Bike Crash Detection

ECE 445 Final Report

Brian Lin, Dhruv Mathur, and Alex Tam
Team 24
TA: Kristina Miller
12/12/19

Abstract

This report explains the design and implementation of the bike crash detector device that we designed. The device will detect a crash and its severity using an IMU. After the crash is detected, the device will send a text to the emergency contacts with the location, time, and severity of the crash. This allows a potentially injured biker to get the help he/she needs. This device was made to be a reliable and inexpensive such that any biker can take advantage of it.

1 Introduction

1.1 Motivation

Biking has become a vital means of transportation. In 2017, approximately 47.5 million Americans cycled on a regular basis [1]. In 2015, there were 467,000 reported accidents involving bicycles [2]. Pedal cyclist fatalities in 2017 made up 2.1% of all traffic related deaths in 2017 [3]. This statistic is disproportionately high compared to the overall population of motorized vehicle users. There have been many expensive and inexpensive innovations to increase communication between cyclists and motor vehicles such as the Varia Rearview Radar ($200) [4] and the Zackees Turn-Signal Gloves ($60) [5], but the overall deaths per year in motor vehicle accidents involving pedal cyclists continues to increase [3]. Of all pedal cyclist deaths caused by motor vehicles, 82% involved the front of the vehicle [3], indicating that visibility is not the main issue in the crash. From 2000 to 2013 commuting rates increased 105% [6]. In these situations, it is rare for cyclists to ride in groups. Group riding not only makes the cycler more visible, it also allows for the rider to gain help immediately. Since this is not the situation for the majority of riders, a device to communicate with emergency contacts in the event of a crash would be beneficial. Cyclists lack the safety innovations that many modern cars benefit from such as the OnStar safety and notification system. There are devices for the bicycles that detect crashes, such as the Garmin Edge 530, one of the most inexpensive options at $300 [7]. However, there is no solo inexpensive device that offers these capabilities without the expensive overhead of active GPS for navigation and hardware to communicate with bike sensors or the rider's smartphone. Our device caters directly to commuters who have no need for expensive hardware to measure performance.

Our goal was to develop a device that can allow cyclists to have access to the critical care they might need immediately after the accident. It would be beneficial for many bikers to have a device that can detect when they are involved in a crash, and notify an emergency contact of the severity of the crash via text message so that the rider can get assistance. Many current solutions rely on devices that attach to helmets, but only 17% of all cycling fatalities involve a helmet [8]. A no hassle solution that is guaranteed to stay on the bike will ensure that the device is present when an accident occurs. Our solution will attach to the bike seat post and detect for crashes based on acceleration and rotation. It will then determine the rider's location and send a notification to the rider's emergency contact with the details of the accident. Our solution will be entirely self-contained and not rely on a connection to the rider's smartphone or other technology on the bicycle.

1.2 Solution

Right now there are no low-cost, attached to bike solutions for accident detection and notification. Devices on the market are traditionally attached to the helmet. However, most cyclists do not use a helmet making it pointless to design an accident detection device on the helmet. Attaching the device to the bicycle ensures that the device will be there when need be.

The device will be independent of an external cellular device to ensure that the emergency signal is sent even if the rider is thrown far from the bike or the rider's mobile device is broken in the crash. Furthermore, to ensure that as many cyclists have access to it, we want to implement a cheap and durable option. A helmet solution costs $50 or more [9] making it impractical to purchase for most consumers.

1.3 High-Level Requirements

- The device must accurately detect a crash with over 1g of force and distinguish crashes from simply dropping the bike or sudden controlled stops.
- The device must quickly send a message within 1 minute of the crash to emergency contact(s) with relevant information from the accident, specifically the time, location, and severity of the crash.
- Device must be compact enough to not be intrusive to the rider.

2 Design

2.1 Introduction and Block Diagram

The device has four main modules: power, control, sensor, and communication. The power module contains the power supply, a 3.7V LiPo battery, and a voltage regulator to step down to 3.3V for supply to ICs. The control module consists solely of the microcontroller, which interfaces with the sensors and communication module. The sensor module consists of the IMU and GPS chips, which are used by the microcontroller to detect crashes and location. The communication module contains the GSM chip and antenna that are used to connect to a cellular network in order to send text messages.

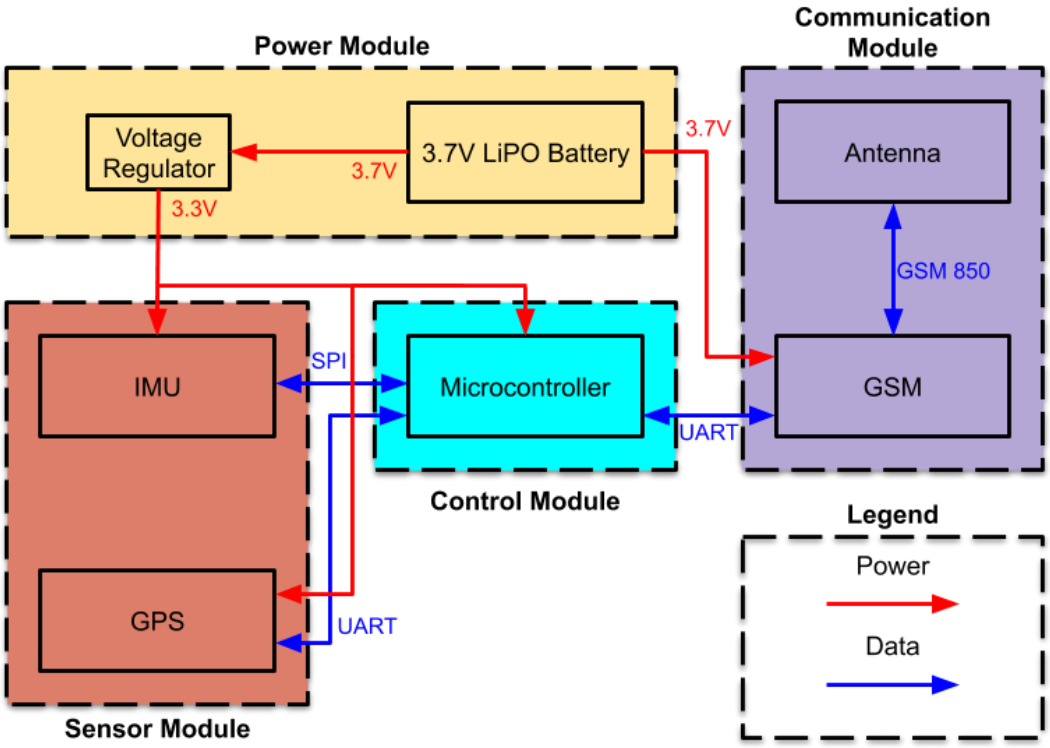

Figure 1: Block Diagram

2.2 Subsystems, Requirements, and Verifications

2.2.1 Power Module

The main power source was a 3.7 V LiPo battery. This provided power for the microcontroller, GPS, and GSM chips and was easily accessible for recharging when the battery was out of power. There was also a low-dropout linear voltage regulator with an output of 3.3 V. The GSM was powered directly off of the battery, while the microcontroller, IMU, and GPS were powered by the voltage regulator. The current requirement came from the requirements for all the chips on the board. The microcontroller drew a max of 14 mA, the IMU drew a max of 5 mA, and the GPS drew a max of 67 mA. The GSM drew a max of 100 mA expect when powering the antenna to send a text message, when it drew a max of 2 A for a brief moment.

The battery we selected had a capacity of 4000mAh. Current draw during normal activity without a crash is less than 20mA, as we only have to power the microcontroller and IMU. Therefore, the battery was able to last at least 200 hours with no crashes before requiring recharging.

Table 1: Power Module R/V

Requirement	Verification	Results
Must provide ≥2 A between 3.6 V - 3.9 V for the ICs and voltage regulator	1. Connect fully charged battery to a fixed load and make sure it provides more than 200 mA 2. Discharge battery and monitor voltage with a voltmeter to ensure it stays between 3.6 V - 3.9 V.	1. 3.7 V 4000 mAh LiPo battery 2. Actual output: 3.6 V - 4.1 V 3. Actual output: > 2A source for GSM
3.3 V regulator must provide ≥100 mA between 3 V - 3.45 V	1. Connect regulators to battery at input, and loads at each output and make sure it provides more than 100 mA. 2. Discharge battery and monitor voltage at both outputs with a voltmeter to ensure it stays between 3 V - 3.45 V.	1. Actual output: 3.29 V 2. Actual output: > 150 mA

2.2.2 Control Module

The microcontroller was an ATmega328P-PU, and was used to process the IMU data, communicate with the GPS chip, and control the GSM chip. Communication with the IMU was over SPI, and communication with the GPS and GSM chips was over software serial. The

microcontroller was responsible for identifying and categorizing a crash, acquiring the current location, and packaging all the relevant information into a text message to send to the saved emergency contact.

Table 2: Control Module R/V

Requirement	Verification	Results
Must support two simultaneous software serial connections	Connect two serial ports to the two ICs and verify active communication to both	1. ATmega328p satisfies requirement
Must communicate at speeds over 1MHz with the IMU over SPI	Record the number of readings in a fixed time frame and check that the date rate is greater than 1MHz	1. SPI communication at 8MHz

2.2.3 Sensor Module

The IMU used was the MPU 9250, containing a 3-axis accelerometer that measures ±16 g (m/s/s scaled by Earth gravity) and a 3-axis gyroscope that measures ±2500 dps (degrees per second). According to Stone and Broughton, fatalities on bicycles increases from 3% to 10% when vehicle speeds 30mph to 50mph [10]. This speed correlates to 10g of acceleration in a normal crash with an average adult male [11], which was the maximum acceleration we wanted to measure accurately, as anything higher would automatically be categorized as a severe crash. The data was smoothed by a software implemented moving average filter to remove any noise in the measurements.

The GPS was a NEO-6M GPS Module, that can communicate over a serial interface with the microcontroller to report the location of the device when a crash occurs. The chip reports location in the standard NMEA format, which can then be parsed to extract the latitude and longitude data.

Table 3: Sensor Module R/V

Requirement	Verification	Results	
Must accurately detect accelerations over 1g and rotations over 45dps	Acceleration testing can be done by performing controlled drops of the IMU. Rotation testing can be done by mounting the IMU to a spinning wheel.	**Testing Thresholds** Fall: 1G in the X or Y plane Light Crash: 0.7G + 45dps Medium Crash 1.2G + 75dps Severe Crash: 1.7G + 90dps	**Realistic Thresholds** Fall: 1G in the X or Y plane Light Crash: 1.5G + 45dps Medium Crash 4G + 75dps Severe Crash: 7G + 90dps
Must provide GPS data in NMEA format accurate to within 10 meters	Parse the acquisition data to ensure the correct encoding and manually verify accuracy of location data when outside using a secondary GPS (Google Maps).	1. Actual 6 decimals of NMEA accuracy a. Within 1m of location	

2.2.4 Communication Module

The GSM was a SIM800L module, which received serial data from the microcontroller with the contents of a text message and the recipient. The GSM module communicated internally with the antenna to send the message over a cellular network.

Table 4: Communication Module R/V

Requirement	Verification	Results
Must connect to a 2G cellular network in populated areas without major interfering structures	Confirm that the SIM card can communicate with the network in various outside locations around campus.	1. -80 dB 2G signal strength outside of buildings
Must be able to send a text message within 1 minute of detecting a crash	Simulate conditions for a crash. Then measure the time for the system to send the notification and ensure it is within 1 minute.	1. Text message sent in 56 seconds from crash simulation

2.3 Crash Thresholding

As there are several different types of crashes and severities, we want to correctly identify the crash. The most important crash we must detect is one that renders the rider unable to contact anyone making the device mandatory to work. A severe crash is when a collision occurs while the bike is travelling around 10 m/s and sends the rider flying. Assuming the acceleration from 10 m/s to 0 m/s occurs in a time frame of 0.1 s, the g's can be calculated by dividing the acceleration by Earth gravity. The acceleration is $a = \Delta v / t$, and Earth gravity is $g = 9.81$ m/s/s. In this case, an acceleration with a change in velocity of 10 m/s over 0.1 s results in a crash with 10.2 g's experienced during impact.

The minimum threshold for a crash is a change from 1 m/s to 0 m/s in a time frame of 0.5 s. This crash results in 1.02 g's experienced.

However, simply detecting a spike in acceleration is not enough to accurately detect a crash. Accelerations of around 1.5 g's can be expected from sharp braking and bumps in the riding surface. Therefore, we also used the gyroscope to measure both the speed of rotation and the absolute angle of the bicycle. Our minimum threshold for rotational speed to detect a crash was 45 degrees per second, which must be seen at the same time as a spike in acceleration.

2.4 Software

The code running on the microcontroller can be divided into two main components. First, the controller establishes a SPI communication protocol with the IMU, and begins reading acceleration and gyroscopic measurements. These are saved into finite length arrays used for a moving average filter. The force data was recorded individually in the X, Y, and Z planes. 25 cycles of force data were recorded in an array, and after 25 cycles, the array was averaged. This averaged data was converted into a vector sum which was used to detect crashes. A similar process was performed with gyroscope data but with 5 cycles.

Using the process T. Islam outlines in "Comparison of complementary and Kalman filter based data fusion for attitude heading reference system" [12], the complementary system was chosen to calculate angle. The complementary system is based on the theory that acceleration data is only valid on the long term and gyroscope data is only valid in the short term. Using a simple moving average, the accelerometer data is filtered and the gyroscope data will undergo numerical integration followed by a moving average. The filtered accelerometer data is summed with the gyroscopic data with the formula angle = .98 * (angle + gyroscopicData*dt) + .02 * (accelerometerData). This formula combines 98% of the past angular data with 2% of the change due to the new acceleration.

Once a crash has been identified and categorized, the microcontroller begins communicating with the GPS chip. It waits for the chip to obtain 5 consecutive readings to ensure that the location has stabilized and is accurate. The latitude and longitude are then saved, and

communication with the GSM begins. The controller configures the chip into SMS mode, and then sends the phone number the text should be sent too. Finally, the message is sent, containing the category of the crash, the GPS location, and other strings to create a coherent message. The full code can be found in Appendix A.

SPI was chosen due to its faster clock cycle and the ability to test the IMU while normally powering the Arduino with the normal 5V. The device was tested extensively with SPI connection. However, when testing with the PCB, the SPI pins of the PCB seemed to be malfunctioning and communication with the IMU ceased. In an effort to fix the problem without removing all components on the PCB, it was decided to switch to I2C and solder the backup IMU to through hole to communicate with the Arduino. This solution worked and allowed us to seamlessly communicate with the IMU.

2.5 Circuit Design

2.5.1 Schematic

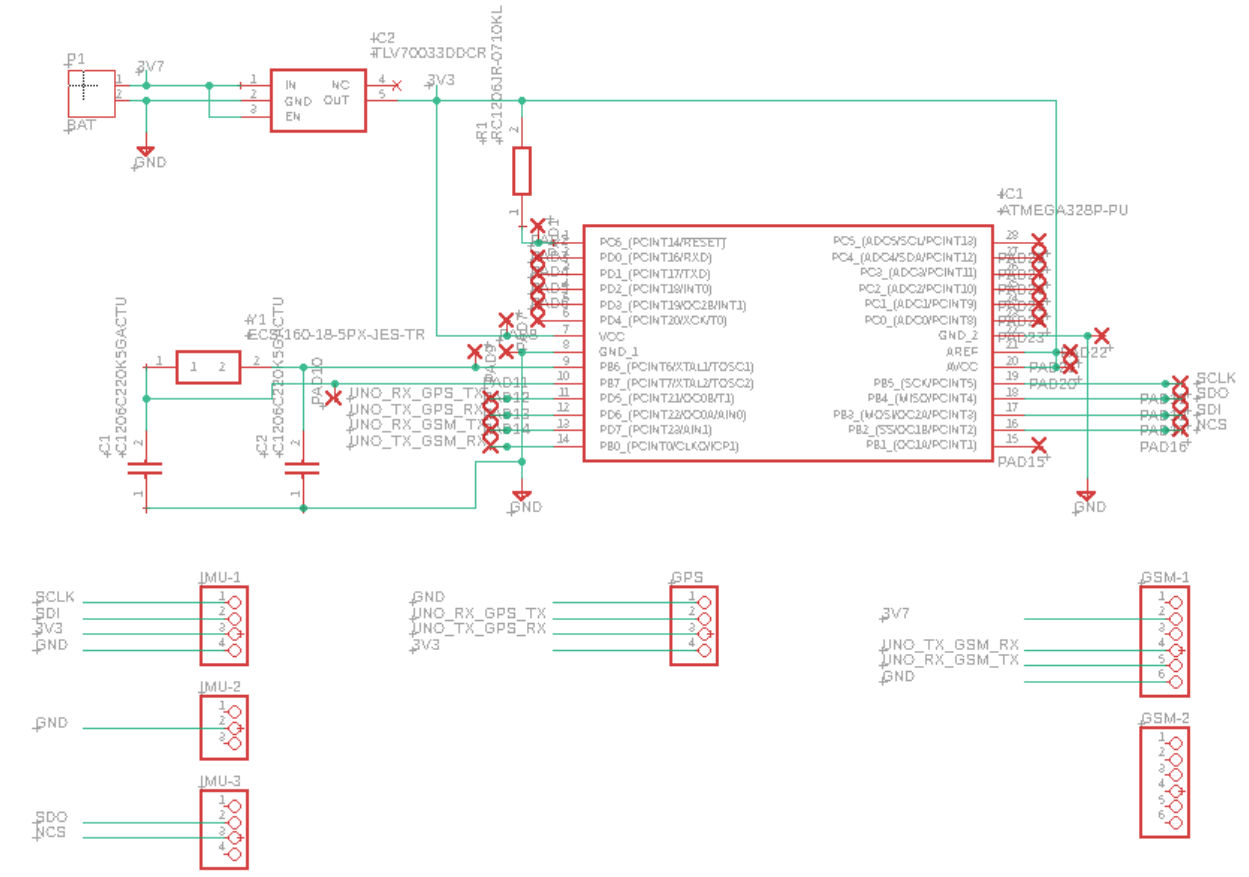

Figure 2: Circuit Schematic

Figure 2 contains the circuit schematic for our PCB. The design was chosen to be as modular as possible. The voltage regulator (IC2) outputs the 3.3 V supply to the microcontroller, IMU,

and GPS chips. The microcontroller (IC3) has a pull-up resistor (R1) on pin 1 (reset), an oscillator (X1) and two capacitors to ground (C1, C2) on pins 9 and 10 (XTAL), and power and ground connections. The other three chips are all on separate breakout boards, requiring only connections for power and communication. These can be found at the bottom of the schematic for the IMU, GPS, and GSM in order from left to right.

2.5.2 PCB Layout

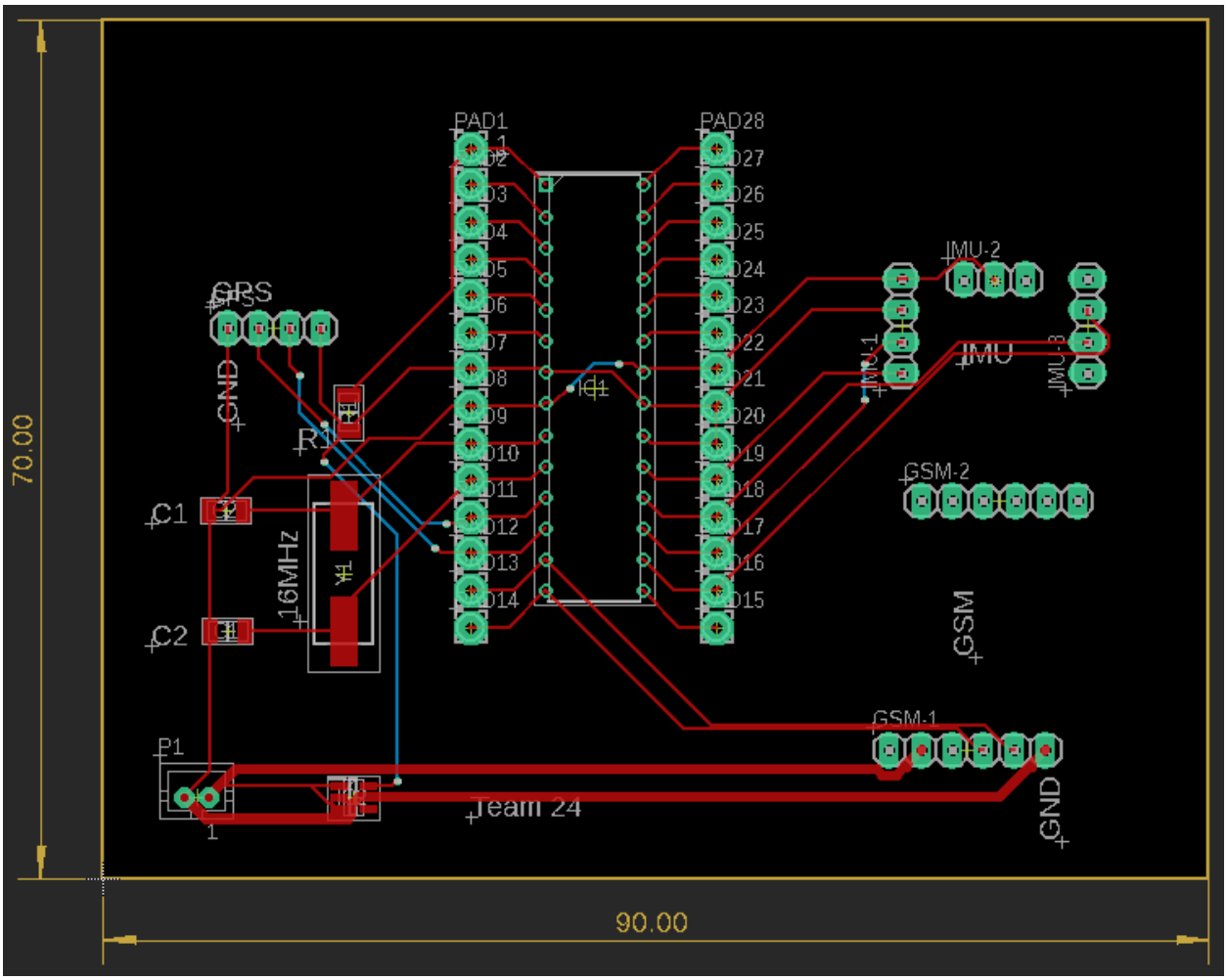

Figure 3: Data Circuit Schematic

Figure 3 contains the PCB layout design. The PCB was designed to be compact yet large enough to easily handle for simulating crashes. The connection to the battery and the voltage regulator are in the bottom left, and the resistor, capacitors, and oscillator SMDs are located to the left of the microcontroller. All four ICs are connected with through hole pads. For debugging purposes, each pin of the microcontroller is also connected to a second through hole pad. All traces are the standard 6 mil width, with the exception of the power and ground connections to the GSM. These are 30 mil traces to accommodate for the occasional 2 A surge drawn by the antenna when sending a text message. Our final board size was 70 x 90 mm, which can be reduced in the future by removing unnecessary through hole pads and reorganizing the locations of components.

3 Cost Analysis

Table 5: Cost Analysis

Part #	Mft	Description	Module	Price	Qty	Total
Breakout MPU-9250	InvenSense	6-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer	Sensor	$ 11.96	1	$ 11.96
ATmega328	Atmel	8-bit AVR microcontroller	Control	$ 11.85	1	$ 11.85
SIM800L	SIM Tech	Quad-band Mini GPRS GSM	Communication	$ 12.73	1	$ 12.73
Neo-6M	Ublox	6 GPS module ROM, crystal	Communication	$ 5.75	1	$ 5.75
TLV70033DDCR	Texas Instruments	LDO Voltage Regulators 200mA Low IQ	Power	$ 0.34	1	$ 0.34
LD1117V33	STMicroelectronics	IC REG LINEAR 3.3V 800MA TO220AB	Power	$ 0.55	1	$ 0.55
YDL 3.7V 4000mAh LiPo	YDL	3.7V 4000mAh LiPo Battery with JST connector	Power	$ 13.39	1	$ 13.39
Labor	UIUC	3 students for 10 weeks at 10 hours/week	Labor	$ 10.00	300	$ 3,000.00
Total						$ 3,056.57

4 Conclusion

4.1 Ethics

As our device transmits data to external sources, we must respect privacy as stated in the ACM Code of Ethics 1.6 [13]. This means that we must only collect the minimum amount of information to make our device work. Our device requires a name, emergency contacts, and current location. The last piece of personal information, location, will only be tracked after the device is activated after a crash is detected which protects the rider's privacy.

ACM Code of Ethics 1.2 [13], states that the device should do no harm. The device does not directly contact Emergency Services which mitigates any false alarms. The device instead contacts emergency contacts which allows for a human intermediate to first make the decision whether emergency services is needed.

Lastly, we should "be honest and realistic in stating claims or estimates based on the available data," according to ACM Code of Ethics 1.3 [13]. As we are detecting data from our gyroscope and accelerometer, we need to provide accurate guidelines and limits for crash detection. Therefore, we must find the threshold in our data set that distinguishes between a crash and normal usage of the bike. This device does not replace a rider's responsibility to call 911 in the event of a severe crash.

4.2 Accomplishments

The project met all the high-level requirements set at the beginning of the project. The device was able to accurately detect a crash with over 1g of force and distinguish crashes from simply dropping the bike or sudden controlled stops. Secondly, the device was able to quickly send a message within 1 minute of the crash to emergency contact(s) with relevant information from the accident, specifically the time, location, and severity of the crash. On average we were able to receive the text in 56 seconds. Finally, we were able to create a compact device that would allow it to be mountable on most bikes.

4.3 Future Design Elements

Due to the budget and time constraints of the project, our team decided to pursue electrical engineering aspects rather than the physical design of crash detection. Because of this, no crash resistant box was formulated and parts were not ordered in duplicate in case one of the components broke during crash testing. This led to us being unable to generate true severe crash level forces in the constraints of the lab. Ultimately, to adequately demo our project, the team decided to lower the force thresholds to detect various crashes. This allowed us to continue testing our device without the fear of breaking a non-replaceable component. In

the future, we will work with the machine shop to design an enclosure for our device that will be strong enough to protect the components without inhibiting the signal of the GSM and GPS.

Future iterations of this design will utilize Bluetooth to connect the device to a simple mobile app. This app will allow for easier modification of the emergency contact list as well as contains an indicator for the battery to tell the consumer when the device must be recharged. The app will also contain settings like a virtual bike lock setting that would automatically text the location of the bike every 2 minutes to the user in the event that their bike was moved without their knowledge. The lock will automatically be disabled when the user is within 5 feet of their bike. This is determined using Bluetooth's direction capabilities.

References

[1] Gough, C. (2018). *Cycling - Statistics & Facts*. [online] Statista. Available at: https://www.statista.com/topics/1686/cycling/ [Accessed 19 Sep. 2019].

[2] Napoli Shkolnik PLLC, “How many people are killed or injured riding bikes?,” *Lexology*, 11-Jun-2018. [Online]. Available: https://www.lexology.com/library/detail.aspx?g=f7666791-e58d-41bb-9551-9e73423b2f79. [Accessed: 18-Sep-2019].

[3] National Center for Statistics and Analysis, “Bicyclist and Pedestrian Safety - nhtsa.gov,” *NHTSA*. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/14046-pedestrian_bicyclist_safety_resources_030519_v2_tag.pdf. [Accessed: 18-Sep-2019].

[4] Garmin and Garmin Ltd., “Varia™ Rearview Radar: Bike Radar,” *Garmin*. [Online]. Available: https://buy.garmin.com/en-US/US/p/518151. [Accessed: 18-Sep-2019].

[5] “Zackees Turn Signal Gloves for Cycling,” *Zackees*. [Online]. Available: https://zackees.com/. [Accessed: 18-Sep-2019].

[6] C. Szczepanski, “Bicycle Commuting Data,” *League of American Bicyclists*, 15-Jun-2013. [Online]. Available: https://bikeleague.org/commutingdata. [Accessed: 18-Sep-2019].

[7] Garmin and Garmin Ltd., “Garmin Edge® 530: Bike Computer with Performance Insights,” *Garmin*. [Online]. Available: https://buy.garmin.com/en-US/US/p/621224. [Accessed: 18-Sep-2019].

[8] “The Stats Behind The Bicycle Helmet,” *Bicycle Universe*, 18-Jan-2019. [Online]. Available: https://bicycleuniverse.com/stats-behind-bicycle-helmet/. [Accessed: 18-Sep-2019].

[9] “ANGi Crash Sensor,” *Specialized Bicycle Components USA*. [Online]. Available: https://www.specialized.com/us/en/angi-crash-sensor/p/170203. [Accessed: 18-Sep-2019]

[10] Stone, M. and Broughton, J. (2013). *Getting off your bike: cycling accidents in Great Britain in 1990–1999*. [online] ScienceDirect. Available: https://www.sciencedirect.com/science/article/pii/S0001457502000325?via%3Dihub [Accessed 19 Sep. 2019].

[11] Schmidt, J. (2019). Collision Reconstruction Concepts (A Series) | DJS Associates. [online] DJS Associates. Available: https://www.forensicdjs.com/blog/collision-reconstruction-concepts-a-series/ [Accessed 19 Sep. 2019].

[12] T. Islam, M. S. Islam, M. Shajid-Ul-Mahmud, and M. Hossam-E-Haider, “Comparison of complementary and Kalman filter based data fusion for attitude heading reference system,” *AIP Conference proceedings*, 2017.

[13] “ACM Code of Ethics and Professional Conduct,” *ACM.* [Online] Available : https://www.acm.org/code-of-ethics [Accessed 19 Sep. 2019].

[14] GitHub. (2019). MPU9250. [online] Available at: https://github.com/bolderflight/MPU9250 [Accessed 3 Dec. 2019].

[15] GitHub. (2019). TinyGPSPlus. [online] Available at: https://github.com/mikalhart/TinyGPSPlus [Accessed 3 Dec. 2019].

Appendix A

Included in this appendix is the software programmed to the microcontroller. There are two libraries included to interface with the IMU and GPS. The first is MPU9250 by bolderflight [14], which allows us to simply instantiate the IMU SPI connection and request readings of acceleration and gyroscope whenever we need to. The second is TinyGPSPlus by mikalhart [15], which simplifies the process of parsing the NMEA data returned by the GPS to get the latitude and longitude information.

```
#include "MPU9250.h"
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

// an MPU9250 object with the MPU-9250 sensor on SPI bus 0 and chip select
pin 10
//MPU9250 IMU(SPI,10);
//int status;

MPU9250 IMU(Wire,0x68);
int status;


float lat = 0.0;
float lng = 0.0;
int gps_count = 0;

bool gsm_ready = 0;

// Create a TinyGPS++ object
TinyGPSPlus gps;

// Create a software serial port called "gpsSerial"
SoftwareSerial gpsSerial(5, 6);

SoftwareSerial gsmSerial(7, 8);

const int numSample = 25;
const int numSampleShort = 5;
const float beta = .98;
const int dt = 1;


int clearCounter = 5* numSample;
float accelXArray [numSample];
float accelYArray [numSample];
float accelZArray [numSample];
float gyrXArray [numSampleShort];
float gyrYArray [numSampleShort];
float gyrZArray [numSampleShort];

bool start =0;
int i =0;
int k=0;
```

```
float totalX =0;
float avgX =0;
float totalY =0;
float avgY =0;
float totalZ =0;
float avgZ =0;

float totalXGyr =0;
float avgXGyr =0;
float totalYGyr =0;
float avgYGyr =0;
float totalZGyr =0;
float avgZGyr =0;



float curX=0;
float curY=0;
float curZ=0;

float curXGyr=0;
float curYGyr=0;
float curZGyr=0;
float angle = 0;
float anglePrev =0;
float magAcc = 0;
float magRot = 0;


void setup() {
  // serial to display data
  Serial.begin(115200);
  while(!Serial) {}

  // start communication with IMU
  status = IMU.begin();
  if (status < 0) {
    Serial.println("IMU initialization unsuccessful");
    Serial.println("Check IMU wiring or try cycling power");
    Serial.print("Status: ");
    Serial.println(status);
    while(1) {}
  }
  Serial.println("IMU Connected");
  float gxb = 0.001; // gyro bias of 0.001 rad/s
  IMU.setGyroBiasX_rads(gxb);
  float gyb = 0.001; // gyro bias of 0.001 rad/s
  IMU.setGyroBiasY_rads(gyb);
  float gzb = 0.001; // gyro bias of 0.001 rad/s
  IMU.setGyroBiasZ_rads(gzb);

  gpsSerial.begin(9600);

  gsmSerial.begin(9600);
```

```
}

void loop() {
  // read the sensor
  IMU.readSensor();
  if (start)  {

  avgX =totalX/numSample;
  avgY =totalY/numSample;
  avgZ =totalZ/numSample;
  magAcc= pow(((avgX*avgX) + (avgY*avgY) +(avgZ*avgZ)),.5)/9.8;
  magRot = pow(((curXGyr*curXGyr) + (curYGyr*curYGyr)
+(curZGyr*curZGyr)),.5);
   if ((abs(1-magAcc)>2)&&(abs(magRot)>60)){
     Serial.print("Bad crash");
     Serial.print("\n");
     start = 0;
     i=0;
     k=0;
     crash_notification(3);
    }
    else if ((abs(1-magAcc)>1.2)&&(abs(magRot)>22)){
     Serial.print("Medium crash");
     Serial.print("\n");
          start = 0;
     i=0;
     k=0;
     crash_notification(2);
    }
    else if( (abs(1-magAcc)>.07)&&(abs(magRot)>10)){
  Serial.print("Mild crash");
     Serial.print("\n");
          start = 0;
     i=0;
     k=0;
     crash_notification(1);
    }
    else if (abs(avgY)>7.5){
       Serial.print("Fall");
     Serial.print("\n");
          start = 0;
     i=0;
     k=0;
     crash_notification(0);
    }


  avgXGyr =totalXGyr/numSampleShort;
  avgYGyr =totalYGyr/numSampleShort;
  avgZGyr =totalZGyr/numSampleShort;


  curX = (IMU.getAccelX_mss()/avgX)-1;
  curY = (IMU.getAccelY_mss()/avgY)-1;
  curZ = (IMU.getAccelZ_mss()/avgZ)-1;
```

```
  curXGyr = (IMU.getGyroX_rads()/avgXGyr)-1;
  curYGyr = (IMU.getGyroY_rads()/avgYGyr)-1;
  curZGyr = (IMU.getGyroZ_rads()/avgZGyr)-1;

 }
  totalX = totalX - accelXArray[i];
  accelXArray [i] = IMU.getAccelX_mss();
  totalX = totalX + accelXArray[i];

  totalY = totalY - accelYArray[i];
  accelYArray [i] = IMU.getAccelY_mss();
  totalY = totalY + accelYArray[i];

  totalZ = totalZ - accelZArray[i];
  accelZArray [i] = IMU.getAccelZ_mss();
  totalZ = totalZ + accelZArray[i];

    totalXGyr = totalXGyr - gyrXArray[k];
  gyrXArray [k] = IMU.getGyroX_rads();
   totalXGyr = totalXGyr + gyrXArray[k];


       totalYGyr = totalYGyr - gyrYArray[k];
  gyrYArray [k] = IMU.getGyroY_rads();
   totalYGyr = totalYGyr + gyrYArray[k];

       totalZGyr = totalZGyr - gyrZArray[k];
  gyrZArray [k] = IMU.getGyroZ_rads();
   totalZGyr = totalZGyr + gyrZArray[k];


  i=i+1;
  if (i>=numSample)
    {
      i=0;
      start=1;
    }

      k=k+1;
  if (k>=numSampleShort)
    {
      k=0;
    }
      anglePrev = angle;
     angle = (beta*(anglePrev+curXGyr*dt)) + ((1-beta)*(curX));


  delay(100);
}

void crash_notification(int cat)
  {

     gpsSerial.listen();
    while (gps_count < 5) {
        while (gpsSerial.available() > 0) {
            if (gps.encode(gpsSerial.read())) {
```

```
                if (gps.location.isValid()) {
                    lat = gps.location.lat();
                    lng = gps.location.lng();
                    gps_count++;
                    delay(100);
                }
            }
        }
    }

    gsmSerial.listen();
    gsmSerial.println("AT");
    delay(500);
    gsmSerial.println("AT+CMGF=1"); // Configuring TEXT mode
    delay(500);
    gsmSerial.println("AT+CMGS=\"+019134499583\"");
    delay(500);
    switch (cat){
      case 0:
        gsmSerial.print("Bike Fall");
        delay(500);
        break;
      case 1:
        gsmSerial.print("Mild Crash");
        delay(500);
        break;
       case 2:
        gsmSerial.print("Medium Crash");
        delay(500);
        break;
      case 3:
        gsmSerial.print("Bad Crash");
        delay(500);
        break;
    }
    gsmSerial.print(" at Location: "); //text content
    delay(500);
    gsmSerial.print(lat, 6);
    delay(500);
    gsmSerial.print(", ");
    delay(500);
    gsmSerial.print(lng, 6);
    delay(500);
    gsmSerial.write(26); // end text content
}
```