

12-15-2014

Snowboard, Ski, and Skateboard Sensor System Application

Adrien Doiron
Santa Clara University

Michael Fernandez
Santa Clara University

Victor Ojeda
Santa Clara University

Robert Ross
Santa Clara University

Follow this and additional works at: https://scholarcommons.scu.edu/mech_senior



Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Doiron, Adrien; Fernandez, Michael; Ojeda, Victor; and Ross, Robert, "Snowboard, Ski, and Skateboard Sensor System Application" (2014). *Mechanical Engineering Senior Theses*. 42.
https://scholarcommons.scu.edu/mech_senior/42

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Mechanical Engineering Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Santa Clara University
DEPARTMENT of MECHANICAL AND ELECTRICAL ENGINEERING

Date: December 15, 2014

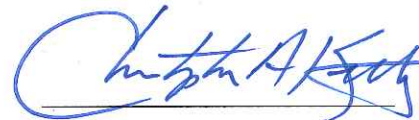
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Adrien Doiron, Michael Fernandez, Victor Ojeda, Robert Ross

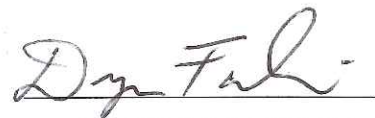
ENTITLED

Snowboard, Ski, and Skateboard Sensor System Application

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
BACHELOR OF SCIENCE IN MECHANICAL OR ELECTRICAL
ENGINEERING



THESIS ADVISOR



ME DEPARTMENT CHAIR



EE DEPARTMENT CHAIR

Snowboard, Ski, and Skateboard Sensor System Application

by

Adrien Doiron, Michael Fernandez, Victor Ojeda, Robert Ross

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements

for the degree of

Bachelor of Science in Mechanical or Electrical Engineering

School of Engineering

Santa Clara University

Santa Clara, California

December 15, 2014

Snowboard, Ski, and Skateboard Sensor System Application

Adrien Doiron, Michael Fernandez, Victor Ojeda, Robert Ross

Departments of Mechanical and Electrical Engineering

Santa Clara University

2014

ABSTRACT

The goal of this project was develop a sensor for the commercial market for skiers, snowboarders, and skateboarders that can give them the data such as speed, elevation, pressure, temperature, flex, acceleration, position, and other performance data such as trick characterization. This was done by using a variety of sensors, including a GPS, flex sensors, accelerometer, and others to provide data such as speed, position, position, and temperature. The sensors were placed in an external polycarbonate casing attached to the ski or board by using an adhesive pad on the bottom of the casing. These sensors then transmit the data via a microcontroller to either an LCD screen displaying a simple application or a memory system. The user can then access and analyze this data using Matlab code to interpret its relevancy. Using this system, performance data was recorded to analyze tricks such as spins and jumps.

Table of Contents

Abstract	iii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
Chapter 1: Introduction	1
Section 1.1 : Introduction/Background	1
Section 1.2: Review of Literature	1
Section 1.3: Problem Statement	4
Chapter 2: System Level	5
Section 2.1: Customer Definition and Needs	5
Section 2.2: System Requirements	7
Section 2.3: System Level Sketch	8
Section 2.4: Functional Analysis	9
Section 2.5: Tradeoff Analysis	10
Section 2.6: Team Goals	12
Chapter 3: Electrical System Metrics and Hardware	13
Section 3.1: Scope of the Project	13
Section 3.2: Metrics and Hardware Involved	14
Section 3.3: Component Breakdown and Block Diagram	15
Section 3.4: Component Breakdown	16
Section 3.5: Design Conflict	19
Chapter 4: Electrical System Physical Prototype	20
Section 4.1: Hardware Prototype Design	20
Section 4.2: Description of the Arduino Code Structure	23

Section 4.3: Design Iterations	27
Chapter 5: Housing/Casing Subsystem	28
Section 5.1: Preliminary Design	28
Section 5.2: Case Requirements	29
Section 5.3: Final Iteration	32
Chapter 6: System Integration and Testing	36
Section 6.1: Casing Finite Element Testing	36
Section 6.2: Thermal Testing	44
Section 6.3: Vibration and Damping Testing	50
Section 6.4: Dynamic Testing	54
Section 6.5: Sensor Testing	57
Chapter 7: Jump Tests and Data Analysis	57
Section 7.1: Interpreting the Accelerometer Information	57
Section 7.2: Data Graphs and Matlab Results	59
Chapter 8: Business Plan	63
Section 8.1: Introduction	63
Section 8.2: Costing Analysis	64
Section 8.3: Company Goals and Objectives	65
Section 8.4: Product Description	66
Section 8.5: Potential Markets	67
Section 8.6: Competition	67
Section 8.7: Sales/Marketing Strategies	69
Section 8.8: Manufacturing Plans	69
Section 8.9: Product Finances	70
Section 8.10: Service and Warranties	71
Section 8.11: Financial Plan/Investors Return	71

Chapter 9: Engineering Standards and Constraints	74
Section 9.1: Engineering Standards and Constraints	74
Section 9.2: Manufacturing	74
Section 9.3: Health and Safety	74
Section 9.4: Economic Factors	75
Section 9.5: Usability	76
Chapter 10: Conclusion	77
Section 10.1: Future Work/Upgrades	77
Section 10.2: Personal Reflection	78
Bibliography	82
Appendix 1: PDS	A1-85
Appendix 2: Timelines	A2-86
Appendix 3: Budget	A3-88
Appendix 4: Sensor System Coding	A4-89
Appendix 5: Vibration Table Data	A5-121
Appendix 6: Sensor Matlab Code	A6-124
Appendix 7: Consumer Needs Data	A7-128
Appendix 8: Dimensioned Final Casing	A8-129

List of Figures

Figure 1: Product Concept	6
Figure 2: System Level Sketch of Snowboard Sensor System	9
Figure 3: Hierarchal System Description	9
Figure 4: Adafruit Trinket Microcontroller	13
Figure 5: Arduino Mega Microcontroller	13
Figure 6: Component Block Diagram	15
Figure 7: Force-Sensitive Resistor and Neopixel 8 Stick	16
Figure 8: BMP180 Pressure and Temperature Sensor Peripheral	17
Figure 9: Adafruit Ultimate GPS Breakout Board	17
Figure 10: ADXL 326 Gyroscope and Accelerometer Module	18
Figure 11: Electrical System Prototype	20
Figure 12: Nose/Tail LED Strip System	21
Figure 13: Gyroscope Module and Calibration Button	22
Figure 14: Quick-Polled BMP180 Data Graphs	24
Figure 15: GPS Information Calibration Screen	25
Figure 15: Integrated System GUI Flowchart	26
Figure 17: Initial Snowboard Housing Design	28
Figure 18: Initial Ski Housing Design	29
Figure 19: Final Housing Design; Ski, Snowboard, and Skateboard	33
Figure 20: External Power Button for Casing System	34
Figures 21 and 22: Top View (Left) and Side View (Right)	35
Figures 23 and 24: Back View (Left) Isometric View (Right)	35
Figure 25: Vertical Loading Free Body Diagram	38
Figure 26: SolidWorks Displacement Analysis for Vertical Load on the Preliminary Square Casing	40
Figure 27: SolidWorks Displacement Analysis for Vertical Load on the Preliminary Square Casing	41
Figure 28: SolidWorks Stress Analysis for Horizontal Load on the Hexagonal Casing	42
Figure 29: SolidWorks Displacement Analysis for Horizontal Load on Hexagonal Casing	43
Figure 30: SolidWorks Displacement Analysis for Horizontal and Vertical Load on Final Casing	44
Figure 31: SolidWorks Stress Analysis for Horizontal and Vertical Load on Final Casing	44
Figure 32: Temperature Changes Within Casing To Equilibrium	47

Figure 33: Sinusoidal Wave Displaying Amplitude and Wavelength	51
Figure 34: Illustration of Vibration Test	52
Figure 35: Sensor Acceleration Data Due to Vibration and Board Acceleration Data Due to Vibration	53
Figure 36: Attenuation of Vibration Data	54
Figure 37: Acceleration Data for Static Drop Test	55
Figure 38: Acceleration Data for Static Shake Test	56
Figure 39: Acceleration Data for Snowboard Jump Test	59
Figure 40: Acceleration Data for Skateboard Jump Test	60
Figure 41: Acceleration Data for Snowboard Spin Test	61
Figure 42: Acceleration Data for Skateboard Spin Test	62
Figure 43: Projected Income and Expenses Over 15 Years	72
Figure 44: Projected Investors Return Over 15 Years	73

List of Tables

Table 1: Summary of Key Requirements	7
Table 2: Microcontroller Comparison	11
Table 3: Sensor Options	11
Table 4: Casing Material Comparison	30
Table 5: Material Properties for Acrylic Plastic	38
Table 6: Material Properties and Measurements	46
Table 7: Heat Losses due to Conduction	48
Table 8: Material and Fluid Properties	49
Table 9: Heat Losses due to Convection and Total Rate of Heat Loss	50
Table 10: Sensor System Test Data	57
Table 11: Preliminary Budget for Prototype Costs	64
Table 12: Final Budget for Prototype Costs	65
Table 13: Unit Pricing and Revenue Generation	70
Table 14: Production Cost and Projections	70
Table 14 (continued): Production Cost and Projections	71

Chapter 1: Introduction

Section 1.1: Introduction/Background

Skiing, snowboarding, and skateboarding are three of the most popular action sports, attracting millions of participants in North America alone (Statista). Because there is such a great and international interest in these action sports, companies invest heavily in research for new technology to bring the newest and best gear to athletes every year. Such new gear varies greatly, ranging from ski and board designs to the development of protective gear. Presently, with the advancement of computer technology, electronic devices have found their way into a variety of sports. In performance sports, such as running, gathering useful data and providing it to users is essential. Nike's sensor technology is one example of useful equipment created to benefit performance athletes by collecting data to assess their progress in training. For action sports, like skiing, snowboarding, and skateboarding, there are few options available for their athletes. Creating and designing such technology allows competitors and enthusiasts alike to track information and metrics on their performance such as speed, range of board movement, and the effects of elevation change.

Section 1.2: Review of Literature

Studying performance technology, such as Nike's, raises the question, "can this sort of technology be applied to action sports?" Technology is already on the market for some action sports, like cycling, which leads to question if this can also be applied to skiing, snowboarding, and skateboarding. In designing a new product, Nike sensors are the main source of inspiration. However, due to programming complications, using such technology is not feasible. By researching the current market, finding sensors used specifically for skiing, snowboarding, and skateboarding yields very few products. Those that are available, are yet to pass the prototyping stage and onto the market – leaving room for innovative creativity and design. A team from Michigan State University, in association with the Air Force Research Laboratory (AFRL), designed a number of prototypes – with some including features such as a global positioning system (GPS) (Bekkala). Nokia, in collaboration with the action sport powerhouse Burton, created a sensor system called PUSH Snowboarding – a system which monitors a snowboarder's

ride speed, heart rate, “head rush”, board orientation, and foot pressure (“Nokia X Burton – TVCs.”).

Neither product created by the teams above are on the market, as of yet, which leaves competing companies the opportunity to pursue research on the technology within these sensors. Combining sensor technology for performance and action sport athletes causes engineers, and athletes alike, to question whether the technologies be used comparatively. In fact, they cannot.

The most dominant of complications to arise is the types of motions that occur when comparing the movement of skiers, snowboarders, and skateboarders to that of a runner or basketball player. Participating in an action sport requires complex body motions in order to control one’s balance, direction, and speed. In order to change the orientation of the skis or board, a combination of movements from the torso, arms, and head must be accounted for, along with the important positioning of one’s legs and feet. Such a combination of movements and physical placement of limbs is not necessarily taken into account when designing sensors to track the fluid motions of running. Therefore, in order to create a product for action sport athletes the sensor would need to be modified in order to track and record the necessary data.

In skiing, snowboarding, and skateboarding, there are specific types of data that an athlete wants to track, the first of these being speed. However, recording one’s speed is not merely enough. Being able to maintain a log of one’s speed at specific points during the “run” is necessary in evaluating an overall performance. Secondly, in order to determine how far an athlete, specifically a skier or snowboarder, has descended on a run at a particular speed, data on elevation change and positioning must be collected. This sort of data collection also helps in evaluating how much airtime an athlete has following a launch or jump. The third piece of data collected is time: how long a run took, overall time spent in practice, etc. Collecting the temperature of the athlete’s environment follows, helping to evaluate if temperature has a substantial effect on performance. And lastly, the final piece of data is board flex, a tracking system for how much a user turns in accordance to the amount their board or skis flex. Combining the above data allows athletes, whether professional or recreational, to track their accomplishments and

improvements while enriching their overall experience. Table 1 on page 5 outlines our sensor requirements and options for choice.

In researching current products or projects with a similar objective to this design project, there are limited results in the market for such a specific audience. In terms of finding another snow sports product with similar functionality, the sensor created by Nokia and Burton, named PUSH Snowboarding, has four separate components that measures speed, orientation of the snowboard, heart rate, and altitude (“Nokia X Burton – TVCs.”). This particular project, however, is a continuing work in process still in its prototyping phase and therefore not yet reliable. Therefore, a more dependable comparison is one created by Garmin systems.

Although there is no snow-sport specific device made by Garmin, they boast being a primary leader in sport sensor technology. Beginning with the Garmin Forerunner, this product comes in the design of a runner’s watch. It measures what most advanced running sensors do now: calculating heart rate, speed, and route (“Garmin Forerunner”). What it does now, in addition, is read steps per minute, ground contact time, and height increases and decreases during the run (“Garmin Forerunner”). Using these three advanced measurements helps maximize the runner’s pace and rhythm with the comfort of looking at an LED screen watch. With all of these measurements, no phone is needed, as data is recorded straight to your watch.

The next Garmin product to compare is the Garmin Edge, a sensor for cycling. Similarly, this product does not require a phone or subsequent app while performing the exercise, as it records its information straight to the device. This particular product, however, is designed more like a car’s GPS navigation, as it not only looks like one but also attaches to a bike’s handlebars during the ride. It contains preloaded maps for both on and off-road trails, allowing the rider to adventure and explore without the worry of getting lost, with turn-by-turn instructions if needed. This product is heavier than the Forerunner by one ounce at 3.5 ounces, but also has a rechargeable battery – up to 17 hours – and is waterproof (“Garmin Edge”). As for the sensors specifically, the Edge displays current, maximum, and average speeds, distance, elevation, and time (“Garmin Edge”). Other sensors like power, heart rate, and cadence can be added to the Edge, but sold separately. This product attempts to create a device in a relatively new market for

bike sensors, similar to how this design project is trying to specifically target the snow sports market.

The last product used in comparison is the Nike Plus sensor. In comparison, it is the lightest in weight at .23 ounces, as well as the cheapest product on the market at \$19.00 (“Nike Plus”). The Edge reigns in at a price of \$300 while the Forerunner costs \$450 (“Garmin Edge”, “Garmin Forerunner”). The Nike Plus functions synchronize with a phone, as it displays all progress made in running by transferring the information to the phone wirelessly. The core sensors are very similar to the two Garmin products. However, unlike the other products the sensor is not water resistant, a requirement needed in order to accommodate snow activities. Also, the Nike Plus sensor does not have a direct display like the other two, which is why a phone is necessary to keep track of the progress.

Section 1.3: Problem Statement

The goal of this project is to develop and test a waterproof and shock-resistant system of sensors to be attached to a pair of skis, snowboards, and skateboards, in order to provide the user with useful real-time data about their runs – data which includes the runs’ speed, elevation, position, temperature, and board flex. In order to achieve this, several preliminary design goals are made to serve as milestones. The first design goal is to outline the customers’ needs, which involve estimating and setting standards for sensor accuracy, system durability, and overall price of the system. The next design goal is the functional analysis assignment; one that ensures the system detects information and can display it properly via the LCD screen. Following this, the goals then target the three main attributes of the system: mechanical, dynamic, and thermal components. For the mechanical components, the housing design is created and then prototyped into a physical model and then put under a mechanical stress analysis. With the thermal component, the system is designed and analyzed in order to confirm its capability of surviving cold weather and snow. Lastly, the dynamics component is analyzed to see if the accelerometers can effectively translate jumps, turns, and rotations with the exact movements of the board. The last major design goal is experimentation. To achieve such a goal involves testing for the thermal functionality of the system using a refrigerator, vibration table testing, and the accelerometer dynamic testing.

Chapter 2: System Level

Section 2.1: Customer Definition and Needs

The target customers are broken into two categories. First are the professionals: skiers, snowboarders, and skateboarders who may wish to track certain statistics or data so they have the ability to perfect their technique to enhance their performance. Data collecting information such as altitude, ski or board bend, speed, and GPS position greatly benefits professional athletes who participate in such events as the slalom – where such factors as the ones listed determine how effectively and efficiently the racer makes it to the finish line. The GPS unit works in tandem with the database during ski runs to provide professionals with an outline of their exact path in relation to the known course, their speed, and more, so that they can map their run and determine the optimum course

The second target customer is the active amateur skier, snowboarder, and skateboarder who wish to track their progress in order to improve, or simply to enjoy viewing how they performed. This type of customer is most likely a user of similar devices, such as a Garmin product, for other activities like running or cycling. This particular consumer is one who frequently uses sensor systems during a regular snow season, estimating use to be about 2-4 times per week for approximately 6 months – varying according to snow conditions, where they can expose the sensor to heavy powder, high winds, or high impacts. This, also, depends on the personal style of the rider, be they a free rider, park-rider, or a backcountry-rider.

To satisfy the needs of the customer, the sensor must be able to function in all of the conditions listed above, and be reliable enough to withstand any environment it is exposed to – particularly moisture and impact forces. Second, the longevity of the sensor, and its battery, is integral to customer satisfaction. For the purpose of this project's sensor, the goal for minimum longevity is a single snow season, which can last from October to June. In that time, the sensor system stores and makes the relevant data easily accessible to the user; this is done by creating an application in the sensor system itself along with Matlab code to provide trick analysis. The data, then, is organized so that the user can individually access a certain collection of data or multiple sets of data on one screen. The user can also activate or deactivate the sensor in order to target a certain section of their run. Figure 1 below shows the time lapse of a skateboard Ollie overlaid

with accelerometer data indicating when the jump was performed, landed, and the hang time in between each motion. This also can be implemented to sense the motion of other tricks. The goal is to provide an easy way for trick characterization to occur so that riders can data log their ski or snowboard runs or their skateboard sessions.

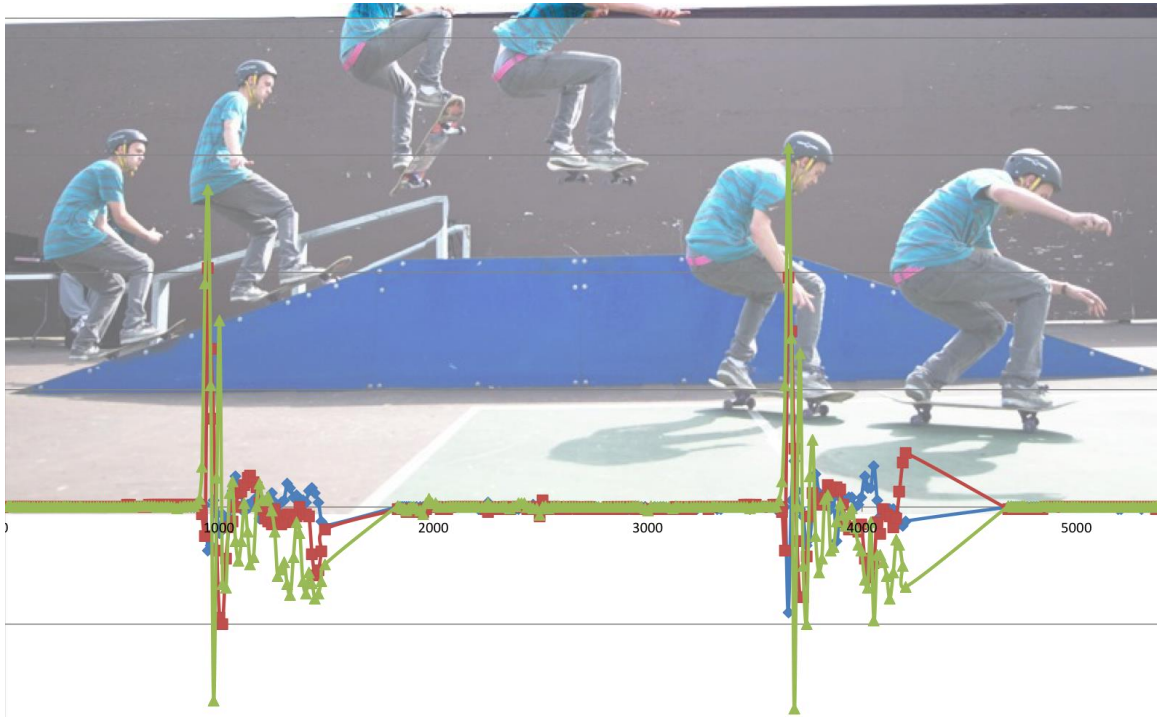


Figure 1: Product Concept

To find the specific needs of target audience, a survey was distributed amongst our peers on two social networks, Facebook and Twitter. The questions asked included; how many hours do you spend practicing your sport in its respective season?, would you like data on your velocity to be taken?, would you like to know your tricks' hang time?, would you like to know your board or ski rotational direction and degree amount?, where are you willing to have sensors placed (all that apply)?, and finally rank cost, size, durability, aesthetics, and simplicity from one to six in order of importance. The data from this questionnaire is tabulated in Appendix 7 with the Table on the next page summarizing key requirements based on responses given. According to this survey, the majority of potential costumers wanted the sensor system to record data on their velocity, jump hang time, and trick performance such as spin direction or amount. Also, the majority also preferred the sensors to be placed on the board or ski itself rather than on their person.

Table 1: Summary of Key Requirements

Rank of Importance	Customer Requirements	General Customer Preference	Our Requirements
1	Speed Data	Cost	Waterproof Casing
2	Jump Hang Time	Size	Multisensor Functionality
3	Board/Ski Orientation	Durability	Trick Analysis

Due to today's Social Media Culture, athletes take pride in advertising their progress and activities. Including a feature on the sensor that tracks and posts performance data to social networking sites, such as Facebook or Twitter, appeals to the target consumer – particularly those who have the means to afford a quality product. Assuming the target consumers have the means to afford such a product, they are likely to spend an average of \$300 or more. However, providing a top-tier product for a lower price of \$150 – 200 is significantly more alluring to the average consumer. Therefore, the sensor in question for this project must be represented adequately in our budget, seen in Appendix 3, at a cost of about \$200.

Section 2.2: System Requirements

There are certain specifications that the sensor must meet in order for the sensor to be considered a fully functioning prototype. To begin, it must satisfy the data requirements that the sensor displays to the user. Currently this data is speed, elevation, time, position, and acceleration. This data requires reliable storage so that the user is able to return to and review the data via computer or by a phone application, and it must also satisfy the accuracy constraints for each set of data. The system reads the speed to an accuracy of +/- 0.5 mi/hr, the altimeter to +/- 50 feet, and the timer keeping precision within a second.

The second set of requirements is structural, primarily dealing with the forces, stresses, cooling, and water factors. In skiing and snowboarding, the skis/boards are subject to a significant amount of force and stress. These are generated through a variety of factors such as turning, which causes the skis/board to bend and torque, or landing harshly after launching off a jump, which generates an extreme amount of impact force

upon landing. It is important that the sensor, and the casing it is placed in, is able to withstand these factors. Unfortunately, there is no current data on what kind of stresses and forces are generated in these situations, so these constraints are determined through experimental computations. The structure of the outer casing must also combat the effects of the extreme cold and frozen environment that the sensor can be subjected to. The cold temperatures impede electronic communication between the components, and if water should get into the casing and reach the sensor, there will be an immediate short in the system and effectively destroy the sensor. Therefore, it is vital that the casing around the sensor be waterproof and insulates the sensor well enough from moisture and temperature so as not to affect its performance.

Coinciding with data and structural requirements are the cost and price constraints. Performance sensors range in cost through great amounts, depending on what equipment they are paired with and what sort of data they provide to the user. In the case of the Nike Plus sensor, the sensor itself holds a price tag of \$19, but this does not include the required purchase of running shoes – which cost on average over \$100. There are other sensors available for other sports, such as cycling, which include a GPS unit as well as speedometers, altimeters, and heart rate monitors. A comparative cost is found by looking at the Garmin Forerunner and Garmin Edge, both running and cycling sensors. The Forerunner totals a staggering \$450 while the Edge compares at \$300 – both costs significantly higher than the physically smaller Nike Plus sensor. Therefore to reiterate, for the purposes of the ski, snowboard, and skateboard sensor of this project, the cost as seen in Appendix 3 is \$200.

Section 2.3: System Level Sketch

Figure 2 on the next page shows a photograph of the physical prototype, excluding the casing, including the various sensors and other components labeled. Eventually these components are mounted on the ski, snowboard, or skateboard, housed in a protective casing. The nose/tail bend sensors are mounted under a plastic laminate along with one of the bend meter LED bars on each end. The temperature and pressure sensor, GPS, gyroscope, LCD and Arduino microcontroller are soldered to a breadboard and mounted inside a waterproof and shockproof container. In the case of the snowboard

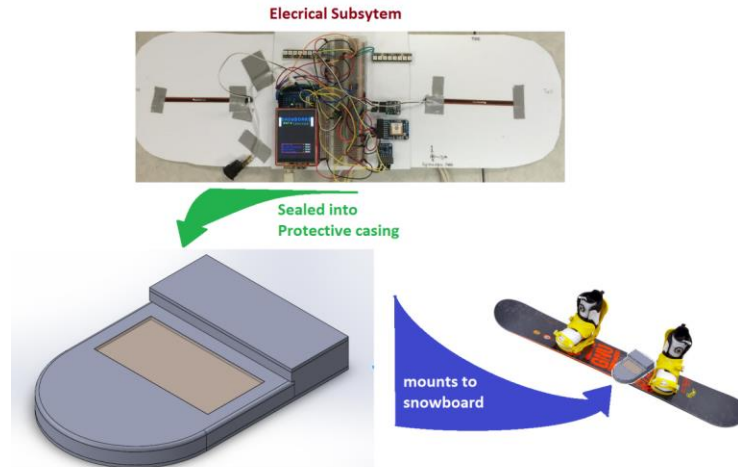


Figure 2: System Level Sketch of Snowboard Sensor System

and skateboard, this casing is then mounted to the board using a heavy-duty adhesive pad in front of the rider's rear foot to act as a "stomp pad", on which the rider rests their rear foot on while riding with it unbound. The electrical components are housed securely inside the casing to prevent it moving about while the board is in use.

Section 2.4: Functional Analysis

The end result of this design project is a product to be installed by the end-user on their ski, snowboard, or skateboard and communicate the information about their runs to the LCD screen. The goal is to create a single enclosure to be adhered to the board, in the place of the stomp pad. For skiers, the same circuitry works with a slightly different enclosure, mounted to the rear of the ski. This is done through a hierarchy system, displayed below in Figure 3.

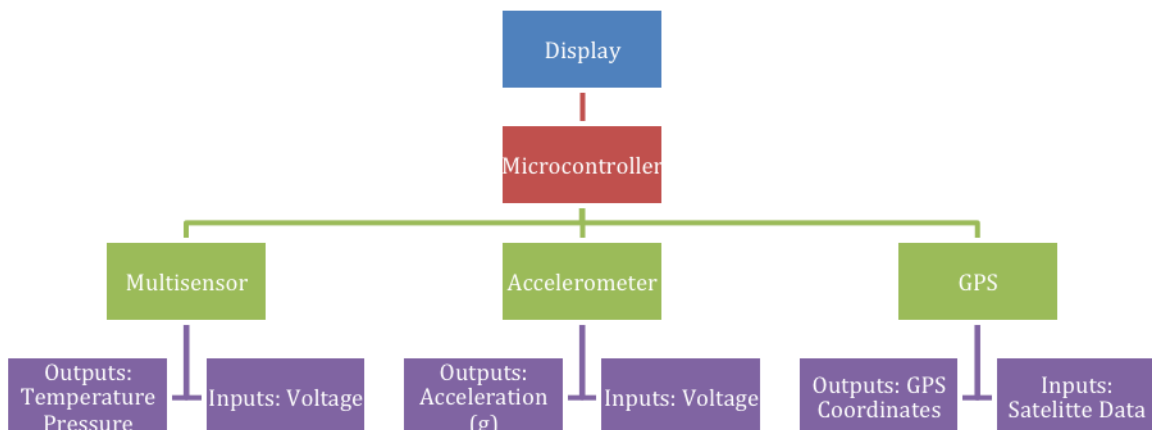


Figure 3: Hierarchical System Description

The system comprises of the three main sensors which link to the microcontroller: the multisensory, the accelerometer, and the GPS. The multisensory receives voltage inputs from the circuit within the sensor, which are then sent to the microcontroller and interpreted as temperature and pressure readings. The accelerometer, similarly, uses voltage inputs that are translated into acceleration readings in three axes before being sent to the microcontroller. The GPS, however, is different, receiving its inputs via a satellite connection with two or more satellites that send the information to the GPS unit. That information, too, is sent to the microcontroller. Once all data has been sent, it is processed via the Arduino code that was previously uploaded to the microcontroller. This allows the raw data outputs from the sensors to be displayed as meaningful data. The newly processed data is then sent to the LCD screen display, which can be viewed by the user.

Section 2.5: Tradeoff Analysis

The fundamental goal of this project is a slim, sleek design that is easy to use without sacrificing functionality. When selecting electronic components, power and functionality are sacrificed in order to minimize size. While working with existing knowledge, using commercial microcontrollers as the backbone of the system is the superlative choice in order to maximize functionality. Because these are commercially available, they exist as a standardized platform on which the whole system is based. By working with these existing systems, they are proven to be reliable for simplicity's sake, without losing functionality of the design. In terms of low cost and lightweight design, the Adafruit Trinket was the first option of controller. After further research, it was decided that the Trinket did not provide sufficient power to utilize the sensors wanted. The Arduino Mega Microcontroller was decided on as it has the ability to provide power through as simple USB power supply at between 7 and 12 Volts. The Arduino Mega also has 256KB of memory on board which provides sufficient space to store data for a short period of time (Arduino - Compare). The Mega's has the ability to store 4KB of data while the power is turned off as well. The Arduino Mega also has 16 analog inputs along with 54 digital inputs which provide significant support for multiple sensors (Arduino - Compare). When compared to components of similar pricing, such as the Arduino Leonardo and the Arduino Uno, the Mega trumps them in all of its specifications. The

Leonardo has only 12 analog inputs and 20 Digital, the Uno has only 6 analog inputs and 14 digital. Each also supports only 32KB of data storage which did not provide enough for our requirements (Arduino - Compare). Table 2 below shows the specifications for the above-mentioned microcontrollers excluding the Trinket.

Table 2: Microcontroller Comparison

MC	Operating Voltage	CPU Speed	Analog I/O	Digital I/O	EEPROM	SRAM	Flash
Uno	7-12 V	16 Mhz	6 and 0	14 and 6	1 KB	2 KB	32 KB
Leonardo	7-12 V	16 Mhz	12 and 0	20 and 7	1 KB	2.5 KB	32 KB
Mega 2560	7-12 V	16 Mhz	16 and 0	54 and 14	4 KB	8 KB	256 KB

This project is designed to be added to a user's existing equipment and, as such, must be versatile, but also very durable. By choosing to locate the circuitry enclosure within the stomp pad, it is subjected to considerable force. And in order to optimize durability of the circuitry enclosure, a durable acrylic composite is used as the main

Table 3: Sensor Options

Information Desired in System	Sensor Component Options	Basis for Choice
Nose/Tail Bend	Force Sensitive Resistor	Simple choice, changes analog voltage value based on bed
Bend Light Display	LEDs in all varieties	The Neopixel Stick 8 was chosen as the LED display for the bed sensors for their compact linear design with few contact points to worry about
	Adafruit Neopixel Stick 8	
Gyroscope / Accelerometer	MMA7361	Upon deciding, we were not certain of our final hardware configuration. We selected the ADXL line because it communicated wiz the arduino's analog voltage inputs rather than he SCL/SDA protocol which has less total inputs. We then chose the ADXL326 for the appropriate range.
	L3GD20H	
	ADXL 335 +/- 3g	
	ADXL 326 +/- 16g	
	ADXL 377 +/- 200g	
Speed / Location	Adafruit GPS Ultimate Breakout Board	Same features, half the price
	Dexter GPS Shield for Arduino	
Additional Information	BMP180 - Temperature, Pressure, Elevation	These sensors do not have tons of application in snowboarding as a sport, but temperature, pressure, and elevation, are all pertinent data about one's snowboarding experience
	Adafruit GPS Breakout - more accurate elevation	

portion of the multi-layered enclosure. The sensors incorporated give the ability to record the user's position, speed, direction, elevation, and temperature. With an elevation/temperature sensor, the bend sensors, and the GPS module, the user's entire

route down the mountain, including humps and hang-time, are able to be mapped with minimal hardware. Table 3 on the page above shows sensor options.

Section 2.6: Team Goals

As a team, we aim to accomplish several goals. The first being to create a fully functioning tested prototype; with fully functioning observing that the sensor is to accurately give the relevant data, within the prescribed constraints for accuracy, and be able to store the data with minimal anomalies or data loss. This goal may be augmented in the case it is decided to add on more sensors.

The second goal is to stay within the set budget. More specifically, the goal is to create a product that is made as inexpensively as possible – keeping the manufacturing and parts cost down to maximize the potential profit of this product. It is not intended to sacrifice the quality of the product, however, meaning that the quality of the components and materials used will be high enough to not cause the product to break or malfunction. With this, keeping the cost of the products relatively low is necessary in order to sell to consumers at a lower and more favorable price, making it more affordable and attractive.

The third goal is to heighten the ability to refine or add on to the prototype that is initially developed. The reasoning behind this is the desire to create a product that not only functions when it is in use by customers, but one that offers multiple features that will appeal to a variety of customers with room to apply other sensors in the future. Some of the features currently implemented include a GPS system, which has potential to increase the accuracy of the data, an altimeter, an accelerometer, a temperature sensor, and two bend sensors.

The fourth and final goal is to utilize the data received from the sensors and create a system or program that determines what “tricks” the user has performed by examining parameters of the data taken. Each trick has a “signature” of movement in the three axes, if performed correctly, making this information useful to the rider as he/she is able to understand and improve their skill level and/or correct mistakes that would normally have gone unnoticed.

Chapter 3: Electrical System Metrics and Hardware

Section 3.1: Scope of the Project

We wanted to get as much information as possible out of the sensor system, but we had to work within the constraints of making it a lightweight system on as minimal a budget as possible. Before we could determine the range of metrics we would be able to glean from sensors, we needed to choose a control scheme for the system. We opted for a microcontroller as the center of the system because of prior experience with them. In the interest of ultra-low-cost and lightweight design, we initially looked into the Adafruit Trinket to be our microcontroller. After we got a better idea of what we would need from our system, it became clear that the Trinket was not powerful enough. We then decided to work with an Arduino Mega, a higher-end microcontroller capable of running many hardware peripherals simultaneously.

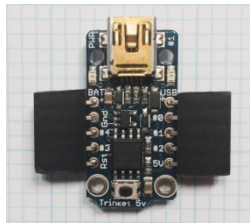


Figure 4: Adafruit Trinket Microcontroller

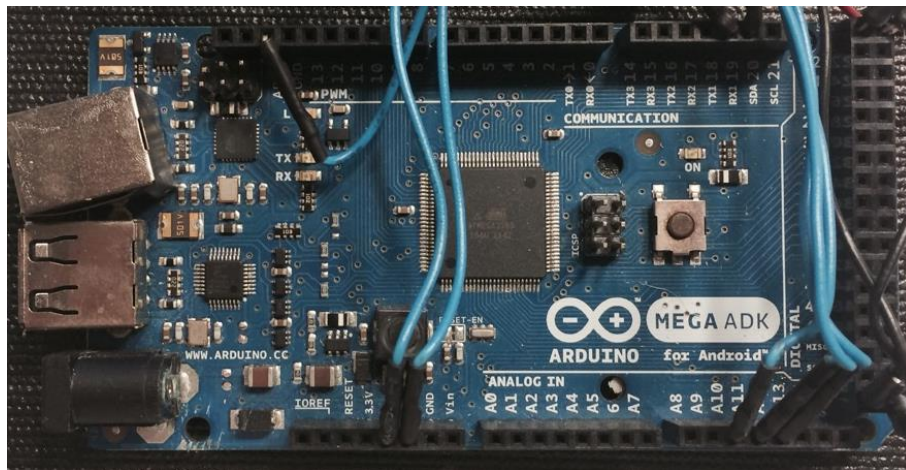


Figure 5: Arduino Mega Microcontroller

With the decision to use the high-power Arduino Mega and our inability as a team to incorporate interfacing data via a phone app due to design scope limitations, we instead opted to use an LCD screen designed for Arduino in our system. Now capable of a visual readout for the user, the system gained a sense of being a self-contained product.

Section 3.2: Metrics and Hardware Involved

In order to potentially become a competitive product, our system would need to take advantage of as many sources of data as are available, having made the choice to work with the Arduino Mega. We wanted primarily to be able to give as much information as is possible about the user's movement on the board. We also incorporated a GPS peripheral to track date, time, the user's speed, altitude, as well as the additional info of latitude/longitude. We decided to incorporate bend detection on the nose and tail in the case of use with a snowboard. Another additional component we decided to incorporate gave us the additional metrics of ambient temperature, and air pressure, which can be relevant in skiing/snowboarding as well.

Movement:

We decided to use an accelerometer peripheral to get information about the orientation of the board and the forces acting on it. This is the component that also enables us to determine the duration of a jump, as well as the direction of a spin, if present. The GPS peripheral enabled us to track speed, accurate to within .3 miles per hour.

Bend:

Bending on the nose or tail of the board comes about with various tricks. We used a system that incorporated a force-sensitive resistor to determine the flex on the nose or tail of the board. In the code it measures it generally, incrementing in ranges of 10% flex from 0 to 100%. From 20% - 100% flex, an LED strip on the board illuminates 8 LEDs in sequence.

Ambient Metrics:

We found one component capable of tracking temperature, pressure, and giving a rough calculation of the user's elevation. These metrics are gathered to render three time plots, being: temperature, pressure, and elevation. Elevation from the GPS is used when possible, because it is more accurate. These plots enable the user to view changes in these metrics over time (one hour, as coded) as they progress down a ski run or downhill longboard course.

The hardware making up these measurement subsystems will be discussed in greater detail in the next chapter.

Section 3.3: Component Breakdown and Block Diagram

The data-tracker operates via a network of four sensor-peripheral subsystems managed by an Arduino 2560 Mega Microcontroller. The microcontroller has a C program uploaded to it, which dictates the use of the sensor peripherals connected to its inputs and outputs. The complete hardware list for the various sensor subsystems consists of the following:

- Arduino 2560 Mega Microcontroller
- Seedstudio 2.8" TFT LCD display
- 2 x Force-Sensitive Resistors (as nose/tail bend sensors) with pulldown resistors
- 2 x Adafruit Neopixel Stick 8 (RGB LED arrays)
- Adafruit Ultimate GPS Breakout v3
- ADXL 326 Gyroscope/Accelerometer Module
- BMP180 Temperature and Pressure Sensor

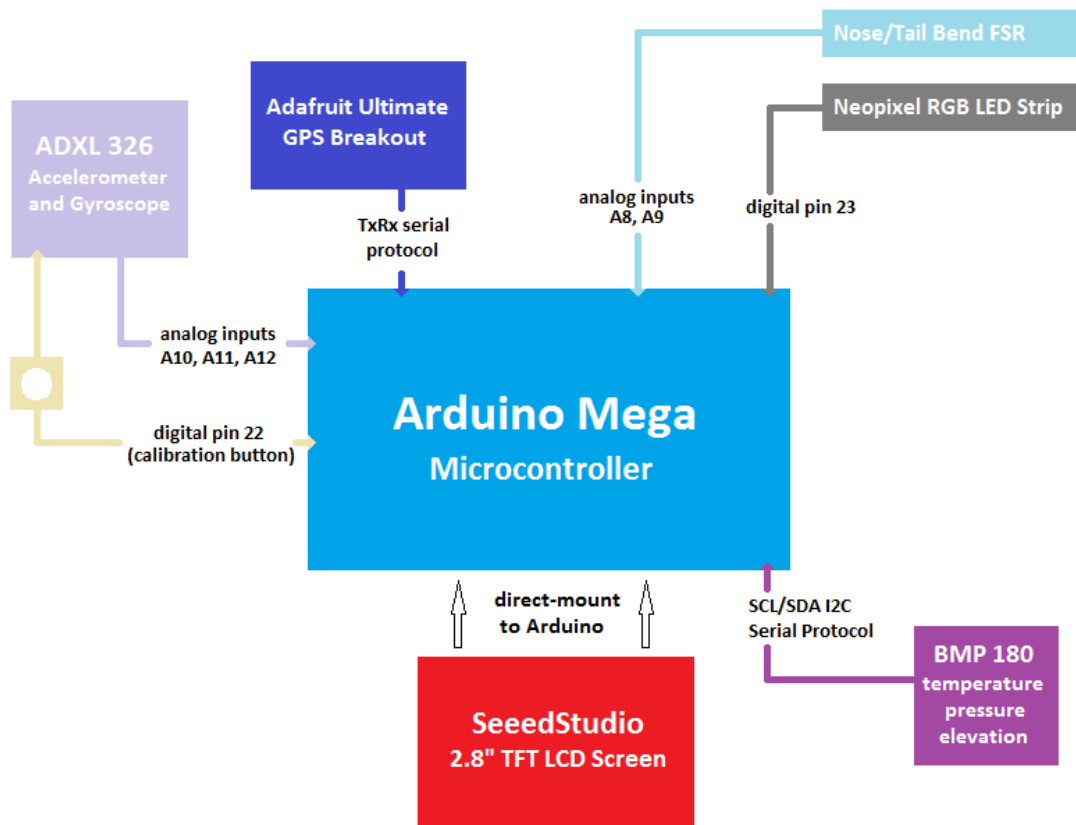


Figure 6: Component Block Diagram

Section 3.4: Component Breakdown

As discussed earlier, the Arduino 2560 acts as the brains of the entire system. Based on the C code, which will be discussed in greater detail in the following chapter, the Arduino operates hardware peripherals (the sensors, in our case) by controlling or interpreting the voltages at its multiple inputs and outputs. The majority of the outputs of this board are taken up by the system's most hardware-intensive peripheral, the Seeedstudio 2.8" TFT LCD display, used to display the information as measured and interpreted by the system.

The LCD display requires a significant amount of space – roughly half the inputs of the Arduino and taking up two-thirds of the area, meaning that the screen is conveniently mounted directly to the board, which simplifies the needs of casing design by being mountable as a single component. While the TFT mounts directly, the remaining components connect through leads to the additional inputs and outputs that remain on the Arduino. The first of these components we will breakdown and analyze is the simplest, the bend-detection system.



Figure 7: Force-Sensitive Resistor and Neopixel 8 Stick

The components in figure 4 make up the bend sensor on the nose and tail of the board. Each end of the board is equipped with a 10cm FSR secured with waterproof plastic laminate. The FSR is given a 5v potential and the voltage drops as the resistor is bent. The Arduino board divides the range of values into sections, between flat and the bend of a full tail flex on a snowboard. Through experiments, it is determined that a proper maximum bend, and the subdivisions between flat and full, correspond to how many LEDs are illuminated. This was done in order to figure a sense of how deep the last nose or tail bend was and to add a stylish component to make the product stand out by using the colored lights.



Figure 8: BMP180 Pressure and Temperature Sensor Peripheral.

The above component, and the next most complicated component in our breakdown provides the Arduino with information about both the temperature and barometric pressure of the surrounding environment. The information is communicated to the Arduino via the SCL and SDA pins using I2C communication. This provides fast communication between the component and the Arduino using only a few inputs, which is ideal given the choice of the LCD screen. The component comes with its own libraries of functions for converting the sensor data to useful information, which means the board can be easily made to work with a number of different systems. The code provided by Adafruit, the manufacturer, is easy to modify and use within the display loop code.



Figure 9: Adafruit Ultimate GPS Breakout Board

To get position and speed information, a Bluetooth module was selected as the best choice. The Adafruit Ultimate GPS Breakout Board provides accurate and reliable information via a satellite link from one of twenty-two dedicated channels.

The unfortunate downside to the convenience brought by GPS is that the information is only available while the satellite link can be established. In some cases, the satellites cannot be reached and the breakout board can therefore not function. The majority of the time, the satellite link is established without a problem, however, code needed creating for this error case in the system display loop.



Figure 10: ADXL 326 Gyroscope and Accelerometer Module

To get information about the forces exerted on the board during use, we used the ADXL326 gyroscope module. We selected this module based on the +/- 16g sensitivity range which best fit our purposes compared to the other models in the product range which ranged from +/- 2g to +/- 200g. The peripheral outputs three analog voltages corresponding to readings on the X, Y, and Z axes. It also outputs a 3v output signal which is used by the analog reference of the Arduino Mega to compare the axis outputs against when interpreting the axis readings.

The AREF pin does this, however, it is taken up by the bulky TFT display. The inability to compare against the 3Vo pin on the ADXL326 resulted in a design conflict (discussed in greater detail in the next section) and ended up with a loss of precision on the gyroscope readings. For the purpose of dynamic testing, the TFT display was removed so the gyroscope/accelerometer could function with the greatest possible accuracy. This is a design conflict we encountered which is discussed in greater detail in the next section.

Section 3.5: Design Conflict

Due to the nature of taking up so many of the primary pins on the Arduino, the Seeedstudio TFT screen presented us with a design conflict. The screen required use of the AREF (analog reference) pin in order to function. The AREF pin is used to provide the comparison point by which analog input voltage is judged as it is interpreted. The gyroscope module as well typically demands use of the AREF pin in order to provide more accuracy by judging against a reference voltage provided by the module itself. Without use of the reference voltage, on-the-fly scaling of gyroscope measurements were not possible via hardware, and instead required workarounds to function. The first of these workarounds, present in the display loop code is the use of a simple arithmetic scaling to readjust the input voltage to scale it directly by $5/3.3$ volts because the 0 to 3.3v incoming voltages were interpreted on the range of 0 to 5v, which is the AREF the TFT is operating by. The simple scaling of this theoretically would reset the analog readings to the expected range, but there was still a loss of accuracy. This loss of accuracy originates from the fact that the one-time calculation is relied on as a best guess for rescaling the voltage, but in actuality, all minute fluctuations in the function of the gyroscope peripheral would show up only in the input data being fed from the sensor and would not show up in the AREF which would serve to adjust for them by minutely fluctuating along with the input data. Since the AREF is not usable via hardware for on-the-fly readjustment, any fluctuations in the sensor data cannot be adjusted for.

Chapter 4: Electrical System Physical Prototype

Section 4.1: Hardware Prototype Design

In order to prove the feasibility of the product's designs, a physical prototype of the board was constructed using the aforementioned sensor peripherals and electrical components mounted to a flexible plastic backing. The peripherals were wired to a central large breadboard which provided ample room to prototype each of the components with the Arduino.

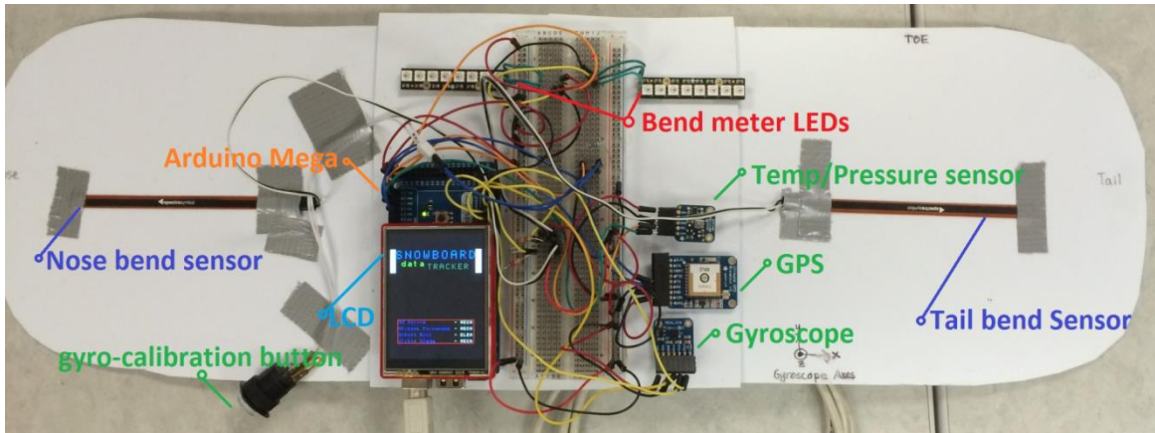


Figure 11: Electrical System Prototype

Most of the components making up the prototype board in figure 1 provided their own libraries of functions online by which they operate. Generally these library functions were all that the component needed to get up and running and spitting out whatever information it gives. It was then our task to program the display loop to work with the variables and information laid out in the peripheral's libraries.

Knitting each of these various libraries of functions into a single programming script involved the use of a display loop controlling the LCD as the central framework for the code. The various functions that were run by each of the peripherals can begin to be calculated while another is being displayed, and then they can be displayed in turn. As coding progressed, blocks of code and functions were timed to get a sense of how long they require to process. By knowing this, the code that gathers the data from each of the sensors and make the calculations could be run while the LCD is idling displaying a screen.

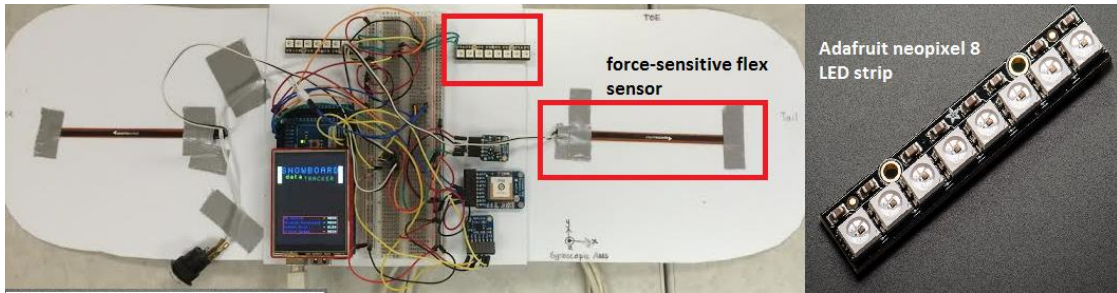


Figure 12: Nose/Tail LED Strip System

The Nose/Tail LED strip system in figure 2, being simply a visual element, may seem like an element that would not necessarily be a design priority, but, whether or not it is commonly articulated, style is an omnipresent element of snowboarding. The unique attire and broad array of outfits seen on the mountain demonstrates the held importance, by many riders, of personal style. Its prevalence in the sport is even seen in the vernacular, with the combination of style and ease to a rider's trick being lauded as 'steazy.' To technologically expand a rider's personal relationship with snowboarding, it should do so on all levels, not simply relate to athletic or statistical aspects of the sport. In order for a new product to be truly revolutionary in the constantly overhauled industry of microelectronics, it must be noticed and enjoyed widely and organically.

The circuit involved is comprised of two force-sensitive-resistors (or FSRs) designed to increase in resistance as it is flexed. The two FSRs are mounted on the ends of the board, near the corresponding LED strip, via clear plastic laminate to the nose of the board, the other to the tail. One pin of each FSR is connected to 5v (Arduino logic HIGH voltage), the other pin connected through a pullup resistor to ground. This forms a voltage divider between the pullup resistor and the FSR, which is measured and interpreted by the Arduino Mega via analog input A8 for the tail, and A9 for the nose. The Arduino C code involved compares the analog reading against ten value ranges representing 10% increments between flat (0 degrees) and a full bend (~45 degrees). Values between 0 and 20% are ignored (no lights illuminate) attempting to eliminate false-positives from ordinary bumps and wobbles on the ride rather than intentional nose or tail bends. Values between 20% flex and 100% flex (compared in ten percent increments) illuminate NeoPixel LEDs 0 through 7 down the strip, corresponding to one tenth each of the remaining 80%, with colors fading from green through yellow to red.

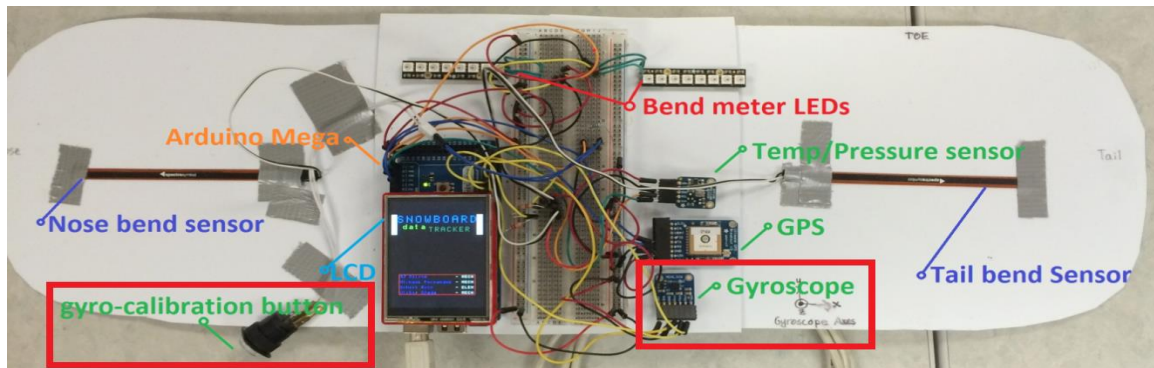


Figure 13: Gyroscope Module and Calibration Button

The ADXL 326 Gyroscope/Accelerometer module proved to be an effective selection for judging the angle of the snowboard relative to the horizon along three axes (nose to tail, toe to heel, and vertical) and could do so quickly and accurately. The complicated and state-of-the-art micro-technology that enables the precision of this component is described by the retailer, Adafruit.com.

“The sensor consists of a micro-machined structure on a silicon wafer. The structure is suspended by polysilicon springs which allow it to deflect in the when subject to acceleration in the X, Y and/or Z axis. Deflection causes a change in capacitance between fixed plates and plates attached to the suspended structure. This change in capacitance on each axis is converted to an output voltage proportional to the acceleration on that axis.”

Adafruit Learn: ADXL 326

(<https://learn.adafruit.com/adafruit-analog-accelerometer-breakouts>)

Though the hardware is complicated and state-of-the-art, a library of functions is included to simplify the coding process and data gathering. The key information the accelerometer is used for is the information just before and after an impact, to give information specific to the movement the rider just performed. In our current stage of prototyping, the accelerometer requires a serial connection with a computer, because the data logging is done by a third-party program called CoolTerm. The program can log incoming serial data in spreadsheet form. Further development of the system would be needed to store and parse information on the fly as we do not currently have a system which can interpret this data in a useful way on the Arduino, nor the memory to do so for multiple runs. The present prototype gives a readout via the LCD display or more

complete information by interfacing via CoolTerm on a computer. We do however have the ability to analyze a jump or drop performed on the board in a test for the forces that impact it. By opening them up in Matlab and analyzing the data on each axis as an array, the net directions and magnitudes of forces can be determined. This Matlab method of analyzing the generated spreadsheets is discussed in greater detail in the next chapter.

The remaining components, namely the Adafruit Ultimate GPS Breakout v3, and the BMP180 temperature/pressure sensor work by using the included library functions, but breaking the data apart to be displayed independently, by the LCD. The functions themselves grab samples of data but it is in our display loop that the data from those functions is sorted and worked with. The structure of the display loop will now be discussed in greater detail.

Section 4.2: Description of the Arduino Code Structure

Standard Arduino scripts operate the inputs and outputs of a circuit board based on a programming loop that it runs continuously. Before beginning the programming loop, it first executes a 'setup' function which contains whatever initial settings, declarations, or other lines of code are needed to then begin the main programming loop. Each of the components we built our system out of has their own basic demo script online, which demonstrates the functionality of the device, containing a setup section of code and a loop section to be uploaded to the Arduino. Our task involved rewriting the various setup scripts as their own which then each get called in a master setup script. We then rewrote the various loop routines as callable functions to be called repeatedly when that information is called for, as determined by a display loop routine controlling the LCD which provided the framework of the master loop.

The LCD display loop provided the structure to our programming loop because it is a straightforward matter to query the devices one at a time, while rotating through which one gets displayed. The device operates in a demo-ready display mode which contains two major changes from what would be used in a device on the mountain. The first of these changes is that the BMP180 loop (the graphs of temperature, pressure, and elevation data points over time) is sampled once every second instead of once every minute. This choice was made to gather a number of data points quickly by the first time the graph was displayed in the rotation. On an actual mountain-ready design, the delay

time on polling this data would be changed to one minute rather than one second, so as to gather a plot of the last hour. This change of rapidly gathering the data does display the functionality of the device, but the graphs appear much more erratic than those gathered over the course of an hour.

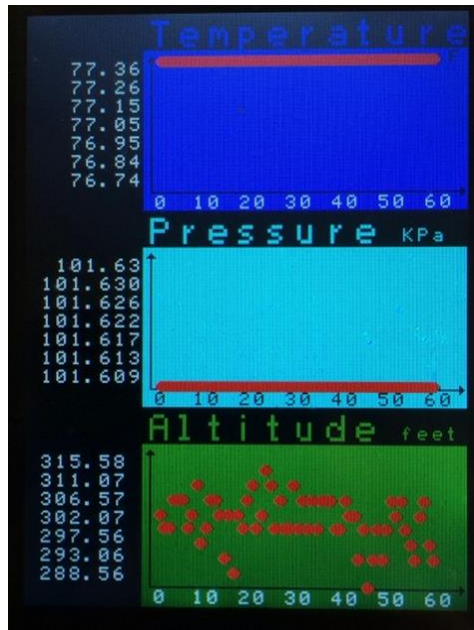


Figure 14: Quick-Polled BMP180 Data Graphs

As can be seen from the graph, the altitude plot is rather inconsistent. Over a longer time and trips down various ski runs, the altitude graphs would look more gradual and cohesive. The least accurate and reliable aspect of the BMP 180 is the altitude calculation. Since it is simply calculated mathematically, there are two solutions utilized to provide more accurate values for elevation. The primary solution solves the problem with much greater precision than the BMP180 is capable of, by getting altitude information directly from the GPS peripheral, provided it detects a reliable satellite link. If the GPS does not detect a reliable link to the GPS network, it instead calculates altitude from a floating point value of the specified sea-level pressure at that location and time. It unfortunately requires a given specified sea-level pressure, and is most accurate when the given day's information is looked up and input into the code. This is not feasible for a product-ready design, so the GPS is relied on as the primary source of elevation data.

The code directly involved in the calculation and display of these values is tedious and lengthy. It is found in the appendix, rather than taking the space to list it here.

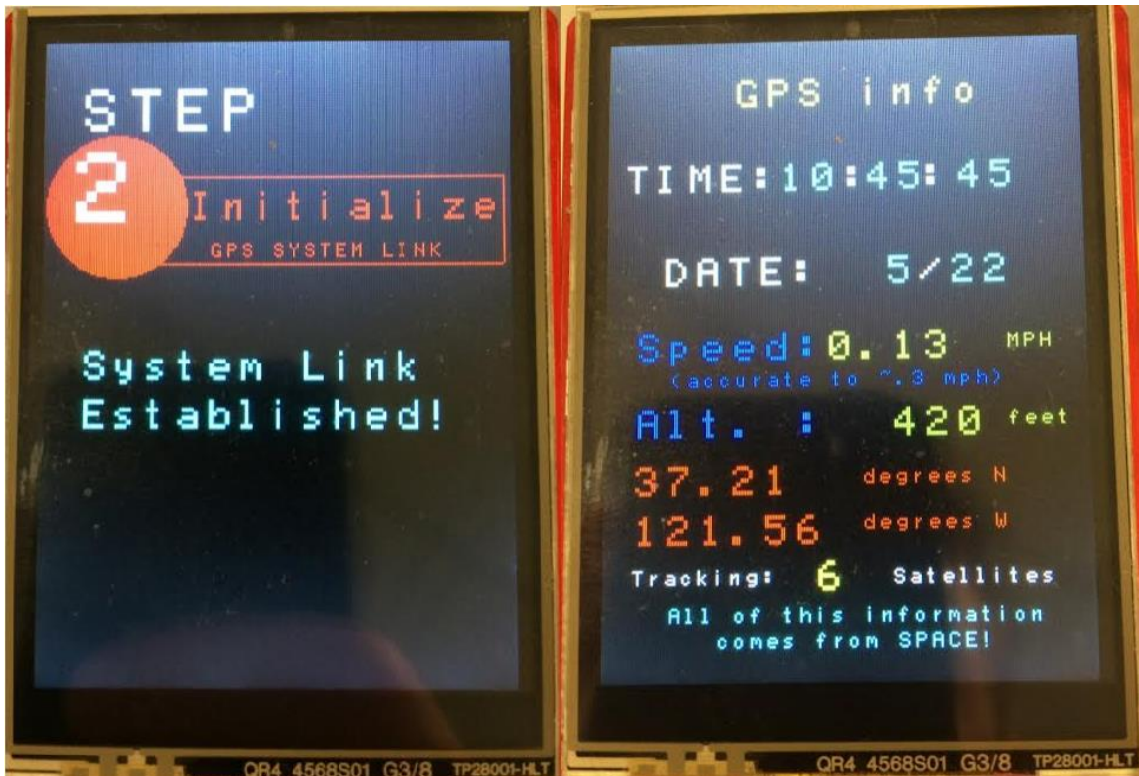


Figure 15: GPS Calibration and Information Screen

Above is pictured the display screen sequence for establishing a GPS link. The Adafruit Ultimate GPS breakout v3 is a low-power board based around the MTK3339 chipset that communicates with satellites having access to 66 communication channels, capable of ten updates per second. The Adafruit_GPS library contains handy functions to draw the specific values for date, time, position, elevation, etc. out of the NMEA sentence transmitted by the satellite, so they can be worked with and displayed by the Arduino during the GPS display loop shown in the figure. This information, though presented in raw form is valuable, and could easily be used to establish additional information about the user's Ski/Snowboard experience. An example of what features could stem from this include, for example, a database of different popular ski locations, which, using this GPS data, could automatically narrow down the user's location to a particular mountain or resort. Additionally, the user's speed down the mountain is measured with an accuracy of .3 mph or better. Using the date and time, as well as trends in elevation, the system could determine when given runs begin and end as well as distance traveled, top speed, and average speed. The entirety of the display loop can be seen in Figure 15, the integrated system GUI flowchart on the next page.

Integrated System GUI Flowchart



Figure 16: Integrated System GUI Flowchart

Section 4.3: Design Iterations

The physical prototype demonstrated in the previous chapter was initially what our team considered to be the final design of the project. The concept behind the design was centered around the idea that we as a team were trying to create a fully self-contained micro-controller system based on ideas of our own design. In that regard we as a team succeeded. The programming and data management were contained entirely in the micro-controller and sensor peripherals. When we connected the system to an initial flexible prototype board and powered it with a battery pack, the design took on a feeling of having been an achievement. By holding the actual device and seeing the data readouts, illustrated in the GUI System Flowchart in the previous chapter, our project gained a sense of being along the lines of a potential product. It was rather satisfying to see it in this state.

The project in this form was, however, considered an incomplete project by our advisor and was short of a few of the design goals that we had began with. Our project needed to be extended, and as different members of the team went home over the summer, the hardware was reduced down as we focused on getting more useful data from the accelerometer.

To complete our project, we decided to augment it to include greater functionality of parsing the data on the accelerometer. We used a third party program called CoolTerm which was capable of connecting to the Arduino serial connection and log the incoming accelerometer values into a spreadsheet for Excel. We then wrote a Matlab script that was capable of parsing the excel formatted spreadsheet and from summing and analyzing values in the data arrays, we were able to determine with reasonable accuracy the direction of spins performed and the duration or hang-time of a jump. This is all discussed in greater detail in the next chapter.

The current state of our project is the accelerometer data only. Previously we had our self-contained design as discussed in the physical prototype section. That design was displayed and demonstrated in the oral presentation last spring. The primary illustration of that design's functionality is seen in the illustrated system GUI Flowchart. A more in-depth discussion of the accelerometer data parsing happens in the following chapter.

Chapter 5: Housing/Casing Subsystem

Section 5.1: Preliminary Design

For sensor systems designed for skiing or snowboarding, the enclosure consists of a durable three-layer system. The bottom layer is a flat plate that is the mounting place for all of the sensors and the battery, the middle layer is a rubber toughened cyanoacrylate adhesive to provide a watertight seal, and the top layer is the dome shape that encloses the sensors from the elemental world.

For the preliminary designs, each housing for both the snowboard and the ski implement a dome shaped cover that will have one single opening for a recharge inlet so that the case never has to be fully opened. The reason that a domed top is used is to deflect impacts to the top surface, where a flat top surface would not deflect these impacts. A hexagonal design is to be used on the snowboard, as shown below in Figure 34.

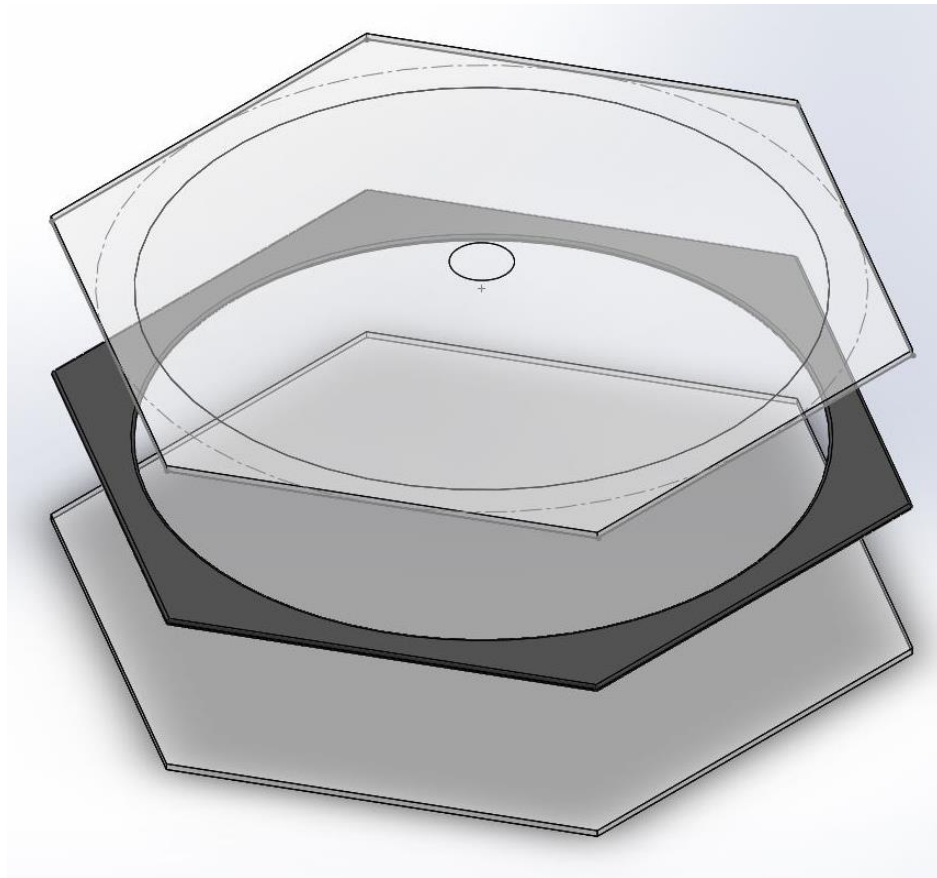


Figure 17: Initial Snowboard Housing Design

The purpose of such a large encasing is that it replaces the stomp-pad on the users board. This is so that there is less extra clutter on the board itself. Besides its shape, the main difference between the ski and snowboard enclosures is the grip that is on top of the hexagon in order to ensure the rider stability in using the new sensor stomp-pad. This grip is custom created from stompdesign.com in order to make sure the charging port is still accessible.

For the ski, a simple domed square is designed (shown below in Figure 35) as the goal is to make the sensor enclosure as small as possible.

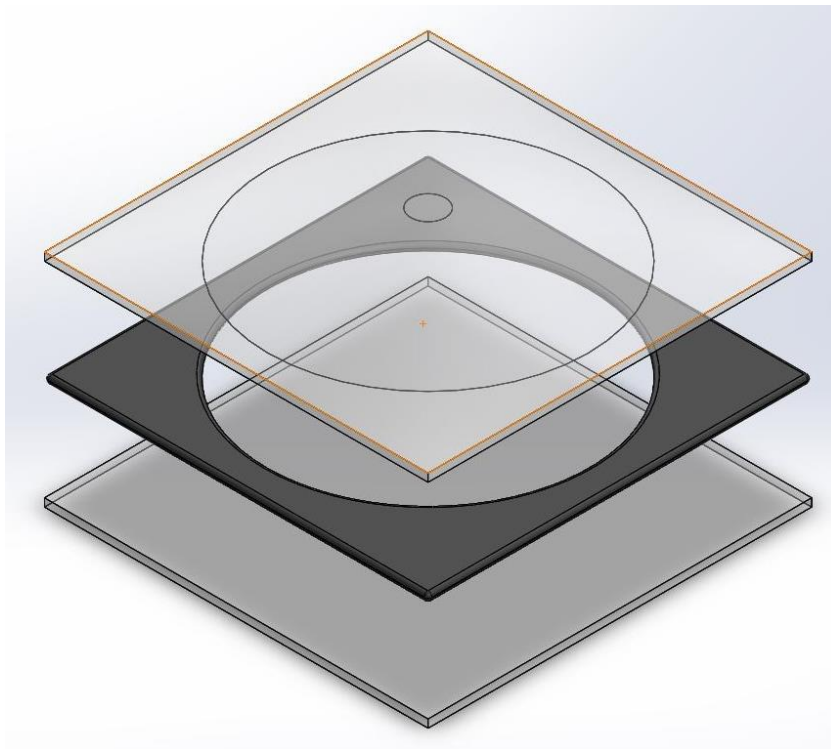


Figure 18: Initial Ski Housing Design

Section 5.2: Case Requirements

The housing for the microcontroller and other sensors must withstand abuse from both the human user and the elements around the ski or board. This means that the housing must be highly impact resistant and 100 percent waterproof to ensure the integrity of the sensor system. A range of materials is considered, the original choice being polycarbonate as the best fit for the sensors' protective housing. Metals, such as aluminum, were not valid options as they are vulnerable to deformation or are too heavy. Other plastics, such as polyethelene, were not viable as they are too soft and

can be damaged easily, as well as being difficult to machine. Polycarbonate is a highly impact resistant material that is injection molded while still having properties to make it machine-able once it solidifies.

Table 4: Casing Material Comparison

Material	Advantages	Disadvantages
Aluminum	Light, Machinable	Vulnerable to Corrosion, Impacts, Deformation
Polycarbonate	Light, Durable, Impact Resistant (Heavy Impacts)	High Cost, Vulnerable to Heat (Manufacturing)
Polyethelene	Cheap, Machinable	Soft, Vulnerable to Impacts and Deformation
Acrylic	Durable, Impact Resistant (Medium Impacts), Clear	Vulnerable to Heat, Abrasions, Severe Impacts

While the specific mixture of polycarbonate is yet to be determined, there are a few key properties that are not negotiable; the material must be UV resistant, must have no glass fiber reinforcement (as this reinforcement handles heat stress well but impact stress poorly, and the glass fibers cause the material to shrink approximately 0.3% during the curing process), and the material must be BPA free.

To manufacture the casing using the polycarbonate, there were two options: machining the various pieces and assembling them using an adhesive, or using an injection molding process. The advantage of using the machining process is that it does not require specialty pieces to manufacture the casing, allowing the easier manufacturability on a smaller scale. The injection molding process is the perfect option for a larger scale operation, as you could produce more casings faster as well as immensely minimizing the deformations, but it would require the creation of specialty molds. As a result of the incapability and lack of practicality in making a specialty mold for our purpose of making singular prototypes, the machining process is the selected option. Once the prototyping phase is completed in the future, the manufacturing process would be changed over to an injection molding process.

However, as a consequence to choosing the machining process, issues arise with the material polycarbonate. At the target thickness, it is considered a bit on the

thin side for a polycarbonate undergoing the machining steps sought. The results of performing these intricate cuts would cause a lift on the edges of the polycarbonate that are being cut as well as making a mess of the edges. The effect of lifting the edges of the polycarbonate sheet bends the original flat state of the sheet, and carries the possibility of creating stress cracks which would ultimately compromise its overall strength. On the other hand, the thickness can be considered too large for a polycarbonate in terms of induced heat. The created heat from milling the polycarbonate is not ideal as it causes discoloration and possibly small bubbling. Its thickness also undoubtedly rules out laser cutting as a possibility. Polycarbonate strongly absorbs infrared radiation, which is the same frequency that the laser runs on, in turn making cutting polycarbonate very ineffective. This is not to mention that any polycarbonate having a thickness greater than a millimeter carries the possibility of catching fire.

The milling issues were not great enough to overcome, though the potential of producing inconsistent cuts was deemed a significant negative. In the end however, it was simply finding a more attractive solution that made ditching the polycarbonate so easy. The material acrylic obtained similar desirable properties that the polycarbonate possessed. It was a lightweight, transparent, and shatter resistant material to use in the place of a glass structure which would be prone to breaking easily. In terms of safety, the acrylic is significantly more impact resistant than glass-type materials. If it were to break, it would do so in relatively blunt large pieces instead of tiny slivers of material that are comparatively much more dangerous. In addition, because the thickness is well in between .08" and .5", it passes the safety requirements of ANSI Z97.1 for window glazing materials (Cryo).

Acrylic was also chosen over the original polycarbonate material because it is less prone to scratching. Considering that we are assembling an item for private consumption, the aesthetics are considerably important. The more scratch resistant it is, the more pleasing to the eye it becomes. In addition to looking better, it better serves its function of displaying the information from the LED screen inside the housing to its user. The acrylic also provides a more rigid structure as polycarbonate is more flexible. Although polycarbonate is more impact resistant and cracks less

easily, acrylic was chosen as it is cheaper and still provides enough impact resistance when compared to glass. Acrylic was also chosen for its weather resistance, it has the ability to withstand exposure to strong sunlight, extreme cold, and quick temperature changes, as well as providing a waterproof shield when sealed properly (Cryo). These are very important factors when considering the product can potentially be exposed to both ends of the weather spectrum in its use, ranging from the cold and melting snow to hot and sunny days. The only other weakness to this material is that acrylic sheets can expand and contract in cold, heat, and humidity (for a 48" panel, approximately 0.002" per each degree Fahrenheit change). In spite of this behavior, these values were found to be negligible to the small size of the casing structure (Cryo).

For the adhesive to connect the acrylic pieces of each complete housing, another acrylic solution is used. The solution needed to be able to provide a waterproof seal as well as withstand severe impacts and temperatures. There were several different choices, such as 3M Plastic Adhesive 1099, which cures quickly and provides decent qualities, or 3M Plastic Adhesive 2262, which is a clear adhesive for materials that flex frequently. The choice we made though was Apollo 2241. Apollo 2241 is a highly viscous, rubber-toughened ethyl cyanoacrylate adhesive that provides high shock and thermal resistivity when bonding with plastics in harsh environments. This adhesive will also provide a watertight seal around the sensors. 2241 is chosen as it has a high tensile shear strength of 3700 psi while still having a large operating temperature range between -65 degrees F and 280 degrees F ("Apollo 2241").

Section 5.3: Final Iteration

To connect the sensor enclosure to the board or ski, the casing had to remain firmly in place without it becoming dislodged but also had to be removable. For this, there were two options: either bolt the casing to the board by drilling into it, or use a silicone rubber with an adhesive coating. To maintain the integrity of the board or ski, as well as reduce the amount of labor to install the product, the silicone rubber with an adhesive coating was determined to be the better option. This is also a bad idea due to the acrylic material's cracking weakness to any hole drilling near the edges. Therefore, to connect the sensor enclosure to the board or ski itself, a silicone rubber

with a pressure sensitive acrylic adhesive coating is used. Because the silicone rubber has a high resistance to impacts and a good resistance to the stresses that will occur on the board, it is the most viable option. The ultimate bonding strength is measured to be 150lbs/inch of width. Figure 36 below shows the final iteration of the casing.

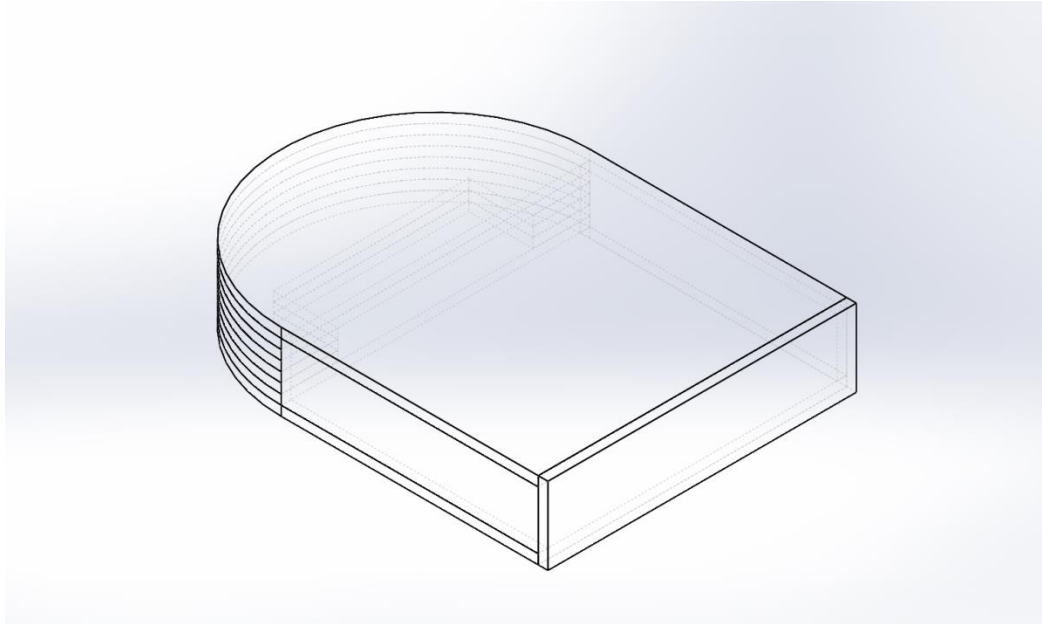


Figure 19: Final Housing Design; Ski, Snowboard, and Skateboard

The fully dimensioned final casing is shown in Appendix 8.

In order to access the power button to turn on the battery, a hole hovering over the button is drilled. The reason for this is so that the top of the casing does not have to be removed to turn on the system. This is a clean hole with no threads in order to allow a pin that is to be machined with the lathe, to freely slide vertically. The pin design is a cylinder with a wider but shorter length cylinder base at the bottom, which measures slightly smaller in diameter to the diameter of the power button. The longer, thinner top cylinder of the pin travels through the hole in a tight fit, with the other half of its length poking over the top of the casing's ceiling. This creates a simple fixed mechanism that is restricted to only moving up and down. The material used is Teflon. This material was selected among other poly-carbonates because of its soft texture, as well as its extremely low coefficient of friction making it effortless for the owner of the product to push the pin down. The top of the pin is also chamfered around the edges in order for the user's fingers to avoid sharp edges. This button is pictured in Figure 37 below.

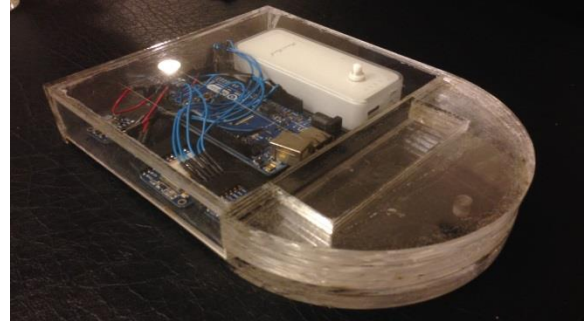
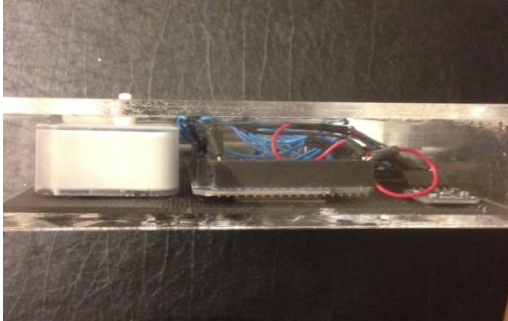


Figure 20: External Power Button for Casing System

Final developed housing pictures are shown in the Figures on the page below. Figure 38 shows a top view, Figure 39 shows a side view, Figure 40 shows a back view, and Figure 41 shows an isometric view. Mechanical drawings are in appendix 8.



Figures 21 and 22: Top View (Left) and Side View (Right)



Figures 23 and 24: Back View (Left) Isometric View (Right)

Chapter 6: System Integration and Testing

For testing the prototypes and the final product, various components and their functionality are tested so that they fall within acceptable ranges of the benchmarks set initially. This includes testing of the individual sensors, their connections to the microcontroller and the LCD screen, and the integrity of the external sensor housing. These categories determine whether or not the prototype is fully functioning or whether there is a flaw in the design.

The first components that are tested are the housing and casing for the external sensors. This is an essential part of the system, as it contains a multitude of essential components. Therefore, the housing and casing need to be able to withstand severe impacts, loads, shearing forces, and torsional forces. These are tested for through a simulated test via SolidWorks. It also has to be waterproof as well as resistant to condensation internally to ensure the sensors inside the casing will not be damaged by any moisture source. The second testing runs the system on a vibration table. This particular test is conducted by simply lowering the interior temperature of the casing from a higher ambient temperature. To do this, the casing is placed into a freezer, whose temperature is controlled, and records the temperature change until it reaches steady state. Lastly is the dynamics test, using an accelerometer within the casing to record the forces in the x, y, and z direction. This is used to determine what motion the casing and sensors are undergoing. This could be anything from a jump, spin, or a turn. It also required not only the use of a snowboard but a skateboard as well.

Section 6.1: Casing Finite Element Testing

The housing and casing for the external sensors are an essential part of the system, as it protects a multitude of components. It acts as the only barrier between the chaotic, fast paced outside environment and the set of sensors and chips delicately interwoven together on the inside. Therefore, the housing and casing need to be able to withstand severe impacts, loads, shearing forces, and torsional forces. It also has to be waterproof as well as resistant to condensation internally to ensure the sensors inside the casing will not be damaged by any source of moisture. For the purpose of staying on the safe side, a load force of 100 N is selected for this test, which was a value believed to be within safe parameters to identify areas of probable failure within the designs. A more

realistic value would be 686 N or more (the equivalent of a person weighing 155 lbs or more standing on the case). This is done in order to simulate the more extreme impact scenarios such as having the snowboard fly off of the rider's feet, or having the board and casing slam into a tree or park features. Although the entire casing will be analyzed, the projected primary location of weakness will be directly in the center of the top of the casing, a location that has a distance furthest away from any support. This will be a highlighted area of concern. Considering the thickness of the top of the casing translates to 4.826 millimeters from .19", a reasonable breaking point deflection lies in the 1 millimeter range. The point of failure lies around a millimeter of displacement as a result of acrylic's correlation to glass. At the same thickness of .250", glass and acrylic sheets underwent several different weighted dropped ball tests. The results consistently had the acrylic being 18.2 times stronger to impact than the glass. Considering that the glass fracturing point was a .003 mm deflection, a deflection of .054mm by comparison is needed for the acrylic to feel that same force. Using these deflection distances to the material thickness as a ratio, the calculated deflection needed to fracture the acrylic is determined. The force 100 N still remains on the lower side for these scenarios, but there has to be some shock absorption material within the case if the housing experiences a higher force we did not anticipate. As long as the resulting deformation is not anything major to the point of crippling the casing, it will be considered a success.

Finite element analyses were then conducted to test the durability of the casing we manufactured. This includes applying loads vertically onto the upward faces of the casings, as well as horizontal loads. The deflection and stress are then simulated for an applied load on the tops as well as the sides of the casings. This is done for both the square and hexagonal base preliminary casings as well as the final snowboard casing. The 100N force is then applied with uniform distribution over the domed surface on top. This is to simulate if an object impacted the casing from the top directly onto the casing. A simulation is also done on to simulate if the same load is applied to one of the horizontal faces on the base of the casing, simulating an object striking the side of the casing. This is done assuming the casing is created out of acrylic plastic, who's material properties make it ideal for objects that need be impact resistant. Table 4 below shows the material properties for Acrylic Plastic.

The free body diagram in Figure 16 on the page below illustrates the type of loading as well as the fixtures on the casing. As shown, the load is uniformly applied

Table 5: Material Properties for Acrylic Plastic

Acrylic Plastic	
Young's Modulus (N/m ²)	3×10^9
Poisson's Ratio	0.35
Shear Modulus (N/m ²)	8.9×10^8
Density (kg/m ³)	1200
Tensile Strength (N/m ²)	7.3×10^7
Yield Strength (N/m ²)	4.5×10^7

to the domed surface of the casing, normal to any given point on the domed surface. The fixtures, illustrated by the green arrows, show where the casing is supported. These are the base and the dome support, which is a vertical support in the center of the casing to provide additional support at the weakest part of the dome.

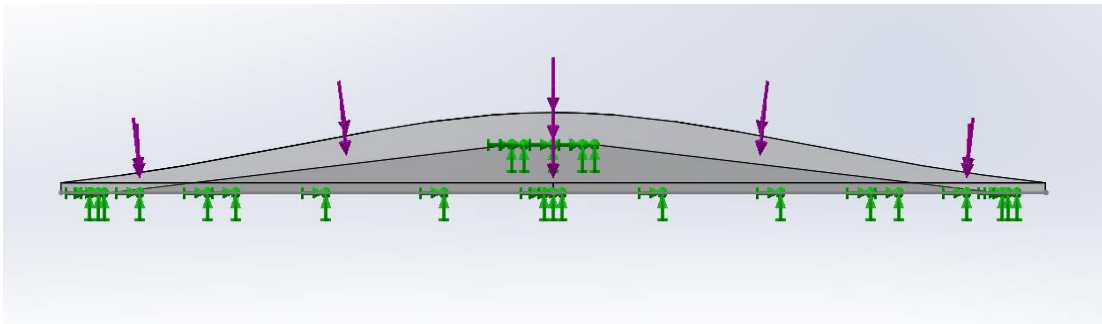


Figure 25: Vertical Loading Free Body Diagram

As seen from the diagram in Figure 16 and where the applied loads and fixtures are located, several predictions are made as to what the simulation would look like. For the displacement done by the vertical loading, it is predicted that the maximum amount of deflection will not occur at the center of the dome because of the center support in the middle of the casing. Therefore, maximum deflection will occur in a circular location around the center support. This is true for both casings as they both contain a central support for the dome. As for the horizontal loading on the rectangular face of the base, the load makes the casing deflect most of the force towards the upper portion of the horizontal face as the bottom of the base is fixed.

When considering stresses, the pattern of stress concentration follows the similar trend of the displacement. For a vertical loading, the maximum stress occurs around the central support, but not where maximum deflection occurs. It is more likely to be closer to the top of the dome. For the horizontal loading, the maximum amount of stress most likely occurs toward the bottom part of the rectangular face. This occurs because of the deformation, the stress concentration moves towards the fixed surface, which in this case is the bottom base.

In doing the analysis for stress and deformation, two sets of equations had to be defined, one for the vertical loads and one for the horizontal loads. For the vertical loading, the equations used to determine stress and displacement are those of a simply supported circular plate, given as:

$$\delta = \frac{F_0(a^2-r^2)}{64D} \left(\frac{5+v}{1+v} a^2 - r^2 \right) \quad (\text{eq. 1})$$

$$\sigma_{max} = \frac{3}{8} (3 + v) \frac{F_0 a^2}{t^2} \quad (\text{eq. 2})$$

where F_0 is the uniformly distributed load, a is the maximum radius, r is the given radius, D is the flexural rigidity, v is Poisson's Ratio, and t is the thickness of the plate. For the horizontal loading, the equations used to determine stress and displacement are those for a simply supported rectangular plate, given as:

$$\delta = \frac{F_0}{24D} (x^4 - 6a^2x^2 + 5a^4) \quad (\text{eq. 3})$$

$$\sigma_{max} = \frac{0.75F_0b^2}{t^2[1.61(b/a)^3 + 1]} \quad (\text{eq.4})$$

Where a is the length of the plate, b is the width of the plate, and t is the thickness of the plate. The reason σ_{max} is used is so that the key interest in the stress analysis is to determine if the stress will exceed the criteria for a safe product, therefore only the max value is required to determine this.

For the preliminary casing designs, the finite element analysis shown in Figures 17 and 18 revealed the behavior that would be expected while undergoing loading.

As seen from Figures 17 and 18, the majority of the displacement and stress from the vertical loading occurred around the central support of the casing. Also, from the

simulation, it is found that the max displacement is 0.00086mm and the max stress is 207959 N/m². As both of these parameters are within the failure criteria, this simulation is considered successful. The preliminary hexagonal casing had a similar behavior, but is within the safe failure criteria as shown in Figures 19 and 20.

As seen in Figures 17 and 18, the majority of the stress occurred towards the bottom of the rectangular face and the maximum displacement occurred at the top edge of the casing, especially towards the outer corners. It is determined that the maximum stress from the simulation is 996427 N/m², and the maximum deflection is 0.00048mm, which both are within the failure criteria.

By doing these preliminary casing analyses, several key aspects about the behavior of the material and design is determined. From doing the four simulations, two for the rectangular and two for the hexagonal casing, it is determined that the results are reasonable and within the failure criteria. None of the simulations exceeded the failure limits of 1mm or 4.5×10^7 N/m² for displacement or stress, which are based upon the material properties of acrylic. From these simulations, there are several areas of interest that could be discerned from the models. The first is the amount of stress that

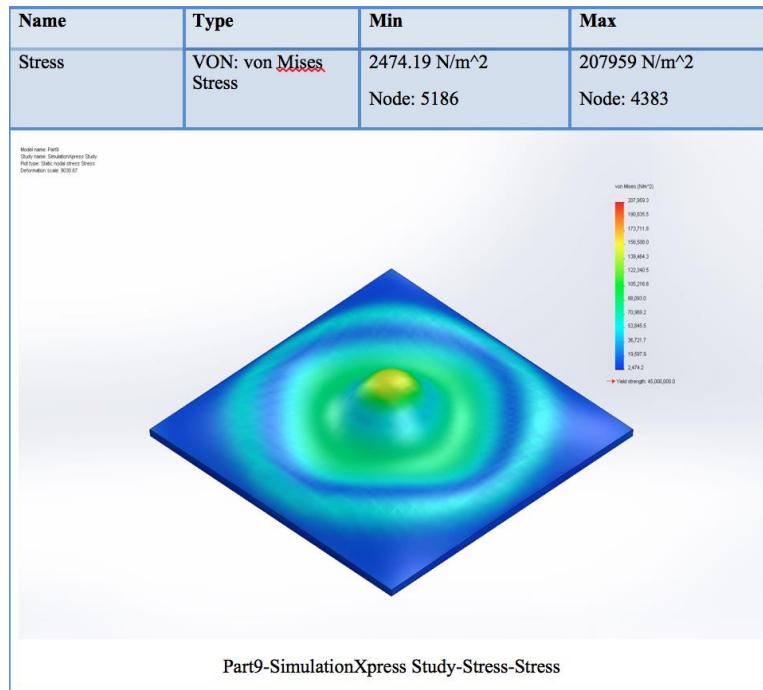


Figure 26: SolidWorks Stress Analysis for Vertical Load on the Preliminary Square Casing

occurred when applying a horizontal load to the square casing. While it did not exceed the failure limit, it is the closest to it by far. While it may not damage or cause any permanent deformation to the casing, the amount of stress could indicate that it might be easily removed from its adhesive base, dislodging the entire casing. This is particularly worrying because the stress is mainly concentrated towards the bottom edge of the casing. Another area of concern is the deflection that occurred when applying a vertical load to the hexagonal casing, which achieved the greatest amount of deflection in any of the simulations. While it only achieved a deflection of 0.0147mm, it undeniably illustrates a problem area in the design. The major concern would be if a heavier load is applied or if the load is a concentrated-point load. This would magnify these parameters, and could eventually lead to failure.

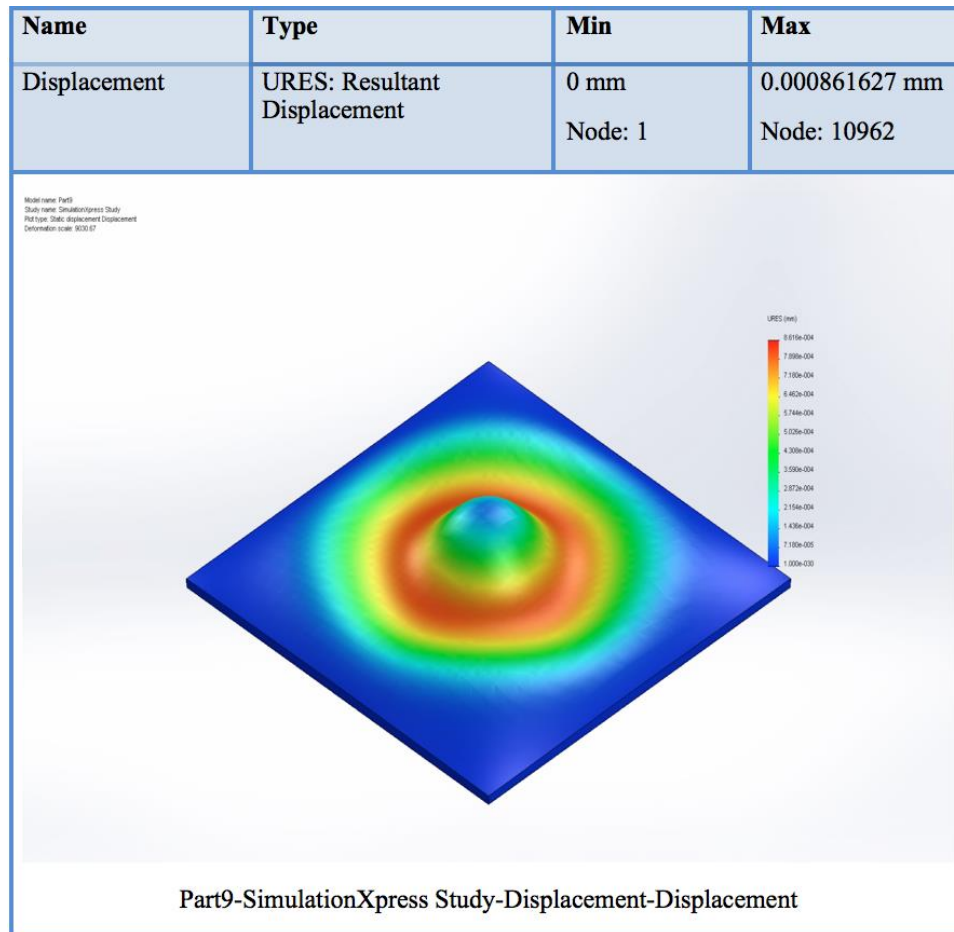


Figure 27: SolidWorks Displacement Analysis for Vertical Load on the Preliminary Square Casing

Besides those few problem areas, the majority of the analysis showed that the casings are structurally sound and will not fail while undergoing a 100N load. The most important result in this experiment is that the casing will not break open during this extreme scenario, in order to preserve the electronics and its vulnerability to snow. However, there are still several areas that could do with refinement or improvement to ensure that the casing is able to handle even greater loads, and minimize the risk of failure. The first improvement that could be done would be to add additional supports in the casing under the dome. This would minimize some of the more extreme areas of deflection, but could lead to stress concentrations around the supports, creating more potential areas of failure. Another improvement would be to change the type of material used that has a higher tensile strength and yield strength.

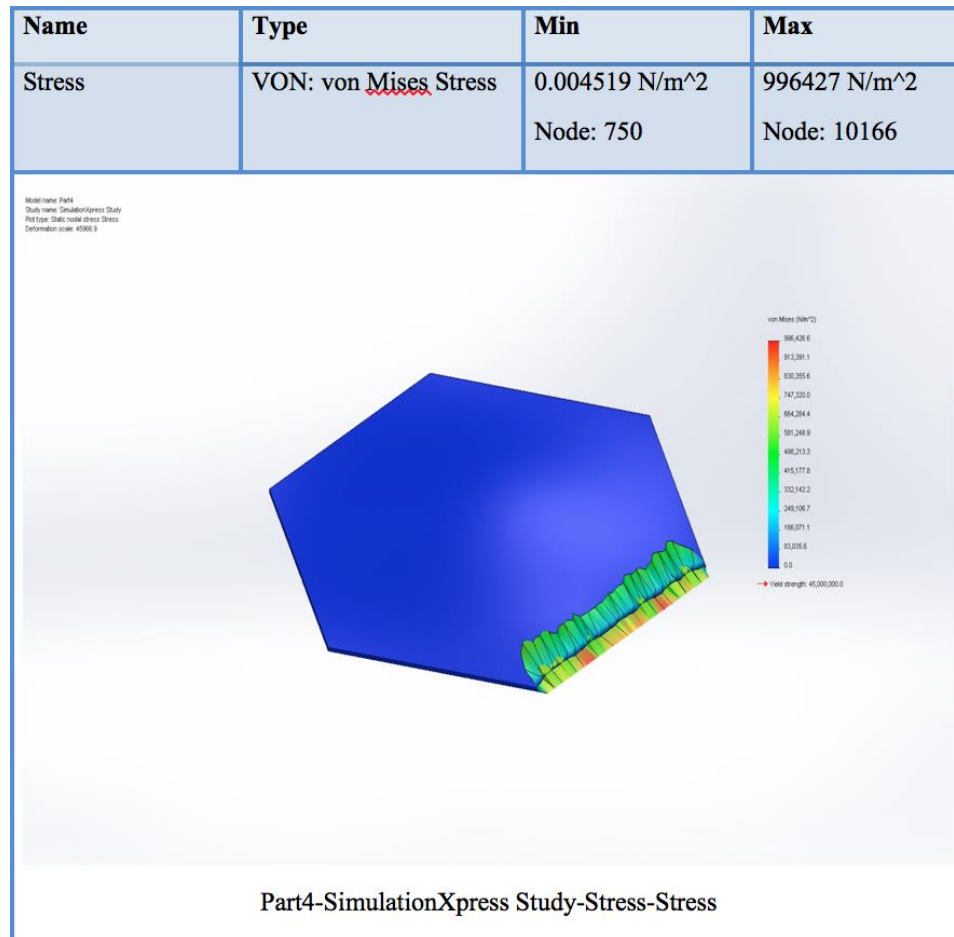


Figure 28: SolidWorks Stress Analysis for Horizontal Load on the Hexagonal Casing

This would limit deflection, but could raise the change impact fracturing, as raising the yield and tensile strength could cause the material to become brittle. Another option would be switching the material from plastic to metal, but metal would be heavier, more susceptible to permanent deformation, and would create other modes of failure, such as corrosion.

A finite element analysis is then conducted on the third iteration of the casing design to see how that would behave under loading. To do this, several sections of the casing are selected and then had loads applied to them as shown in Figures 21 and 22. The two that are looked at are the top of the electronics portion of the casing, and the arced wall of the electronics casing.

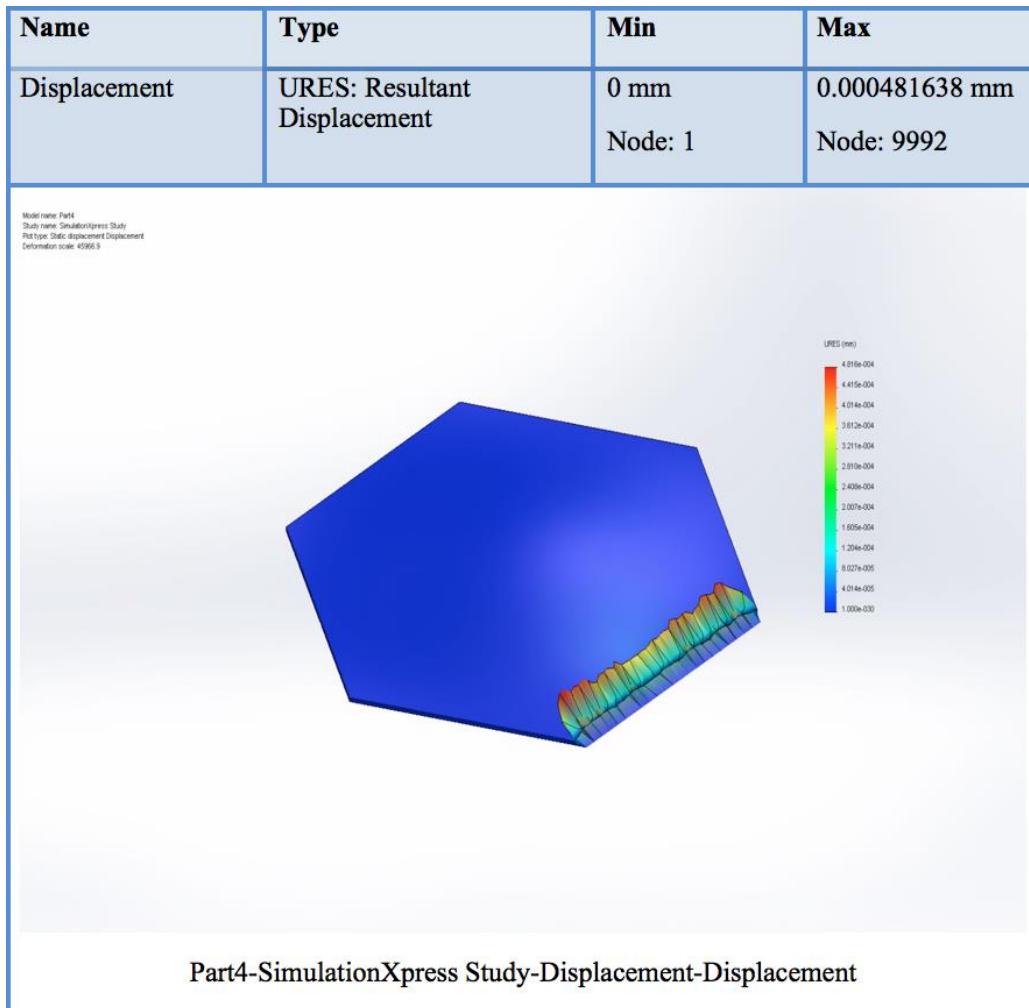


Figure 29: SolidWorks Displacement Analysis for Horizontal Load on Hexagonal Casing

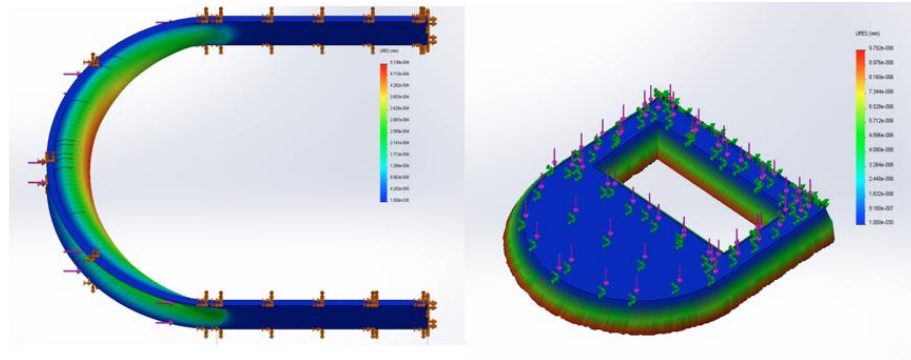
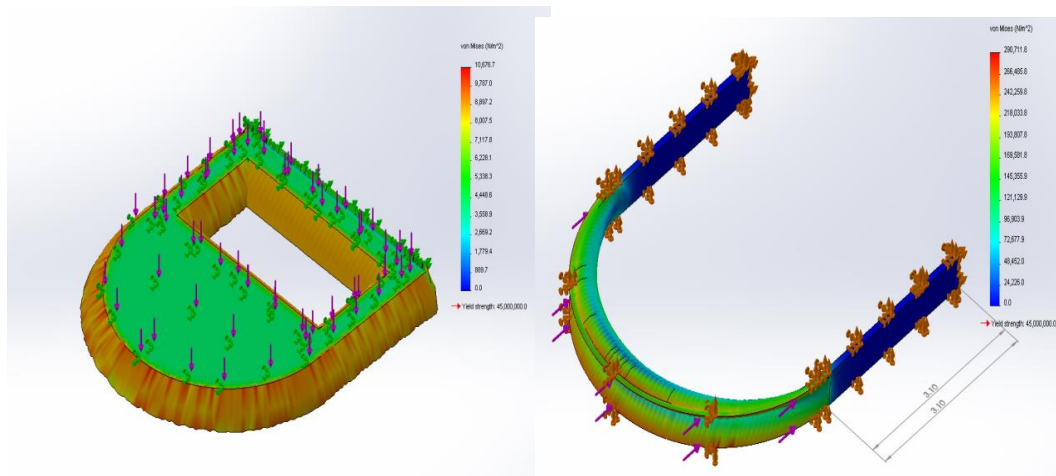


Figure 30: SolidWorks Displacement Analysis for Horizontal and Vertical Load on



Final Casing

Figure 31: SolidWorks Stress Analysis for Horizontal and Vertical Load on Final Casing

In conclusion, there were more weak areas found that anticipated in some of the prototypes. However, they are negligible considering that the greatest point of deflection was at .0147mm. This is well below the point of failure that was calculated at around 1 mm, a point that would cripple the housing. Although cracking will come from these types of impact forces, the more important result is that the housing does not collapse onto itself, and then expose the expensive internal system that is viable to damage once it's exposed.

Section 6.2: Thermal Testing

Heat plays a major role in the functioning of electronic components, and can be the cause of failure and inaccuracy. As such, many electrical components have a set range

of temperature that they can operate within. For example, an Apple product, such as an iPhone, operates between temperatures of 32 and 95°F (0 to 35°C), and in the case that the temperature of the device exceeds either of these boundaries, the devices will shut down to protect itself. The main reason for this is the effect that temperature has on electrical current. In cold temperatures, the flow of electrons slows, which in the case of a battery means that the battery has more time to release its charge, thus leading to a quicker draining of the battery. This affect can vary from having a minimal to severe impact on battery life. Another effect is that at colder temperatures, condensation may also occur, which can damage any electrical device. The goal of this test is to determine if cold temperatures will adversely affect the sensor system and cause damage or decreased performance. The most essential and vulnerable component to powering the sensor system is the battery, and therefore is required to successfully function among the coldest temperatures that can occur during snowboarding. This low temperature requirement is negative 8 degrees Celsius, 1 degree lower than the lowest average temperature for weather in Tahoe.

This test is conducted by simply lowering the interior temperature of the casing from a higher ambient temperature. To do this, the casing is placed into a freezer, whose temperature is controlled, and records the temperature change until it reaches steady state. There is also a time component as well, where the casing is exposed to cold temperatures for an extended period of time to see if the prolonged exposure will have any effect, as well as determine the longevity of the battery at those conditions. The sensors are also observed frequently to determine if any of the components fail or malfunction due to the cold temperature.

Before the test is actually conducted though, there are several concepts and theoretical calculations that must be done. The first is the concepts of heat transfer and heat loss, which is the transfer of thermal energy from one surface, fluid, etc. to another. This generally entails that heat energy is lost by one source and gained by another until it reaches a state of equilibrium. The rate of heat loss is what determines how quickly an object or surface cools or heats up. This is dependent on several factors, such as heat loss due to conduction through the material, and heat loss due to convection from air hitting the surfaces of the case.

The first type of heat loss, conduction, describes the transfer of thermal energy through a material or materials. This rate of heat loss is determined by several factors, such as thermal conductivity (k), the area of the materials (A), the thickness of the material (d), the hot temperature (T_h), and the cold temperature (T_c). These relationships are described by the equation:

$$\dot{Q} = \frac{kA(T_h - T_c)}{d}$$

where \dot{Q} represents the rate of heat transfer. To do this calculation, several of these parameters had to be taken from material properties and conditions of the experiment, shown below in Table 5.

Table 6: Material Properties and Measurements

$A_{\text{base}} = A_{\text{top}} \text{ (m}^2\text{)}$	0.2603
$A_{\text{wall}} \text{ (m}^2\text{)}$.0097
$A_{\text{toe}} \text{ (m}^2\text{)}$.0051
$A_{\text{total}} \text{ (m}^2\text{)}$	0.2751
$k_{\text{acrylic}} \text{ (W/m}^2\text{K)}$	0.2
$k_{\text{rubber}} \text{ (W/m}^2\text{K)}$.13
$d_{\text{base}}=d_{\text{wall}}=d_{\text{rubber}} \text{ (m)}$.0025
$d_{\text{toe}} \text{ (m)}$.0635
$T_h \text{ (}^\circ\text{C)}$	27
$T_c \text{ (}^\circ\text{C)}$	-8

A_{base} is the area of the base of the casing, which is also the same area as the area of the rubber pad. A_{top} is the area of the top of the casing, which is equal to that of the area of the bottom of the casing. A_{wall} is the area of the surrounding side wall of the casing. A_{toe} is the area of the front toe piece. The two thermal conductivity numbers are for acrylic and rubber, as they have different thermal properties. The thickness of the base piece (d_{base}) is the same as the thickness of the sidewall and the rubber pad. The thickness of the toe piece (d_{toe}) is larger than the thickness of the other pieces (0.0635m). Figure 18 shows the hot to cold temperature change, (T_h , T_c) represent the temperature change from 27°C to -8°C.

After the material properties and temperature changes are determined, the rate of heat loss is calculated for each section: the top, base, rubber pad, toe piece, and the side wall. This is then summed to give an overall rate of heat loss due to conduction. As seen

from Table 6 on the next page, the piece with the least amount of heat loss is the toe piece. This can mainly be attributed to the increased thickness of the material. The two pieces that have the greatest rate of heat loss are the base and top pieces, with a rate of 875 watts. This value is less than the rate of heat loss of the rubber because of the less thermal conductivity that the rubber has when compared with acrylic. The rates of heat losses are then summed to give a total rate of heat loss of 2324 watts (2.324kW).

The second type of heat loss is heat loss due to convection, which is the transfer of thermal energy due to the flow of fluids/gases. This is generally one of the more dominant forms of heat transfer, especially when dealing with structures in contact with a moving fluid. This type of heat transfer is described by the equation:

$$\dot{Q} = hA(T - T_{\infty})$$

where h is the convective heat transfer coefficient, A is the area of the surface of the structure, T is the temperature of the object, and T_{∞} is the surrounding temperature of the environment. This relationship though, is dependent on the convective heat transfer coefficient, which varies according to certain conditions.

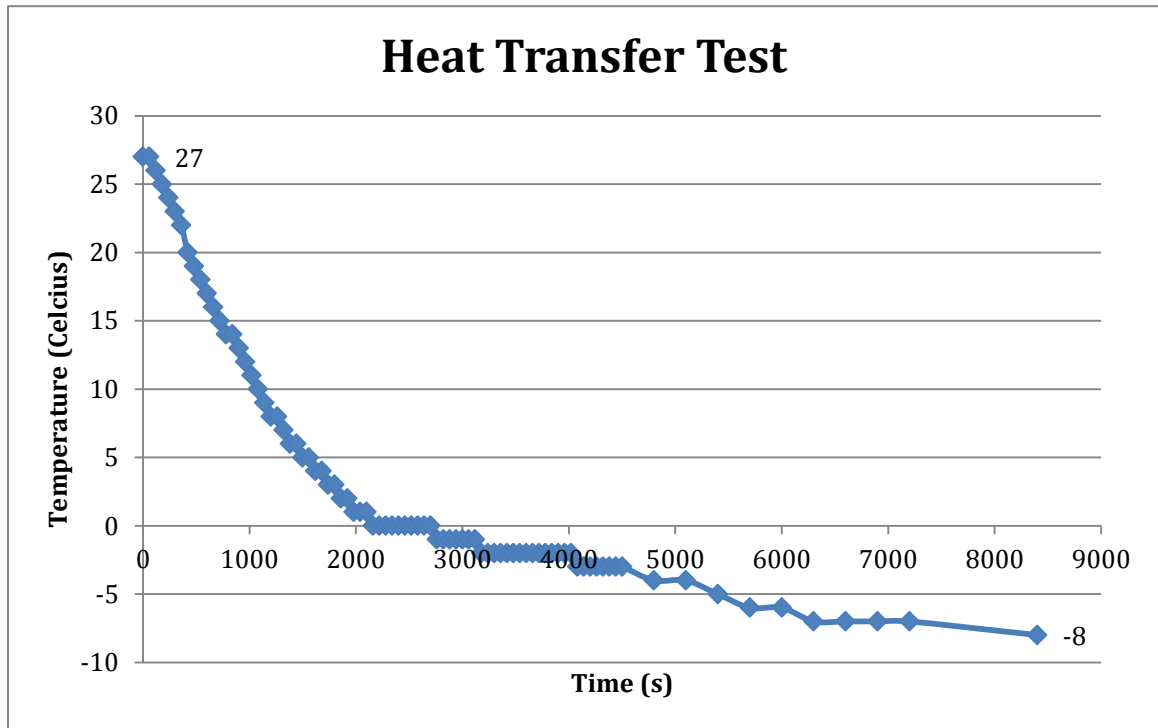


Figure 32: Temperature Changes Within Casing To Equilibrium

Table 7: Heat Losses due to Conduction at t=0

$\dot{Q}_{\text{base}} = \dot{Q}_{\text{top}}$ (W)	-728.84
\dot{Q}_{rubber} (W)	-473.75
\dot{Q}_{toe} (W)	-0.56
\dot{Q}_{wall} (W)	-27.16
$\dot{Q}_{\text{total,cond}}$ (W)	-1959.15

The convective heat transfer coefficient relies on two conditions: whether the fluid (in this case air) is flowing against a vertical or horizontal wall, and whether the flow is turbulent or laminar. For the case of this test, assuming that the flow is laminar and that two separate case distinctions are calculated: one for a vertical wall and one for a horizontal wall. For a vertical wall, the equation to determine the convective heat transfer coefficient is:

$$h_{\text{conv,vert.}} = \frac{k}{L} \left(0.68 + \frac{0.67 Ra_L^{\frac{1}{4}}}{\left(1 + (0.492/Pr)^{\frac{9}{16}} \right)^{\frac{4}{9}}} \right)$$

where k is the thermal conductivity of the object's material, L is the characteristic length, Ra_L is Rayleigh number (dimensionless value associated with buoyancy driven flow), and Pr are Pr and tl numbers (ratio of viscous to thermal diffusivity). This equation is dependent on the following factor, though: Rayleigh number. Rayleigh number generally determines whether or not a fluids flow is turbulent or laminar. For a vertical wall, $Ra_L \leq 10^9$ is considered to be laminar flow. Anything greater than that is considered to be turbulent flow (for flow over a vertical plate). To find Rayleigh number, two other equations must be used:

$$Ra_L = Gr_L Pr$$

$$Gr_L = \frac{g\beta(T_s - T_\infty)}{\gamma^2}$$

where Gr_L is Grashof number (dimensionless number, approximates ratio of buoyancy to viscous forces), g is the acceleration due to Earth's gravity, β is the thermal expansion coefficient, and γ is kinematic viscosity (ratio of dynamic viscosity to density of fluid). Using these equations, the convective heat transfer coefficient is found for a vertical wall.

In the case of a horizontal plate, the conditions and equations change accordingly. To find the heat transfer coefficient, the equation depends on the Rayleigh number. In the case of the horizontal plate, the Grashof number must be found and then determine the Rayleigh number to define the equation for the horizontal heat transfer coefficient. In the case of the horizontal plate, the Rayleigh number satisfied the condition of $10^5 \leq Ra_L \leq 2 \times 10^7$, which gives the equation:

$$h_{conv,horz.} = \frac{k0.54Ra^{1/4}}{L}$$

This will give the convective heat transfer coefficient for laminar flow over a horizontal plate.

Once these equations, variables, and properties (shown in Table 7 on the next page) are determined, the rate of heat loss could be calculated. As seen from Table 8, the greatest heat loss came from the airflow over the top of the casing (63 watts). The heat loss from the other sections (toe piece and wall) is minimal in comparison with the heat loss from the top of the casing.

Table 8: Material and Fluid Properties

$\gamma_{air,0} (m^2/s)$	9.49×10^{-6}
$\alpha (m^2/s)$	15.67×10^{-6}
$\beta(1/^\circ K)$	3.67×10^{-3}
$h_{conv,vert.} (W/m^2K)$	9.7562
$h_{conv,horz} (W/m^2K)$	6.3890
$Pr_{air,0}$	0.715
$k_o (W/m^2K)$	0.0243

Table 9: Heat Losses due to Convection and Total Rate of Heat Loss At t=0

\dot{Q}_{top} (W)	-63.26
\dot{Q}_{toe} (W)	-2.12
\dot{Q}_{wall} (W)	-4.04
$\dot{Q}_{total,conv}$ (W)	-69.43
\dot{Q}_{total} (W)	-2028.58

In conclusion, it was determined that the sections that were most vulnerable to heat loss were the top and walls of the casing. The anticipated thermal weaknesses of the top and walls of the casing is not enough to disrupt the entire system. The casing as a whole provided a strong enough barrier to prolong the amount of time it took the weather to fully infiltrate the interior, taking over 2 hours to fully influence the inside. Even more revealing is how the heat transfer seemed to plateau just below zero degrees Celsius. The interior of the casing drops from 27 degrees to negative 3 in about an hour, and then it takes the same amount of time to get from negative 3 degrees to negative 8. This shows how much harder it becomes to transfer heat, even from the vulnerable top of the casing, once it gets around zero degrees. Fortunately, for this Raspberry Pi battery, it sports a functional temperature range as low as negative 20 degrees Celsius. Based off of how easily it adapted to the temperature at negative 8 degrees Celsius, this test successfully displayed the system's ability to handle the rigors of extreme cold weather that is experienced in snowboarding environments. In the future, if it was decided that insulation would be needed to help regulate the heat loss through the casing, insulation would be added to the most vulnerable areas that were determined in this test, the top and walls of the casing.

Section 6.3: Vibration and Damping Testing

The goal of the vibration test is to determine what kind of acceleration and forces the system will be subjected to due to vibrations, what kind of effect it has on sensors, and whether or not damping is required to reduce the forces on the sensors. This is an essential test, as the casing will have to undergo various forces and torques, which could harm the system. This will also be evaluated at the test's result.

The main goal of damping is influence the oscillation by reducing or restricting vibration. There are several types of damping, which range from mechanical, musical, to structural, etc. The type that is linked to this test is structural damping by the application of using a rubber pad between the base piece of the casing and the bottom of the side wall of the casing. This is done to insulate the sensors and electrical components from the vibrations, which will absorb some of the forces and reduce the amplitude of the oscillating forces. The question, though, is how much damping would it provide, and is damping necessarily required? To understand this, some relationships have to be clarified.

Force is generally described by the equation:

$$F = ma$$

where F is the force, m is the mass of the object, and a is the acceleration. However, in the case of force damping, the relationship becomes:

$$F_d = -cv$$

where F_d is the damping force, c is the damping coefficient, and v is the velocity of the object. The damping coefficient is the main factor that will determine how much damping will occur, which varies depending on the type of material that is used. In the case of this test where rubber is the material that is being used as a damper, the damping ratio ranges anywhere from 0.01 to 0.08.

Oscillations generally follow simple harmonic motion, generally displayed as a sinusoidal wave.

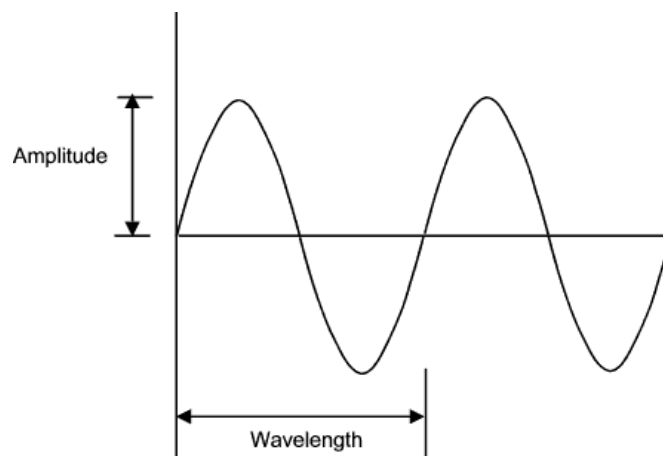


Figure 33: Sinusoidal Wave Displaying Amplitude and Wavelength

In the case of a vibration, the motion of the object would follow such a pattern as shown in Figure 24, elevating to a max peak and dipping to minimum amplitude. The acceleration follows a similar pattern, except as a cosine wave, with max acceleration occurring at the initial time. With this in mind, there are two possible types of system it could be: a 1st Order system (defined by a single parameter and a forcing function $f(t)$), and a 2nd Order system (defined by two state variables and a forcing function $f(t)$). In the case of this particular test, the vibration will resemble a 2nd order system, as it will depend on two state variables: the spring constant (k) and the damping coefficient (c). The equation then becomes:

$$m\ddot{x} + c\dot{x} + kx = F(t)$$

where m is the mass of the object, c is the damping coefficient, and k is the spring constant. These two variables are then determined by two equations:

$$k = \frac{mg}{x}$$

where m is the mass of the object, g is the acceleration due to gravity, and x is the extended length of the spring, and

$$c = 2\sqrt{km}$$

where c is the damping coefficient. Using these values, along with the measured mass of the casing, the force of vibration, acceleration, and damping are predicted. Figure 25 below shows the experimental vibration test setup.

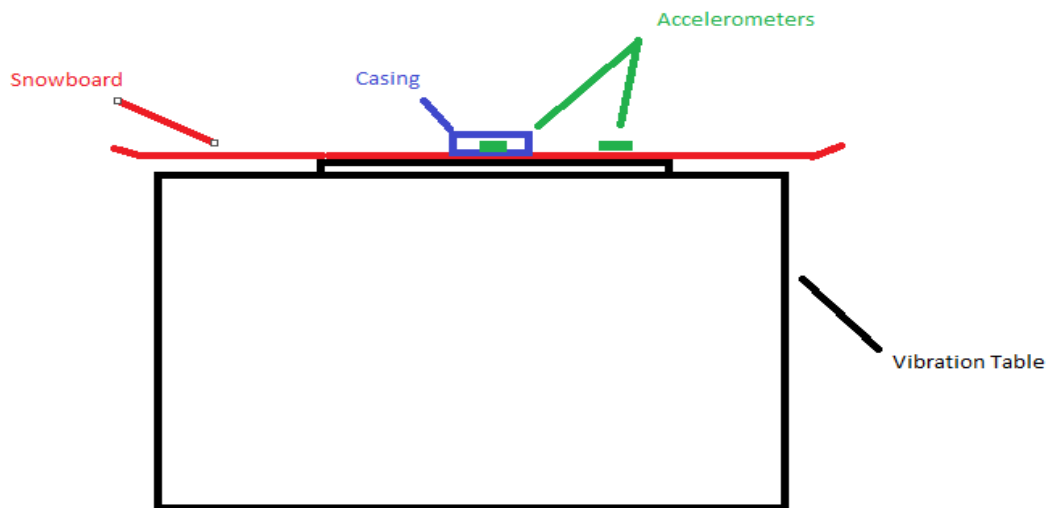


Figure 34: Illustration of Vibration Test

Figure 26 on the page below represents the acceleration responses during the test, one that has an accelerometer attached to the sensor casing and another that has an accelerometer attached directly to the board. As seen from the accelerometer that is attached to the sensors, it has significantly smaller amplitude than that of the accelerometer attached to the board, which indicates that there is some damping that is occurring. The peak acceleration value of the sensors is 0.218g and the peak

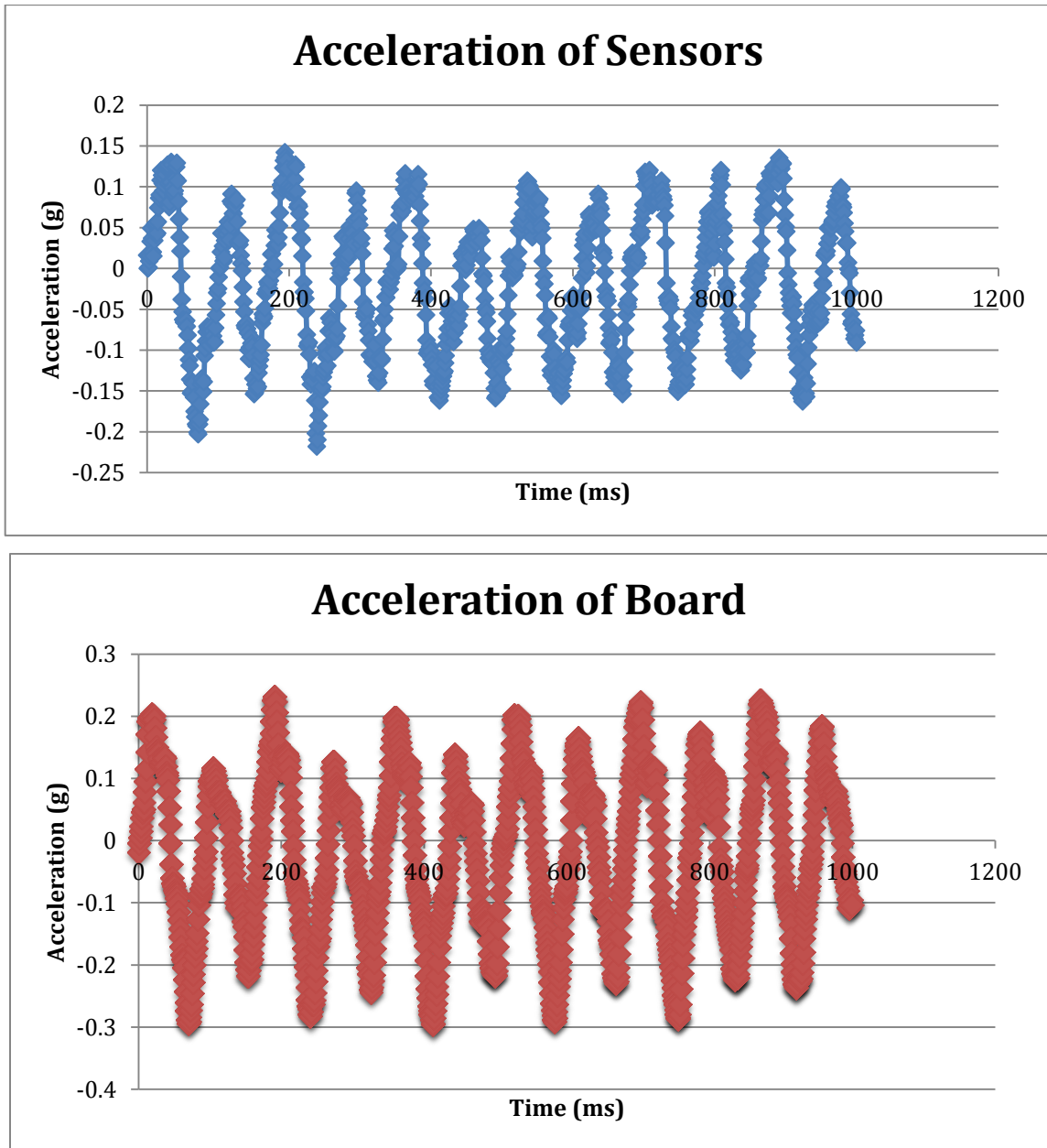


Figure 35: Sensor Acceleration Data Due to Vibration and Board Acceleration Data Due to Vibration.

that is occurring. The peak acceleration value of the sensors is 0.218g and the peak acceleration of the board is 0.289g. This is a 24.6% reduction in acceleration, which is a significant amount of force reduction and damping.

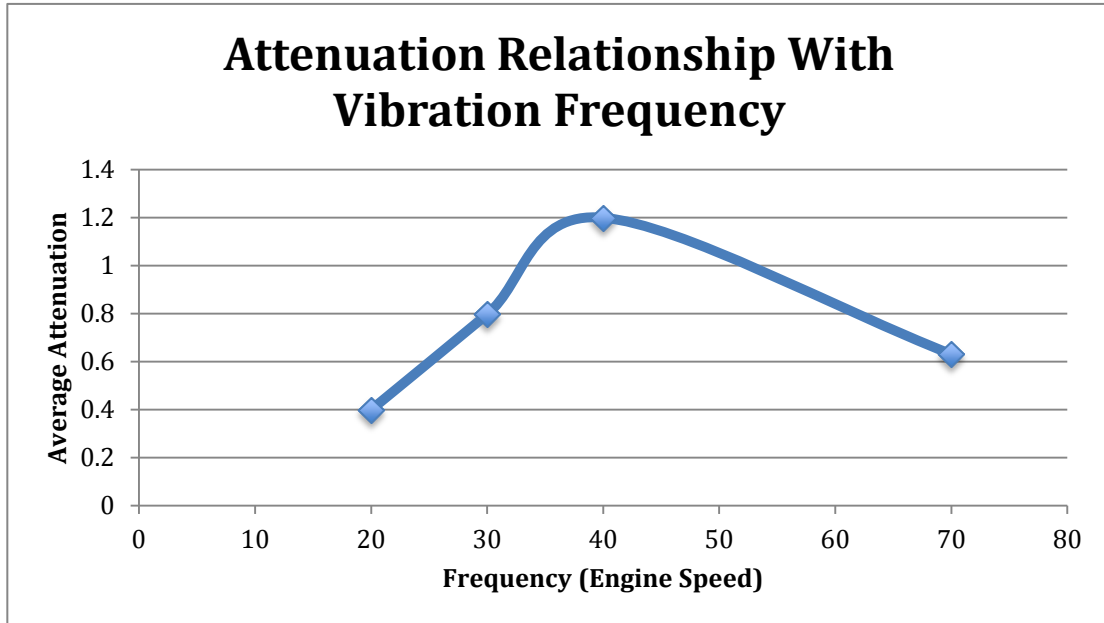


Figure 36: Attenuation of Vibration Data

This can also be shown as a ratio by comparing the two data groups from the two accelerometers. This is known as the attenuation of the two responses. Its main use is to determine the reduction of forces in situations such as the vibration/damping test. As seen in Figure 27, the attenuation varies with time and is not constant. Most of the data points that show the attenuation are usually within the range of less than ± 1 , which indicates a reduction in force. However, there are several data points, which are greater than ± 1 , which indicates an increase in force. The explanation for this is that the data readings must have been out of phase with each other, which would allow for a ratio greater than ± 1 . With these values, as well as those determined from the dynamic testing, it can be concluded that the sensors and casing are not in any danger of becoming damaged or destroyed.

Section 6.4: Dynamic Testing

The ideal functionality of the accelerometer peripheral is to identify a trick made based on the movement data interpreted by the sensor. Distinguishing exactly between the many possibilities of movement for tricks requires some precise processing and data

parsing methods that are beyond the scope of our team for this project. However, it is still possible to determine useful movement information as an essential design component, and gathered what information could be managed.

This could be anything from a jump, spin, or a turn.

It also required not only the use of a snowboard but a skateboard as well. The testing comprised of 6 different sets of testing: static drop test, shake test, snowboard jump and spin test, and a skateboard jump and spin test. The static drop test is done by simply taking a snowboard with the casing attached and dropping it from a height of 1.5m and measuring the impact acceleration. The shake test is comprised of shaking the snowboard and casing attached from a static position. The snowboard and skateboard jump test consisted of attaching the sensor system to a skateboard and snowboard, jumping twice, and recording the acceleration on impact. The 180° spin test consisted of conducting a 180° spin twice on the skateboard and snowboard. The first spin is by 180° in the counter clockwise direction and the second spin is in the clockwise direction.

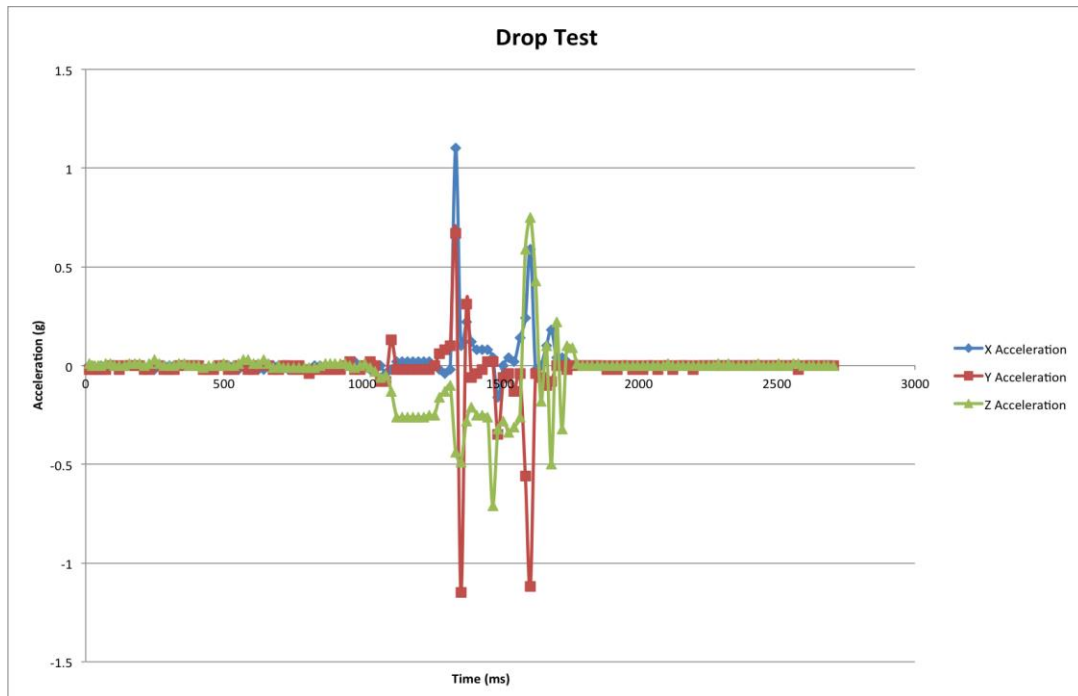


Figure 37: Acceleration Data for Static Drop Test

In the static drop test shown in Figure 28 above, the expected acceleration that is a negative value in the z direction, which would be followed by a sharp spike in acceleration at the moment of impact. In the case of this test though, we see that there is a

negative value in the z direction, but at the point of impact, there are two positive spikes in acceleration in the x and y directions, not in the z direction. This is contrary to what is predicted. The explanation for this is that the snowboard did not land on its bottom surface evenly. Instead, it landed on one of its edges, which would give values in x and y direction. From the point of impact, the largest acceleration is 1.15g in the y-axis. The second spikes in acceleration after 1.5 seconds are secondary impacts from when the board rebounded and hit the ground again.

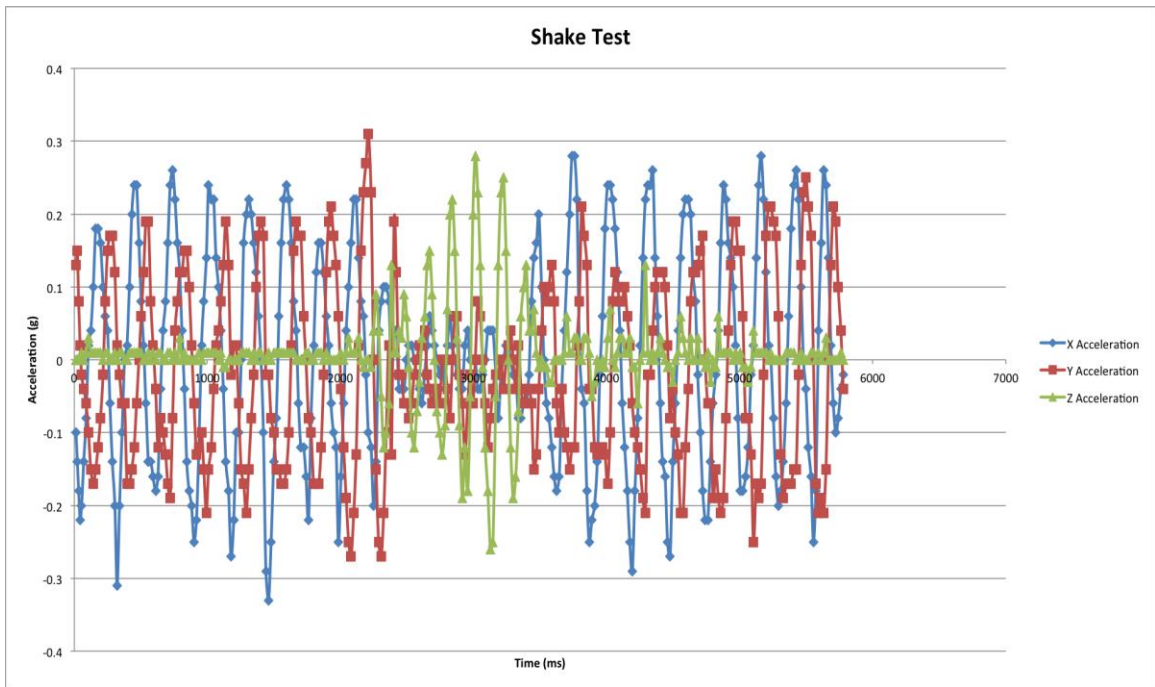


Figure 38: Acceleration Data for Static Shake Test

In the static shake test shown above in Figure 29, the board is shaken first in the x and y-axis, and then for an intermittent period in the middle in the z-axis. This test is done mainly to demonstrate that the sensor system accurately reads acceleration data over a certain period in 3 axes. As the data shows, from 0 to 2.2 seconds, the case is shaken in the x and y axis, producing the positive and negative cycles in acceleration. From 2.5 to 3.5 seconds, the rise in acceleration in the z direction and the decrease of values in the x and y direction clearly indicate the transition from shaking the case in the horizontal plane to shaking in the vertical direction. This then transitioned back into shaking in the x and y directions.

Section 6.5: Sensor Testing

While integrating the different components of the electrical subsystem into the GUI running on the LCD, there are a few design problems we encountered which we needed to find solutions to. The first of these design snares came from realizing that the AREF pin (analog reference), against which the Arduino compares the voltages at the analog input pins, is required by the gyroscope but physically taken up by the 2.8" TFT LCD screen. This requires a software workaround by adding lines to manually scale the inputs coming from the gyroscope module. Unfortunately, by forcing this solution in software by simply scaling the inputs, accuracy could be lost by not using the exact reference voltage provided by the gyroscope peripheral. The accuracy of the gyroscope, even with a totally ideal calibration, is still only accurate to about +/- 10 degrees. This is a major way in which future revisions to the system would need to be made to improve the overall data coming from the gyroscope potentially using slightly different hardware.

Table 10: Sensor System Test Data

	Standard	Test
<i>Temperature (F)</i>	<i>+/- 3</i>	<i>+/- 0.5</i>
<i>Speed (mph)</i>	<i>+/- 5</i>	<i>+/- 0.25</i>
<i>Elevation (ft)</i>	<i>+/- 10</i>	<i>+/- 8</i>
<i>Pressure (Pa)</i>	<i>+/- 500</i>	<i>+/- 100</i>
<i>Bend (Degrees)</i>	<i>+/- 5</i>	<i>+/- 5</i>
<i>Acceleration (g)</i>	<i>+/- 1</i>	<i>-</i>

Accuracy of the various components is tested by comparing the values as displayed by the device against those measured using reliable instruments. The temperature sensor is found to be accurate to at least .5 degrees Fahrenheit. The pressure sensor is accurate to within ~1 kPa as compared to reported local weather conditions. The GPS relies on satellites for accuracy, but given a good connection, it is accurate to 4 decimal places on longitude and latitude, within ~.3 to .5 mph, and conveys accurate date and time information.

Chapter 7: Jump Tests and Data Analysis

Section 7.1: Interpreting the Accelerometer Information

In order to extrapolate useful information from the accelerometer peripheral, we transferred the excel spreadsheets into Matlab in order to work with the numbers directly to see what can be determined from them. The first column of the graph represents the timestamp associated with the sample in milliseconds; the next three columns correspond to the X, Y, and Z-axes. Using Matlab, we wrote a script centered around a useful function, 'xlsread', which can read a specified column from an excel spreadsheet and import it as an array. By summing the acceleration over the whole sample for each axis, we could get an idea of the move represented by the data, which correlated to the known test carried out.

When opening up the arrays of data in Matlab to look for trends, there were a number of effects in the data that matched expectations of the trick. When executing a simple jump on a snowboard or skateboard, there were observation forces of the largest magnitude at the z-axis. Z forces during a skateboard jump were 2 to 4 times greater than those of a snowboard jump. This is in line with expectations, as a skateboard requires a sharp striking of the ground to jump whereas a snowboard uses a more gradual spring-like action.

The first dynamics test carried out was the vertical drop test. By summing the outputs of the axes in Matlab, a net impact in the z direction with a moderate magnitude was observed. There was also a substantial clockwise twist after impact represented by a positive reading on the x-axis and a negative reading on the y-axis. It is not as substantial as the jump and turn readings, being just a minor deflection from the drop, but it shows up in the data.

After writing the initial code to calculate the sum of the axis values for the X, Y, and Z-axes of the accelerometer, to determine the net direction of the forces, we then modified it to generate a rough calculation of the time boundaries of the Z impact. The goal was to determine when the Z value on the accelerometer would begin climbing, and again when it was nearly done. This method gave us a reliable time window on the jump, because the forces register as huge spikes at the beginning and end of the jump.

The method we used for this was simply running two sums scans for the z-axis. The first sum scan generates the total z value experienced in the movement. From there, it calculates a second sum that is, at each step, compared against the original sum to determine when it was 10% of the way toward the sum, and again for 90% of the way toward the sum. This value is then scaled by 120% to make up for the two missing 10% sections. Despite the crude nature of this calculation, it provides a decent gauge of the time window, accurate to within 10%, because the impacts experienced are rather sharp on the Z-axis for jumping the board. Sample Code is shown below, the rest of the code is listed in Appendix 6.

Matlab Code example

```
filename = 'SnowJump1.xlsx';
xSum = 0;
ySum = 0;
zSum = 0;
zScan = 0; %second sum to be calculated to get bounds
zMax = 0;
tStart = -1;
tStop = -1;
tData = xlsread(filename,'A:A'); %time, first data column
xData = xlsread(filename,'B:B'); %x axis, second column
yData = xlsread(filename,'C:C'); %y axis, third column
zData = xlsread(filename,'D:D'); %z axis, fourth column
for i = 1:size(xData)-1
    if(abs(zData(i))>zMax)
        zSum = zSum + abs( zData(i) );
    end %this loop establishes the total z value
end
for i = 1:size(xData)-1
    zScan = zScan + abs( zData(i) );
    xSum = xSum + xData(i);
    ySum = ySum + yData(i);
    if(abs(zScan)>abs(.05*zSum) && tStart == -1)
        tStart = tData(i);
    end
    if(abs(zScan)>abs(.95*zSum) && tStop == -1)
        tStop = tData(i);
    end
end %this loop sums the x,y axis data and looks at the z
data to determine approximate bounds
tJump = tStop-tStart; %subtracts to determine jump window
```

Section 7.2 Data Graphs and Matlab Results

We transferred samples of information from the performed tests as data arrays from the Arduino using a third-party data program called CoolTerm. We were able to open these results up as excel spreadsheets and render graphs from the data arrays. The following graphs show the X, Y, and Z acceleration samples over time. The graphs each consist of roughly five hundred data samples taken over the course of about seven seconds.



Figure 39: Acceleration Data for Snowboard Jump Test

The snowboard jump test involved a basic jump on the snowboard without any twist. The sensor data registered a moderate impact along the z-axis, and only very slight deflection in x and y.

In the snowboard jump test shown in Figure 30 above, the data that is expected to be seen are sharp spikes in the z direction of acceleration on lift off and impact, with some residual acceleration in the x and y axis that occur from not landing or taking off perfectly vertically. As seen from the data, the greatest amount of acceleration is in the z-axis, which accounts for the vertical acceleration. This occurred for both jumps. The largest acceleration seen during the test is 5.54g in the z-axis. This shows us that the accelerometer selection was ample and appropriate given its range of +/- 16g.

We opened up the snowboard jump script in Matlab (snowjump1.m) which accesses the data arrays as laid out in the excel spreadsheet/graph (snowjump1.xlsx). When we ran the script, it output data consistent with what we would expect from the test and from looking at the graph. After analysis of the 450 samples in the data array, it registered a takeoff time of ~ 1.308 s into the test and a landing time of ~ 4.032 s, which translates to a 2.724s time window on the jump. This is pretty much in line with what we would expect from the graph.

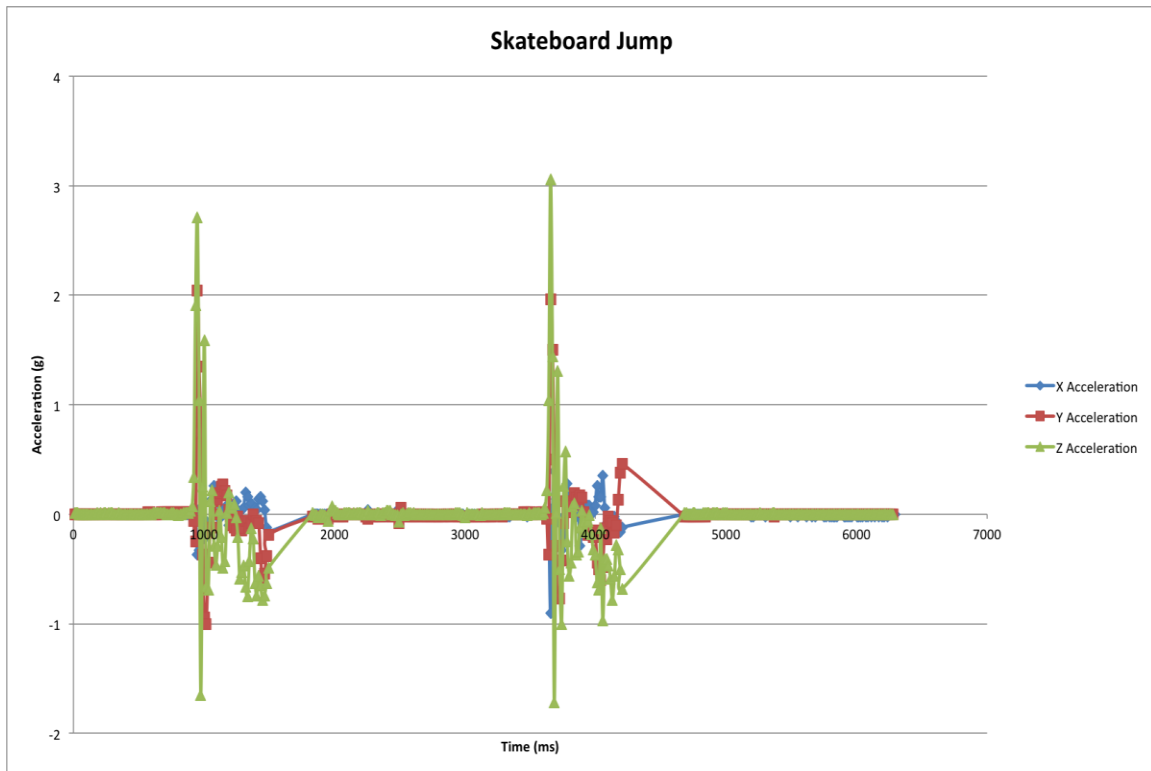


Figure 40: Acceleration Data for Skateboard Jump Test

In the skateboard jump test shown in Figure 31 above, the expectation is that there is similar data to that of the snowboard jump test, except that there is more residual x and y acceleration, likely due to the fact that the skateboard is in motion at the time of the jump and after landing. As seen from the data, there is significant acceleration in the z-axis, consistent with a lift and impact of a jump. There is also though a significant amount of acceleration in the y-axis. This illustrates the acceleration that occurs on a skateboard in the direction of horizontal motion, which in this case is forward in the positive y direction. In physical terms, this means that at the impact of the jump, the skateboard accelerated forward, as that is the direction of motion at the time of takeoff.

The expectations we had from analyzing the graph were again verified by the Matlab analysis. The window on the jump runs from ~.935s to ~4.145s for a window of 3.210 seconds, as determined by analysis of the 403 samples. The forces on the x and y axes registered minimal deflections, with roughly the same y impact and slightly more impact on the x axis.

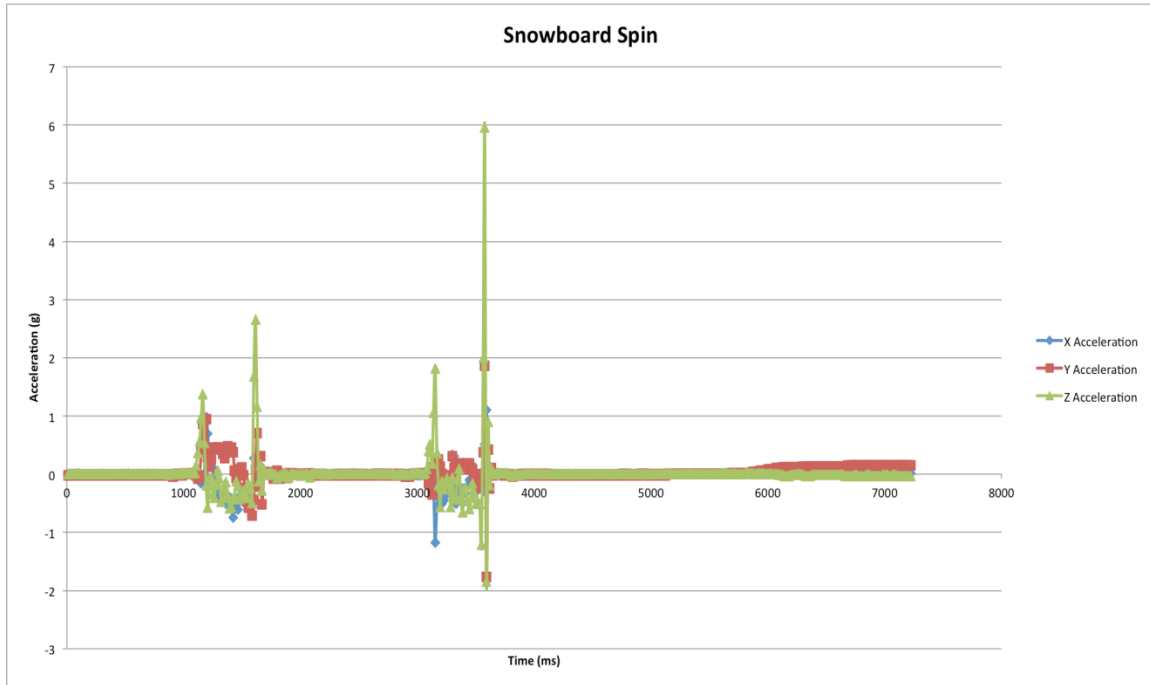


Figure 41: Acceleration Data for Snowboard Spin Test

In the snowboard spin test shown in Figure 32 above, it is expected that there would also be a significant acceleration in the z-axis, as the spin requires a jump to rotate 180°. There should also be a significant amount of acceleration in the x and y-axis. These values depend on the type of spin though, and whether it is a spin in the clockwise or counter clockwise directions. In the case of a counter clockwise spin, the acceleration profile would be a positive acceleration value in the x-axis and a negative acceleration value in the y-axis.

The snowboard spin test was a counterclockwise jump and turn of the snowboard, which would result in a large negative x value and a large positive y value. When we opened this up in Matlab we found results that confirmed this. The snowspin1.m script registered a time window from ~1.150s to ~3.605s for a jump lasting 2.455 seconds. The x value summed to a large negative value, with a large positive value in the y axis. The

skateboard spin test is similar but again registers a greater force of impact at the z axis than the snowboard does, this time roughly double, whereas the lateral forces are relatively more moderate (about two-thirds in magnitude of the snowboard spin).

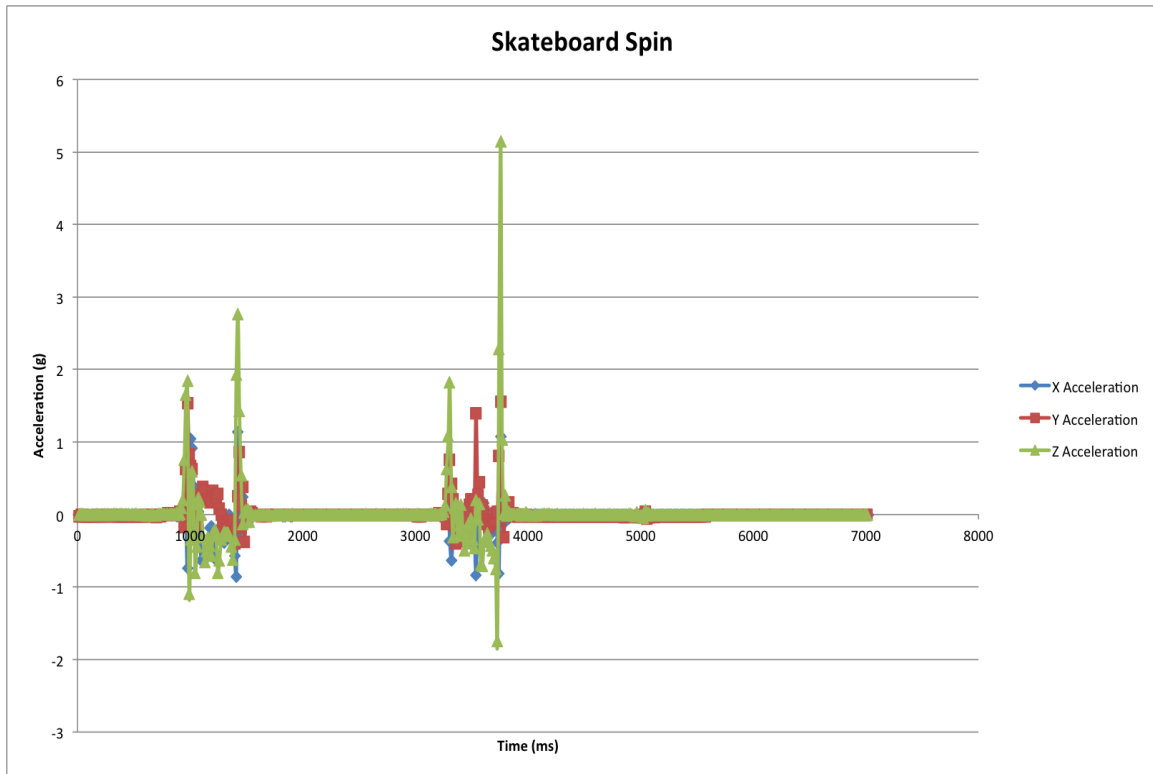


Figure 42: Acceleration Data for Skateboard Spin Test

In the skateboard spin test shown in Figure 33 above, it is expected to see similar results to that of the snowboard spin test. The Matlab analysis determined as expected that this registers as a spin and jump with a time window of 3.210 seconds ranging from $\sim .935s$ to $\sim 4.145s$. These values match what we would expect from the graph, and from our knowledge of the test.

Given the reliability of these calculations matching what we would expect, we can conclude that we have effectively designed a form of trick analysis from the accelerometer data. We have shown that our Matlab analysis technique is effective at determining the presence of a spin, and the time window during which a jump occurs for either a snowboard or skateboard. This technique would not prove as effective for skis however, because skis do not jump by a sharp impact slapping against the ground to take off, and the graphs would likely not register such tidy spikes around the time window.

Chapter 8: Business Plan

Section 8.1: Introduction

Skiing, snowboarding, and skateboarding are three of the most popular action sports, attracting more than 90 million participants in North America, alone (Statista). Because there is such a great and international interest in these action sports, companies invest heavily in research for new technology to bring the newest and best gear to athletes every year. Such new gear varies greatly, ranging from ski and snowboard designs to the development of protective gear. Presently, with the advancement of computer technology, electronic devices have found their way into a variety of sports. Today, the up-and-coming technology for athletes is sport sensor systems, which track a user's performance and maintaining record of their statistics. Creating and designing this new technology allows competitors and enthusiasts alike to track information and metrics on their performance such as speed, range of board movement, and the effects of elevation change. For action sports, like skiing, snowboarding, and skateboarding, there are few options available for their athletes.

In skiing, snowboarding, and skateboarding, there are specific types of data that an athlete wants to track, the first of these being speed. However, recording one's speed is not merely enough. Being able to maintain a log of one's speed at specific points during the "run" is necessary in evaluating an overall performance. Secondly, in order to determine how far an athlete, specifically a skier or snowboarder, has descended on a run at a particular speed, data on elevation change and positioning must be collected. This sort of data collection also helps in evaluating how much airtime an athlete has following a launch or jump. The third piece of data collected is time: how long a run took, overall time spent in practice, etc. Collecting the temperature of the athlete's environment follows, helping to evaluate if temperature has a substantial effect on performance. Board flex is the fourth, and final, category of data collection. Board flex is a tracking system for how much a user turns in accordance to the amount their board or skis flex. Combining the above data allows athletes, whether professional or recreational, to track their accomplishments while enriching their overall experience. However, in order to acquire the desired data, several sensors and electrical components are required: a global positioning sensor

(GSP), an accelerometer, a temperature/pressure/altimeter multi-sensor, a microcontroller, and bend sensors are such components.

In researching the current market, finding sensors used specifically for skiing, snowboarding, and skateboarding yields very few products. Those that are available, are yet to pass the prototyping stage and onto the market – leaving room for innovative creativity and design. A team from Michigan State University, in association with the Air Force Research Laboratory (AFRL), designed a number of prototypes – with some including features such as a global positioning system (GPS) (Bekkala); while Nokia, in collaboration with the action sport powerhouse Burton, created a sensor system called PUSH Snowboarding – a system which monitors a snowboarder’s ride speed, heart rate, “head rush” board orientation, and foot pressure (“Nokia X Burton – TVCs”).

Section 8.2: Costing Analysis

Creating an overall budget for the development of a prototype requires two separate units. The first is the preliminary budget, shown in Table 9 which consists of the components needed for the initial prototype design, their estimated cost, actual cost, and pending expenses.

Table 11: Preliminary Budget for Prototype Costs

EXPENSES				
Category	Description	Estimated	Spent	Pending
Electronics	GPS MTK3339	\$ 30.00	\$ 29.95	
	Nike + Sensor	\$ 19.00	\$ -	
	Olympus LI 42 B	\$ 12.00	\$ 5.88	
	Adafruit BMP085	\$ 20.00	\$ 9.95	
	Adafruit Trinket 5v	\$ 8.00	\$ 7.95	
	BLE112 Module	\$ 14.00	\$ 13.95	
Casing	Polycarbonate	\$ 10.00	\$ -	\$ 10.00
	Adhesive Pad	\$ 15.00	\$ -	\$ 15.00
Misc.				
	TOTAL	\$ 128.00	\$ 67.68	\$ 25.00
Net Reserve (Deficit)			\$ (67.68)	\$ (25.00)

As the design changes, the budget updates in order to give an accurate final budget for the prototype, as shown in Table 10. The primary changes consist of the removal of the Nike Plus sensors, and the corresponding Bluetooth modules, and the addition of the LDC display, bend sensors, and an upgraded Mega microcontroller.

Table 12: Final Budget for Prototype Costs

EXPENSES				
Category	Description	Estimated	Spent	Pending
Electronics	GPS MTK3339	\$ 30.00	\$ 29.95	
	LCD Display	\$ 25.00	\$ 25.00	
	Raspberry Battery	\$ 30.00	\$ 29.95	
	Adafruit BMP085	\$ 20.00	\$ 20.00	
	Adafruit Mega	\$ 25.00	\$ 7.95	
	Bend Sensors	\$ 14.00	\$ 15.95	
	Wiring	\$ 20.00	\$ 19.95	
Casing	Polycarbonate	\$ 10.00	\$ 10.31	
	Adhesive Pad	\$ 15.00	\$ -	
Misc.				
	TOTAL	\$ 189.00	\$ 159.06	\$ -
	Net Reserve		\$	
	(Deficit)		(159.06)	\$ -

Section 8.3: Company Goals and Objectives

The vision of this company is to develop a foundation of excellent product quality while maintaining the customer's needs, as well. As avid skiers, snowboarders, and skateboarders, we know that the product has to withstand hazardous and strenuous conditions as well as expected wear and punishment from the users. Keeping this in mind, the company's primary goal is to create a product to withstand such conditions while maintaining quality performance. Second, it is important for customer's desires to be heard and implemented; and with that, the

subsequent goal of the company is to utilize customer inputs to further improve and develop the sensor system. Our customers are the reason we are in business. Maintaining their satisfaction and listening to their desires is what drives the motivation to create a high quality product.

Keeping these two goals in mind and using them as motivation, this company sees itself in five years as a fully functioning company that manufactures and produces 10,000 units every year. To successfully reach this vision, the product is to be sold at a few select retailers and resorts as a testing period. In the following four years, the distribution will expand to more retailers, with the production capability to increase to an additional 5,000 units. Within the third and fourth years of production, a second sensor system is to be developed, and during this time will undergo prototyping to be introduced to the market and hold firm to a loyal clientele base, who are interested in a company that is not afraid to grow within the ever growing, and competitive, market.

Section 8.4: Product Description

The developed product is a sensor to be advertised on the commercial market for skiers, snowboarders, and skateboarders that gives provides the data they desire; such as speed, elevation, foot pressure, temperature, flex, acceleration, and position. This is achieved by using a variety of sensors; including a GPS, flex sensors, accelerometer, and more in order to provide the said desired data. The sensors are placed in an external polycarbonate casing attached to the ski or board by using an adhesive pad on the bottom of the casing. These sensors than transmit the data via a microcontroller to either an LCD screen displaying a simple application or a memory system, which then user can access and analyze using Matlab code to interpret relevant data. Using this system, performance data was recorded to analyze tricks such as spins and jumps.

The advantage of having such a sensor is that it is specifically designed for the sports of skiing, snowboarding, and skateboarding. This meaning that the user would not have otherwise useless information that they would receive from a sensor designed for performance sports such as running, which might display data such as 'Pace' and 'Lap Time'. The sensor system and its casing are also specifically

designed to deal with the forces and stresses that this particular sensor system would have to endure while being used. Therefore, it is able to operate well and provide the user the desired data, no matter the conditions the user may find themselves in.

As of yet, there are no patent conflicts with the current sensor system. In the future, a patent(s) is required.

Section 8.5: Potential Markets

Currently, there is no other sport sensor system on the market that is specifically designed for skiing, snowboarding, and skateboarding. The only competition comes from large-scale GPS manufacturers, such as Garmin, who have systems that only track performance and position through GPS. Therefore, there is a great niche in the market that could be taken by this new product.

As there is no existing sensor system currently on the action sport market, it is difficult to tell who would be willing to buy a similar item, and how many of them be sold. According to Garmin's annual report for 2011, they sold approximately 16 million units, generating approximately \$2.76 billion in revenue (Annual Garmin Report). This, however, spans multiple products from automotive and marine GPS units, to cycling and running units. In the first year, it is the goal of the company to sell 10,000 units with the potential to add an additional 5,000 units at \$160 a unit, generating predicted revenue of \$1.6 million. This is achieved by first selling at select retailers and resorts within the United States and Canada. From there, continue the production plan of 10,000 units per year with an additional 5,000, if possible, with a second model due to be released in either the third or fourth year of production.

Section 8.6: Competition

Limited results are uncovered in researching previous and current products or projects in the selected action sports market. In terms of finding another snow sports product with similar functionality, the PUSH Snowboarding sensors, backed by Burton and Nokia, is a project with goals similar to those of our company. PUSH snowboarding has four separate components that measure speed, orientation of the snowboard, heart rate, and altitude. The project, however, is still a work in progress, stalemating in its prototyping phase and proving unreliable for sale.

In choosing a system to compare with this company's design, Garmin is the optimum choice. Although there is no snow sports specific device made by Garmin, it boasts being one of the primary leaders in the sensor technology in use for this design. Beginning with the Garmin Forerunner, this product comes in the design of a running watch. It measures what most advanced running sensors do now, calculating the user's heart rate, speed, and route. In addition, it reads the user's steps per minute, ground contact time, and vertical oscillation. Using these three more advanced measurements hopes to maximize the runners pace and rhythm at the comfort of looking at an LED screen on the watch. With all of these measurements, no phone is needed as it sends the data directly to the watch.

The next Garmin product used to compare is the Garmin Edge, designed for bike riding. Similarly to the Forerunner, this product does not require a phone while out doing exercise, as it records its information straight on the device. This particular device, however, is designed more like a car's GPS navigation, as it not only looks like the part, but also attaches to the user's bike handlebars while riding. It contains preloaded maps for both on and off road trails, allowing the bike rider to go on adventures and explore without the worry of getting lost with turn-by-turn directions if needed. This product is heavier than the watch by one ounce at 3.5 ounces, but also has a rechargeable battery that lasts up to 17 hours and is also waterproof. As for the sensors, it displays the user's speed, max speed, average speed, distance, elevation, and, of course, time. Other sensors like power, heart rate, and cadence are added on but sold separately. This product attempts to create a device in a relatively new market of biking sensors, similar to how this company's new design is trying to specifically target the snow sports market.

The last product to compare is the original Nike Plus sensor. The sensor, itself, is the smallest on the market, weighing in at .23 ounces, as well as the cheapest at \$19. In comparison, the Edge carries a price tag of \$300 while the Forerunner is \$450. The sensor functions in sync with a mobile phone, as it displays all of the progress made while running by transferring the information over wirelessly or via a Bluetooth network. The Nike Plus calculates the calories burned, pace, distance traveled, and elapsed time of the workout. The core sensors, too, are very similar to

the two Garmin products. However, unlike the other products the sensor is not water resistant, which is needed for snow activities. This Nike product, though, is easier to integrate into other objects because of its much smaller size; although it does not have a direct display like the two Garmin products, which is why a phone is needed to keep track of the progress.

Section 8.7: Sales/Marketing Strategies

There are several ways in which this new product will be advertised and sold. The first is through commercial advertising on outlets such as radio, billboards, magazines, and television. These would be focused primarily in areas where there is a large concentration of skiers, snowboarders, and skateboarders. Advertising would also take place at various ski resorts throughout the United States and Canada. These would expand with time to include most of the United States, Canada, and other foreign markets.

There needs to be several sales persons to deal with the various financial and marketing aspects of the business. These personnel are dedicated to the promoting and selling of the units to the various vendors throughout the United States and Canada. For the first year, we predict that the need of five marketing persons and three financial professionals to manage this aspect of the business.

The distribution is centralized from a single distributor, who is based, preferably, from somewhere in the Western United States. This would allow for easier distribution to a larger number of retailers and resorts throughout the United States. It is the desire of our company to have a second distribution center in Canada, most likely in the Vancouver area. Instead of distributing the product independently, we use an existing distributor in order to optimize product coverage.

Section 8.8: Manufacturing Plans

The sensor systems, themselves, will be built in the Western United States at a central plant, with the separate parts shipped to the central plant for assembly and shipping after being bought from external manufacturers. The assembly consists of wiring the various sensors together, connecting them to the microcontroller and LCD screens, and then placing the assembled product within the casing. This finished assembly would then be placed in its packaging and stored until shipped via the

distributor. It is expected to take approximately 3 months to construct the assembly production line, and from there take approximately 30 minutes to construct 1 unit per assembly line with 1,000 units kept on hand as on-site inventory.

It is predicted to take approximately \$5 million to get the operations started, and requires additional funding to keep up with expansion as other assembly lines and increased required parts are added to the expenses.

Section 8.9: Product Finances

There are two major parts to consider in the production of this product. The first is the income generated from the sale of the product, as shown in Table 11.

Table 13: Unit Pricing and Revenue Generation

Income	Revenue
Individual Product Sales Cost (per unit)	\$200
Sale of 10,000 Units	\$2.0 Million

For the production of the product itself, price is \$200 per unit. For a year’s production, this equals \$2 million for 10,000 units sold. This, however, does not include the possibility for an additional 5,000 units that could be produced for an additional \$1 million in revenue.

Table 14: Production Cost and Projections

Part	Expense
GPS	\$30
LCD Display	\$25
Battery	\$30
Temperature/Pressure/Altimeter Sensor	\$20
Microcontroller	\$8
Bend Sensor (x2)	\$16
Wiring	\$20
Casing	\$10
Total Retail Cost	\$159
Total Wholesale Cost (40%-30% Discount)	\$111.30 - \$95.40
Total Wholesale Cost for 10,000 Units	\$1,113,000 - \$954,000

Table 14 (continued): Production Cost and Projection

Equipment	\$500,000
Personal Costs (1 year, 25 people)	\$500,000
Facilities Cost	\$1Million
Total Yearly Cost (1 st Year)	\$3.11Million - \$2.95 Million
Total Yearly Cost	\$1.61 Million - \$1.45 Million

The second point to consider are the production expenses and costs shown above in Table 12. For this business plan, we expect to produce 10,000 units per year, which have a retail cost of \$159 per unit. However, if the wholesale prices are taken into account, the cost per unit drops to a range of \$95.40 -111.30. This leads to a total yearly unit cost of \$954,000 - 1,113,000. Other expenses that taken into account are equipment, personnel, and facilities expenses, which amount to \$2 million. This leads to a total expense of \$2.95 million - 3.11 million within the first year and \$1.45 million - 1.61 million every subsequent year after that.

Section 8.10: Service and Warranties

The goal of the product is for it to last at least one ski season, but it is preferable for the sensors to exceed that goal. In the case that the product is damaged or made inoperable, the product would need to be sent back to the factory where it is either repaired or discarded – depending on the type of damage and its severity. The cost of repairs and the person or persons liable depends on the type of damage and whether it is caused by user negligence or is a product defect. If there is a defect in the product, the company is responsible for covering the cost of repairs at no extra cost to the customer. However, if the damage is caused by user negligence, the user is responsible for the cost of repairs or consequent replacement. Types of negligence include intentional destruction of the product, such as striking the casing with a blunt or sharp object with the intention of harming the product, applying an excessive amount of weight to the product, or any other form of intentional harm.

Section 8.11: Financial Plan/Investors Return

The financial plan for the company depends drastically on the funds supplied by investors and the degree of their investments. According to the financial

projections, the company is not able to financially support itself until it becomes profitable, around year 3 or 4 of production. The expected total expenses during this time ranges from \$5.85 million to \$7.94 million, depending on production and personnel expenses. Therefore, to support the business in its beginning stage, an initial investment of \$8 million is needed from investors. Ideally, the investments are split between 80 investors at \$100,000 per investment. With this initial investment, it is predicted that the company can turn a profit ranging from \$60,000 to \$700,000 in its fourth year of production.

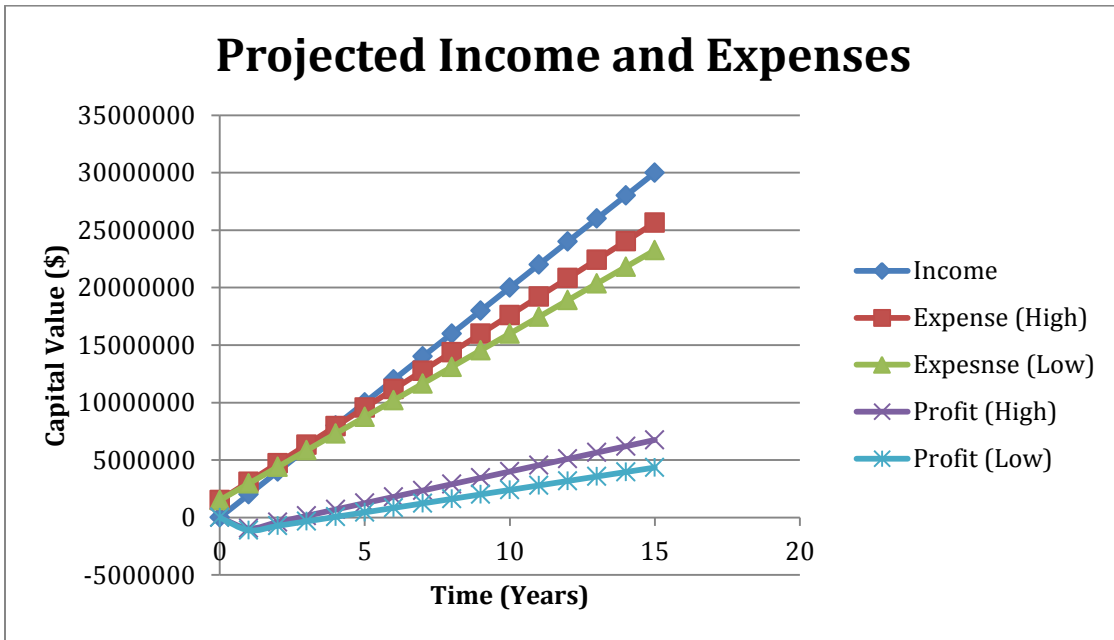


Figure 43: Projected Income and Expenses Over 15 Years

According to the projections shown in Figure 42, the company could turn a \$20 million plus profit by the 7th year of production and \$50 million plus profit by its 12th year of production. These projections, however, do not include the possibilities for expansion. With this in mind, the break-even point for investors will occur between 8-10 years of production as shown in Figure 43 on the next page.

Depending on the unforeseen expenses and costs that are likely to occur, it is believed that for each investor’s investment of \$100,000 they can potentially receive a return between \$200,000 and \$500,000 by the 15th year of production.

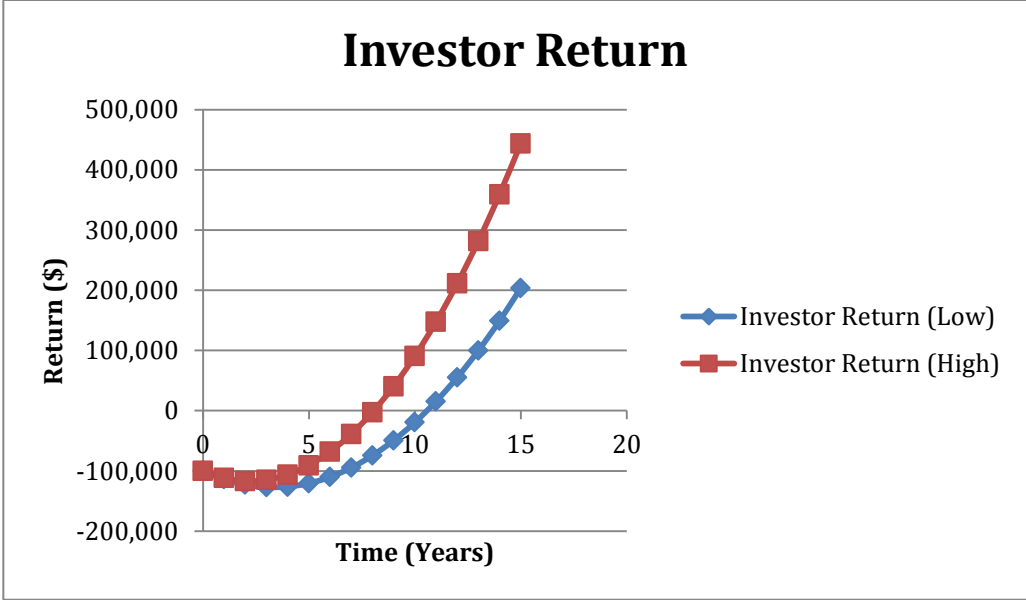


Figure 44: Projected Investors Return Over 15 Years

Chapter 9: Engineering Standards and Realistic Constraints

Section 9.1: Standards and Constraints

There are several standards and constraints that must be met in order for the project to be ethically and economically sound. As evident by the world today, all businesses and projects have implications in a variety of fields. In the case of developing a sensor for the application of skiing, snowboarding, and skateboarding, there are three major categories that set the standards and constraints for this project: manufacturability, health and safety implications, and economic factors. These elements determine the feasibility and the eventual outcome of the project.

Section 9.2: Manufacturing - Victor

There are a small handful of concerns that surround the manufacturability of the sensor to be developed. The first, parts must be available and readily accessible in order to produce a sufficient quantity of the product, itself. This is determined by the vendors of the separate components and materials, especially the sensor suppliers, and whether they are accommodating to the needs requested. In order for manufacturing to be a successful endeavor, it is important, too, that the components themselves cost less than the consumer price for the product. This, however, depends entirely on the simplicity of the design of the system; because the higher the complexity, the longer the product takes to manufacture.

The materials chosen for the casing are the second concern when manufacturing the casing, itself. Acrylic is a hard plastic, but is susceptible to heat, abrasions, and surface scratches. This makes it a difficult material to manufacture, as heat from friction generated from cutting the various sections can cause the material to melt. Such warping and melting results in a distorted shape, leading to a poor fit and more time and money spent to correct the result. Secondly, there is the possibility the acrylic could be scratched during the manufacturing process, which is most likely caused from the handling of the material itself. The consequence of this is an undesirable appearance, which subsequently harms the sale of the product, itself.

Section 9.3: Health and Safety - Adrien

There are also issues surrounding the health and safety implications of developing a product for an action sport. Whenever a product or component is not

designed with the addition of a sensor system in mind, there is the potential that the particular product or component's integrity is compromised. This, in turn, could lead to a structural failure, which then leads to bodily injury or death. There is also the potential that the adhesive fails on the external casing, becoming dislodged and potentially striking the user of the sensor or striking another person nearby the user. With this potential danger, certain steps must be taken in order to ensure that these risks are minimized or negated to protect the user, the manufacturer, and the designer/developer.

These health and safety issues extend to the manufacturing process, as well. Within the manufacturing process, people are exposed to hazardous materials, such as acrylic particles that are extremely dangerous when inhaled. The same hazard exists when handling the adhesive, a plastic epoxy, whose fumes are, also, considered dangerous. This, therefore, creates a hazardous work environment for manufacturers that could potentially lead to lasting health effects and/or death – resulting in lawsuits and harsh fines, affecting the business and its reputation significantly.

Section 9.4: Economic Factors - Robert

The third category that regulates the product is economic factors that might limit and constrain the development and production of the sensor system. These are factors that affect how much profit is generated from the product, itself. One of the causes that affect this is the use of rechargeable batteries to power the components. Using such batteries are more cost efficient, as the product will not require disposable batteries, making it cheaper to produce as well as cheaper for the customer. There is also an economic benefit to using acrylic to construct the casing, as it is easier to produce than creating a casing out of metal. Making one out of metal is a complicated process which requires cutting, shaping, and welding, whereas creating a casing out of polycarbonate only requires cutting the pieces and epoxying them together. This makes it a more time and cost-effective form of production.

Another economic factor to be considered is the acquisition of parts and materials. As shown in the business plan, there are a variety of components needed to build the casing and sensor system. It then makes sense that the acquisition of parts be done at the lowest possible cost. To do this, it is not logical to purchase these

components and materials at market retail price, as a single individual consumer might. Instead, it is more sensible to purchase these items at wholesale cost, such as a corporation or company might do so. By doing this, it will help to reduce the expenses of production, which can, in turn, increase profits and/or reduce the cost of the product to the consumer.

Section 9.5: Usability - Michael

The fourth category relates to the products usability, as it needs to have a clear mode of function and clear data retrieval and analysis. This standard correlates to how many people will recommend and/or buy the product. If the product is native or easy to use, more consumers are likely to consider buying it. There is an entire engineering discipline devoted to usability, in which designers and coders simplify the human to computer interaction in order to make the product available for the product. Usability often means implementation of instructions on how to use the product or creating a manageable interface so that all or most of the target audience is involved. An upgrade to the usability of the casing's design came in the form of a single button, used in order to easily turn on and off the sensor system while enclosed inside the case.

As a designer, decisions on whom the audience for the sensor system is for greatly determines the usability of the product. If the goal is to aim only for technological consumers, then there are differences in functionality, what parts the product is made of, and where the system is used; as opposed to an everyday user who may require a simpler product.

Chapter 10: Conclusion

The goal of this project is to develop a sensor for the commercial market, targeting skiers, snowboarders, and skateboarders, that will give them performance data – such data including speed, elevation, pressure, temperature, flex, acceleration, and position. This is done using a variety of sensors, including a GPS, flex sensors, accelerometer, and others to provide data such as speed, position, foot pressure, position, and temperature. Originally, Nike Plus sport sensors are used in the system, but due to compatibility issues, it could not be incorporated. The sensors in use are placed in an external acrylic casing attached to the ski or board by using an adhesive pad on the bottom of the casing. These sensors then transmit the data via a microcontroller to either an LCD screen displaying a simple application or a memory system, which then user can access and analyze using Matlab code to interpret relevant data.

Several tests were conducted to test the functionality and survivability of the sensor system and casing. The purpose of the first test is to assess whether the sensor system would survive under extreme weather conditions, by placing the casing with the sensors in a freezer and lowering the temperature. The second test that conducted is a dynamic set of tests, which accessed the acceleration response read by the accelerometer during the performance of tricks and jumps. The final test is a vibration-damping test, which evaluated the functionality of the sensors under vibration and to determine how much damping, if any, occurs on the casing and sensors.

Section 10.1: Future Work/Upgrades

Throughout the duration of the project, the team was able to construct a functional sensor-system prototype that could give the desired data that is specified. This was accomplished after several components had to be, either, upgraded, discarded, or replaced. A case constructed of acrylic is made to house the sensor system, but fails on the first intense loading test that is performed on it, revealing a flaw in the casing design. In the future, improvements could be made to the system. Such improvements consist of upgrading the casing material to a higher strength polycarbonate, or using a different epoxy that would hold the pieces together better.

Another other improvement that could be made to the design would be upgrading the battery to a smaller battery with more capacity. This would have several benefits to the design, such as the reduction in size would allow for more space within the casing, which would allow more room for other components to be added to the design. The increased capacity would also allow for longer usages and extend the battery life out even more.

The final change would be changing the interface from a button/LCD screen interface to a Bluetooth-phone interface. This would reduce the amount of parts that would be needed within the casing, increasing the amount of space available within the casing. It would also allow the utilization of the user's phone, which would greatly increase the usability and allow the user to view the data from a remote device, rather than from the casing itself.

Section 10.2: Personal Reflection

Victor Ojeda

When deadlines are assigned to tasks that all build up toward a common end goal, it is crucial to at least come close to the target dates assigned to the smaller tasks in order to successfully reach the final culminated achievement on time. It's ok to not reach the desired deadlines from time to time, but when these deadlines are consistently not met, there is a core dilemma that is causing this repeated failure. The number one reason for this failure specific to our group has to be the lack of teamwork. Each one of us is capable of carrying out the tasks that were needed as well as tasks that other group members performed, however we struggled to get on the same page whether it was making it to meetings together or being unable to tackle the simplest of assignments. We had miscommunication problems and were ambiguous with how we could all contribute and mesh our work together without actually meeting with each other. Personally, I put school work in front of the senior design. Prioritizing is one thing, but there's also a point when you put it second to everything in schoolwork and I should have done a much better time during the school year balancing the two. Not just in being present at meetings which I usually was, but in being a voice of opinion in how we should organize responsibilities which we lacked as a whole because we were all too complacent with taking the back seat.

Not dealing with these issues obviously destine any project for failure, as well as possibly creating other unwanted products. This ranges from feeling anxious and pressure from not finishing on time to feeling angry and causing arguments among group members at the disappointment of failure. This leads to procrastination and will only make things even worse. That is why there is so much importance in setting a precedent early in the process in order to not let the lack of cohesion as a group spiral out of control. If we were able to better communicate our commitment to the project early, it would have lead to better work distribution and an overall greater experience in working on this project together.

Michael Fernandez

There are many issues that have arisen from our lack completion of our Senior Design project. The first and foremost problem is that I (we) have failed to graduate on time. Which also puts myself and my teammates in a lower position in finding roles in the engineering workforce, not to mention the stress and anxiety an incomplete produces. While these things in minor amounts can produce results, the large amount I have personally experienced has greatly affected myself in the fields losing sleep and avoiding parental confrontation. It also provides knowledge that I did not do what I should have and was supposed to do. Procrastination is both detrimental to health and work, in this case senior design, but also in the future, things such as ignoring changes in health or missing work deadlines could cause us to become ill or lose our jobs. If I had done more work earlier in the year, I could have helped save my team from still having an incomplete, I could possibly have a job for fall, and could proudly hold my diploma. For future projects, the knowledge of the faults of procrastination will be useful in preventing this situation from happening again in a work environment. If we had created a more prominent team dynamic and role system, we could have much more easily accomplished our goals and would most likely not be in this position now. We have been continually playing a game of catch-up, but in reality the deadline caught up with us because of our ineffective teamwork and personal work. Finally, I have learned that not only does my procrastination directly affect me, but it affects my team mates as well, giving more of the work load

to them at times. If I and my teammates had completed this assignment earlier, all of our lives would be easier and less stressful.

Robert Ross

The workload for our project proved to exceed our initial expectations as a group and needed to be continued into the summer. With this being the case, each of us as individuals would benefit from an assessment of aspects of the project with an eye for which trends we as a team fall into which hinders our consistency of work output. In the future, our projects will require more precise and minute management of time and deadlines. We were not able to operate completely within the confines of the already generous Senior Design course time allotments. Given that our future endeavors will be arbitrarily paced and more competitive by nature, it is crucial that we as individuals improve our time-management skills to better master the demands of larger-scope projects.

The chief area by which we as a team would have improved is by establishing a team identity early on. Taking early action to establish our own personal interest and stake in the project as well as in the team community would foster a more comfortable and pleasant mindset with regard to the tasks at hand. With a comfortable team-oriented mindset in which we each operate with respect to our known and communicated goals for the project, work would be more comfortable to pursue often, with pressure applied in smaller pieces at a more leisurely pace. If we had assumed responsibility and personal stake in the project early on, our group as a whole would have been more cohesive throughout the entire process.

Instead, we put off meeting for the most part until we already had a clear and present deadline impending. This is, of course, the natural pace of a lot of schoolwork, with drive to work coming along with the pressures of the deadline, but it is less than adequate for a project of this scope. Instead of following a consistent schedule by which we operate as a team, something which only gets more difficult to establish as time goes on, our group instead procrastinated at many of the smaller steps which in turn put our group behind pace overall. As this work draws to a close and we are finishing up the unfortunately lingering catch-up game we wound up in, we can use the experience here as individuals as we engage in further projects. The

difficulties we met as a team resulting from procrastination are self-evident, and will serve to bolster ambitious management of time, particularly in the early stages as the team finds its identity.

Adrien Doiron

There were several major issues with the project that lead to the late completion and missing of deadlines. The first was the lack of cohesion between our teammates. Because of this lack in teamwork, many of the tasks were not accomplished, work was incomplete, and there was mass confusion when trying to do a task. This lead to a major part of work being undone, and placing teammates in hard positions where some were doing large portions of the work involved and others were doing very little.

The second major issue with the project was the failure on my part as the team leader of the project. My major role was to coordinate the project's tasks with other members of the team so that the project could proceed efficiently. In this, I failed entirely, as tasks were not accomplished on time and there was much confusion between teammates. I attribute this to my own lack of leadership skills and my underestimation of the amount of planning and communication needed to run a project. In hindsight, I should have spent more time working as a team leader rather than just trying to do the project on my own, or someone else on the team should have taken on the responsibility as the team leader.

Bibliography

1. Adafruit. "Adafruit MTK3339 Chipset". *Adafruit Industries*. December 5, 2013
http://www.adafruit.com/images/medium/790_MED.jpg
2. "Adafruit Trinket - Mini Microcontroller - 5V Logic -." *Adafruit Industries Blog RSS*. Adafruit Industries, n.d. Web. 06 Dec. 2013.
3. "Apollo 2241." *Adhesive*. Cyberbond L.L.C., n.d. Web. 06 Dec. 2013.
4. "Arduino - Compare." *Arduino - Compare*. N.p., n.d. Web. 3 Oct. 2014.
<<http://arduino.cc/en/Products.Compare>>.
5. Beadmore, Roy. "Loaded Flat Plates." *Loaded Flat Plates*. N.p., 20 Feb. 2013.
Web. 03 Mar. 2014.
6. Bekkala, Michael, Michael Blair, Michael Carpenter, Matthew Guibord, Abhinav Parvataneni, and Shanker Balasubramaniam, Dr. Speed and Distance Sensor for Skiers and Snowboarders. Air Force Research Laboratory, Michigan State University, 11 Dec. 2009. Web. 6 Dec. 2013.
7. "BLE112 Bluetooth Smart Module." BLE112 Bluetooth Smart Module – Bluegiga. Bluegiga Technologies, n.d. Web. 06 Dec. 2013.
8. Bluegiga. "Bluegiga BLE112." Bluegiga Industries. December 5, 2013
http://techforum.bluegiga.com/files/bluegiga/Press%20kit%20pictures/BLE112_RGB_S.png
9. "BMP085 Barometric Pressure/Temperature/Altitude Sensor-5V Ready." *Adafruit Industries Blog RSS*. Adafruit Industries, n.d. Web. 06 Dec. 2013.
10. Cyro. *Acrylite FF Physical Properties Brochure* (n.d.): n. pag. *SD Plastics*. Web. 13 June 2014.
<<http://www.sdplastics.com/acryliteliterature/1121DFFPhysicalProperties%5B1%5D.pdf>>.
11. Frank, Michael. "Adventure Journal." Gear Preview: Garmin Forerunner 620 Running Watch. *AdventureJournal*, 16 Sept. 2013. Web. 26 Oct. 2013.
12. "Garmin Edge." *Garmin*. N.p., n.d. Web. 26 Oct. 2013.
13. "Garmin Forerunner." *Garmin*. N.p., n.d. Web. 26 Oct. 2013.
14. "Garmin(R) Edge(R) Touring and Edge Touring Plus -- New GPS Devices Designed For Navigating ByBike." *The Wall Street Journal*. N.p., 28 Aug.

2013. Web.
15. *Glass vs. Acrylic Comparison* (n.d.): n. pag. Web.
<<http://www.acrylite.net/sites/dc/Downloadcenter/Evonik/Product/ACRYLITE/acrylite®-acrylic-vs-glass.pdf>>.
 16. Hacknmod. "Iphone Wireless Router". Apple Inc. December 5, 2013.
<http://hacknmod.com/wp-content/uploads/2008/10/iphone-wireless-router.jpg>
 17. Nike. "Nike Sport + Sensors". Nike Inc. December 5, 2013
http://media.t3.com/img/resized/ni/xl_NikeSensor_1.jpg
 18. "Nike Plus." *Nike*. N.p., n.d. Web. 26 Oct. 2013.
 19. "Nokia X Burton - TVCs." Nokia X Burton - TVCs. N.p., n.d. Web. 03 Feb. 2014.
 20. "Number of Skiers & Snowboarders in the USA, 2013." *Statista*. Scarborough Research, n.d. Web. 6 Mar. 2014.
<<http://www.statista.com/statistics/227427/number-of-skiers-and-snowboarders-usa/>>.
 21. "Olympus LI-42B Rechargeable Lithium-Ion Battery." *Olympus*. Olympus Industries, n.d. Web. 06 Dec. 2013.
 22. Report, Garmin Annual. *GRMN* (n.d.): n. pag. Web.
<https://www8.garmin.com/aboutGarmin/invRelations/reports/2011_Annual_Report.pdf>.
 23. Roark, Raymond J., and Warren C. Young. *Roark's Formulas for Stress and Strain*. New York: McGraw-Hill, 1989. Print.
 24. Shopify. "Adafruit BMP085 Barometric Pressure/Temperature/Altitude Sensor". Adafruit Industries. December 5, 2013
http://cdn.shopify.com/s/files/1/0215/6458/products/BMP085_1_1024x1024.jpg?v=1383858181
 25. "Silicone Gaskets and Pads, Custom Gaskets, Gasket Manufacturers | Stockwell Elastomerics." *Silicone Gaskets and Pads, Custom Gaskets, Gasket Manufacturers | Stockwell Elastomerics*. Stockwell Elastomerics, Inc., n.d. Web. 06 Dec. 2013.
 26. "STOMPGRIP Online | Welcome." *STOMPGRIP Online | Welcome*. Stompgrip

Inc., n.d. Web. 06 Dec. 2013.

27. "Ultimate GPS Module - 66 Channel W/10 Hz Updates - MTK3339 Chipset."

Adafruit Industries Blog RSS. Adafruit Industries, n.d. Web. 06 Dec. 2013

Appendix 1:
PDS

MECH 194 - Advanced Design I
Product Design Specification - Targets & Benchmarks
Fall 2013

Design Project: Nike Snowboard/Ski Sensor Student Names: Doiron, Fernandez, Ojeda

Benchmark System 1: Garmin Forerunner Date: October 18th, 2013

Benchmark System 2: Garmin Edge

Benchmark System 3: Nike + Sensor

Characteristic / Parameter	Parameter Units	Design Criticality	Design Target	Benchmark 1 Range	Benchmark 2 Range	Benchmark 3 Range
Speed	m/hr	High	50 m/hr	65 80 m/hr	< 80 m/hr	< 25 m/hr
Elevation	ft	High	14,000 ft	< 10,000 ft	< 10,000 ft	—
Time Clock	Hours	High	24 hrs	24 hrs	24 hrs	24 hrs
Shock Lifetime	Months	Medium	8 months	< 2 years	< 2 years	1000 hrs
Cost	\$	Medium	50	\$450	\$200	\$19
Price	\$/in	Medium	10 in ³	3.02 in ³	7.4 in ³	7.4 in ³
Size	oz	High	> 4 oz	2.5 oz	3.5 oz	0.23 oz
Mass	oz	High	Impact Stress, Buck	—	—	Impact Stress, Force
Strength	—	Medium	Adhesive	Wristband	Clamps on	Insert into side
Installation	—	Medium	W/comp & phone	Not compatible	17 hrs, rechargeable	1000 hrs
Reliability	—	Low	> 5 m/hr	> 0.5 m/hr	w/comp	w/comp & phone
Compatibility (speed)	m/hr	Medium	> 50 ft	250 ft	> 0.5 m/hr	90% accuracy
Accuracy (Elevation)	ft	Medium	No Markers	Replaceable	Replaced	Replaced
Maintenence	—	Low	Phone/LCD Screen	Watch Screen	GPS Screen	Phone Display
Ergonomics	Yes/No	Low	Yes	Yes	Yes	No
Weather Resistant	Yes/No	High	—	—	—	—

Appendix 2: Timeline
Timeline for Fall Quarter

Date	Goal
10/30	Complete Preliminary Design 1: Casing
11/8	Complete Preliminary Design 2: Circuitry
11/15	Complete Preliminary Design 3: Interface
11/22	Complete Customer Analysis
11/22	Preliminary List of Components Needed to Construct Prototype
11/29	
12/6	

Timeline for Winter Quarter

Date	Goal
1/10	Complete Preliminary Design 3: Circuitry
1/17	Finalize Prototype Drawings
1/24	Complete Assembly Drawings
1/31	Finalize List of Components Needed to Construct Prototype
2/14	Acquire Components and Materials Needed
2/21	Finalize Manufacturing Process
2/28	Begin Construction of Prototype

Spring Timeline

Date	Goal
3/14	Complete Construction: Casing
3/28	Complete Construction: Circuitry
4/4	Complete Construction: Interface
4/11	Begin Prototype Testing
4/18	Design of Prototype 2
4/25	Complete Prototype 2
5/2	Test Prototype 2
5/9	Design of Prototype 3 (Time allowing)
6/2	Open House

Appendix 3: Budget

Income

Source of Income	Amount
Sale of Product	\$160 (Est.)

Expenses

Part	Cost
Adafruit MTK3339 GPS peripheral	\$30
Nike + Sensor	\$19
Olympus LI 42B Camera Battery	\$12
Adafruit BMP085 Pressure/Temperature/Altitude Sensor	\$20
Adafruit Trinket (5v)	\$8
BLE112 Bluetooth Smart Module	\$14
Polycarbonate Casing	\$10
Packaging	\$20
Total Cost	\$133

Appendix 4: Sensor System Coding **System Master Loop Coding**

```
#include <stdint.h>
#include <stdlib.h>
#include <floatToString.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_NeoPixel.h>
#include <SoftwareSerial.h>

#include <Adafruit_GPS.h>
#include <TouchScreen.h>
#include <TFT.h>
#include <Wire.h>

#define USE_SERIAL 0

//>>> Definitions for GPS
unsigned long GPSSlinkWait = 0;
unsigned long GPSDisplayTime = 0;
unsigned long timer = millis();
#define GPSECHO true
Adafruit_GPS GPS(&Serial1);
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy
int GPSsetupBool=0;

//>>> Definition for Pressure/Temperature
#define SEA_LEVEL_PRESSURE 1015.5
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);
unsigned long beginTime = 0;
unsigned long BMP180PollTime = 0;
int pollNumber = 0;
float pressureArray[60]; //holds 60 minutes of data, declares to -999 which is flagged
in the displaybmp180 function to not
float tempArray[60]; //holds 60 minutes of data
float altitudeArray[60]; //holds 60 minutes of data
float maxPressure = -100000, minPressure = 100000, maxTemp = -120, minTemp =
120, minAlt = 30000, maxAlt = -30000; //values designed to get rewritten on first
pass

//>>> Definitions for Display
unsigned long dispGPSStartTime = millis();
unsigned long callTime = 0;
```

```

unsigned long bmp180start = millis();
byte prevDisplay = 0;
unsigned long BMP180DisplayTime = 0;
static unsigned int TS_MINX, TS_MAXX, TS_MINY, TS_MAXY;
static unsigned int MapX1, MapX2, MapY1, MapY2;
TouchScreen ts = TouchScreen(17, A2, A1, 14, 300);
char printBuffer[25]; // <---- used to convert float values to strings for LCD to
display

//>>> Definitions for End_Bars
#define endBarsPin 23
#define brightness 23 // <----- This is WAY low for in sunlight
Adafruit_NeoPixel endBars = Adafruit_NeoPixel(8, endBarsPin, NEO_GRB +
NEO_KHZ800);
int i=0, noseReading = 0, tailReading = 0,tailMidVal = 0,noseMidVal = 0,fullBends =
0,pixelVal = -1;
unsigned long lastSet;
const int tailBendBar = A8;
const int noseBendBar = A9;

//>>> Definitions for Accelerometer
int xdegree=0,ydegree=0,zdegree=0;
unsigned long AccelPollTime = millis();
const int xInput = A10;
const int yInput = A11;
const int zInput = A12;
const int buttonPin = 22;
boolean flatFLAG = 0;
// Raw Ranges:
// initialize to mid-range and allow calibration to
// find the minimum and maximum for each axis
int xFlat=0, xRawMin = 512, xRawMax = 512, yFlat=0, yRawMin = 512, yRawMax
= 512, zFlat=0, zRawMin = 512, zRawMax = 512;
// Take multiple samples to reduce noise
const int sampleSize = 25;

void setup(){
  Display_setup();
  // Display the title screen for 6 seconds
  delay(6000);
  Serial.begin(115200);
  beginTime = millis();
  Accelerometer_setup();

```

```

End_Bars_setup();
Pressure_setup();
GPS_setup();
}

void loop() {
//manages timers incase either they or millis() overflow
if (GPSlinkWait > millis()) GPSlinkWait = millis();
if (GPSDisplayTime > millis()) GPSDisplayTime = millis();
if (beginTime > millis()) beginTime = millis();
if (BMP180PollTime > millis()) BMP180PollTime = millis();
if (dispGPSStartTime > millis()) dispGPSStartTime = millis();
if (bmp180start > millis()) bmp180start = millis();
if (BMP180DisplayTime > millis()) BMP180DisplayTime = millis();
if (lastSet > millis()) lastSet = millis();
if (AccelPollTime > millis()) AccelPollTime = millis();

Accelerometer_loop();
End_Bars_loop();
if(GPSsetupBool==0)
{
while(millis()-beginTime < 5000)
GPS_loop();
GPSsetupBool = 1;
GPS_setup_display();
}
GPS_loop();
End_Bars_loop();
// Every 80ms take data point
if(millis()-BMP180PollTime >= 80)
{
//this loop takes ~40ms
Pressure_loop();
BMP180PollTime = millis();
if(pollNumber==60)
{
pollNumber = 0;
}
}
End_Bars_loop();
if(millis()-GPSDisplayTime > 10000 && prevDisplay == 2 && millis()-
AccelPollTime > 10000)
{
prevDisplay = 0;
GPSDisplayTime = millis();
}
}

```

```

Display_GPS();
}
if(millis()-AccelPollTime > 10000 && prevDisplay == 0 && millis()-
BMP180DisplayTime > 10000)
{
  prevDisplay = 1;
  AccelPollTime = millis();
  if(!flatFLAG)
  {
    xFlat = ReadAxis(xInput)*5/3.3;
    yFlat = ReadAxis(yInput)*5/3.3;
    zFlat = ReadAxis(zInput)*5/3.3;
    flatFLAG = 1; // <---- This means it has just set the current orientation (flat on the
ground per instructions) as flat
  }

  End_Bars_loop();
  Display_Accelerometer();
  End_Bars_loop();
}
if(millis()-BMP180DisplayTime > 10000 && prevDisplay == 1)
{
  prevDisplay = 2;
  BMP180DisplayTime = millis();
  //Resets the number of counts (and also where the array is indexed)
  Display_BMP180();
}
}

```

Accelerometer Code Script associated with ADXL326 peripheral

```
void Accelerometer_setup()
{
  //This long Display-heavy routine is the GUI For calibrating the gyroscope
  Tft.fillRectangle(0,0,239,319,BLACK);
  Tft.drawString("STEP",18,24,3,WHITE);
  Tft.fillCircle(52,84,36,RED);
  Tft.drawString("1",26,56,5,WHITE);
  Tft.drawRectangle(60,68,176,44,RED);
  Tft.drawString("Calibrate",92,76,2,RED);
  Tft.drawString("INTERNAL GYROSCOPE",92,102,1,RED);
  delay(750);
  Tft.drawString("Place the board on a flat",30,126,1,WHITE);
  Tft.drawString(" surface, and press the",30,138,1,WHITE);
  Tft.drawString(" calibration button for",30,150,1,WHITE);
  Tft.drawString(" about half a second.",30,162,1,WHITE);

  // Display calibration progress fill-circles
  Tft.drawCircle(14,138,12,RED);
  Tft.drawCircle(14,138+28,12,RED);
  Tft.drawCircle(14,138+28+28,12,RED);
  Tft.drawCircle(14,138+28+28+28,12,RED);
  Tft.drawCircle(14,138+28+28+28+28,12,RED);
  Tft.drawCircle(14,138+28+28+28+28+28,12,RED);
  while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
  until the button is pressed
  for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
  ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries
  Tft.fillRectangle(28,126,220,110,BLACK);
  delay(300);
  Tft.fillCircle(14,138,12,GREEN);

  Tft.drawString("Hold the board vertically",30,126,1,WHITE);
  Tft.drawString(" on the TOE-side edge.",30,138,1,WHITE);
  Tft.drawString(" Press calibration button",30,150,1,WHITE);
  Tft.drawString(" again, for half a second.",30,162,1,WHITE);

  while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
  until the button is pressed
  for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
  ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries
  Tft.fillRectangle(28,126,220,110,BLACK);
  delay(300);
  Tft.fillCircle(14,138+28,12,GREEN);
```



```

Tft.drawString("Hold the board vertically",30,126,1,WHITE);
Tft.drawString(" on the HEEL-side edge.",30,138,1,WHITE);
Tft.drawString(" Press calibration button",30,150,1,WHITE);
Tft.drawString(" again, for half a second.",30,162,1,WHITE);

while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
until the button is pressed
for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries
Tft.fillRect(28,126,220,110,BLACK);
delay(300);
Tft.fillCircle(14,138+28+28,12,GREEN);

Tft.drawString("Hold the board with the",30,126,1,WHITE);
Tft.drawString(" NOSE on the GROUND and ",30,138,1,WHITE);
Tft.drawString(" tail in the air, directly",30,150,1,WHITE);
Tft.drawString(" vertically. Once again,",30,162,1,WHITE);
Tft.drawString(" press the calibration",30,174,1,WHITE);
Tft.drawString(" button for half a second.",30,186,1,WHITE);

while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
until the button is pressed
for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries
Tft.fillRect(28,126,220,110,BLACK);
delay(300);
Tft.fillCircle(14,138+28+28+28,12,GREEN);

Tft.drawString("Hold the board with the",30,126,1,WHITE);
Tft.drawString(" TAIL on the GROUND and ",30,138,1,WHITE);
Tft.drawString(" nose in the air, directly",30,150,1,WHITE);
Tft.drawString(" vertically. Once again,",30,162,1,WHITE);
Tft.drawString(" press the calibration",30,174,1,WHITE);
Tft.drawString(" button for half a second.",30,186,1,WHITE);

while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
until the button is pressed
for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries

```

```

Tft.fillRectangle(28,126,220,110,BLACK);
delay(300);
Tft.fillCircle(14,138+28+28+28+28,12,GREEN);

Tft.drawString("Hold the board upside-down",30,126,1,WHITE);
Tft.drawString(" exactly parallel with the",30,138,1,WHITE);
Tft.drawString(" ground. One more",30,150,1,WHITE);
Tft.drawString(" time, press the",30,162,1,WHITE);
Tft.drawString(" calibration button.",30,174,1,WHITE);

while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
until the button is pressed
for(i=0;i<50;i++) { AutoCalibrate(ReadAxis(xInput)*5/3.3,
ReadAxis(yInput)*5/3.3,ReadAxis(zInput)*5/3.3); } // Calibrates with 5 data entries
Tft.fillRectangle(28,126,220,110,BLACK);
delay(300);
Tft.fillCircle(14,138+28+28+28+28+28,12,GREEN);

Tft.drawString("CALIBRATION",40,76+80,2,CYAN);
Tft.drawString("COMPLETE!",62,76+104,2,CYAN);
Tft.drawString("Place the board on a",40,150+64,1,WHITE);
Tft.drawString(" flat surface, and press",40,162+64,1,WHITE);
Tft.drawString(" calibration button",40,174+64,1,WHITE);
Tft.drawString(" to continue!",40,186+64,1,WHITE);

while(digitalRead(buttonPin) == HIGH){End_Bars_loop();} // <---- Delay here
until the button is pressed

xFlat = ReadAxis(xInput)*5/3.3;
yFlat = ReadAxis(yInput)*5/3.3;
zFlat = ReadAxis(zInput)*5/3.3;

}

void Accelerometer_loop() //This function takes ~13ms
{
int xRaw = ReadAxis(xInput)*5/3.3;
int yRaw = ReadAxis(yInput)*5/3.3;
int zRaw = ReadAxis(zInput)*5/3.3;

if (digitalRead(buttonPin) == LOW)
{

```

```

    AutoCalibrate(xRaw, yRaw, zRaw);
}
else
{
    if(USE_SERIAL)
    {
        Serial.print("Raw Ranges: X: ");
        Serial.print(xRawMin);
        Serial.print("-");
        Serial.print(xRawMax);

        Serial.print(", Y: ");
        Serial.print(yRawMin);
        Serial.print("-");
        Serial.print(yRawMax);

        Serial.print(", Z: ");
        Serial.print(zRawMin);
        Serial.print("-");
        Serial.print(zRawMax);
        Serial.println();
        Serial.print(xRaw);
        Serial.print(", ");
        Serial.print(yRaw);
        Serial.print(", ");
        Serial.print(zRaw);
    }

    // Convert raw values to 'milli-Gs"
    long xScaled = map(xRaw, xRawMin, xRawMax, -1000, 1000);
    long yScaled = map(yRaw, yRawMin, yRawMax, -1000, 1000);
    long zScaled = map(zRaw, zRawMin, zRawMax, -1000, 1000);

    // re-scale to fractional Gs
    float xAccel = xScaled / 1000.0;
    float yAccel = yScaled / 1000.0;
    float zAccel = zScaled / 1000.0;

    if(USE_SERIAL)
    {
        Serial.print(" :: ");
        Serial.print(xAccel);
        Serial.print("G, ");
        Serial.print(yAccel);
        Serial.print("G, ");
        Serial.print(zAccel);
    }
}

```

```

    Serial.println("G");
  }

}

//
// Read "sampleSize" samples and report the average
//
int ReadAxis(int axisPin)
{
  long reading = 0;
  analogRead(axisPin);
  delay(1);
  for (int i = 0; i < sampleSize; i++)
  {
    reading += analogRead(axisPin);
  }
  return reading/sampleSize;
}

//
// Find the extreme raw readings from each axis
//
void AutoCalibrate(int xRaw, int yRaw, int zRaw)
{
  if(USE_SERIAL)
  {
    Serial.println("Calibrate");
  }
  if (xRaw < xRawMin)
  {
    xRawMin = xRaw;
  }
  if (xRaw > xRawMax)
  {
    xRawMax = xRaw;
  }

  if (yRaw < yRawMin)
  {
    yRawMin = yRaw;
  }
  if (yRaw > yRawMax)
  {
    yRawMax = yRaw;
  }
}

```

```
}  
  
if (zRaw < zRawMin)  
{  
    zRawMin = zRaw;  
}  
if (zRaw > zRawMax)  
{  
    zRawMax = zRaw;  
}  
}
```

Seedstudio 2.8" TFT LCD display Script:

```
void Display_setup(){

  Tft.init();
  initTouchScreenParameters();
  //Title lines
  Tft.fillRectangle(0,40,14,60,WHITE);
  Tft.fillRectangle(240-16,40,16,60,WHITE);
  Tft.drawString("SNOWBOARD",12,43,3,BLUE);
  Tft.drawString("data",28,69,2,GREEN);
  Tft.drawString("TRACKER",96,77,2,0x003366);
  //Contributors
  Tft.drawRectangle(20,220,199,60,RED);
  Tft.drawString("AJ Doiron",22,223,1,BLUE);
  Tft.drawString("- MECH",168,223,1,WHITE);
  Tft.drawString("Michael Fernandez",22,239,1,BLUE);
  Tft.drawString("- MECH",168,239,1,WHITE);
  Tft.drawString("Robert Ross",22,255,1,BLUE);
  Tft.drawString("- ELEN",168,255,1,WHITE);
  Tft.drawString("Victor Ojeda",22,271,1,BLUE);
  Tft.drawString("- MECH",168,271,1,WHITE);

}

void Display_loop(){
}

void Display_Accelerometer(){
  callTime = millis();
  Tft.fillRectangle(0,0,240,320,BLACK);

  //get x y z values as strings
  int currentX=0,currentY=0,currentZ=0;
  Tft.drawString(" X:",112,44,2,RED);      Tft.drawString("Nose-
Tail",12+8,16,1,RED);      Tft.drawString("in degrees",158,48,1,RED);
  Tft.drawString(" Y:",112,136,2,0x00F420);  Tft.drawString("Left-
Right",12+6,16+92,1,0x00F420);  Tft.drawString("in
degrees",158,48+92,1,0x00F420);
  Tft.drawString(" Z:",112,228,2,YELLOW);
  Tft.drawString("Verticality",12,16+92+92,1,YELLOW);  Tft.drawString("in
degrees",158,48+92+92,1,YELLOW);

  while(millis()-callTime <= 10000)
  //loop for 10 seconds of refreshing the axes
```

```

//SHOULD CALL ENDBARS FUNCTIONS SO THAT THEY ARE STILL
ACTIVE
{
currentX = ReadAxis(xInput)*5/3.3;
currentY = ReadAxis(yInput)*5/3.3;
currentZ = ReadAxis(zInput)*5/3.3;
AutoCalibrate(currentX, currentY, currentZ);
End_Bars_loop();

//x value
if(xFlat <( ReadAxis(xInput)*5/3.3 ))
{
xdegree=map(currentX, xFlat, xRawMax,0,90)+13;// <---- X angle adjustment
factor simply added after the fact MAY RUIN ACCURACY, watch out for it
}
else
{
xdegree=map(currentX, xRawMin, xFlat,-90,0)+13;// <---- X angle adjustment
factor simply added after the fact MAY RUIN ACCURACY, watch out for it
}
dtostrf(xdegree,3,0,printBuffer);
Tft.fillRectangle(120,64,80,24,BLACK);
Tft.drawString(printBuffer,120,64,3,RED);
Tft.fillRectangle(23,31,66,74,RED);
Tft.drawLine(24,68,88,56,BLACK); //x
Tft.drawLine(56,36,56,127,BLACK); //y
if(xdegree >= 0 && xdegree < 45)
{
Tft.fillCircle(82,68 - map(xdegree,0,45,0,32),5,BLUE);
Tft.fillCircle(30,68 + map(xdegree,0,45,0,32),5,BLUE);
}
if(xdegree >= 45 && xdegree < 90)
{
Tft.fillCircle(82 - map(xdegree,45,90,0,32),68 - 32,5,BLUE);
Tft.fillCircle(30 + map(xdegree,45,90,0,32),68 + 32,5,BLUE);
}
if(xdegree >= -45 && xdegree < 0)
{
Tft.fillCircle(82,68 + map(xdegree,-45,0,0,32),5,BLUE);
Tft.fillCircle(30,68 - map(xdegree,-45,0,0,32),5,BLUE);
}
if(xdegree >= -90 && xdegree < -45)
{
Tft.fillCircle(82 - map(xdegree,-90,-45,0,32),68 + 32,5,BLUE);
Tft.fillCircle(30 + map(xdegree,-90,-45,0,32),68 - 32,5,BLUE);
}
}

```

```

}

End_Bars_loop();

//y value
if(yFlat <( ReadAxis(yInput)*5/3.3 ))
{
ydegree=map(currentY, yFlat, yRawMax,0,90)+13;
}
else
{
ydegree=map(currentY, yRawMin, yFlat,-90,0)+13;
}
dtostrf(ydegree,3,0,printBuffer);
Tft.fillRectangle(120,156,80,24,BLACK);
Tft.drawString(printBuffer,120,156,3,0x00F420);
Tft.fillRectangle(23,121,66,74,0x00F420);
Tft.drawLine(24,160,88,160,BLACK); //x
Tft.drawLine(56,128,56,192,BLACK); //y
if(ydegree >= 0 && ydegree < 45)
{
Tft.fillCircle(82,68 - map(ydegree,0,45,0,32)+92,5,BLUE);
Tft.fillCircle(30,68 + map(ydegree,0,45,0,32)+92,5,BLUE);
}
if(ydegree >= 45 && ydegree < 90)
{
Tft.fillCircle(82 - map(ydegree,45,90,0,32),68 - 32+92,5,BLUE);
Tft.fillCircle(30 + map(ydegree,45,90,0,32),68 + 32+92,5,BLUE);
}
if(ydegree >= -45 && ydegree < 0)
{
Tft.fillCircle(82,68 + map(ydegree,0,-45,0,32)+92,5,BLUE);
Tft.fillCircle(30,68 - map(ydegree,0,-45,0,32)+92,5,BLUE);
}
if(ydegree >= -90 && ydegree < -45)
{
Tft.fillCircle(82 - map(ydegree,-45,-90,0,32),68 + 32+92,5,BLUE);
Tft.fillCircle(30 + map(ydegree,-45,-90,0,32),68 - 32+92,5,BLUE);
}

//Again, check and light the end bars
End_Bars_loop();
if(millis()-BMP180PollTime >= 80)
{
//this loop takes ~40ms
GPS_loop();
}

```



```

Pressure_loop();
BMP180PollTime = millis();

}

End_Bars_loop();
//z value
if(zFlat <( ReadAxis(zInput)*5/3.3 ))
{
zdegree=map(currentZ, zFlat, zRawMax,0,90)+13;
}
else
{
zdegree=map(currentZ, zRawMin, zFlat,-90,0)+13;
}
dtostrf(zdegree,3,0,printBuffer);
Tft.fillRectangle(120,248,120,24,BLACK);
Tft.drawString(printBuffer,120,248,3,YELLOW);
Tft.fillRectangle(23,214,66,76,YELLOW);
Tft.drawLine(24,252,88,216,BLACK); //x
Tft.drawLine(56,220,56,283,BLACK); //y

if(zdegree >= 0 && zdegree < 45)
{
Tft.fillCircle(82,68 - map(zdegree,0,45,0,32)+92+92,5,BLUE);
Tft.fillCircle(30,68 + map(zdegree,0,45,0,32)+92+92,5,BLUE);
}
if(zdegree >= 45 && zdegree < 90)
{
Tft.fillCircle(82 - map(zdegree,45,90,0,32),68 - 32+92+92,5,BLUE);
Tft.fillCircle(30 + map(zdegree,45,90,0,32),68 + 32+92+92,5,BLUE);
}
if(zdegree >= -45 && zdegree < 0)
{
Tft.fillCircle(82,68 + map(zdegree,0,-45,0,32)+92+92,5,BLUE);
Tft.fillCircle(30,68 - map(zdegree,0,-45,0,32)+92+92,5,BLUE);
}
if(zdegree >= -90 && zdegree < -45)
{
Tft.fillCircle(82 - map(zdegree,-45,-90,0,32),68 + 32+92+92,5,BLUE);
Tft.fillCircle(30 + map(zdegree,-45,-90,0,32),68 - 32+92+92,5,BLUE);
}

// Every 300ms take data point <---- REMOVE THIS when time scale changes to
get data once a MINUTE
End_Bars_loop();

```

```

End_Bars_loop();

}

}

void Display_GPS(){

GPS_loop();

dispGPSStartTime = millis();
Tft.fillRectangle(0,0,240,320,BLACK);

Tft.drawString(" GPS info ",48,20,2,YELLOW);
Tft.drawString("TIME:",8,64,2,WHITE);
Tft.drawString("DATE:",28,64+48,2,WHITE);
Tft.drawString("Speed: ",14,126+12+12,2,BLUE);
Tft.drawString("MPH",28+172,126+12+14,1,GREEN);
Tft.drawString("(accurate to ~.3 mph)",32,126+12+12+20,1,BLUE);
Tft.drawString("Alt. :",14,126+62,2,BLUE);
Tft.drawString("feet",28+172,126+64,1,GREEN);
dtostrf(GPS.latitude/100,5,2,printBuffer);
Tft.drawString(printBuffer,14,126+62+28,2,RED);
Tft.drawString("degrees N",14+114,126+62+28+2,1,RED);
dtostrf(GPS.longitude/100,5,2,printBuffer);
Tft.drawString(printBuffer,14,126+62+28+24,2,RED);
Tft.drawString("degrees W",14+114,126+62+28+24,1,RED);
Tft.drawString("Tracking: ",14,126+62+30+24+24,1,WHITE);

Tft.drawString(" All of this information",8,126+62+30+24+24+18,1,CYAN);
Tft.drawString(" comes from SPACE!",8,126+62+30+24+24+18+12,1,CYAN);
while (millis()-dispGPSStartTime < 10000)
{

if (GPS.newNMEAreceived() {
if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag
to false
delay(100); // we can fail to parse a sentence in which case we should just wait
for another
}
}
}

```

```

Tft.fillRectangle(86,64,136,16,BLACK);

dtostrf((GPS.hour + 4)% 12 + 1,2,0,printBuffer);
Tft.drawString(printBuffer,28+42+16,64,2,CYAN);
Tft.drawString(":",28+42+16+16+16,64,2,WHITE);

dtostrf(GPS.minute,2,0,printBuffer);
if(GPS.minute < 10)
{
Tft.drawString("0",28+42+16+16+20+8,64,2,CYAN);
}
Tft.drawString(printBuffer,28+42+20+16+16+8,64,2,CYAN);
End_Bars_loop();
Tft.drawString(":",28+42+16+16+32+24,64,2,WHITE);
dtostrf(GPS.seconds,2,0,printBuffer);
if(GPS.seconds < 10)
{
Tft.drawString("0",28+42+30+20+48,64,2,CYAN);
}
Tft.drawString(printBuffer,28+42+30+8+20+48,64,2,CYAN);

Tft.fillRectangle(28+96,64+48,96,16,BLACK);
dtostrf((GPS.month)% 12,2,0,printBuffer);
Tft.drawString(printBuffer,28+96,64+48,2,CYAN);
Tft.drawString("/",28+96+32,64+48,2,WHITE);

dtostrf(GPS.day,2,0,printBuffer);
Tft.drawString(printBuffer,28+96+48,64+48,2,CYAN);

Tft.fillRectangle(110,150,76,16,BLACK);
dtostrf(GPS.speed*1.15078,4,2,printBuffer); //converts knots to mph by
knots*1.15078
Tft.drawString(printBuffer,28+76+6,126+12+12,2,GREEN);

Tft.fillRectangle(110,126+62,76,16,BLACK);
dtostrf(GPS.altitude*3.28084,5,0,printBuffer); //converts m to feet
Tft.drawString(printBuffer,28+76+6,126+62,2,GREEN);

dtostrf(GPS.altitude*3.28084,5,0,printBuffer);

```

```

Tft.drawString(printBuffer,28+76+6,126+62,2,GREEN);

dtostrf(GPS.satellites,2,0,printBuffer);
Tft.fillRectangle(84,262,32,16,BLACK);

Tft.drawString(printBuffer,88,262,2,YELLOW);
if(GPS.satellites==1){
Tft.drawString("Satellite",14+54+20+32,126+62+30+24+24,1,WHITE);}
else {Tft.drawString("Satellites",14+54+42+32,126+62+30+24+24,1,WHITE);}

End_Bars_loop();

}
}

void GPS_setup_display(){
//This long Display-heavy routine is the GPS setup display
Tft.fillRectangle(0,0,239,319,BLACK);
Tft.drawString("STEP",18,24,3,WHITE);
Tft.fillCircle(38,84,36,RED);
Tft.drawString("2",12,56,5,WHITE);
Tft.drawRectangle(60,68,176,44,RED);
Tft.drawString("Initialize",76,76,2,RED);
Tft.drawString("GPS SYSTEM LINK",86,102,1,RED);

delay(500);
GPSlinkWait = millis();
while(!GPS.fix && millis()-GPSlinkWait <= 10000)
{
GPS_loop();
}
if (GPS.fix)
{
Tft.drawString("System Link",20,76+80,2,CYAN);
Tft.drawString("Established!",20,76+104,2,CYAN);
delay(750);
Tft.fillRectangle(20,76+80,200,120,BLACK);

Tft.drawString("TIME:",28,126,1,WHITE);
dtostrf((GPS.hour + 4)%12 + 1,2,0,printBuffer);
Tft.drawString(printBuffer,28+42,126,1,CYAN);

```

```

Tft.drawString(":",28+42+16,126,1,CYAN);

Tft.drawString("DATE:",28,126+12,1,WHITE);
dtostrf((GPS.month)%12,2,0,printBuffer);
Tft.drawString(printBuffer,28+42,126+12,1,CYAN);
Tft.drawString("/",28+42+16,126+12,1,CYAN);

dtostrf(GPS.day,2,0,printBuffer);
Tft.drawString(printBuffer,28+42+24,126+12,1,CYAN);

dtostrf(GPS.minute,2,0,printBuffer);
if(GPS.minute < 10)
{
Tft.drawString("0",28+42+16+8,126,1,CYAN);
}
Tft.drawString(printBuffer,28+42+16+8,126,1,CYAN);

Tft.drawString("Speed: ",14,126+12+12,2,BLUE);
Tft.drawString("MPH",28+172,126+12+14,1,GREEN);
dtostrf(GPS.speed*1.15078,4,2,printBuffer); //converts knots to mph by
knots*1.15078
Tft.drawString(printBuffer,28+76+6,126+12+12,2,GREEN);
Tft.drawString("(accurate to ~.3 mph)",32,126+12+12+20,1,BLUE);

Tft.drawString("Alt. :",14,126+62,2,BLUE);
Tft.drawString("feet",28+172,126+64,1,GREEN);
dtostrf(GPS.altitude*3.28084,5,0,printBuffer); //converts m to feet
Tft.drawString(printBuffer,28+76+6,126+62,2,GREEN);

dtostrf(GPS.latitude/100,5,2,printBuffer);
Tft.drawString(printBuffer,14,126+62+28,2,RED);
Tft.drawString("degrees N",14+114,126+62+28+2,1,RED);

dtostrf(GPS.longitude/100,5,2,printBuffer);
Tft.drawString(printBuffer,14,126+62+28+24,2,RED);
Tft.drawString("degrees W",14+114,126+62+28+24,1,RED);

dtostrf(GPS.altitude*3.28084,5,0,printBuffer);
Tft.drawString(printBuffer,28+76+6,126+62,2,GREEN);

Tft.drawString("Tracking: ",14,126+62+30+24+24,1,WHITE);
dtostrf(GPS.satellites,2,0,printBuffer);
Tft.drawString(printBuffer,14+58+16,126+62+30+24+20,2,YELLOW);

```

```

if(GPS.satellites==1){
Tft.drawString("Satellite",14+54+20+32,126+62+30+24+24,1,WHITE);}
else {Tft.drawString("Satellites",14+54+42+32,126+62+30+24+24,1,WHITE);}

Tft.drawString(" All of this information",8,126+62+30+24+24+18,1,CYAN);
Tft.drawString(" comes from SPACE!",8,126+62+30+24+24+18+12,1,CYAN);

delay(8000);

}
else
{
Tft.drawString("Unable to link",14,126+12+12,2,BLUE);
Tft.drawString("with satellite",14,126+62,2,BLUE);
delay(1500);
Tft.fillRect(14,126+12+12,240,200,BLACK);
Tft.drawString("GPS values could",14,126+12+12,1,BLUE);
Tft.drawString("not be established",14,126+62,1,BLUE);
delay(1500);
}

}

void Display_BMP180(){
  bmp180start = millis();
  Tft.fillRect(0,0,239,319,BLACK);

  Tft.fillRect(64,20,239,86,BLUE);
  Tft.drawString("Temperature",66,4,2,BLUE);
  Tft.drawString(" deg F",66+130,21,1,BLACK);

  Tft.fillRect(64,126,176,86,CYAN);
  Tft.drawString("Pressure",66,110,2,CYAN);
  Tft.drawString(" KPa",66+130,116,1,CYAN);

  Tft.fillRect(64,232,239,88,GREEN);
  Tft.drawString("Altitude",66,216,2,GREEN);
  Tft.drawString(" feet",66+130,222,1,GREEN);

  //X-axis for Temp
  Tft.drawLine(68,96,237,96,BLACK);
  Tft.drawLine(236,95,236,97,BLACK);
  Tft.drawLine(235,94,235,98,BLACK);
  Tft.drawString("0",70,98,1,WHITE);
  Tft.drawString("10",91,98,1,WHITE);

```

```

Tft.drawString("20",116,98,1,WHITE);
Tft.drawString("30",141,98,1,WHITE);
Tft.drawString("40",166,98,1,WHITE);
Tft.drawString("50",191,98,1,WHITE);
Tft.drawString("60",216,98,1,WHITE);

float pixelx = 0, pixely = 0;
for(int i=0; i<60; i++)
{

    pixelx = 74 + 2.5*i;
    pixely = map(tempArray[i], maxTemp, minTemp, 26, 96); //though it appears
backwards, this is how it needs to be
    if(tempArray[i]!=-999)
        Tft.fillCircle(pixelx,pixely,3,RED);

}

//Y-axis for Temp
Tft.drawLine(68,23,68,96,BLACK);
Tft.drawLine(67,24,69,24,BLACK);
Tft.drawLine(66,25,70,25,BLACK);

dtostrf(minTemp,6,2,printBuffer);
Tft.drawString(printBuffer,16,86,1,WHITE);

dtostrf(minTemp+(maxTemp-minTemp)/6,6,2,printBuffer);
Tft.drawString(printBuffer,16,76,1,WHITE);

dtostrf(minTemp+2*(maxTemp-minTemp)/6,6,2,printBuffer);
Tft.drawString(printBuffer,16,66,1,WHITE);

dtostrf(minTemp+3*(maxTemp-minTemp)/6,6,2,printBuffer);
Tft.drawString(printBuffer,16,56,1,WHITE);

dtostrf(minTemp+4*(maxTemp-minTemp)/6,6,2,printBuffer);
Tft.drawString(printBuffer,16,46,1,WHITE);

dtostrf(minTemp+5*(maxTemp-minTemp)/6,6,2,printBuffer);
Tft.drawString(printBuffer,16,36,1,WHITE);

dtostrf(maxTemp,6,2,printBuffer);
Tft.drawString(printBuffer,16,26,1,WHITE);

```

```

End_Bars_loop();

//X-axis for Pressure
Tft.drawLine(68,96+106,237,96+106,BLACK);
Tft.drawLine(236,95+106,236,97+106,BLACK);
Tft.drawLine(235,94+106,235,98+106,BLACK);
Tft.drawString("0",70,98+106,1,BLACK);
Tft.drawString("10",91,98+106,1,BLACK);
Tft.drawString("20",116,98+106,1,BLACK);
Tft.drawString("30",141,98+106,1,BLACK);
Tft.drawString("40",166,98+106,1,BLACK);
Tft.drawString("50",191,98+106,1,BLACK);
Tft.drawString("60",216,98+106,1,BLACK);

for(int i=0; i<60; i++)
{
  pixelx = 74 + 2.5*i;
  pixely = map(pressureArray[i], minPressure, maxPressure, 96+106, 26+106);
  if(pressureArray[i]!=-999)
    Tft.fillCircle(pixelx,pixely,3,RED);
}

//Y-axis for Pressure
Tft.drawLine(68,23+106,68,96+106,BLACK);
Tft.drawLine(67,24+106,69,24+106,BLACK);
Tft.drawLine(66,25+106,70,25+106,BLACK);

dtostrf(minPressure,7,3,printBuffer);
Tft.drawString(printBuffer,8,86+106,1,WHITE);

dtostrf(minPressure+(maxPressure-minPressure)/6,7,3,printBuffer);
Tft.drawString(printBuffer,8,76+106,1,WHITE);

dtostrf(minPressure+2*(maxPressure-minPressure)/6,7,3,printBuffer);
Tft.drawString(printBuffer,8,66+106,1,WHITE);

dtostrf(minPressure+3*(maxPressure-minPressure)/6,7,3,printBuffer);
Tft.drawString(printBuffer,8,56+106,1,WHITE);

dtostrf(minPressure+4*(maxPressure-minPressure)/6,7,3,printBuffer);
Tft.drawString(printBuffer,8,46+106,1,WHITE);

dtostrf(minPressure+5*(maxPressure-minPressure)/6,7,3,printBuffer);
Tft.drawString(printBuffer,8,36+106,1,WHITE);

```



```

dtostrf(maxPressure,7,2,printBuffer);
Tft.drawString(printBuffer,8,26+106,1,WHITE);

End_Bars_loop();

//X-axis for Altitude
Tft.drawLine(68,96+212,237,96+212,BLACK);
Tft.drawLine(236,95+212,236,97+212,BLACK);
Tft.drawLine(235,94+212,235,98+212,BLACK);
Tft.drawString("0",70,98+212,1,WHITE);
Tft.drawString("10",91,98+212,1,WHITE);
Tft.drawString("20",116,98+212,1,WHITE);
Tft.drawString("30",141,98+212,1,WHITE);
Tft.drawString("40",166,98+212,1,WHITE);
Tft.drawString("50",191,98+212,1,WHITE);
Tft.drawString("60",216,98+212,1,WHITE);

for(int i=0; i<60; i++)
{
  pixelx = 74 + 2.5*i;
  pixely = map(altitudeArray[i], minAlt, maxAlt, 96+106+106, 26+106+106);
  if(altitudeArray[i]!=-999)
    Tft.fillCircle(pixelx,pixely,3,RED);
}

//Y-axis for Altitude
Tft.drawLine(68,23+212,68,96+212,BLACK);
Tft.drawLine(67,24+212,69,24+212,BLACK);
Tft.drawLine(66,25+212,70,25+212,BLACK);

dtostrf(minAlt*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,86+106+106,1,WHITE);

dtostrf((minAlt+(maxAlt-minAlt)/6)*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,76+106+106,1,WHITE);

dtostrf((minAlt+2*(maxAlt-minAlt)/6)*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,66+106+106,1,WHITE);

dtostrf((minAlt+3*(maxAlt-minAlt)/6)*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,56+106+106,1,WHITE);

dtostrf((minAlt+4*(maxAlt-minAlt)/6)*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,46+106+106,1,WHITE);

dtostrf((minAlt+5*(maxAlt-minAlt)/6)*3.28084,6,2,printBuffer);

```

```

Tft.drawString(printBuffer,8,36+106+106,1,WHITE);

dtostrf(maxAlt*3.28084,6,2,printBuffer);
Tft.drawString(printBuffer,8,26+106+106,1,WHITE);
while(millis()-bmp180start < 10000) {End_Bars_loop();}
}

void initTouchScreenParameters()
{
  //This function initializes Touch Screen parameters based on the detected TFT Touch
  Schield hardware

  if(Tft.IC_CODE == 0x5408) //SPFD5408A TFT driver based Touchscreen hardware
  detected
  {
    #if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
      ts = TouchScreen(54, A1, A2, 57, 300); //init TouchScreen port pins
    #else
      ts = TouchScreen(14, A1, A2, 17, 300); //init TouchScreen port pins
    #endif
    //Touchscreen parameters for this hardware
    TS_MINX = 120;
    TS_MAXX = 910;
    TS_MINY = 120;
    TS_MAXY = 950;

    MapX1 = 239;
    MapX2 = 0;
    MapY1 = 0;
    MapY2 = 319;
  }
  else //ST7781R TFT driver based Touchscreen hardware detected
  {
    #if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
      ts = TouchScreen(57, A2, A1, 54, 300); //init TouchScreen port pins
    #else
      ts = TouchScreen(17, A2, A1, 14, 300); //init TouchScreen port pins
    #endif

    //Touchscreen parameters for this hardware
    TS_MINX = 140;
    TS_MAXX = 900;
    TS_MINY = 120;
    TS_MAXY = 940;
  }
}

```

```
MapX1 = 239;  
MapX2 = 0;  
MapY1 = 319;  
MapY2 = 0;  
}  
}
```

Script associated with NeoPixel LED Strip and force-sensitive-resistor

```
void End_Bars_setup(){
  endBars.begin();
  endBars.setBrightness(brightness);
  //Calibrate the analog read sensors, set reading to be flat/zero
  for(i=0;i<8;i++)
  {
    //REMEMBER! tail = 0 , nose = 1
    tailMidVal += analogRead(tailBendBar);
    noseMidVal += analogRead(noseBendBar);
    delay(50);
    endBars.setPixelColor(i, Color(31*i,0,192));
    endBars.show();
    if(i==7) { tailMidVal /= 8; noseMidVal /= 8; }
  }
  delay(200);
  for(i=0; i<8; i++)
    endBars.setPixelColor(i, Color(0,0,0));
  endBars.show();
}

void End_Bars_loop(){ //Function takes ~15ms

  if(bendCheck() != -1) { lightBarsFromBend(); lastSet = millis(); }
  if(millis()-lastSet >= 3000){
    //if bars have been illuminated for 3 seconds with no change, turn lights off
    for(i=0;i<8;i++)
      endBars.setPixelColor(i, Color(0,0,0));
    endBars.show();
  }
}

int bendCheck(){ //Function Takes <20ms
  //mid value typically 410, max value 250. Bend is broken up into 10 different levels
  //lighting the neopixels with the middle 8 leves, first two sections as buffers
  //buffer value regions are used so pixels stay off while flat and also to max out a
  little more generously.

  if(analogRead(tailBendBar) <= analogRead(noseBendBar))
  {
    if(tailMidVal - analogRead(tailBendBar) > 2*((tailMidVal-300)/10) && tailMidVal
    - analogRead(tailBendBar) < 3*((tailMidVal-300)/10)) return 0;
    else if(tailMidVal - analogRead(tailBendBar) > 3*((tailMidVal-300)/10) &&
    tailMidVal - analogRead(tailBendBar) < 4*((tailMidVal-300)/10)){ return 1;}
  }
```

```

    else if(tailMidVal - analogRead(tailBendBar) > 4*((tailMidVal-300)/10) &&
tailMidVal - analogRead(tailBendBar) < 5*((tailMidVal-300)/10)){ return 2;}
    else if(tailMidVal - analogRead(tailBendBar) > 5*((tailMidVal-300)/10) &&
tailMidVal - analogRead(tailBendBar) < 6*((tailMidVal-300)/10)) return 3;
    else if(tailMidVal - analogRead(tailBendBar) > 6*((tailMidVal-300)/10) &&
tailMidVal - analogRead(tailBendBar) < 7*((tailMidVal-300)/10)) return 4;
    else if(tailMidVal - analogRead(tailBendBar) > 7*((tailMidVal-300)/10) &&
tailMidVal - analogRead(tailBendBar) < 8*((tailMidVal-300)/10)) return 5;
    else if(tailMidVal - analogRead(tailBendBar) > 8*((tailMidVal-300)/10) &&
tailMidVal - analogRead(tailBendBar) < 9*((tailMidVal-300)/10)) return 6;
    else if(tailMidVal - analogRead(tailBendBar) > 9*((tailMidVal-300)/10)) return 7;
    return -1;
}

else
{
    if(noseMidVal - analogRead(noseBendBar) > 2*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 3*((noseMidVal-300)/10)) return 0;
    if(noseMidVal - analogRead(noseBendBar) > 3*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 4*((noseMidVal-300)/10)){ return 1;}
    if(noseMidVal - analogRead(noseBendBar) > 4*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 5*((noseMidVal-300)/10)){ return 2;}
    if(noseMidVal - analogRead(noseBendBar) > 5*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 6*((noseMidVal-300)/10)){ return 3;}
    if(noseMidVal - analogRead(noseBendBar) > 6*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 7*((noseMidVal-300)/10)){ return 4;}
    if(noseMidVal - analogRead(noseBendBar) > 7*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 8*((noseMidVal-300)/10)){ return 5;}
    if(noseMidVal - analogRead(noseBendBar) > 8*((noseMidVal-300)/10) &&
noseMidVal - analogRead(tailBendBar) < 9*((noseMidVal-300)/10)){ return 6;}
    if(noseMidVal - analogRead(noseBendBar) > 9*((noseMidVal-300)/10)) return 7;
    return -1;
}
//Function returns which pixel to light
}

```

```

void lightBarsFromBend()
{ // | Function takes ~15ms not counting the extra ~20ms from bendCheck() |
  pixelVal = bendCheck();
  if(pixelVal >= 0)endBars.setPixelColor(0, Color(0,255,0));
  if(pixelVal >= 1)endBars.setPixelColor(1, Color(0,255,0));
  if(pixelVal >= 2)endBars.setPixelColor(2, Color(20,255,0));
  if(pixelVal >= 3)endBars.setPixelColor(3, Color(128,255,0));
  if(pixelVal >= 4)endBars.setPixelColor(4, Color(255,255,0));
  if(pixelVal >= 5)endBars.setPixelColor(5, Color(255,128,0));
  if(pixelVal >= 6)endBars.setPixelColor(6, Color(255,25,0));
}

```

```
if(pixelVal >= 7) { endBars.setPixelColor(7, Color(255,0,0)); fullBends++; }  
endBars.show();  
}
```

```
uint32_t Color(byte r, byte g, byte b)  
{  
  //we dont need to cite open-source code from online do we?  
  uint32_t c;  
  c = r;  
  c <<= 8;  
  c |= g;  
  c <<= 8;  
  c |= b;  
  return c;  
}
```

GPS Peripheral Scripts

```
void GPS_setup()
{

    // connect at 115200 so we can read the GPS fast enough and echo without dropping
    chars
    // also spit it out
    if(USE_SERIAL){
    Serial.begin(115200);
    Serial.println("Adafruit GPS library basic test!");
    }
    // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
    GPS.begin(9600);

    // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
    data) including altitude
    GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
    // uncomment this line to turn on only the "minimum recommended" data
    //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
    // For parsing data, we don't suggest using anything but either RMC only or
    RMC+GGA since
    // the parser doesn't care about other sentences at this time

    // Set the update rate
    GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 1 Hz update rate
    // For the parsing code to work nicely and have time to sort thru the data, and
    // print it out we don't suggest using anything higher than 1 Hz

    // Request updates on antenna status, comment out to keep quiet
    GPS.sendCommand(PGCMD_ANTENNA);

    // the nice thing about this code is you can have a timer0 interrupt go off
    // every 1 millisecond, and read data from the GPS for you. that makes the
    // loop code a heck of a lot easier!
    useInterrupt(true);
}

// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
#ifdef UDR0
    if (GPSECHO)
        if (c) UDR0 = c;
    // writing direct to UDR0 is much much faster than Serial.print
```

```

    // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}

void GPS_loop()          // run over and over again
{
  // in case you are not using the interrupt above, you'll
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) if(USE_SERIAL) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trying to print out data
    //Serial.println(GPS.lastNMEA()); // this also sets the newNMEAreceived() flag
    to false

    if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag
    to false
      return; // we can fail to parse a sentence in which case we should just wait for
      another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis()) timer = millis();
}

```



```

// approximately every 2 seconds or so, print out the current stats
if (millis() - timer > 2000) {
  timer = millis(); // reset the timer
  if(USE_SERIAL)
  {
    Serial.print("\nTime: ");
    Serial.print(GPS.hour, DEC); Serial.print(':');
    Serial.print(GPS.minute, DEC); Serial.print(':');
    Serial.print(GPS.seconds, DEC); Serial.print('.');
    Serial.println(GPS.milliseconds);
    Serial.print("Date: ");
    Serial.print(GPS.day, DEC); Serial.print('/');
    Serial.print(GPS.month, DEC); Serial.print("/20");
    Serial.println(GPS.year, DEC);
    Serial.print("Fix: "); Serial.print((int)GPS.fix);
    Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
    if (GPS.fix) {
      Serial.print("Location: ");
      Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
      Serial.print(", ");
      Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);

      Serial.print("Speed (knots): "); Serial.println(GPS.speed);
      Serial.print("Angle: "); Serial.println(GPS.angle);
      Serial.print("Altitude: "); Serial.println(GPS.altitude);
      Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
    }
  }
}
}
}

```

BMP180 Pressure and Temperature Data-Gathering Scripts

```
void Pressure_setup()
{
  int i;
  for(i=0;i<60;i++)
  {pressureArray[i]=-999;}
  for(i=0;i<60;i++)
  {tempArray[i]=-999;}
  for(i=0;i<60;i++)
  {altitudeArray[i]=-999;}

  float temperature;
  sensors_event_t event;

  Serial.println("Pressure Sensor Test"); Serial.println("");
  /* Initialise the sensor */
  if(!bmp.begin())
  {
    /* There was a problem detecting the BMP085 ... check your connections */
    Serial.print("Ooops, no BMP085 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }
}

void Pressure_loop() //Function takes < 100ms
{
  End_Bars_loop();
  float temperature;
  sensors_event_t event;

  bmp.getEvent(&event);
  if (event.pressure)
  {
    pressureArray[pollNumber%60] = event.pressure;
    bmp.getTemperature(&temperature);
    temperature = ((temperature*9)/5)+32;
    tempArray[pollNumber%60] = temperature;

    float seaLevelPressure = SEA_LEVEL_PRESSURE;
    if(GPS.fix && GPS.altitude > -100 && GPS.altitude < 500000 ) {
altitudeArray[pollNumber%60] = GPS.altitude; }
    else { altitudeArray[pollNumber%60] =
bmp.pressureToAltitude(seaLevelPressure, event.pressure, temperature)*3.28084; }
```

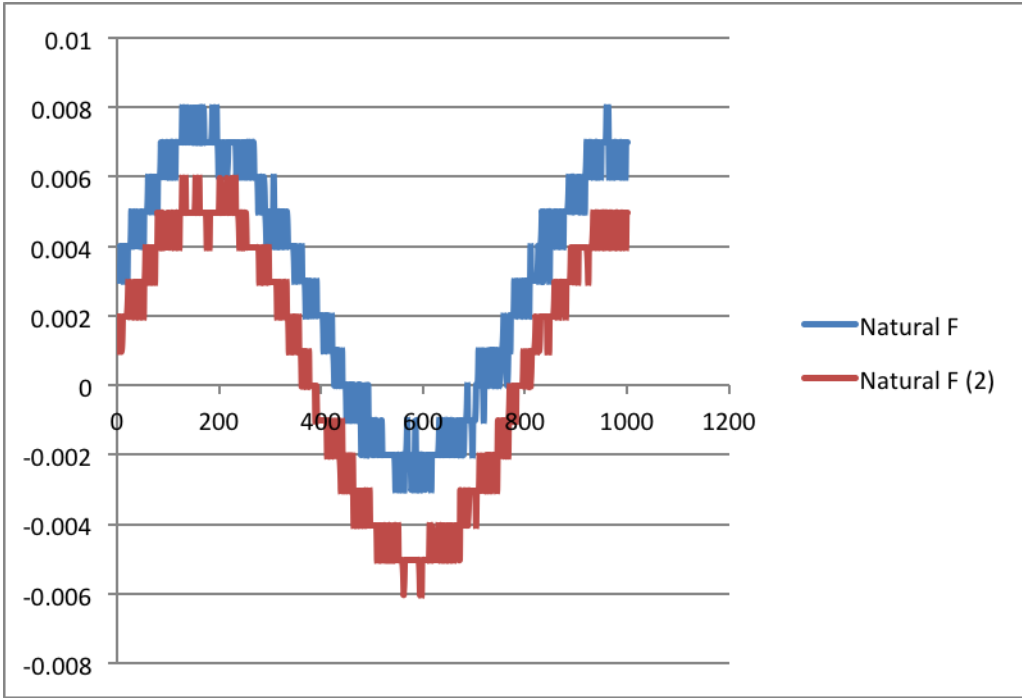
```

}
else
{
  Serial.println("Sensor error");
}
End_Bars_loop();
//Updates max and min values which are to be tracked through the whole day ::
if(pressureArray[pollNumber%60] > maxPressure) { maxPressure =
pressureArray[pollNumber%60]; }
if(pressureArray[pollNumber%60] < minPressure) { minPressure =
pressureArray[pollNumber%60]; }
if(tempArray[pollNumber%60] > maxTemp)      { maxTemp =
tempArray[pollNumber%60]; }
if(tempArray[pollNumber%60] < minTemp)      { minTemp =
tempArray[pollNumber%60]; }
if(altitudeArray[pollNumber%60] > maxAlt)   { maxAlt =
altitudeArray[pollNumber%60]; }
if(altitudeArray[pollNumber%60] < minAlt)   { minAlt =
altitudeArray[pollNumber%60]; }
pollNumber++;

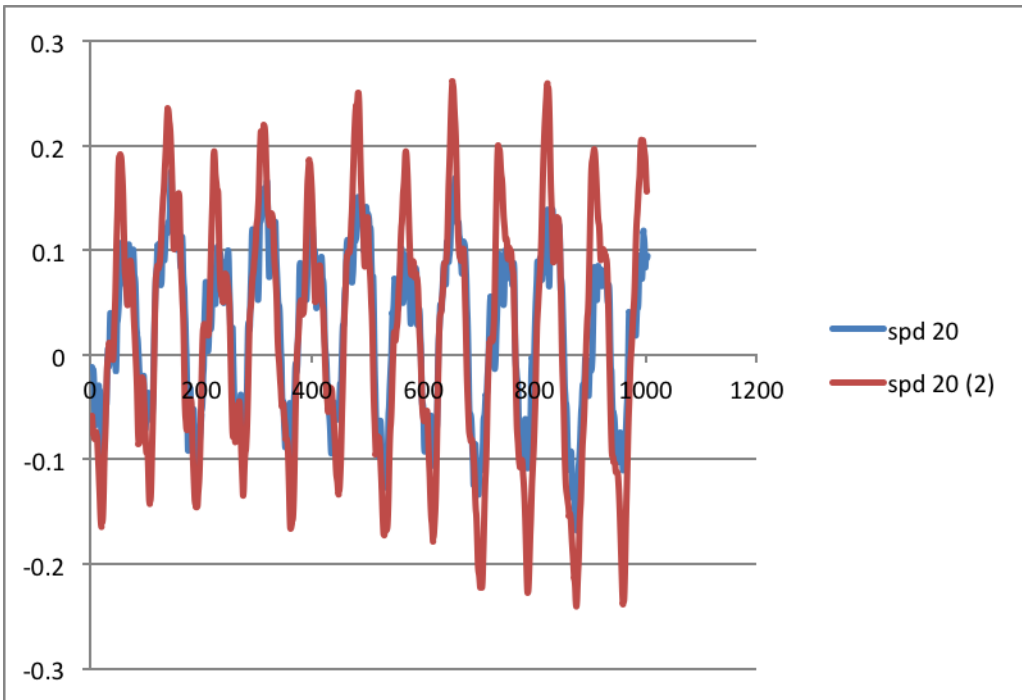
}

```

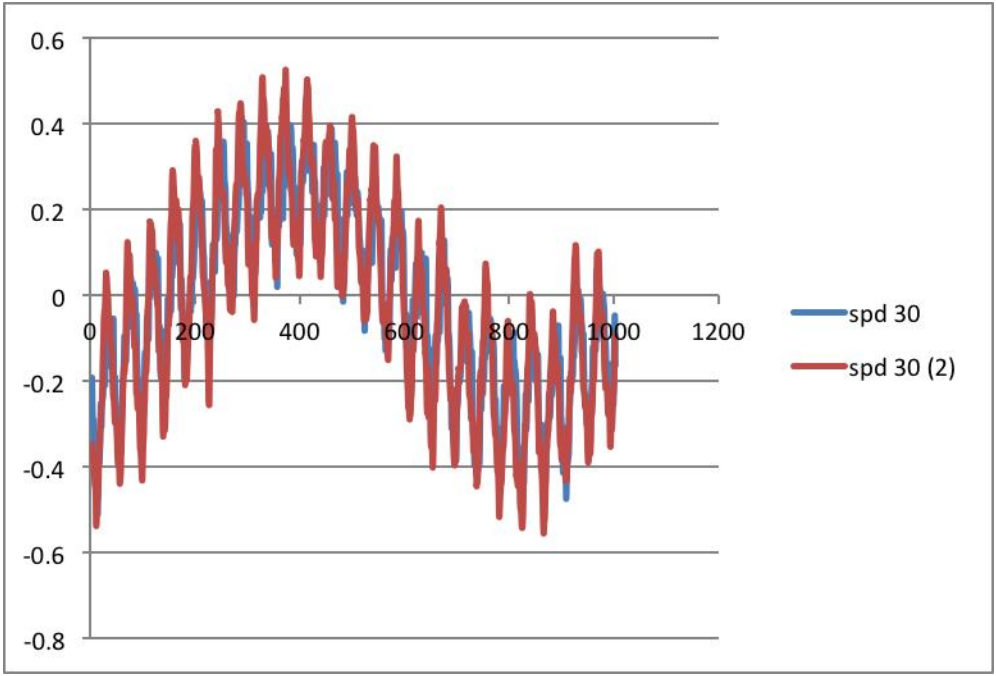
Appendix 5: Vibration Table Data (RED is sensor on board, BLUE is sensor on casing)



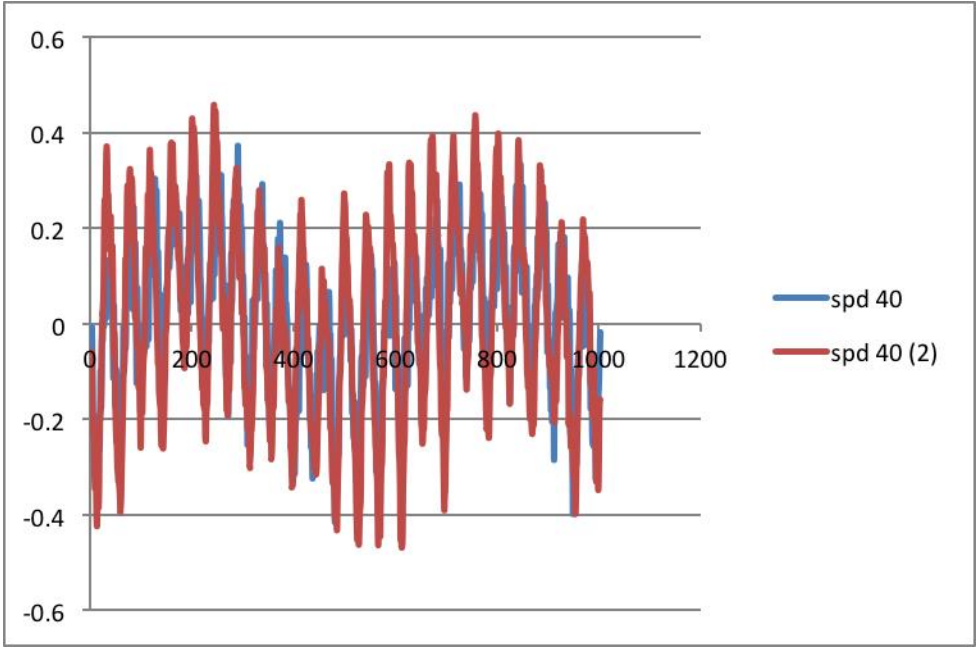
Natural frequency, pulled board down and let go



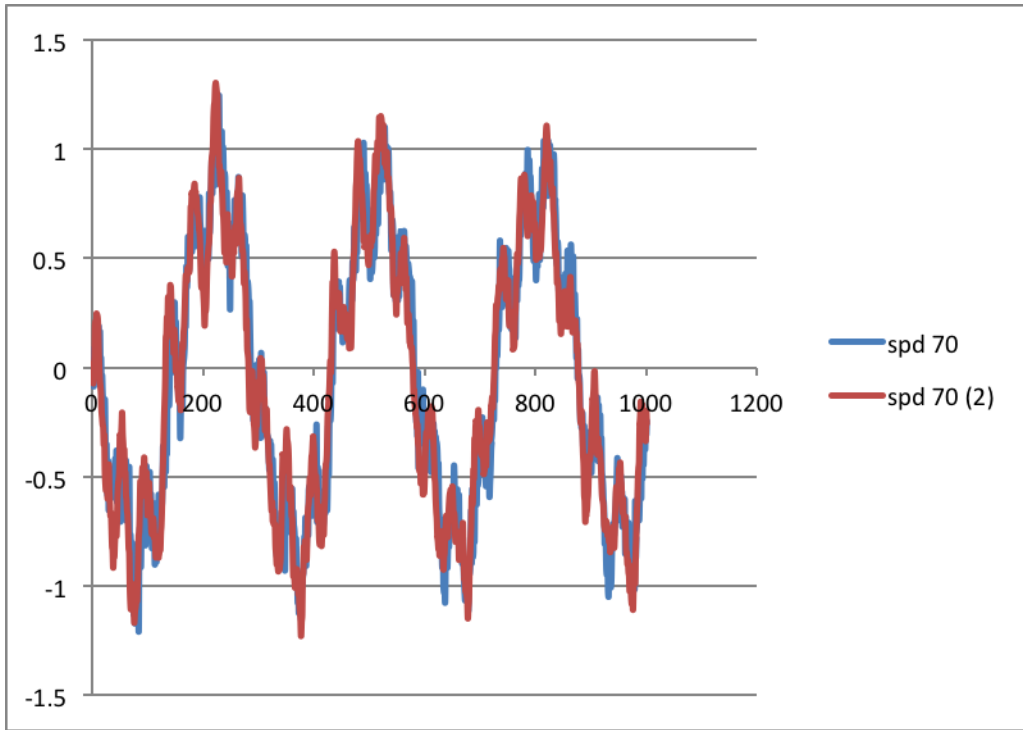
Speed on table is 20



Speed on table is 30



Speed on Table is 40



Speed on the table is 70

Appendix 6: Sensor Matlab Code

Skate Jump

```
filename = 'SkateJump1.xlsx';
xSum = 0;
ySum = 0;
zSum = 0;
zScan = 0; %second sum to be calculated to get bounds
tData = xlsread(filename,'A:A');
xData = xlsread(filename,'B:B');
yData = xlsread(filename,'C:C');
zData = xlsread(filename,'D:D');
zMax = 0;
tStart = -1;
tStop = -1;
for i = 1:size(xData)-1
    if(abs(zData(i))>zMax)
        zSum = zSum + abs( zData(i) );
    end %this loop establishes the total z value
end
for i = 1:size(xData)-1
    zScan = zScan + abs( zData(i) );
    xSum = xSum + xData(i);
    ySum = ySum + yData(i);
    if(abs(zScan)>abs(.1*zSum) && tStart == -1)
        tStart = tData(i);
    end
    if(abs(zScan)>abs(.9*zSum) && tStop == -1)
        tStop = tData(i);
    end
end %this loop sums the x,y axis data and looks at the z data to determine
approximate bounds
tJump = 1.25*(tStop-tStart);
```

Skate Spin

```
filename = 'SkateSpin1.xlsx';
xSum = 0;
ySum = 0;
zSum = 0;
zScan = 0; %second sum to be calculated to get bounds
tData = xlsread(filename,'A:A');
```

```

xData = xlsread(filename,'B:B');
yData = xlsread(filename,'C:C');
zData = xlsread(filename,'D:D');
zMax = 0;
tStart = -1;
tStop = -1;
for i = 1:size(xData)-1
    if(abs(zData(i))>zMax)
        zSum = zSum + abs( zData(i) );
    end %this loop establishes the total z value
end
for i = 1:size(xData)-1
    zScan = zScan + abs( zData(i) );
    xSum = xSum + xData(i);
    ySum = ySum + yData(i);
    if(abs(zScan)>abs(.1*zSum) && tStart == -1)
        tStart = tData(i);
    end
    if(abs(zScan)>abs(.9*zSum) && tStop == -1)
        tStop = tData(i);
    end
end %this loop sums the x,y axis data and looks at the z data to determine
approximate bounds
tJump = 1.25*(tStop-tStart);

```

Snow Jump

```

filename = 'SnowJump1.xlsx';
xSum = 0;
ySum = 0;
zSum = 0;
zScan = 0; %second sum to be calculated to get bounds
tData = xlsread(filename,'A:A');
xData = xlsread(filename,'B:B');
yData = xlsread(filename,'C:C');
zData = xlsread(filename,'D:D');
zMax = 0;
tStart = -1;
tStop = -1;
for i = 1:size(xData)-1
    if(abs(zData(i))>zMax)

```



```

        zSum = zSum + abs( zData(i) );
    end %this loop establishes the total z value
end
for i = 1:size(xData)-1
    zScan = zScan + abs( zData(i) );
    xSum = xSum + xData(i);
    ySum = ySum + yData(i);
    if(abs(zScan)>abs(.1*zSum) && tStart == -1)
        tStart = tData(i);
    end
    if(abs(zScan)>abs(.9*zSum) && tStop == -1)
        tStop = tData(i);
    end
end %this loop sums the x,y axis data and looks at the z data to determine
approximate bounds
tJump = 1.25*(tStop-tStart);

```

Snow Spin

```

filename = 'SnowSpin1.xlsx';
xSum = 0;
ySum = 0;
zSum = 0;
zScan = 0; %second sum to be calculated to get bounds
tData = xlsread(filename,'A:A');
xData = xlsread(filename,'B:B');
yData = xlsread(filename,'C:C');
zData = xlsread(filename,'D:D');
zMax = 0;
tStart = -1;
tStop = -1;
for i = 1:size(xData)-1
    if(abs(zData(i))>zMax)
        zSum = zSum + abs( zData(i) );
    end %this loop establishes the total z value
end
for i = 1:size(xData)-1
    zScan = zScan + abs( zData(i) );
    xSum = xSum + xData(i);
    ySum = ySum + yData(i);
    if(abs(zScan)>abs(.1*zSum) && tStart == -1)

```

```
    tStart = tData(i);  
end  
if(abs(zScan)>abs(.9*zSum) && tStop == -1)  
    tStop = tData(i);  
end  
end %this loop sums the x,y axis data and looks at the z data to determine  
approximate bounds  
tJump = 1.25*(tStop-tStart);
```

Appendix 7
Consumer Needs Data

Sport	Ski	Snowboard	Skateboard	2 of These	All of These	
Quantity	11	13	7	10	3	
Hours Spent per Week in season	Quantity					
1 -> 10	12					
11 -> 20	10					
21 -> 30	5					
31+	13					
Speed Data		Jump Hang Time		Board/Ski Orientation (in air and on ground)		
Yes	37	Yes	34	Yes	31	
No	7	No	9	No	13	
Willing Sensor Placement (all that apply)		Age	Quantity			
Board/Ski	31	19	7			
Boots	29	20	10			
Torso	11	21	21			
None	4	22	4			
		other	2			
Rank of Importance	1	2	3	4	5	6
Cost	25	7	3	2	1	2
Size	3	18	4	8	6	2
Durability	7	6	17	5	5	1
Aesthetics	2	5	9	9	8	7
Simplicity	3	3	4	13	13	4

Appendix 8: Dimensioned Final Casing

All dimensions are in inches

