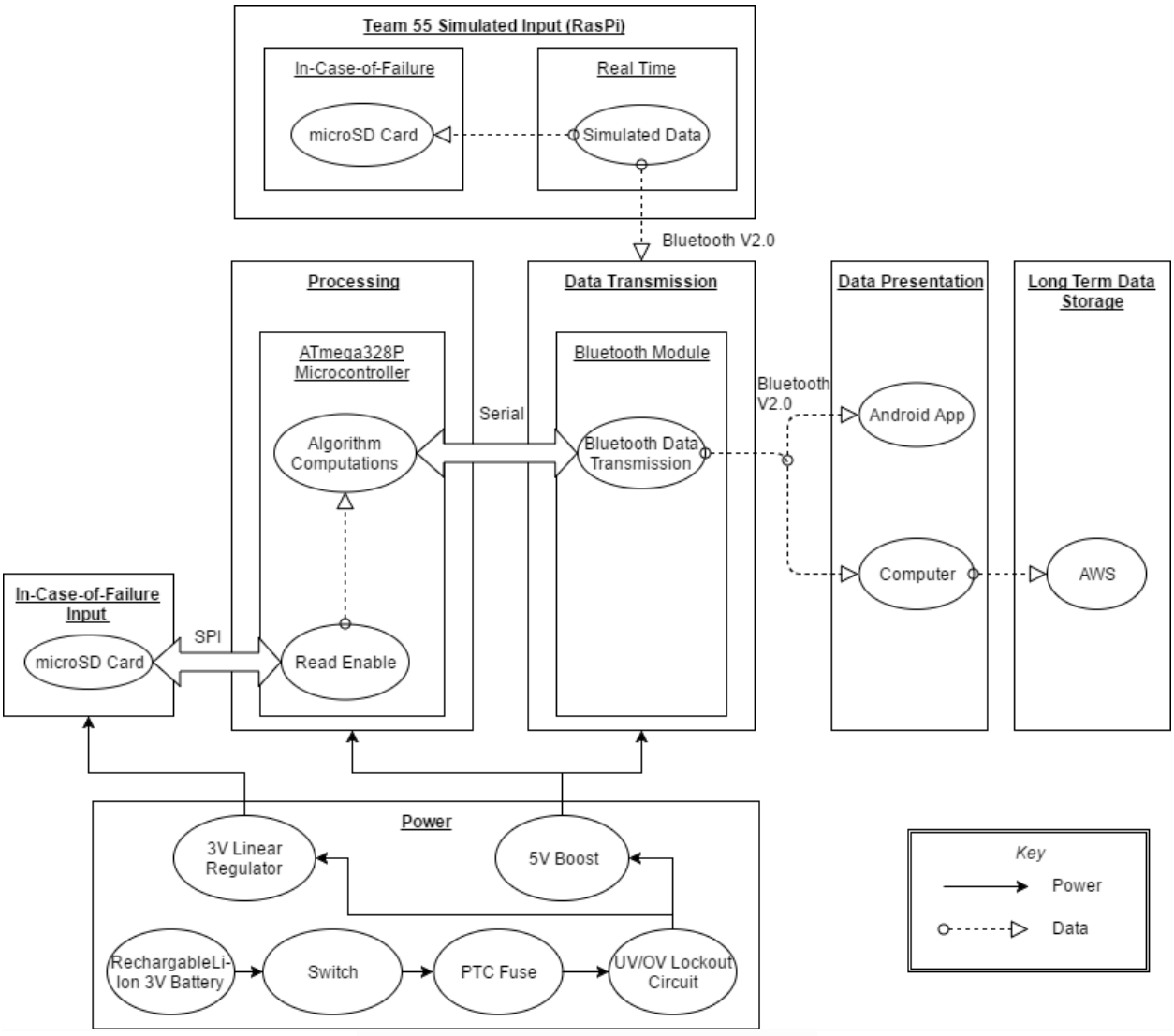

Figure 2-1 High Level Block Diagram

The *Team 55 Simulated Input* block is the source of filtered data that will be wirelessly transmitted to our microcontroller from a Raspberry Pi. In case of transmission failure, the Raspberry Pi will also write the data to a microSD to ensure data is not lost. This block essentially

resembles Team 55's portion of the overall design. If needed, the microSD card from the Raspberry Pi will be inserted into a connector that is soldered to the board that will communicate with the microcontroller through SPI transmission traces. This is the *In-Case-of-Failure* block. The *Data Transmission* block consists of a Bluetooth module that will receive the data wirelessly from the Raspberry Pi and send it to the microcontroller in the *Processing* block. The microcontroller will process the data by running the algorithm, and then transmit the data to a mobile application/computer (*Data Presentation* block) via the Bluetooth module. AWS will be used for *Long-Term Data Storage*. The *Power* block consists of a lithium ion rechargeable 3V battery, a switch to turn off power to the board, a PTC fuse for excess current protection, an UV/OV lockout protection IC for undervoltage and overvoltage protection, a 5 V boost IC to step the voltage up from 3V to 5V for the microcontroller and Bluetooth module, and a 3 V linear voltage regulator to ensure a consistent 3V power supply for the microSD.

2.2 Design Details

2.2.1 Input

To emulate Team 55's wearable device that collects real-time acceleration and heart rate data, three tri-axle accelerometers and a Raspberry Pi will be used. In order to collect accurate accelerometer data, the Raspberry Pi should sample at a frequency of 1000 Hz [3]. Since each accelerometer sample will collect 16 bits per sensing axis, one impact sample will total 144 bits just for acceleration ($3\ accelerometers \times 3\frac{axis}{accelerometer} \times 16\frac{bits}{axis}$). 8 bits will be used for each simulated heart rate sample since they can represent a maximum heart rate of 256 BPM. To calculate how much memory will be stored per game, a few parameters need to be set. Each impact event will include 50 samples of accelerometer data since an average impact lasts less than 50 ms. Additionally, heart rate will be calculated as a running average every second. Since our goal is to learn how a player's heart rate changes from before, during, and after an impact event, heart rate will be calculated every second for an entire minute, resulting in 60 heart rate samples. The average amount of plays in an NFL game is about 65, so if a skill position player (i.e. running back) saw about half of the plays *and* experienced an impact on each of those plays, the player would experience about 30 impacts per game. This amounts to less than half a MB of data per game, which does not impose any strict requirements on the size of the microSD card.

$$\left(144\frac{bits}{sample} \times 50\frac{samples}{impact} + 8\frac{bits}{sample} \times 60\frac{samples}{impact}\right) \times 30\frac{impacts}{game} \times 1\frac{MB}{8\times10^6\ bits}$$

$$= 0.23\ MB\ of\ data\ per\ game \quad (1)$$

If a microSD card needs to be used to recover data that wasn't transmitted in real-time, it can be inserted into the connector on the PCB and used as the source of the data for the algorithm. On our PCB, the microSD card requires a voltage input of 2.7-3.6 V [16]. The voltage being supplied from the lithium-ion battery is 3 V. Therefore, the input voltage needs to be within 10% of 3 Volts to satisfy the microSD power requirement. Lastly, the average amount of data collected for one player over the course of a game will be less than 0.5 MB. This does not impose any strict requirements on the size of microSD card we use.

2.2.2 Data Transmission

The Data Transmission block is comprised of a Bluetooth module. The RasPi will be responsible for sending data collected from the accelerometers to the Bluetooth module which in turn will send the data to the microcontroller for processing. Once the microcontroller has finished processing the data, it will transmit the results to the Bluetooth module and from there to the Bluetooth connected device. It is important to keep in mind that due to the limited memory available to us on the microcontroller, we will be sending a single data point per transmission to ensure that the algorithm fits and we still have space left for a potential state machine and other auxiliary code. Additionally, since it is important to protect the players, the data should all be transferred to the sideline after a major collision within 40 seconds since that is the amount of time between plays in the NFL.

2.2.3 Power

The power block consists of a lithium-ion rechargeable 3V battery, a switch to turn off power to the board, a PTC fuse for excess current protection, an UV/OV lockout protection IC for undervoltage and overvoltage protection, a 5 V boost IC to step the voltage up from 3 V to 5 V, and a 3 V linear voltage regulator to ensure a consistent 3 V power supply. The battery chosen is a 3 volt battery that uses the undervoltage/overvoltage, 3 V linear regulator, and 5 V boost to ensure an accurate 3 V and 5 V is supplied. The linear regulator accepts 2.7-5.5 V [6], the microSD accepts 2.7-3.6 V [16] and the UV/OV protection IC accepts 2.5-34 V [8]. Therefore, the limiting factors on the 3 V input voltage are the microSD card and linear regulator which require the voltage to be within 10% of 3 V. Since the microcontroller accepts 1.8-5.5 V but needs 5 V to run at its highest clock speed [7] and the Bluetooth module accepts 3.6-6 V [9], the microcontroller is the limiting factor on the 5 V power supply. Therefore, the 5 V supply needs to be within 10% of 5 V.

Additionally, the microSD requires up to 200 mA [16], the UV/OV protection IC requires up to 125 μA [8], the Bluetooth module requires up to 35mA [9], the linear regulator requires up to 385 μA [6], the 5V boost module requires up to 15mA [15], and the microcontroller IC requires up to 100 mA [7], so the maximum total current required for the PCB will be approximately 350.51 mA. However, the battery should be capable of supplying more current than this and the PTC fuse is set to fault at 500 mA in case there are additional resistive losses that would draw unforeseen current.

For power protection purposes, the undervoltage/overvoltage lockout protection IC is used. In terms of an open or short applied to the circuit, the PTC fuse protects the circuit in the case of a short. In the event of an open, the UV/OV protection will not allow the circuit to operate and no damage will be done. The linear regulator is used to keep the voltage level constant rather than fluctuating within the range allowed by the UV/OV IC. Additionally, the linear regulator has built-in undervoltage protection. However, since it does not have overvoltage protection, the UV/OV protection is still needed in case someone inserts a higher voltage battery accidentally.

Furthermore, for real world use cases the unit should be able to function throughout the course of a whole game of any sport. Assuming that the upper limit on this requirement is three hours, the battery should be able to power the device at its average load current for three hours. The average load current, assuming real time data is being used (microSD in sleep mode), is 350 μA for the microSD [16], 9.46 mA for the microcontroller [7], 35 mA for the Bluetooth module [9], 125 μA for the UV/OV IC [8], 385 μA for the linear regulator [6], and 3.8 mA for the 5V boost module [15], which comes out to be 49.12 mA total. In terms of scalability, with more accelerometers more power would be required, however for the purpose of our half of the project

we do not supply power to the accelerometers. But, if enough accelerometers are used then a faster processor may be needed which in turn would require more power.

The UV/OV IC has a procedure in the datasheet to determine the values of the resistors for the external resistor divider network [8]. As mentioned above, the lower threshold was determined to be 2.7 V while the high threshold was determined to be 5.5 V. The resistor values were calculated using these threshold values but due to the nature of what standard resistor values are available, with the closest standard values to the calculated values picked, it was determined that 1.3 Mohm, 154 kohm, and 147 kohm resistors would be used. This gives a threshold range of 2.67-5.45 V. Since the resistors are 1% accuracy resistors and there is a maximum voltage offset used in the calculation due to leakage flux, these threshold values should be within 5% of 2.7 V and 5.5 V in order for the UV/OV IC to be considered working properly.

2.2.4 Processing

The *Processing* block consists of an ATmega328 microcontroller IC. This microcontroller will be responsible for all incoming data, processing of this data, and transmission of the enriched data. These tasks will be handled by two main components: a state machine and the algorithm that determines linear and rotational acceleration from accelerometer data. In its current state, the microcontroller's software takes up approximately 12 KB of data, which is about a third of the ATmega328's flash memory (32 KB).

2.2.4.1 Algorithm

Our algorithm has one main objective: to infer the linear and rotational acceleration of the head's center of gravity. This results in six degrees of freedom that describe the motion of a rigid body upon impact [5]. The acceleration at any point, i, on the head, $\vec{a}_i$, undergoing linear and rotational acceleration can be described by:

$$\left|\left|\vec{a}_i\right|\right| = \overrightarrow{r_{a_i}} \cdot \vec{H} + \overrightarrow{r_{a_i}} \cdot (\vec{\alpha} \times \vec{r_i}) \quad (2),$$

where $\overrightarrow{r_{a_i}}$ is the sensing access of the accelerometer, $\vec{H}$ is the linear acceleration of the head's center of gravity, $\vec{\alpha}$ is the rotational acceleration of the head's center of gravity, and $\vec{r_i}$ is the position vector of the accelerometer. All parameters are known except $\vec{H}$ and $\vec{\alpha}$. Equation (2) is an equality relating the acceleration an accelerometer reads and the actual linear and rotational acceleration a rigid body experiences. Ideally, $\vec{H}$ and $\vec{\alpha}$ are chosen such that the equality holds true. Therefore, we define a loss function to minimize the error between the two sides of (2):

$$\min_{\vec{H},\vec{\alpha} \in \Re^3} \sum_{i=1}^{n} \sum_{j=1}^{3} \left[\left|\left|a_{i_j}\right|\right| - \left(\overrightarrow{r_{a_{i_j}}} \cdot \vec{H} + \overrightarrow{r_{a_{i_j}}} \cdot \left(\vec{\alpha} \times \overrightarrow{r_{i_j}}\right) \right) \right]^2 \quad (3),$$

where n is the number of accelerometers being used. To solve this optimization problem, a gradient descent algorithm will be used to estimate $\vec{H}$ and $\vec{\alpha}$.

To ensure that the Atmega328p can handle the necessary computations in real-time, we have simulated our algorithm in Python to better understand how long it will take to process incoming data. The ATmega328p achieves throughput at about 1 MIPS per 1 MHz. It also operates at 20 MHz. This allows for $20 \times 10^6 \frac{instructions}{second}$:

$$1\,\frac{instruction}{cycle} \times 20 \times 10^6\,\frac{cycles}{second} = 20 \times 10^6\,\frac{instructions}{second}\ (4),$$

Our simulated gradient descent algorithm is primarily broken up into two main components (*while* loops). The first component checks if we have achieved the minimum of our loss function. If the minimum has not been achieved, we move in a direction opposite of the gradient by a varying step-size. On average, it takes approximately 100 iterations (steps) to achieve the minimum when using 3 accelerometers. Each iteration is composed of multiple calculations, such as 2-norms, multiplications, subtractions, assignments, and comparisons. These calculations are primarily due to the 1000 function evaluations and 1500 gradient evaluations that take place. To evaluate the function once, it takes approximately 234 operations, due mostly to the cross and dot products. To evaluate the gradient of the function, it takes approximately 297 operations for the same reason. In total, about 750,000 operations (additions, multiplications, subtractions) are evaluated for the gradient descent algorithm to converge at its minimum. Since each impact event totals 50 samples, it will take approximately 1.875 seconds to process a full impact event's data using the algorithm.

$$750{,}000\,\frac{instructions}{data\ point} \times 50\ data\ points\ \div \left(20 \times 10^6\,\frac{instructions}{second}\right) = 1.875\ seconds\ (5)$$

2.2.5 Wearability/Size

As mentioned in the high-level requirements, the device needs to be compact enough as not to inhibit the athlete while they are competing. With that being said, the size of the device comes down to the hardware constraints. For this initial prototype, in terms of size we are limited by hand soldering capabilities. For instance, all resistors and capacitors are size 0603 with a larger footprint for hand soldering whereas if this device were to be mass produced, a pick and place machine would be used along with either 0402 or 0201 sized passive components. Additionally, the microSD slot that is being used is a vertical one because it is all through hole and hand solderable. For production purposes, a smaller surface mount socket could be used so it wouldn't stick up off of the board. Similarly, a surface mount package for the microcontroller could be used for space saving characteristics. Furthermore, the Bluetooth module is connected to the PCB via a cable and four-pin header soldered to the PCB. As this sounds, it is not very space effective. The reason for choosing this method was because the module would have been extremely difficult to hand solder and debug if there were any solder issues. Again, in production this would be soldered to the surface of the board with a pick and place machine and take up much less space.

As for power, the battery holder is being used so the battery can be removed to be charged. For convenience for the end users, in production a battery charging circuit would be implemented on the board along with a connector, so a typical 5V AC to DC power cord could be used to charge the device. The battery could then be soldered to the board, eliminating the size of the battery holder but adding the size of the charging circuit and charger port, ultimately, similar in size.

2.3 Verifications

Table 2.3-1: Requirement and Verification Table

Requirements	Verification	Points
Input		**Total Points: 8**
1. Continuously store the data from each impact (23,200 bits) on microSD and transmit wirelessly to PCB Bluetooth module.	Connect device to Raspberry Pi via Bluetooth and insert microSD into Raspberry Pi. Collect accelerometer data, which should be received wirelessly, in real-time. Remove microSD card from the Raspberry Pi and plug into computer. Check that the data was written to the microSD.	4
2. PCB accurately reads at least 95% of the data from the microSD card.	Connect the programmed microcontroller to the Arduino. Execute the program and compare the data written to the Serial Monitor against the data that was on the microSD card.	4
Data Transmission		**Total Points: 10**
1. Ability to transfer data via Bluetooth at 9600 bit/s baud rate within approximately 40 seconds.	Run the algorithm on a full impact event's dataset and time it by hand.	5
2. Transmit 23,200 bits total for accelerometer and heart rate data for one impact event.	View 50 estimates of linear and rotational acceleration, given by running the algorithm, as well as 50 heart rate samples, all on a Bluetooth connected device.	5
Power		**Total Points: 12**
1. Supply voltage within 10% of 3 Volts.	Apply a voltage meter between ground and the output pin of the linear regulator to verify that the voltage is within 10% of 3 Volts.	2

2. Supply voltage within 10% of 5 Volts.	Apply a voltage meter between ground and the output pin of the 5 V boost IC to verify that the voltage is within 10% of 5 Volts.	2
3. Supply 500 mA at 3 Volts.	Use a breadboard and resistors to set a resistive load of 6 ohms within 5% which would draw 500 mA at 3 V. Use voltmeter to verify voltage is still within 10% of 3 V.	2
4. Supply average current, 49 mA, for three hours.	Use a breadboard and resistors to set a resistive load of 60 ohms within 5% and verify after three hours that the battery is still within 10% of 3 V with a volt meter.	2
5. UV/OV lockout within 5% of 2.7 and 5.5 V	Use a DC power supply as the input to the PCB. Slowly vary the voltage near the limits and record when the UV/OV IC no longer allows voltage to pass through. Ensure these values are within 5% of their respective limits.	4
Processing/Algorithm		**Total Points: 10**
1. Calculate linear and rotational acceleration of the rigid body to within +/- 10% of test data.	Run the algorithm on accelerometers' datasets and compare the estimated linear and rotational accelerations against generated values in Unity.	5
2. Generate test data by creating a Unity simulation and an accelerometer Simulink block.	Demonstrate the process of running the Unity simulation, exporting relevant data to simulink, and storing test data in text files.	5
Wearability/Size		**Total Points: 10**
1. Device fits within 101 x 42 x 20 mm.	A caliper or similar measuring device will be used to ensure the device is within these limits.	5
2. Size scaling compared to power consumption, processing speed, and accuracy.	Explain how increasing the number of accelerometers (accuracy) will ultimately affect the size of the device.	5

2.3.1 Input

As per our initial design considerations, we attempted to use a RasPi to transmit accelerometer data to our microcontroller. Unfortunately, in the process of implementing this feature we ran into a variety of issues that forced us to abandon this original piece of the design.

One of the first issues we experienced was soon after establishing communication between the microcontroller and the RasPi. After our initial tests of sending simple data, such as an on/off signal to control an LED, were successful we attempted to get that data to do a full round of travel, i.e. to transmit from the RasPi, to the microcontroller, and then to a Bluetooth connected tablet. The issue here was three fold. One, the bluetooth module we selected was a slave chip which means that while it can accept connection requests from other devices, it can not attempt to establish those connections itself. Since our state machine controlled the flow of data and was implemented on the microcontroller, it was necessary for it to be able to make Bluetooth connections by itself. Instead, with our Bluetooth module the microcontroller was forced to wait for the RasPi or tablet to make a connection. Now the obvious solution was to purchase a master Bluetooth module, and it was an idea that was pursued until we ran into our second issue. Namely, the problem now was discovering that the average time for Bluetooth to establish a connection is about 0.79 seconds [13]. This added time was simply too long to be feasible because our design would require the microcontroller to connect, drop, and reconnect with the RasPi and tablet multiple times for each set of data. Also, since the full device, i.e. our project combined with Team 55's, would directly connect the accelerometers to the microcontroller, using Bluetooth to transmit that data was seen as an unnecessary task. Now the third and final issue that ultimately forced us to abandon this component entirely was the fact that the microcontroller could not read any of the data from the microSD card that the RasPi had written. This was due to the fact that the RasPi's microSD card stored an operating system that the RasPi used to boot. Although the RasPi could write data to this microSD card, the microcontroller could not access the file system because it could not interpret the overlying operating system. The microSD card was included to serve as a backup for data storage, in case of Bluetooth transmission failure, but since the microcontroller cannot read that data there was no feasible way to incorporate the RasPi into the design.

Despite this setback, we still needed some method of getting accelerometer data to feed into the algorithm. To this end it was suggested that we use a program called Unity in conjunction with a Simulink block to generate test data. Through Unity we gained access to a physics engine which could be used to collect data about the dynamics of a "rigid body". A rigid body is a component that can be attached to objects, such as spheres and cubes, to allow the physics engine to work on the object. For our purposes, we modeled the head as a sphere and the accelerometers as cubes. For the purposes of scalability, we used the Fibonacci Sphere algorithm to generate *n* evenly distributed points on the sphere where we could place *n* cubes. Without the Fibonacci Sphere algorithm there would be no easy way to place an arbitrary number of cubes on the sphere since Unity required exact coordinates to place the cubes. So from Unity we could collect the linear positions, angular velocities, and the center of mass of the cubes and sphere. This data was then exported to Matlab where a script calculated the linear velocity, linear acceleration, and angular acceleration, all of which were passed into the Simulink accelerometer block. This block would then produce accelerometer data, which another Matlab script would save into a text file, for us to pass into our algorithm.

Now although we had test data, it came with its own set of concerns. Initially the main worry was the fact that regardless of how many points of data we fed into the Simulink block, it would always output 101 points. Additionally, feeding one of the test accelerometer values into the algorithm would result in values of linear and rotational acceleration that did not match the

values recorded by Unity and Matlab for the sphere. The lack of any clear and helpful documentation or tutorials for the Simulink block made it difficult to debug the issue we were experiencing, and unfortunately we were unable to determine the problem before our demo. However, it is important to mention that even though we could not verify the accuracy of the algorithm using our test data, we could still use it to measure other parameters such as the algorithm's speed of convergence and the effect of varying the number of accelerometers used. Also, the accuracy of our algorithm was guaranteed to have an error within 20% using 6 single-axis accelerometers as per the 6-DOF paper that our algorithm was derived from [3].

2.3.2 Data Transmission

In our final working design the data transmission block takes processed data being transferred from the microcontroller to a tablet, or other Bluetooth connected device, through a Bluetooth 2.0 connection. The microcontroller contained our state machine and thus handled all data transmission. As part of the TRANSMIT state, the microcontroller would accept a connection to a tablet and wait for a ping before transmitting the processed data. The baud rate was set to be 9600 bits/s, and we attempted to use a faster rate but unfortunately the tablet could only handle a baud rate of 9600 bits/s as a maximum. For more detail about the state machine in general, and the TRANSMIT state in particular, please see section 2.3.4.2.

2.3.3 Power

Upon verifying the requirements for power, a DC supply was used to verify requirements 1, 2, and 5. See Table 2.3.3-1 for the results. As shown, the 3V and 5V sources are within 10% of their respective voltage targets when a 3 V source is applied. Additionally, the UV/OV lockout interrupts the power supply to the rest of the circuit when the supplied voltage is within 5% of 2.7 and 5.5V.

However, when verifying requirements 3 and 4 it became evident that the lithium-ion coin cell battery would not be able to handle the current draw of the microcontroller. When attempting to verify requirement 3 with a load of 6.6Ω connected to the battery, the voltage on the battery immediately dropped to 1.6V. Similarly, when trying to verify requirement 2 with a load of 63.2Ω connected to the battery, the voltage dropped to 2.65V, outside of the UV/OV voltage threshold, within two minutes.

The mistake in choice of battery arose from the misinterpretation of the stored energy capacity (mAh) rating of the battery. Although the battery had enough stored energy to supply the 500mA max that our design may draw, the nature of the battery does not allow it to discharge fast enough to supply that current at rated voltage. Upon further investigation, it was determined that coin cell batteries are generally used for very low current draw applications. Upon redesign, a possible battery choice could be a cellphone battery. Typical cell phone lithium ion batteries are 3.7V so it would fit within the UV/OV range. Additionally, smartphone current draw just using internet with the backlight at max brightness can exceed the 500mA draw, therefore a 3.7V lithium-ion cell phone battery would be sufficient for our application [12].

Table 2.3.3-1: UV/OV Sweeping Input Voltage Results

Input [V]	2.80	2.79	3.00	4.00	5.00	5.30	5.33	5.34	5.35
3 V [V]	0.00	2.789	2.984	3.112	3.128	3.131	3.131	3.131	0.00
5 V [V]	0.00	5.052	5.044	5.062	5.098	5.209	5.304	5.313	0.00

2.3.4 Processing

The completed processing engine was approximately 12 KB in size and occupied about a third of the ATmega328p's flash memory. Since our final design did not include an incoming stream of live data from a RasPi, the processing engine was built around a microSD card that was the source for data. However, the original plan for a state machine and algorithm as the core of the processing engine was completed.

2.3.4.1 Processing: State Machine

The purpose of the state machine in our design was to handle the processing of data. Since the ATmega328 had limited memory capacity, we needed to design an efficient state machine. We decided to process data very systematically. This allowed for a total of only four states. Upon the PCB powering on, the state would be set to WAIT. The PCB would remain in the WAIT state until a Bluetooth connection was established with a sideline device and until the connected device sent a "ping" to the PCB. Once received, the PCB would enter the READ state. Since our microcontroller was programmed to handle the processing of data from three accelerometers, the microSD card on our PCB contained three files. Each file contained accelerometer data from its respective accelerometer and each line in the file described one sampled data point (x, y, and z values). The READ state was designed to read in one line of data from each file. At this point, the processing engine held accelerometer data for the x, y, and z directions for three different accelerometers. Next, the PCB would transition into the PROCESS state. In the process state, all of the accelerometer data would be sent into the algorithm so that an estimate of $\vec{H}$ and $\vec{\alpha}$ could be made. Upon convergence, the PCB's state would be set to TRANSMIT and it would transition. In the TRANSMIT state, $\vec{H}$ and $\vec{\alpha}$ would be transmitted to the sideline application via Bluetooth. Additionally, the heart rate data at that time instance would be sent via Bluetooth. Since our PCB did not actually collect heart rate data, the PCB transmitted random heart rate data. After the completion of transmission, the PCB would transition back to the READ state. If there was more accelerometer data to read and therefore process, the PCB would read the data, transition to the PROCESS state, and then transition to the TRANSMIT state. This would occur over and over until the end of the three files were reached and there was no more data to process. This would result in the PCB returning to the WAIT state.

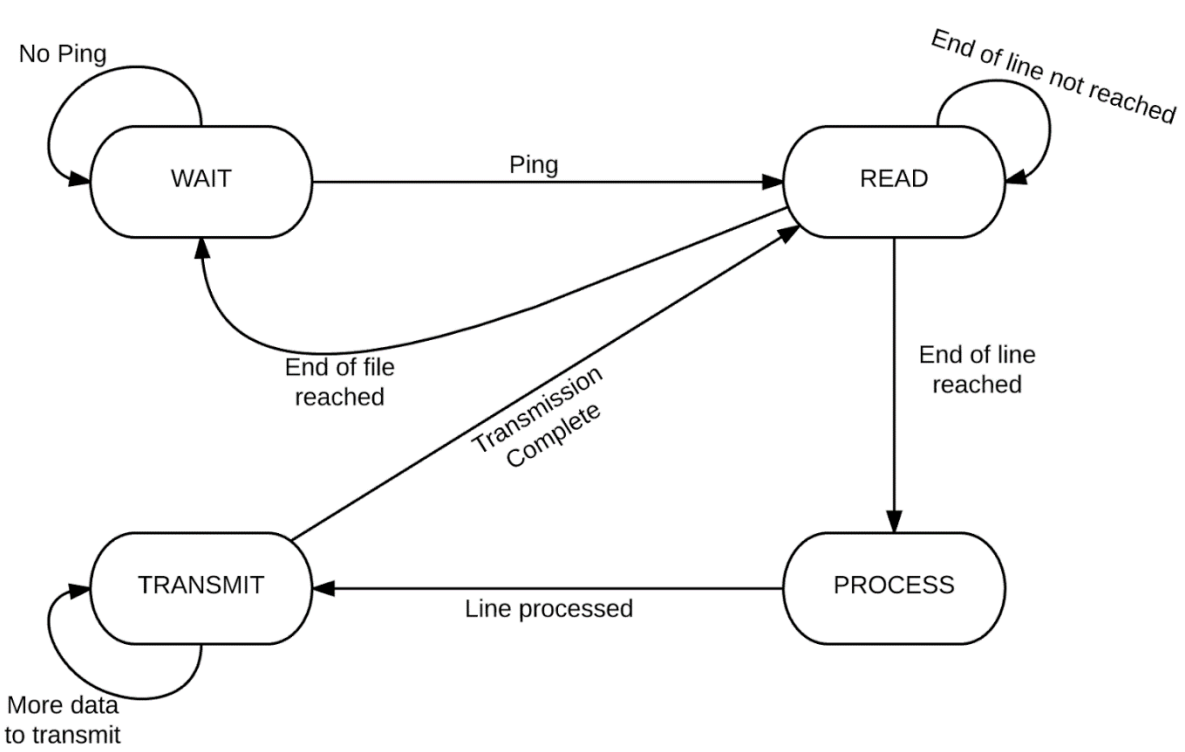

Figure 2.3.4.1-1: State Machine

2.3.4.2 Processing: Algorithm

During the design phase of this project, the processing algorithm was programmed in Python to simulate data and the estimation of the rotational and linear acceleration of a rigid body. Upon completion and consistent convergence of the algorithm, it was transferred to C code. This was done because the ATmega328p required a lower-level programming language.

When it came time to verify how long it took to process all of our data, we measured multiple processes. First, we measured how long our PCB took to read all of the data on the microSD card, process all of the data with the algorithm, and transmit the enriched data to the sideline device via Bluetooth. This entire process took roughly 47 seconds, which was longer than the required 40 seconds. 40 seconds was chosen as the requirement because there are 40 seconds between consecutive plays in the NFL and if a player experienced a possibly-concussive hit on a given play, we wanted our device to show the coaches/trainers on the sideline that their player might be hurt. When we didn't meet this requirement, we decided to make a few additional measurements. In Python, we measured how long it took our computer to process 50 samples of data for three accelerometers (same amount of data processed on the PCB). It took a total of 19.2 seconds in Python, which was much slower than the estimated 1.875 seconds estimated during the design phase for the ATmega328p. This underestimation is most likely due to the microcontroller's 8-bit architecture and significant use of 32-bit floating point numbers. The calculation of 1.875 seconds was based on operations such as additions, subtractions, and multiplications, but was not broken down into assembly language register shifts and arithmetic, especially not for 32-bit floating point numbers on an 8-bit architecture. However, it was still very close to the 40 second requirement and a faster processor would have helped significantly.

In order to evaluate how the algorithm would scale with more accelerometers, the Python algorithm supported testing with six and nine accelerometers. When scaled to six, the algorithm

converged with less than half of the iterations it took for three accelerometers. More importantly, the algorithm processed 50 data samples in 14.9 seconds, which was faster than 19.2 seconds for three accelerometers. When scaled to nine accelerometers, the algorithm again took less iterations to converge, but processing time went up to 32.35 seconds.

Overall, we learned that six tri-axis accelerometers would be the ideal implementation for our design since it processed data the fastest. Additionally, research suggested that six single-axis accelerometers be used for sufficient accuracy. Although we were not able to test our results for accuracy against our test data, we know that from research our six tri-axis accelerometers would result in sufficient accuracy and processing speed.

2.3.5 Wearability/Size

As seen in Figure 2.4.3-1, the device PCB fits within the specified margins. However, as it was discovered too far into the semester to redesign, the microSD needs a logic converting IC to convert the 3V logic used with the microSD to the 5V logic used with the microcontroller. That is why a breakout board is soldered onto the microSD spot and hanging off of the PCB. A further iteration of the design would have both the microSD slot and the logic converting IC soldered on the PCB within the size constraints.

In terms of scalability, if more accelerometers are needed for more accurate data, then more data will need to be processed. If more data needs to be processed, then to do it in a similar time frame a faster processor would need to be used. With a faster processor comes a larger power consumption and possibly different supply voltages. With a larger power consumption, comes more robust voltage regulators capable of handling the higher current, as well as a battery with a higher stored charge capability. Ultimately, for the purpose of our initial prototype, the device fits within 101 x 42 x 20 mm in order to adhere to being as small as possible while accounting for all the hardware. However, for production purposes, the device would be able to be made smaller with the help of a pick and place machine and design for production. Finally, if upscaling in the number of accelerometers is necessary, it is likely there would need to be an upgrade in hardware which in turn would be an increase in size as well. See Table 3.3-1 for an estimation of how power scales with the number of accelerometers.

2.4 Cost Analysis

The labor cost of development of this project is predicted to be proportional to $30/hour for each member of the team over a period of 14 weeks. Thus the total cost of labor for the whole team is estimated to be:

$$3 * \$30/\text{hour} * 20 \text{ hours/week} * 14 \text{ weeks} = \$ 25{,}200 \quad (6)$$

Table 2.4-1 Cost of Parts.

Part	Vendor	Quantity	Cost (Individual)
3 V Li-Ion Battery	Mouser	1	$9.28
Battery Holder (36-301-ND)	Digikey	1	$1.79
Microcontroller (ATmega32)	Microchip	1	$1.83
PCB's	PCBWay	1	$10
Various passive components	Mouser	16	$2.14 (total for 16)
Power Inductor (DO1608C-103MLB)	Mouser	1	$1.59
Sandisk 8GB microSD	Amazon	1	$5.49
microSD Slot (472192001)	Avnet	1	$0.98
PTC Fuse (MF-FSMF035X)	Mouser	1	$0.42
Switch (CL-SA-12C-02)	Digikey	1	$2.52
UV/OV (LTC4365CTS8#TRMPBF)	Arrow	1	$2.13
Accelerometer (ADXL375BCCZ)	Analog Devices	3	$7.92
HC-06 Serial Bluetooth Module	Amazon	1	$9.99
5 V Boost (MAX1724EZK50+T)	Mouser	1	$3.67
3 V LDO (TPS79530DCQR)	Digikey	1	$2.70
28 PDIP Socket (1-2199298-9)	Mouser	1	$0.71
4-Pin Header (22-03-2041)	Mouser	1	$0.30
16 MHz Crystal (ECS-160-20-4X)	Mouser	1	$0.69
Dual Mosfet (IRF7910TRPBF)	Mouser	1	$1.23
Total	N/A	37	$114.63

We only plan on building one fully functional device by the end of the semester so the total development cost is predicted to be $25,314.63. This price may increase if replacement parts are required and so this price should be taken as a lower bound.

3 Conclusions

3.1 Ethics

Since our device is primarily a data logger/processor we do not foresee any major ethical concerns. As far as our data processing is concerned, we are not going to make any inferences or claims based on our data. Rather we are simply collecting and transmitting useful data that can be provided to other individuals or organizations to utilize in their research. Any claims or conclusions that are made based off of our data is not our ethical responsibility. We will go so far as to assure that the shared data has not been altered or tampered with in any way to support any particular study or conclusion. Additionally, since the data we are collecting is health information of athletes we can fall under the protection of HIPPA [2]. Under HIPPA (Health Insurance Portability and Accountability Act of 1996) the healthcare information of an individual, such as heart rate, is protected by federal law. An individual can choose to share their healthcare information with apps and companies such as ours and this ensures the protection of our data as well as the privacy of the athletes.

3.2 Safety

The only real piece of our design that could potentially harm someone is the lithium ion battery. A relatively small explosion or a small fire can be caused by sudden capacity loss from thermal runaway which could be caused by a short across the battery. Depending on how close an individual is to our device if such a failure was to occur, there could be some minor injuries. To mitigate the risk of such an event and ensure the safety of our users we will include a PTC fuse to disconnect the hardware from the power source in the case of a short. This will also have the added benefit of protecting the rest of the circuit from damage. Another instance where the battery could explode or catch fire would be if a collision physically damaged the battery. This would be protected against by a mechanical housing. Also, since we are providing a detailed description of our inputs and outputs, all potential risks associated with our device are clearly defined. This minimizes any major risk of injury due to ignorance.

3.3 Further Design Considerations

For further iterations of this design, as mentioned above, the microSD connector and logic converting IC would be soldered on the board. Additionally, a different battery, possibly smart phone lithium-ion or other architecture battery, would be used to meet our current consumption needs. Furthermore, three traces had to be cut and three jumper wires had to be soldered due to a pin labeling error in the schematic. This, of course, would also be addressed in a following iteration. Also, as mentioned above, when a final design is completed it would be scaled down using smaller passive components and closer placement since a pick-and-place machine would be used as compared to hand soldering.

As to scaling the device to include more accelerometers, more power, faster processing, and a larger size would also be needed to maintain a consistent run-time. See Table 3.3-1 for a rough estimation of how power, processing speed, and size all scale with the number of accelerometers. The max power consumption was looked up for a processor of that speed, added to the power consumption of the rest of the ICs on the PCB, and added to the power consumption of the accelerometers. The processor speed was scaled linearly since the amount of data would increase linearly. The size was determined by accounting for the additional size of the faster processor on the PCB and the additional size of more accelerometers. The speed of the

algorithm was computed from simulations in Python. Memory storage was calculated based on the upper limit of data that would be collected for one player during one football game.

Table 3.3-1: Power, Speed, and Size Scaling with Increasing Accelerometers

# of Accelerometers	Power [W]	Processing Speed [MHz]	PCB Size [cm^3]	Algorithm Speed to find $\vec{H}$ and $\vec{\alpha}$ [ms]	Memory Storage [MB per player per game]
3	1.401125	16	84.8445	19.2	0.2304
6	1.50225	32	88.449	14.9	0.45
9	1.523375	48	92.2035	32.35	0.66

3.4 Take-Away

Overall, this project has been a huge learning experience. Before taking this class none of us had ever participated in brainstorming an idea, formulating a design, and bringing that design to reality. Through this process we have learned quite a bit about what it takes to create a wearable device. For instance, we learned that size consideration is an ever present concern that impacts nearly all other aspects of our design. In terms of our project, size was an issue because it limited how powerful a microcontroller we could use. To take it to an extreme, the most processing power we could get would be from a computer. However, it is obviously unfeasible to ask people to lug around a computer to use our "wearable" device. So instantly, we have to sacrifice a lot of processing power for the sake of making our device comfortable for a person to wear. Additionally, the size constraints impact power in a similar fashion. This time, the issue is that a small microcontroller can still require a fair bit of power to function and that power can only be supplied by a large battery. Once again if taken to the extreme, it could be that we have a compact and powerful microcontroller that fits the design criteria but the power it requires can only be supplied by a large battery that violates our size constraint. So again, size limits the microcontroller that is viable for our design. The process of determining all these different constraints and working within them was an initial challenge that we did not foresee and it was something that impacted the remainder of our project. Thus, we realize now the importance of this beginning stage of design and the proper parameters that need to be considered at this stage.

Another important take-away for us is the ability to quickly pick up new technologies and the ability to modify the design in light of new information. Going into this project, none of us had any prior experience working with Bluetooth and so we had to work fast in order to learn enough about it in order to utilize it in our project. Part of what we learned forced us to do a major redesign in the form of dropping a design component completely, i.e. the RasPi. Having to knock out the RasPi then required us to quickly think up of a new way to generate test data, which we did by way of Unity, Matlab, and Simulink. Frankly, we could have saved ourselves some hassle if we had worked faster on learning and implementing Bluetooth. Accordingly, the lesson here was to quickly learn about any new technology that we are relying on and to be prepared for a redesign due to new information.

All in all this project has been a tremendous learning experience. We have learned quite a bit about various aspects of design as well as general skills and good practices, i.e. quickly learning new technologies, making progress on all fronts, and seeking expert advice to name a few. Although it may have been difficult at times, this project has helped give us valuable insight into the design of wearable devices and the trade offs that exist between size, power, and processing speed. Although we may not have been able to complete the project exactly as we initially envisioned it, we are proud of the progress that we made and of our final device as it exists. We firmly believe that another iteration of this design could meet all of our requirements and be a practical wearable device that can aid concussion research for years to come.

4 Supporting Material

4.1 Design Schematic and Layout

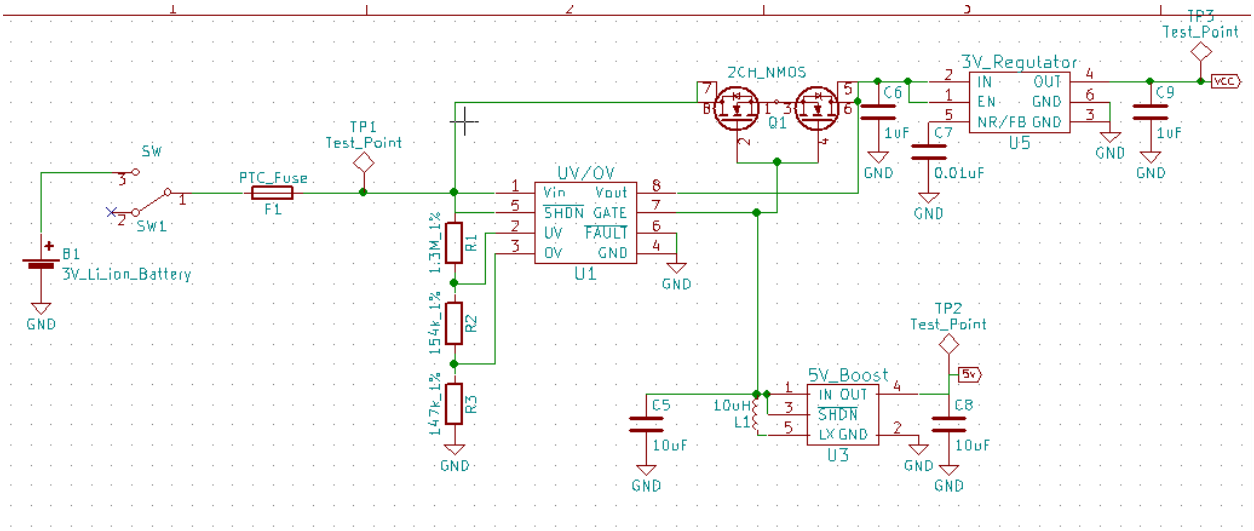

Figure 4.1-1 Power Circuit Schematic

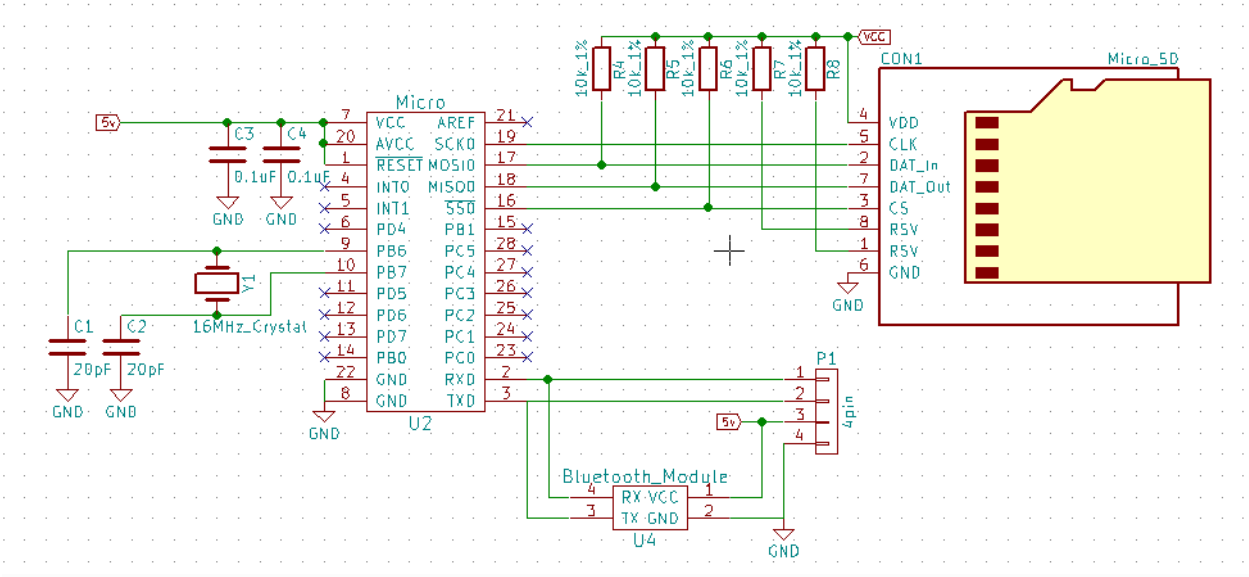

Figure 4.1-2 Microcontroller, Bluetooth Module, and microSD

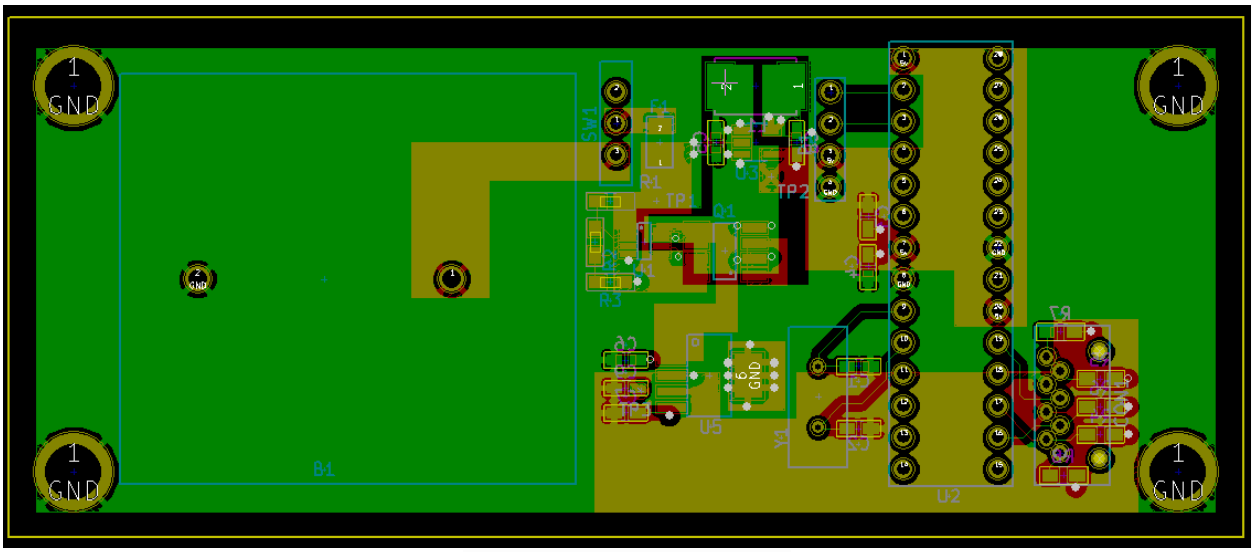

Figure 4.1-3 PCB Layout

4.2 Test Data Generation Block

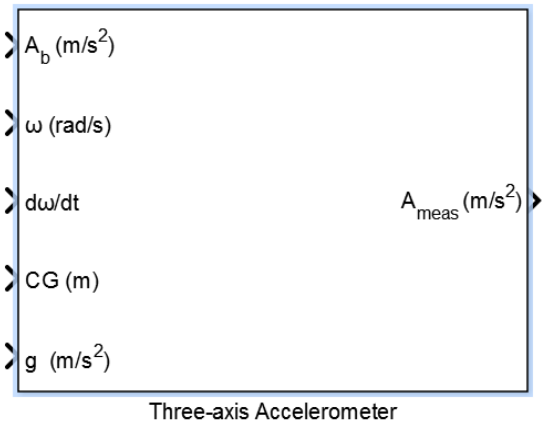

Figure 4.2-1 The simulink block diagram of an accelerometer.

4.3 Device

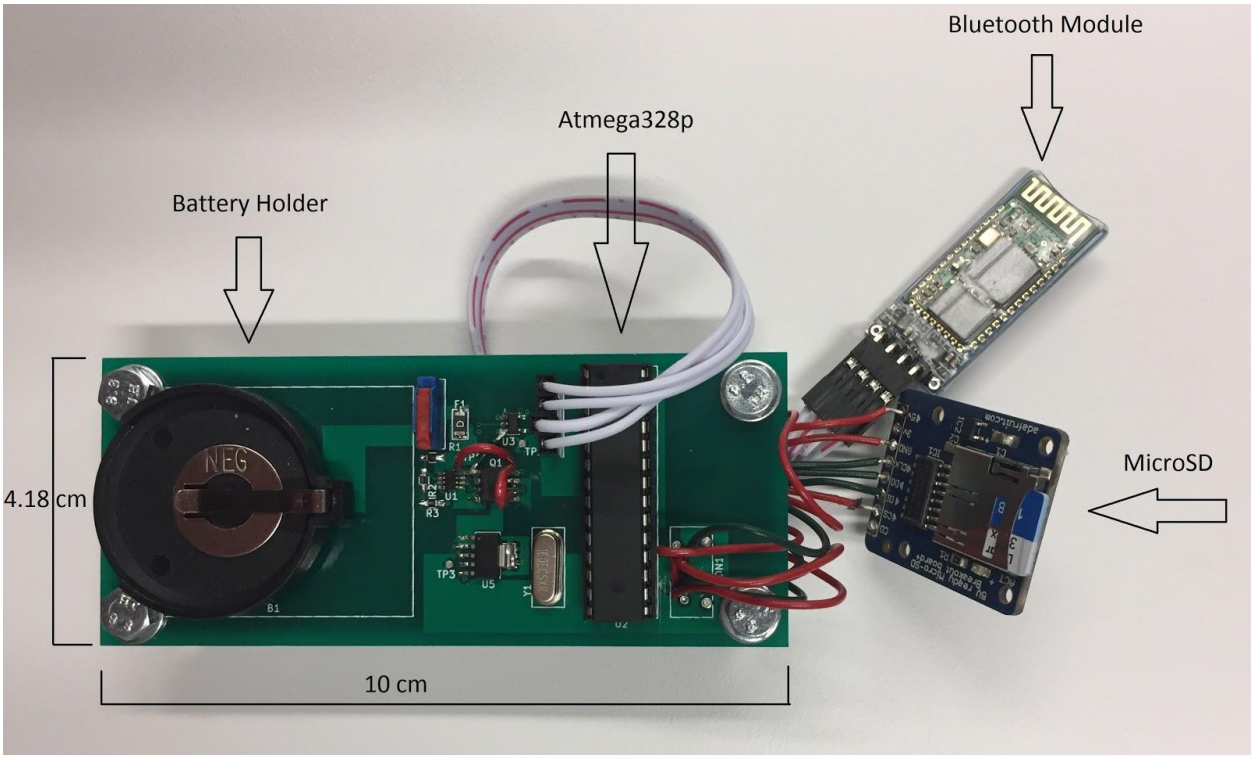

Figure 4.3-1: Finished Device

5 References

[1] Beckwith, J.G., Greenwald, R.M. & Chu, J.J. Ann, "Measuring Head Kinematics in Football: Correlation Between the Head Impact Telemetry System and Hybrid III Headform,"*PubMed*,2012. [Web]. Available: https://www.ncbi.nlm.nih.gov/pubmed/21994068. Accessed Feb 8, 2017.

[2] "Your rights under HIPAA," in *US Department of Health And Human Services*, HHS.gov, 2017. [Web]. Available: https://www.hhs.gov/hipaa/for-individuals/guidance-materials-for-consumers/. Accessed Feb. 8, 2017.

[3] S. Rowson, G. Brolinson, M. Goforth, D. Dietter, and S. Duma, "Linear and angular head acceleration measurements in collegiate football," *Journal of Biomechanical Engineering*, vol. 131, no. 6, p. 61016, 2009. [Web]. Available: https://biomechanical.asmedigitalcollection.asme.org/article.aspx?articleid=1. Accessed Feb. 8, 2017.

[4] Burger, Thomas. "How Fast Is Realtime? Human Perception and Technology." *PubNub*. PubNub, 2015. [Web]. Available: https://www.pubnub.com/blog/2015-02-09-how-fast-is-realtime-human-perception-and-technology/. Accessed Feb. 09, 2017.

[5] Chu, J. J., J. G. Beckwith, and R. M. Greenwald. "A Novel Algorithm to Measure Linear and Rotational Acceleration Using Single Axis Accelerometers." *Journal of Biomechanics*, 2006. [Web]. Available: http://www.jbiomech.com/article/S0021-9290(06)85195-X/abstract. Accessed Feb. 24, 2017.

[6] "TPS795 Ultralow-Noise, High-PSRR, Fast, RF, 500-mA Low-Dropout Linear Regulators." *Texas Instruments.* Ti.com, 2015. [Web]. Available: http://www.ti.com/lit/ds/symlink/tps795.pdf. Accessed Feb. 24, 2017.

[7] "ATmega328/P." Atmel. Microchip.com, 2016. [Web]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Summary.pdf. Accessed Feb. 24, 2017.

[8] "Overvoltage, Undervoltage and Reverse Supply Protection Controller." *Linear Technology*. Linear.com, 2013. [Web]. Available: http://cds.linear.com/docs/en/datasheet/4365fa.pdf. Accessed Feb. 24, 2017.

[9] "HC-06 Module Data Sheet." *Guangzhou HC Information Technology Co., Ltd*. Olimex.com, 2011. [Web]. Available: https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf. Accessed Feb. 24, 2017.

[10] Williams, Mark. "Guide to Interfacing a Gyro and Accelerometer with a Raspberry Pi." *Ozzmaker.com*, 2016. [Web]. Available: http://ozzmaker.com/guide-to-interfacing-a-gyro-and-accelerometer-with-a-raspberry-pi/. Accessed Feb. 24, 2017.

[11] Wu, Lyndia C., Kaveh Laksari, Calvin Kuo, Svein Kleiven, Cameron R. Bass, and David B. Camarillo. "Bandwidth and Sample Rate Requirements for Wearable Head Impact Sensors." *Journal of Biomechanics*, 2016. [Web]. Available: http://www.sciencedirect.com/science/article/pii/S0021929016307448. Accessed Feb. 24, 2017.

[12] Carroll, Aaron, Heiser, Gernot. "An Analysis of Power Consumption in a Smartphone." NICTA and University of New South Wales, 2010. [Web]. Available: https://www.usenix.org/legacy/event/atc10/tech/full_papers/Carroll.pdf. Accessed April 24, 2017.

[13] Welsh, Erik, Murphy, Patrick J., Frantz, Patrick. "Improving Connection Times for Bluetooth Devices in Mobile Environments." Rice University, 2001. [Web]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=A9227FBFA56F370E2BFE458C9B39901E?doi=10.1.1.1015.2811&rep=rep1&type=pdf. Accessed April 24, 2017.

[14] "Specification of Bluetooth System." Bluetooth, 2004. [Web] Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40560. Accessed April 24, 2017.

[15] "MAX1722/MAX1723/ MAX1724." *Maxim Integrated Products*. Mouser.com, 2016. [Web]. Available: http://www.mouser.com/ds/2/256/MAX1722-MAX1724-97057.pdf. Accessed March. 26, 2017.

[16] "Samsung SD & MicroSD Card product family." *Samsung Electronics*. Farnell.com, 2016. [Web]. Available: http://www.farnell.com/datasheets/1836582.pdf. Accessed March 26, 2017.