University of Colorado Boulder

For the College of Engineering and Applied Science

Cloud Cover

GEEN 1400-080 Final Project

Instructor: Daniel Godrick

10 December 2018

Cloud^5:

Josie Johnson, Wade Pearce, Cole Pragides, Reese Turney, Brian Wittmer

I. ABSTRACT

The Cloud Cover is a modular, attachable product that fits on most bike helmets, and will alert a pre-designated emergency contact through text message that the user has been involved in a crash wherein the head was impacted. The texts state where the impact occurred on the user's head, as well as the GPS coordinates of the user. These texts are efficient and informative to whoever is receiving the messages so that the user can get immediate aid for their injuries. We, Cloud^5, recognize that biking alone brings with it a number of risks; injuries are very common, and head injuries can be very severe, where seconds can make the difference between saving and losing a life. We believe that the Cloud Cover is vital to user safety, and can possibly save their life.

The Cloud Cover mainly consists of a circuit including an Arduino Pro Mini, an accelerometer, force sensors, a FONA 808 GPS/GSM shield, a button, and a power switch. The majority of this system is housed in a 3D printed box that attaches to the back of the helmet via zip ties. The system is powered by a 3.7 volt rechargeable LiPoly battery which runs for a significant period of time. The text gets sent to the user's emergency contact after around 12 "g"s, or acceleration divided by 9.8 m/s^2 , has been crossed and one of the force sensors placed throughout the helmet registers a collision impact.

Throughout our many iterations of code, circuitry, and designs, we were able to create a product that satisfies all of our initial design requirements. These were, comfort, consistency, and ease of use, which all fall under the umbrella of user safety. The area we can most improve upon would be the large size of the box and circuitry; if given more time, we could miniaturize the entire system even further to make the Cloud Cover even more comfortable for the user, and even possibly integrating the system into the helmet itself instead of having it attach to the outside.

II. INTRODUCTION

Since Boulder is such a naturally beautiful area, almost everyone enjoys the outdoors. On campus, the number of bicyclists is high; the high traffic on every path can make biking an obstacle course. However, if you do happen to crash on campus, help is usually only a couple of

seconds away. Unfortunately, biking alone in the mountains is a different story: there's not always someone to report the crash or lend a helping hand. With head injuries in particular, time is of the essence. Seconds wasted searching for a missing person can make a difference in whether someone lives or dies.

With Cloud Cover, one is never truly alone. Our attachable module provides safety for bikers by sending a text message to a pre-specified emergency contact after a crash has been detected. The text not only says that someone needs help, but also sends the user's exact GPS coordinates. In addition, force sensors placed throughout the inside of the helmet locate the point of greatest force on the head of the user and include it in the message, which eliminates more time spent determining what needs to be treated during crunch time.

Because the Cloud Cover must be worn for hours on end by users, comfort was one of the most important factors we considered when designing a helmet addition. We also strove to make our product easily understandable so that anyone could use it. Finally, in order for Cloud Cover to be trusted and respected by medical professionals, we made sure to keep precision and consistency in mind.

As a group, we recognize how big of a safety issue unseen or unreported head injuries can be. The initial intended users of this product were mountain bikers who are concerned for their safety and would like additional precautionary measures. However, Cloud Cover also quells the insecurities people feel about their loved ones going off on trips alone. We intend to make the product even more modular and be able to attach to many helmets; this way, the Cloud Cover could eventually be used for any type of helmet like skiing, especially backcountry or even horseback riding. From rock slides to avalanches, many things can go wrong when you're adventuring alone; we hope that Cloud Cover will give make our users feel protected and provide them with added peace of mind.

III. DESIGN REQUIREMENTS

When first discussing our project, we mainly focused on the safety of the user; all of our design requirements revolve around this simple principle. To maximize the safety of the general population, we wanted to make the Cloud Cover as attractive to consumers as possible. To make

our product user-friendly in all aspects of the design, we wanted to ensure that people who tried it out would compliment its comfort, consistency, and ease of use.

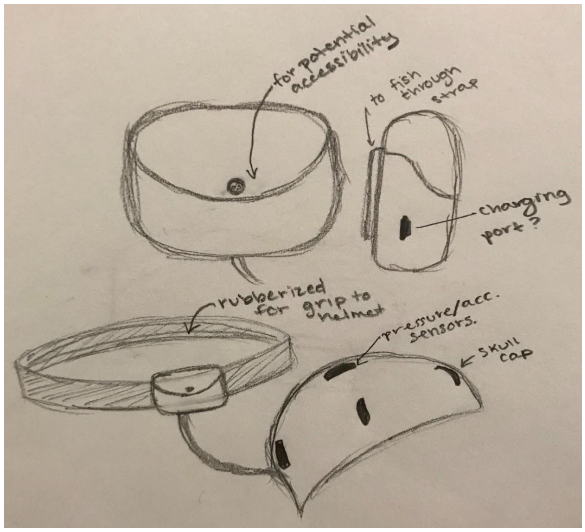


Figure 1: Initial design sketch 1, with the skull cap and strap included.

The comfort requirement stood out the most to our team because if the rider did not feel comfortable wearing it, we would not have any buyers and users would continue to ride alone unsafely. Before talking to potential users, our first main design iteration included a skull cap including the force sensors, which would have added an additional layer of heat and discomfort. Therefore we decided to eliminate the skull cap and add the force sensors to the inside of the helmet so the user would neither feel them nor suffer from heat or irritation. To improve comfort, the weight of our product was closely examined. If the product put too much weight on the user's head, neck strains and pain could occur. So, with the use of Computer Aided Design through OnShape, we designed a small shell to conceal our electronics inside. We decided 3D printing would be optimal for the execution of this design because of how lightweight, yet strong and durable, PLA filament is. How the product feels on the user's head is the biggest selling point when considering what can set our product apart from others.

In order for our product to be trusted, Cloud Cover needed to remain consistent. Therefore our team needed to examine and perfect our entire design inside and out multiple times to ensure users would be safe using our product. During initial planning, we understood the urgency of head-related injuries and wanted the Cloud Cover to perform whenever truly needed. This performance consistency was based on the equations we utilized to convert our readings into interpretable data. We knew that our product needed to work for users of all ages and sizes, meaning the weight of the person could not be a factor in our code (Equation 1). Without this requirement, our product would not have had a large enough market to be successful.

Finally, the ease of use of our design is a major requirement because we wanted anyone to feel confident in understanding our product. To accomplish this, Cloud Cover only

incorporates two versions of user input: a power switch and a “kill” button. When first designing the product, we did not include and buttons or switches because we did not think to add them into our design. During the design process, however, we realized a “kill” button was needed to cancel the alert message if the thresholds were breached but nobody was injured, and a power button to save battery. Ease of use when attaching the Cloud Cover to any existing helmet was also vital; many iterations also went into executing this requirement, but we ended up choosing zip ties, which are readily available to all users and velcro to adhere the force sensors to the inside of the helmet. With a simple attachment process and two self explanatory buttons, all users will find that the Cloud Cover is a product for everyone.

If a user were to inspect our design and find faults in any of the above criteria, our product would be overlooked. Our three main goals of comfort, consistency, and ease of use are hopefully omnipresent in every aspect of Cloud Cover.

IV. DESIGN ALTERNATIVES AND PROCESS

When we first chose the Cloud Cover as our Final Project topic, the first design choice we had to make was where we were going to place the circuits on the helmet. At first we thought of attaching the case (with the electronics and hardware within it) on the top of the helmet, because we thought that there it would have the least chance of falling. We quickly realized that not only would this impede the aerodynamics of the biker, but that head injuries also often occurred with impacts on the top of the head, placing the electronics at risk of breaking. We determined that the area least frequently impacted from head injuries caused by bike-related accidents was the back of the head because most people tend to fall forwards or sideways.

The next decision we were faced with was how we wanted to attach the casing to the helmet. We thought of having an interior skull cap through which we would layer the force sensors so that the rider could put Cloud Cover on under any helmet. In this design, the majority of the hardware would still be held within a box attached by fishing a strap through 3D printed slits.

Our second final design contender had the box and force sensors secured externally to the helmet with a series of straps running around the helmet longitudinally and latitudinally.

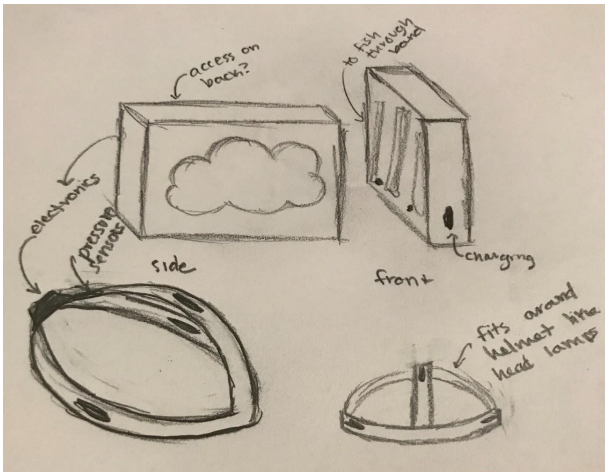
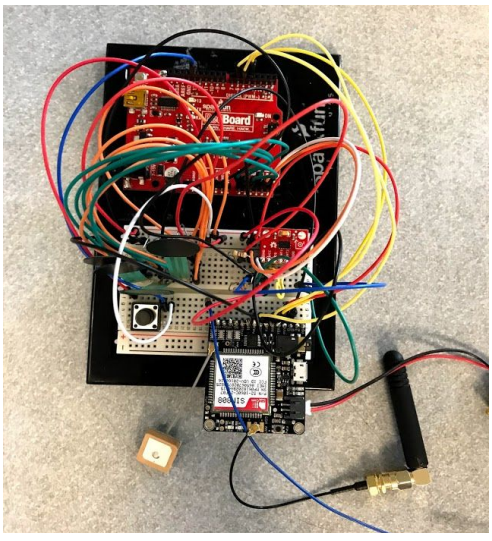


Figure 2: Initial design sketch 2, with head straps containing the force sensors

However, this external system was ruled as too susceptible to being caught on an obstacle like a branch while the user was biking. The straps would also have been much harder to make attachable onto every helmet. So we began to pursue the internally attached skull cap design.

As time went on, we were forced to move away from the skull cap design because it would have made it difficult to troubleshoot, and would have added hours to our assembly. We stayed true to the attachment of the force sensors within the helmet, and secured them with velcro on the existing pads that all helmets come with. We moved towards strapping the case on the back of the helmet using a system of interwoven rings that prevented the case from coming loose. Ultimately, we realized that zip ties were easier to attach and detach the case with, and are easily accessible for users so that they can attach and detach at will.

Figure 3: Circuit boards before miniaturization to the Arduino Pro Mini and breadboard



V. FINAL DESIGN

A. Describe the Project in Detail

Cloud Cover, at its core, is driven by coding and circuitry. The coding for Cloud Cover went through many iterations, but ended working very consistently. It begins in the definition stage, before the void setup{...}. Several

libraries need to be included to run various components. We needed Adafruit_FONA.h for the FONA, Wire.h for the force sensors, and Adafruit_Sensor.h and Adafruit_ADXL345_U.h for the accelerometer. We also defined the emergency number and the Fona's TX, RX, and RST pins in this section. The program then sets pins and global variables for the force sensors based on their inactive resistance. Lastly, we initialize pins and variables for the button and accelerometer to reference later.

During void setup{...}, the program initializes the FONA. Opening the serial monitor shows this step, and the communication back and forth between the FONA and the Arduino can be seen. It is important to note that serial communication baud rate across this entire program (with the exception of the FONAserial) is set to 115200, regardless of the default accelerometer and force sensors' settings as they still work at the increased baud rate. The FONA communicates at the same rate but has an internal baud rate of only 4800. The accelerometer is initialized next, also shown in the serial monitor, and is the last portion of the setup process.

The void loop{...} has two basic sections, the first being the crash detection portion of the loop. The Arduino cycles readings from the both the accelerometer and all four force sensors followed by a 100 millisecond delay, which helps with program stability. The accelerometer returns a simple force rating in "g"s while the force sensors return a value based on how hard they are pressed. When one of the force sensors returns a higher value than the others that is also above 250 pressure "g's,", the code changes a variable to the number of the force sensor, which then resets at the top of the loop assuming the accelerometer did not also exceed the threshold.

Figure 4: The Cloud Cover on a helmet with the locations of the force sensors shown.

If the accelerometer did exceed the threshold with the force sensors, then the communication portion of the loop has been triggered and takes over. The threshold is currently set at 12 "g"s, giving us a slightly-too-sensitive reading when referring to possible head injury. For maximum safety, we set the threshold slightly low so that a minor, but still potentially dangerous, crash wouldn't be overlooked by the device because the user can easily cancel the alert system. The threshold can

easily be adjusted by just changing the number at the beginning of the communication loop. The loop itself is simply an if{...} statement referencing the variables from the accelerometer and the force sensors. It starts by turning on the GPS on the FONA. We kept the GPS off unless triggered to conserve battery. After initializing the GPS, the FONA then uses a pair of while{...} loops and the millis{...} function to create a delay that does not block the Arduino from reading other pins. During this delay, if the cancel button is pressed, it changes a variable from a 0 to a 1, then waits for the rest of the delay.

At the end of the delay, there is a second while{...} loop that engages the actual communication feature if the cancel button has not been pressed. If it has, the program skips to the bottom, resets the cancel variable, and goes back to the top of the loop where it measures acceleration and force, allowing a rider that had a false alarm to keep riding without a care in the world. If the cancel button was not pressed, the GPS triangulates the user's location, and then sends one of four initial text messages to the emergency contact defined before the setup. The text message sent is dependent on what force sensor was triggered, and each one describes a different area of the rider's head. The exact message is as follows: "CloudCover has detected a possible head injury focused on the _location of impact on head_. The user is at the following gps coordinates: '___'." The FONA then sends a text message with the rider's GPS coordinates which have been parsed out of the much larger GPS data array. The program then returns to the top of the loop and measures the accelerometer and force sensors.

The other major portion of Cloud Cover is the physical circuitry within. The Adafruit FONA not only has a built in GPS and Cellular module, but also utilizes a 3.7 volt lithium-polymer battery which can be used to power the entire circuit through the battery output pin. We used a solderable breadboard to create power and ground "rails," similar to what one might use on a traditional breadboard. Force sensors are powered from the power rail, connect to their data pins on the Arduino, and all connect through resistors to the ground rail. The accelerometer is similarly powered and grounded through the rails on the breadboard with jumper cables running to data pins on the Arduino. The Adafruit FONA connects to the power (VIO pin) and ground rails (KEY and GND pins) of the breadboard. The battery pin also connects to the power rail to power the circuit. Since that line is "hot" as long as a battery is

plugged in and charged, there is a simple power switch installed along this line that when turned off, cuts the power, opening the entire circuit. The TX, RX, and RST pins all run to pins 9, 8, and 7, respectively, on the Arduino Pro Mini. The Arduino also connects to the power and ground rails through the RAW and GND pins. There is a face of the Pro Mini that can be plugged into an FTDI Breakout adapter for easy coding, but is not currently attached on our working version to minimize the occupied space.

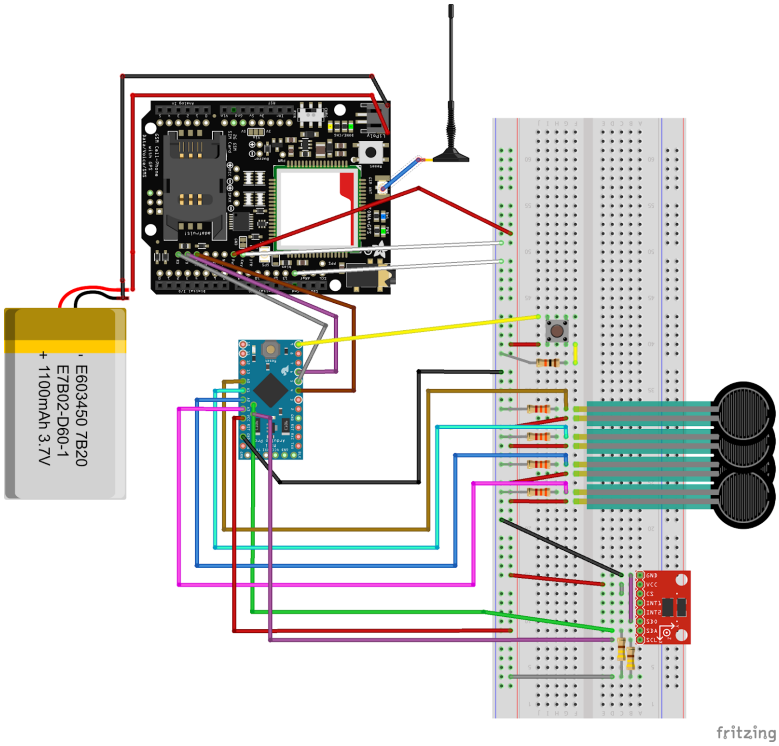


Figure 5: The completed circuit diagram, excluding the power switch added in a later iteration.

B. Testing and Analysis

(1) The most critical portions of the Cloud Cover design are the accelerometer and the FONA 808 GPS/GSM shield. The ADXL 345 accelerometer, a ±16g, triple-axis sensor, triggers the entire sequence through the FONA which then sends a text with the user's coordinates to the selected emergency contact. Without these two components, the fundamentals of the Cloud Cover do not work and the other aspects of the circuitry, such as the force sensors, become useless. Therefore, it was vital to

test both devices extensively throughout the design process to ensure minimum error. If an issue arose during the iterative process, the error could be found through this "process of elimination."

Figures 6(top) and 7(bottom): the accelerometer and FONA shield

(a) To test the accelerometer, we frequently went back to the "sensortest" code we created exclusively for the accelerometer (the code did not contain any other components) throughout each iteration of our circuit and main Breakout code. In doing so we were able to pinpoint whether there was an issue in the newest iteration of the entire project, in the wiring itself or in the updated code (see Appendix).

(b) Testing the FONA involved a similar process; we would upload the basic FONA code "FONAtest" to the entire circuit each time we made an iteration to ensure that the FONA still retained its functionality even after miniaturizing further.

The Cloud Cover did fail at in-class Expo, despite our careful efforts. To exactly identify the main issue with the circuit, we went back to the beginning and combed through the circuit, examining each individual wire, connection, and pin on each component, especially on the accelerometer and the FONA. It took quite a bit of time, but the problem was found to be the failure of the Arduino Pro Mini to communicate with the FONA shield. We found a solution to this underlying issue by replacing the original battery (which had been messily soldered back together after some careless handling), with one that had cleaner connections.

(2) Directly related to the accelerometer are the main equations used throughout the design process and in the actual function of the Cloud Cover:

$$\vec{a} = \frac{d\vec{v}}{dt}, \quad \vec{a}\,(in\ "g"s) = \frac{\sqrt{a_x{}^2 + a_y{}^2 + a_z{}^2}}{9.8} \quad \text{(Equations 1, 2)}$$

Since the ADXL is a triple-axis accelerometer, it measures the helmet's acceleration in the x, y, and z directions. However, as we are interested in the *net* acceleration in "g"s, we use Equation 2. This value is used to determine the threshold of acceleration that will trigger the FONA to begin triangulating the user's location and sending a text to the

predesignated emergency contact. This threshold for a possible head injury is somewhere between 10 and 30 “g”s when biking, which is what was set in the final code (see Appendix) to begin the sequence. However, for testing purposes, we lowered the threshold for acceleration to 1.2 “g”s so that a hard shake would trigger the sequence and we didn’t have to hurt any of the team members during testing. Equation 1 was used instead of Newton’s Second Law ($\vec{F} = m\vec{a}$), or the impulse equation ($J = \Delta p = m\Delta v$), in order to eliminate the need for the mass of the user. This elimination meant that the acceleration threshold could be set to a singular value and not changed based on the user’s weight, eliminating the need for Cloud Cover to calibrate to its users.

C. User Feedback

We gathered user feedback throughout the design process, from initial sketches to the population we reached at the Design Expo. Each of the group members tested Cloud Cover with their friends and family as well as frequent bikers on and around campus. Much of the feedback we received about our beginning designs was more large-scale, overall design related ideas and thoughts, whereas towards the end each user focused on smaller “tweaks” that we could add to our existing product.

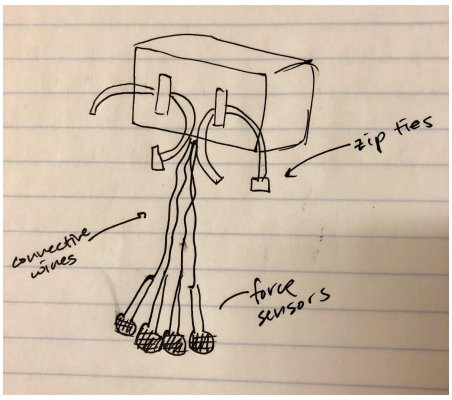


Figure 8: Final design sketch, with attachment method of zip ties

Some of our initial designs for the Cloud Cover consisted of a “skullcap” of sorts that the user would wear underneath the helmet that contained the force sensors needed to detect any impacts. However, after gathering some opinions from those around us, we decided that the skullcap idea would be too restrictive and provide unnecessary heat and discomfort for the user, especially when mountain biking in the heat. This presented a dilemma: the force sensors needed to be placed either inside or outside the helmet itself. We considered placing them outside the helmet, however other users, including Mr. Godrick (a frequent mountain biker himself) were worried that the wiring would get caught on obstacles such as tree branches or the ground itself, yanking the user’s head in a certain direction rapidly and possibly

causing a neck injury as well. Finally we settled upon a solution that would have the force sensors inside the helmet but non-interfering to the user's overall experience: using the existing Velcro cushioning inside each bike helmet. This solution also made the Cloud Cover more modular and accessible to all users with varieties of helmets.

Many of our potential users remarked that the Cloud Cover would be great for many of their friends and family, especially those involved in bike accidents or who have had brain injuries from crashes. But if this product was to be used throughout the user's bike ride, then what would happen if the text message was sent by accident? We knew that users could hit a large bump during their ride, or drop their helmet on the floor, forgetting to turn off the Cloud Cover device. As a result we implemented a "kill" switch of sorts: once the sequence is triggered by the accelerometer and force sensors, the FONA is delayed by 15 seconds before sending the text to the emergency contact. If the button is pressed during that 15 seconds, the entire trigger sequence is canceled and the components continue to gather data just as before. This cancel button removes any sort of worry that the user has about causing their loved ones undue stress.

The most requested update to the Cloud Cover from users throughout the design process was to miniaturize even further. We as a team acknowledged this point up until the Expo and we still acknowledge it looking into the future. For these future iterations, we can cut down the long wires that currently occupy the majority of the space inside the Cloud Cover box. Unfortunately, we are restricted by the FONA and its battery's large size, but hopefully after further miniaturization efforts we can decrease the size and weight of the Cloud Cover for the user's ease and comfort.

VI. CONCLUSIONS

We have successfully created Cloud Cover, a device attachable to bike helmets that alerts the user's emergency contacts if a crash has occurred with a message stating the user's GPS location head injury site. The alert message is sent when an acceleration large enough to signify possible head injury occurs, and when enough pressure is applied to the user's head. In the process, we honed our soldering skills, developed a familiarity with coding, improved our understanding of circuitry and wiring, learned how to design using CAD, perfected our 3D

Printing Skills, and bonded as a team through collaboration. Although we have assembled a well-functioning product, our user feedback still indicates many areas for possible improvements.

With more time we would have loved to add a tail light for additional safety, and reduce size and weight by minimizing our hardware even further. We would definitely pass on to all future creators the phrase: “Fail Early, Fail Often.” We failed early and often, and this was integral to our success at Expo. Without the deadlines imposed by Mr. Godrick the week before, we are not sure what our final product would have been. It felt like we failed too often sometimes, but we failed just early enough to bring our original vision for the project to life.

VII. BUDGET/ BILL OF MATERIALS

- Adafruit Fona: $91 - included our lipoly battery.
- Helmets for testing and demonstration: $45
- Electronic Components (2 Arduino Pro Minis, Force Sensors, Accelerometers, etc.): $112
- PLA - $12
- Total budget: $260. Target: $375.

VIII. TIMELINE

The original timeline for our final project as proposed on the preliminary design review was rarely met. Beginning October 20-26, we aimed to finalize our design plan, order all parts and begin to test the FONA, force sensors, and accelerometers. From October 27-November 7 we planned to get each individual component working and the begin prototyping, which led into November 8-11 where we intended to put all components together and test. Lastly, from November 12-15 our team set aside time for aesthetics and further improvements based on the feedback from our peers.

Alongside our proposed timeline above, our team paid close attention to the due dates given to the class as a whole, such as the Hardware Component and Functional Project Demos. The abundance of work and iterations that came along with Cloud Cover, however, was an aspect that our team never planned; a consequence for these difficulties that we did not account

for was not following our intended timeline. We ended up not meeting any in-class deadlines with what we hoped for, but always succeeded on those aspect a few days later. Our actual timeline did look like the initial one, but instead pushed about a week and half back because we kept facing unexpected setbacks. Because of the delay, we failed to finish our project before Thanksgiving break like we had hoped, leaving us with little time to perfect the aesthetics. Although we lost that time, during the last week before Expo we altered our design in some smaller ways to provide a polished look for the public.

In that last seven days before the final Expo, our team made a lot of advances. Before that last week we had completed approximately 85% of our project; we had all its aspects ready, but the most important parts, the circuits and Arduino, were failing to work *together* after we moved them from the main breadboard and larger Arduino RedBoard to the smaller components through soldering. Once the problems were addressed in that final week and the code (see Appendix) was working perfectly, we had just enough time to reprint our CAD casing and attach it to the helmet in a more aesthetically pleasing manner. Fortunately, we finished the last 15% of Cloud Cover in time for the public Expo, and it performed seamlessly.

IX. APPENDIX

Cloud Cover 10 (Final Code):

```
#include "Adafruit_FONA.h"

#define FONA_RX 8   //defines Softwareserial pins for fona to arduino.  TX must be pin nine per the fona hookup guide.
#define FONA_TX 9
#define FONA_RST 7

#include <Wire.h>  //includes libraries to run the fona, ADXL345, and pressure sensors.
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

/* Assign a unique ID to this sensor at the same time */
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);

void displaySensorDetails(void)
{
  sensor_t sensor;
  accel.getSensor(&sensor);     //sets up the accelerometer
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" m/s^2");
```

```
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" m/s^2");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" m/s^2");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

const int FSR_PIN1 = A0; // Pin connected to FSR1/resistor divider1     //defines pins for force sensors
const int FSR_PIN2 = A1; // Pin connected to FSR2/resistor divider2
const int FSR_PIN3 = A2; // Pin connected to FSR3/resistor divider3
const int FSR_PIN4 = A3; // Pin connected to FSR4/resistor divider4

const float VCC = 3.2; // Measured voltage of Ardunio 3.3V line          //initializes resistance variables for
force sensors based on inactive resistance.
const float R_DIV1 = 3230.0; // Measured resistance of 3.3k resistor for Sensor 1
const float R_DIV2 = 3230.0; // Measured resistance of 3.3k resistor for Sensor 2
const float R_DIV3 = 3230.0; // Measured resistance of 3.3k resistor for Sensor 3
const float R_DIV4 = 3230.0; // Measured resistance of 3.3k resistor for Sensor 4

// this is a large buffer for replies
char replybuffer[255];

// We default to using software serial. If you want to use hardware serial
// (because softserial isnt supported) comment out the following three lines
// and uncomment the HardwareSerial line
#include <SoftwareSerial.h>
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;

// Hardware serial is also possible!
//  HardwareSerial *fonaSerial = &Serial1;

// Use this for FONA 800 and 808s
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);
// Use this one for FONA 3G
//Adafruit_FONA_3G fona = Adafruit_FONA_3G(FONA_RST);

uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);

uint8_t type;

int ledPin = 13;   // LED connected to digital pin 13
int inPin = 13;     // pushbutton connected to digital pin 13 for cancel button.
int val = digitalRead(13);
int cancel = 0;
int z;
float accelread;
```

#define sendto "2176910781" //defines global "variables" that functions within any loop below can access. This is also where the phone number is changed within the code.

```
void setup() {

  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
  pinMode(inPin, INPUT);        // sets the digital pin 13 as input

  while (!Serial);

  Serial.begin(115200);  //The next 20 lines or so initialize the fona when it powers up.  This can be seen on the fona itself when the power and network leds are solid on.
  Serial.println(F("FONA basic test"));
  Serial.println(F("Initializing....(May take 3 seconds)"));

  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while (1);
  }
  type = fona.type();
  Serial.println(F("FONA is OK"));
  Serial.print(F("Found "));
  switch (type) {
    case FONA800L:
      Serial.println(F("FONA 800L")); break;
    case FONA800H:
      Serial.println(F("FONA 800H")); break;
    case FONA808_V1:
      Serial.println(F("FONA 808 (v1)")); break;
    case FONA808_V2:
      Serial.println(F("FONA 808 (v2)")); break;
    case FONA3G_A:
      Serial.println(F("FONA 3G (American)")); break;
    case FONA3G_E:
      Serial.println(F("FONA 3G (European)")); break;
    default:
      Serial.println(F("???")); break;
  }


  // Optionally configure a GPRS APN, username, and password.
  // You might need to do this to access your network's GPRS/data
  // network.  Contact your provider for the exact APN, username,
  // and password values.  Username and password are optional and
  // can be removed, but APN is required.
  //fona.setGPRSNetworkSettings(F("your APN"), F("your username"), F("your password"));
```

```
  // Optionally configure HTTP gets to follow redirects over SSL.
  // Default is not to follow SSL redirects, however if you uncomment
  // the following line then redirects over SSL will be followed.
  //fona.setHTTPSRedirect(true);

 Serial.begin(9600);
  pinMode(FSR_PIN1, INPUT); //Define each Serial Pin as an Input location for the Force Sensor
Calculations and Data
  pinMode(FSR_PIN2, INPUT);
  pinMode(FSR_PIN3, INPUT);
  pinMode(FSR_PIN4, INPUT);

#ifndef ESP8266
  while (!Serial); // for Leonardo/Micro/Zero
#endif
  Serial.begin(115200);
  Serial.println("Accelerometer Test"); Serial.println("");

  /* Initialise the sensor */
  if(!accel.begin())
  {
    /* There was a problem detecting the ADXL345 ... check your connections */
    Serial.println("Ooops, no ADXL345 detected ... Check your wiring!");
    while(1);
  }

  /* Set the range to whatever is appropriate for your project */
  accel.setRange(ADXL345_RANGE_16_G);
  // accel.setRange(ADXL345_RANGE_8_G);
  // accel.setRange(ADXL345_RANGE_4_G);
  // accel.setRange(ADXL345_RANGE_2_G);

  /* Display some basic information on this sensor */
  displaySensorDetails();

  /* Display additional settings (outside the scope of sensor_t) */
  Serial.println("");
}

void loop() {

   /* Get a new sensor event */
  sensors_event_t event;
  accel.getEvent(&event);

  /* Display the results (acceleration is measured in m/s^2) */
 // Serial.print("X: "); Serial.print(event.acceleration.x/); Serial.print("  ");
  //Serial.print("Y: "); Serial.print(event.acceleration.y/); Serial.print("  ");
```

```
  //Serial.print("Z: "); Serial.print(event.acceleration.z/); Serial.print("  ");Serial.println("m/s^2 ");
float accelread =
sqrt(sq(event.acceleration.x/9.8)+sq(event.acceleration.y/9.8)+(sq(event.acceleration.z/9.8)));
Serial.println(accelread);
Serial.println(cancel);


if (accelread < 1.05){
 int fsrADC1 = analogRead(FSR_PIN1);
  // If the FSR has no pressure, the resistance will be
  // near infinite. So the voltage should be near 0.

    // Use ADC reading to calculate voltage:
    float fsrV1 = fsrADC1 * VCC / 1023.0;
    // Use voltage and static resistor value to
    // calculate FSR resistance:
    float fsrR1 = R_DIV1 * (VCC / fsrV1 - 1.0);
//    Serial.println("Resistance of Sensor 1: " + String(fsrR1) + " ohms");
    // Guesstimate force based on slopes in figure 3 of
    // FSR datasheet:
    float force1;
    float fsrG1 = 1.0 / fsrR1; // Calculate conductance
    // Break parabolic curve down into two linear slopes:
    if (fsrR1 <= 600)
      force1 = (fsrG1 - 0.00075) / 0.00000032639;
    else
      force1 =  fsrG1 / 0.000000642857;
    Serial.println("Force of Sensor 1: " + String(force1) + " G"); //Print out detected force at Sensor 1
    Serial.println();
    // No pressure detected


  int fsrADC2 = analogRead(FSR_PIN2);
  // If the FSR has no pressure, the resistance will be
  // near infinite. So the voltage should be near 0.
    // Use ADC reading to calculate voltage:
    float fsrV2 = fsrADC2 * VCC / 1023.0;
    // Use voltage and static resistor value to
    // calculate FSR resistance:
    float fsrR2 = R_DIV2 * (VCC / fsrV2 - 1.0);
//    Serial.println("Resistance of Sensor 2: " + String(fsrR2) + " ohms");
    // Guesstimate force based on slopes in figure 3 of
    // FSR datasheet:
    float force2;
    float fsrG2 = 1.0 / fsrR2; // Calculate conductance
    // Break parabolic curve down into two linear slopes:
    if (fsrR2 <= 600)
      force2 = (fsrG2 - 0.00075) / 0.00000032639;
```

```
    else
      force2 =  fsrG2 / 0.000000642857;
    Serial.println("Force of Sensor 2: " + String(force2) + " G"); //Print out detected force at Sensor 2
    Serial.println();
    // No pressure detected


  int fsrADC3 = analogRead(FSR_PIN3);
  // If the FSR has no pressure, the resistance will be
  // near infinite. So the voltage should be near 0.
    // Use ADC reading to calculate voltage:
    float fsrV3 = fsrADC3 * VCC / 1023.0;
    // Use voltage and static resistor value to
    // calculate FSR resistance:
    float fsrR3 = R_DIV3 * (VCC / fsrV3 - 1.0);
//    Serial.println("Resistance of Sensor 3: " + String(fsrR3) + " ohms");
    // Guesstimate force based on slopes in figure 3 of
    // FSR datasheet:
    float force3;
    float fsrG3 = 1.0 / fsrR3; // Calculate conductance
    // Break parabolic curve down into two linear slopes:
    if (fsrR3 <= 600)
      force3 = (fsrG3 - 0.00075) / 0.00000032639;
    else
      force3 =  fsrG3 / 0.000000642857;
    Serial.println("Force of Sensor 3: " + String(force3) + " G"); //Print out detected force at Sensor 3
    Serial.println();

  int fsrADC4 = analogRead(FSR_PIN4);
  // If the FSR has no pressure, the resistance will be
  // near infinite. So the voltage should be near 0.
    // Use ADC reading to calculate voltage:
    float fsrV4 = fsrADC4 * VCC / 1023.0;
    // Use voltage and static resistor value to
    // calculate FSR resistance:
    float fsrR4 = R_DIV4 * (VCC / fsrV4 - 1.0);
//    Serial.println("Resistance of Sensor 4: " + String(fsrR4) + " ohms");
    // Guesstimate force based on slopes in figure 3 of
    // FSR datasheet:
    float force4;
    float fsrG4 = 1.0 / fsrR4; // Calculate conductance
    // Break parabolic curve down into two linear slopes:
    if (fsrR4 <= 600)
      force4 = (fsrG4 - 0.00075) / 0.00000032639;
    else
      force4 =  fsrG4 / 0.000000642857;
    Serial.println("Force of Sensor 4: " + String(force4) + " G"); //Print out detected force at Sensor 4
    Serial.println();
```

```
   if (force1==0 && force2==0 && force3==0 && force4==0) {    //The next block of code until the
elseif statement are boolean statements that determine which force sensor experienced the most force and
make sure it exceeds a minimum threshold of 250 to keep a twist of the head from activating.
     Serial.println("No Force Detected");
   }
   else

   if (force4 > force3 && force4 > force2 && force4 > force1 && force4 > 250) {
     Serial.println("User Head Impacted at Force Sensor 4 Location");
     z = 4;
   //Determine which Sensor Location received the most blunt force trauma
   }
   if (force3 > force4 && force3 > force2 && force3 > force1  && force3 > 250) {
     Serial.println("User Head Impacted at Force Sensor 3 Location");
     z = 3;
   //Determine which Sensor Location received the most blunt force trauma
   }
   if (force2 > force4 && force2 > force3 && force2 > force1 && force2 > 250) {
     Serial.println("User Head Impacted at Force Sensor 2 Location");
     z = 2;
   //Determine which Sensor Location received the most blunt force trauma
   }
   if (force1 > force4 && force1 > force3 && force1 > force2 && force1 > 250) {
     Serial.println("User Head Impacted at Force Sensor 1 Location");
     z = 1;}
   //Determine which Sensor Location received the most blunt force trauma
   delay (100);
 }
else if (accelread >= 12 & z != 0) {   //once both the accelerometer threshold (set in this line) and the a
pressure sensor have been tripped, this if statement executes the emergency communication feature of
CloudCover.

   // turn GPS on

long int initialmillis = millis();   //initializes wait time for an unblocked delay.
         if (!fona.enableGPS(true))
        Serial.println(F("Failed to turn on"));
     accelread = 0;
  while (millis() - initialmillis < 45000){  //leaves an option during the startup delay for a button to be
pressed.
     if(digitalRead(13) == HIGH){
       delay(5);

       cancel = 1;    //once the cancel button is pressed, the variable changes to one, excluding the loop
below and sending the code back to the top.
       Serial.println(cancel);
```

```
      }

    while (millis() - initialmillis > 30000 & cancel != 1)     {            //If the cancel button is not pressed, the texting function below executes.
   // check for GPS location
      char gpsdata[120];
      fona.getGPS(0, gpsdata, 120);
      if (type == FONA808_V1)
        Serial.println(F("Reply in format: mode,longitude,latitude,altitude,utctime(yyyymmddHHMMSS),ttff,satellites,speed,course"));
      else
        Serial.println(F("Reply in format: mode,fixstatus,utctime(yyyymmddHHMMSS),latitude,longitude,altitude,speed,course,fixmode,reserved1,HDOP,PDOP,VDOP,reserved2,view_satellites,used_satellites,reserved3,C/N0max,HPA,VPA"));
      Serial.println(gpsdata);

      if (z = 1){                          //the next four lines determine what the third text will be based on which force sensor was triggered.
        if (!fona.sendSMS(sendto, "CloudCover has detected a possible head injury focused on the left of the forehead.  The user is at the following gps coordinates:")) {
        Serial.println(F("Failed"));
      } else {
        Serial.println(F("Sent!"));
      }
      }
       else if (z = 2){
        if (!fona.sendSMS(sendto, "CloudCover has detected a possible head injury focused on the right of the forehead.  The user is at the following gps coordinates:")) {
        Serial.println(F("Failed"));
      } else {
        Serial.println(F("Sent!"));
      }
      }
      else if (z = 3){
        if (!fona.sendSMS(sendto, "CloudCover has detected a possible head injury focused on the left of the rear of the head.  The user is at the following gps coordinates:")) {
        Serial.println(F("Failed"));
      } else {
        Serial.println(F("Sent!"));
      }
      }
      else if (z = 4){
        if (!fona.sendSMS(sendto, "CloudCover has detected a possible head injury focused on the right of the rear of the head.  The user is at the following gps coordinates:")) {
        Serial.println(F("Failed"));
      } else {
        Serial.println(F("Sent!"));
      }
```

```
      }
      if (!fona.sendSMS(sendto, gpsdata[4,5])) {  //second text with parsed gps data.  This will send a bunch of commas (an empty array) if the gps has not acquired a fix during the 30 second delay.
        Serial.println(F("Failed"));
      } else {
        Serial.println(F("Sent!"));
       delay(120000);
      }
            }
 }}
 cancel = 0;     //resets the cancel variable each loop in case the button was pressed so the code will keep cycling.
```

X. WORKS CITED/CONSULTED

"Adafruit FONA 808 Cellular + GPS Breakout." *Adafruit Learning System*, Adafruit Industries, 29 May 2015, learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/overview.

JIMBLOM. "Force Sensitive Resistor Hookup Guide." *Tutorials,* SparkFun Electronics, learn.sparkfun.com/tutorials/force-sensitive-resistor-hookup-guide/all.

THEDARKSAINT. "ADXL345 Hookup Guide." *Tutorials*, SparkFun Electronics, learn.sparkfun.com/tutorials/adxl345-hookup-guide.

Button. TUTORIALS > Built-In Examples > 02.Digital > Button *Tutorials,* Arduino https://www.arduino.cc/en/Tutorial/Button.