Bicycle Street Notification System

By
Anant Jani
Savannah Russell
Stephanie Wong

Final Report for ECE 445 Senior Design, Fall 2017
TA: Channing Philbrick

13 December, 2017
Project No. 8

Abstract

The Bicycle Street Notification System is a sensor and light-based system with the purpose of increasing cyclist visibility and safety, while minimizing the risk of potential accidents. Our design, as described in this report, utilizes three main sensors to interface with automatic lights and includes a speedometer to put cyclists more on par with motorized vehicles. The system has manual control buttons on the handlebars for left and right turn signals that can be automatically shut off after the completion of a turn using a gyroscope. The brake light uses data from the accelerometer to indicate when the light should be turned on. The speedometer displays the current speed of the cyclist by reading data obtained from the magnetic reed switch. All sensors and lights are fully integrated into one system that is capable of being used on the street.

Table of Contents

1. Introduction

1.1 Objective

Even as cars become more available to the average civilian, many people still choose to ride bicycles for either economical, physical, or environmental reasons. There are over 66 million cyclists in the United States [1], which equates to over 12% of Americans biking outdoors [2]. Many cities and towns have designated biking trails and specialized areas for cyclists. But more often than not, a cyclist must share the road with pedestrians and motorized vehicles. By sharing the same road with cars, they also share the same accident risks that car drivers face but without the same warning signs and safety measures. A bicycle rider must depend on signals formed by the cyclist's arms and hands. These signals require the cyclist to take his hands off the bike and lose focus of the road, making cyclists a hazard to themselves and others sharing the road.

With a desire to make it safer for cyclists to share the road, our solution is to create a bicycle light system similar to that of a car. The system includes left and right turn signals, a brake light, and hazard lights. There will be buttons for user control of the turn signals, an accelerometer-monitored braking system, automatic shut off for the turn signals once the cyclist has completed a turn, as well as a speedometer. The lighting fixtures will make cyclists more visible on the road and allow other road users to be more aware of their intentions. The automatic turn signal shut off and the brake light allow cyclists to focus on what is happening on the road instead of the movement of their hands.

1.2 High-Level Requirements

- The product shall automatically shut off the blinking left or right turn signal after the gyroscope detects a return from at least a 5° orientation change.
- The product shall turn on the brake light once the accelerometer detects deceleration of at least ¼ mi/h or greater.
- The product shall display a speedometer reading with no more than ±1 mi/h error.

2. Design

2.1 Block Diagram

The project can be separated into several main components, as illustrated in Figure 1. The battery supplies the necessary energy to the power source, which is able to power the entire system. Sensors interface with the microcontroller in order to send the correct information to the peripherals. Particularly, the accelerometer, gyroscope, and magnetic reed switch interface with the microcontroller to correctly operate the brake light, turn signals, and speedometer, respectively. With the time we had remaining, we were able to develop two additional features that were not part of the original requirements: a photocell and battery gauge. The photocell has been fully integrated in our system and will be further explained in this report. However, while the prototype of the battery gauge showed to be successful, it has not been fully incorporated in our final product.

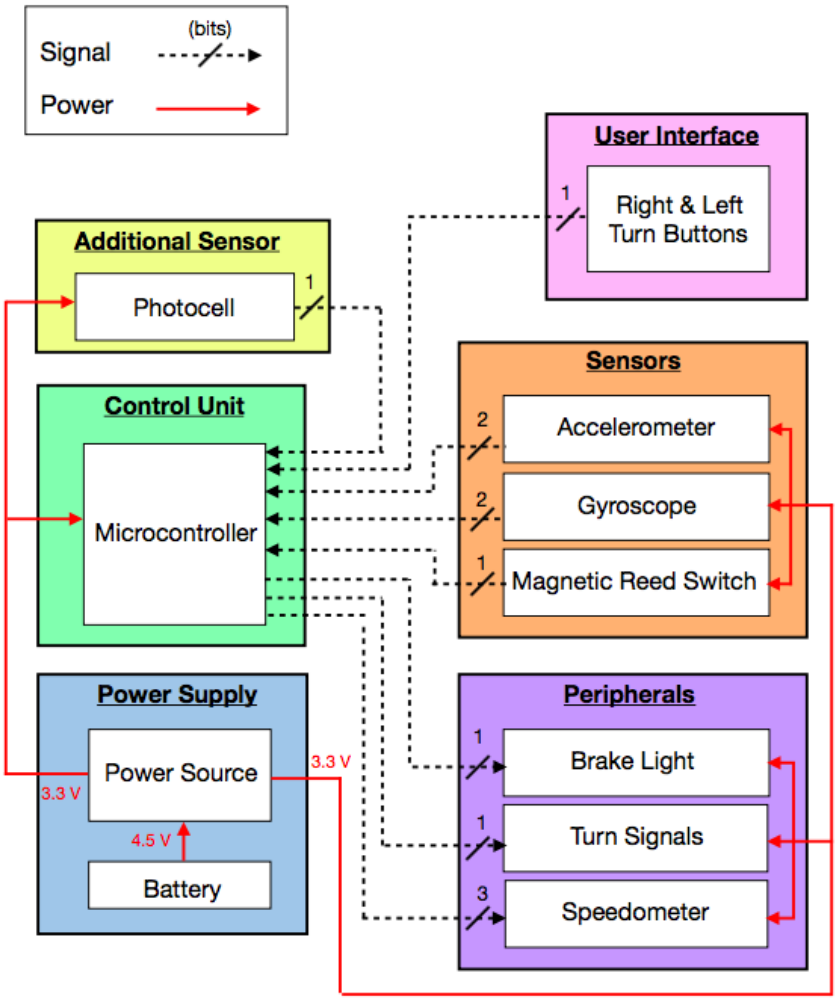

Figure 1: High-Level System Block Diagram

2.1.1 Power Supply

To power the entire system we utilize three AA batteries. These alkaline batteries are widely available, inexpensive, and are functional for our project since our system only requires a low-draw of power. Three AA batteries supply a nominal voltage of 4.5 V with an estimated capacity of 2,850 mAh rated at 25 mA of continuous drain [3]. Since the notification system is entirely battery powered, we aim to have a reasonably long battery life. The light-emitting diode (LED) turn signals and speedometer display require the most current at 20 mA each. Thus, the nominal load current required will be 20 mA. The estimated battery life for our system is approximately 428 h. This is only an estimate, as the average capacity used for the power calculations is the rated capacity at 25 mA of continuous drain. Equation (1) calculates the estimated battery life as

$$H = \frac{C}{A} \tag{1}$$

where H is the estimated hours of battery life, C is the total charge, and A is the nominal current.

The battery voltage is greater than the maximum input voltage for multiple components, so a voltage regulator is required. The voltage regulator (TLV71333PDBVR) outputs a steady voltage of 3.3 V, which is optimal for the microcontroller (ATMega328P) and our other sensors. The voltage regulator is useful for our battery-powered system because it prevents the flow of large currents from the batteries to our components. The coupling capacitors improve the input ripple and minimize noise to provide a steadier output [4].

While testing the design of our power supply module, we accidentally shorted the printed circuit board (PCB) and destroyed the voltage regulator component. With the project deadline quickly approaching, there was not enough time to place an order for new parts. Utilizing our resources, we found that a voltage regulator on the Arduino board had a similar layout to our original voltage regulator. The new voltage regulator (KB33) had the same layout as our PCB trace, which allowed us to incorporate it into our design [5]. Figure 3 shows the final circuit schematic for our power supply module, including the new capacitor value.

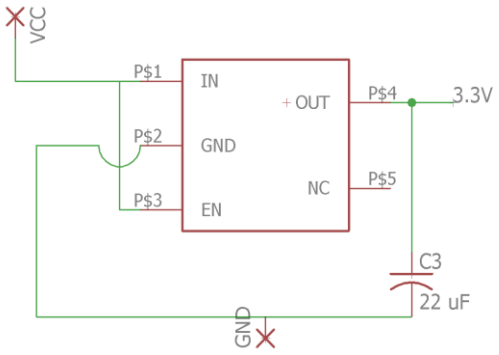

Figure 2: Final KB33 Power Supply Schematic

2.1.2 Microcontroller

The microcontroller (ATMega328P) is responsible for sending and receiving all signals in our system. The ATMega328P is an optimal choice for our system because it has the necessary amount of input and output pins and is capable of being powered by the regulator output voltage of 3.3 V. A 16 MHz crystal and various capacitors are used to help stabilize the microcontroller clock signal through the use of a low power crystal oscillator. At 16 MHz, we use two 22 pF capacitors as recommended by the datasheet [6]. Figure 3 illustrates the circuit schematic for the microcontroller.

The final program for our microcontroller is detailed in Appendix C. This program receives data from the sensors and displays the desired output to the peripherals.

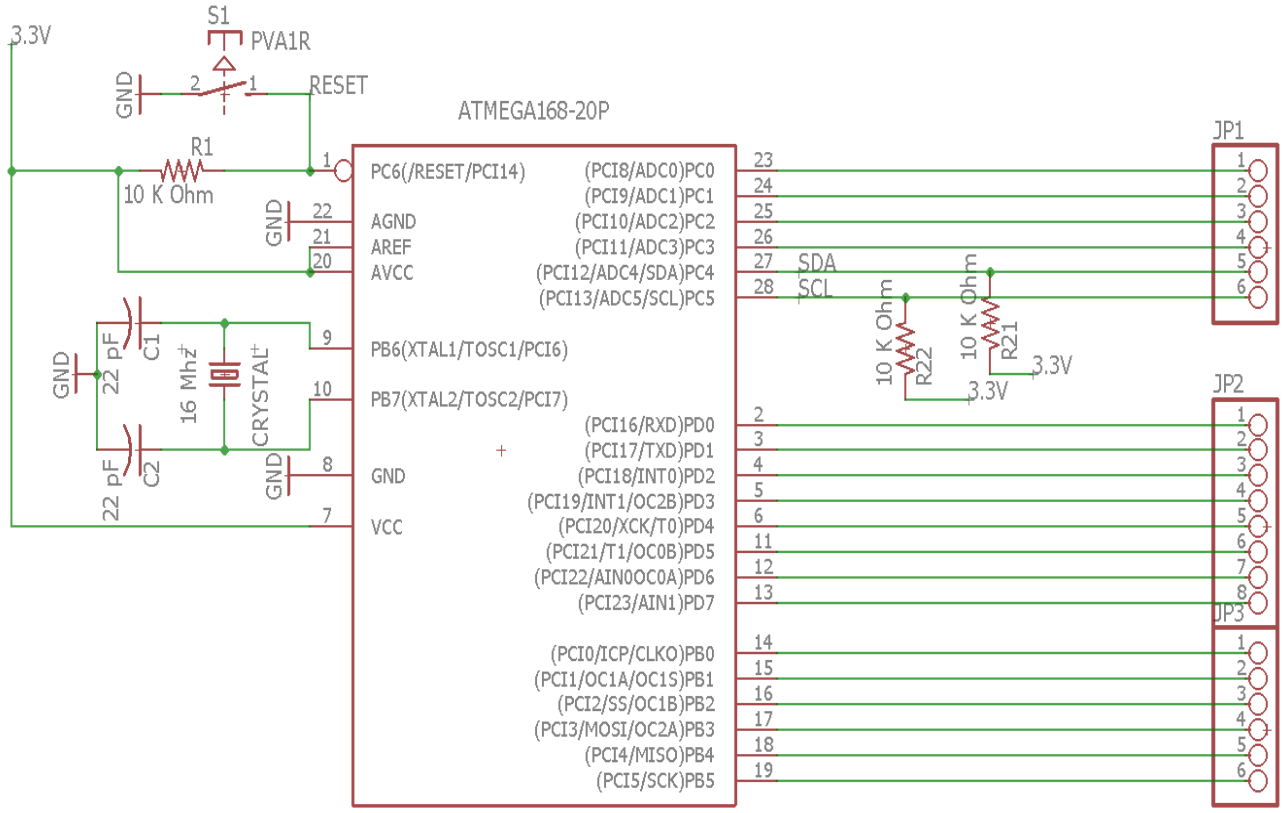

Figure 3: ATMega328P Microcontroller Schematic

2.1.3 Turn Buttons

A cyclist can utilize the turn buttons to manually power the LED signals on and off. We chose to use simple push buttons (KS-01Q-01) for this purpose. As shown in Figure 4, the buttons are active low, which means the microcontroller reads a low signal only when the button is pressed. Designing the button schematic to be active low results in a more responsive button by preventing the microcontroller from reading floating voltages. The resistor limits the current in the turn button schematic.

An alternative design consideration was the possibility of using toggle switches to manually control the LED turn signals. However, switches would cause issues with the microcontroller by remaining in the high or low state until the user manually toggles the switch to the desired position. Because the cyclist

may automatically shut off a turn signal via the gyroscope, the toggle switch may remain in an undesired position.

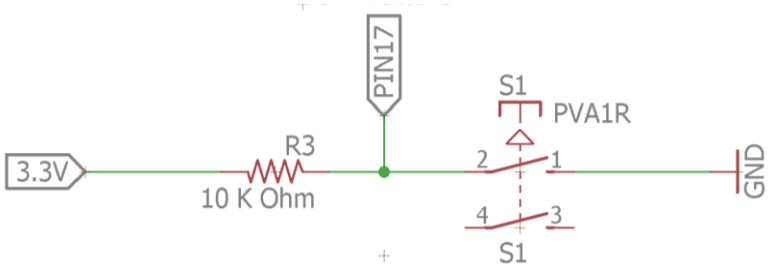

Figure 4: Turn Button Schematic

2.1.4 LED Turn Signals

The LED turn signals receive data from the turn buttons via the microcontroller to be either blinking on or turned off. White LEDs (100F5T-YT-WH-WH) are placed under a diffuser casing because white-colored LEDs offer the greatest luminous intensity at 8 cd compared to any other color [7]. Figure 5 shows the LED signal schematic for each turn signal. The two 220 Ω resistors are used to limit the current through the circuit. This prevents the LEDs from drawing too much current and burning out.

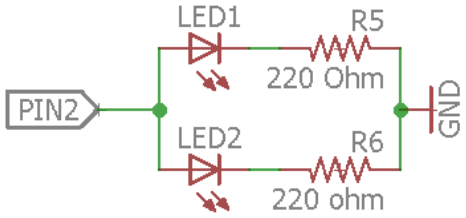

Figure 5: LED Turn Signal Schematic

2.1.5 Gyroscope

The gyroscope is a combined gyroscope and accelerometer sensor (MPU-6050), which is able to detect an orientation change when the cyclist completes a turn. This component was best-suited for our design because it is low cost, low power, and high-accuracy. For the purpose of detecting a turn, the gyroscope is able to detect a full scale range of ±2000 °/s and operate on 3.3 V [8]. A typical bicycle turn will be less than 90°, so the detection range of the gyroscope will be able to detect the orientation change of a turn.

Originally, we planned to purchase the MPU-6050 sensor and solder our own PCB with the schematic shown in Figure 6. However, the sensor package was a quad flat no-leads (QFN), which would require a socket to more easily solder. The socket costs upwards of $70, while a breakout board would be a fraction of that cost [9]. We decided to purchase the breakout board, which uses the same schematic as shown in Figure 6. This decision would allow us to better focus our time and efforts on improving the design and aesthetics.

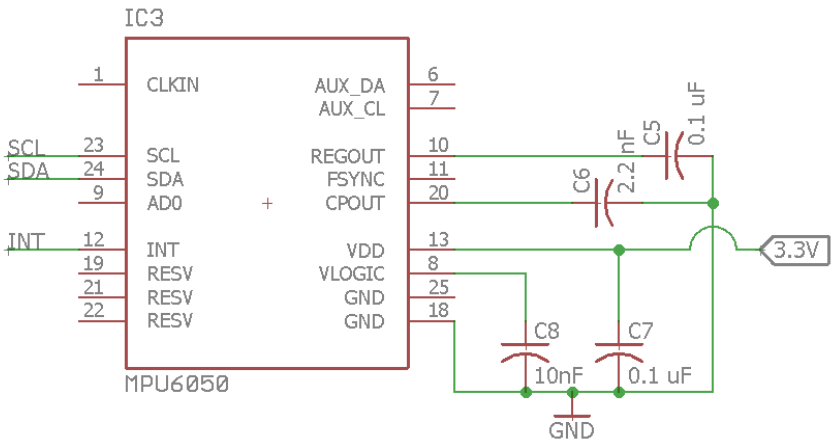

Figure 6: MPU-6050 Gyroscope/Accelerometer Schematic

2.1.6 Brake Light

Figure 7 shows the brake light schematic. It uses the same schematic layout as the LED turn signals. Two LEDs are used for better visibility during the day and night. The brake light turns on when the accelerometer detects a deceleration of at least ¼ mi/h.

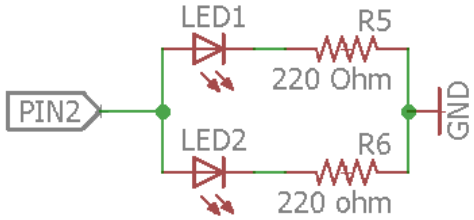

Figure 7: LED Brake Schematic

2.1.7 Accelerometer

As previously mentioned in Section 2.1.5, the accelerometer is a two-in-one sensor with the gyroscope. The accelerometer can read a wide range of acceleration data ranging from ±0.016 kgf [8]. This is sufficient because 0.016 kgf equates to nearly 157 m/s^2, and a cyclist typically would not be cycling at such high speeds. This conversion is calculated using Equation (2). The acceleration can be calculated by

$$A = 9.81 \times a \tag{2}$$

where A is the acceleration in m/s^2, 9.81 is the conversion factor from 0.001 kgf to m/s^2, and a is the acceleration in gf.

2.1.8 Magnetic Reed Switch

The magnetic reed switch is a simple switch that reacts to a magnetic impulse. Like any other switch, it has two states: opened and closed. Our switch is a naturally open switch, which means that the magnets will not send an impulse to the microcontroller until a magnet passes the switch. By positioning four magnets on the back wheel at 90° apart from each other, Equation (3) shows the calculation for the angular velocity as

$$\omega = A \div T \tag{3}$$

where ω is the angular velocity, A is the angle between each magnet, and T is the time difference between each magnet pass. Using the angular velocity, Equation (4) calculates the linear velocity as

$$V = \omega \times R \tag{4}$$

where V is the linear velocity, ω is the angular velocity, and R is the radius at which the reed switch is placed. Finally, Equation (5) finds the current speed in miles per hour by

$$S = V \times F \tag{5}$$

where S is the speed in miles per hour, V is the velocity in inches per second, and F is the conversion factor equivalent to 0.0568182.

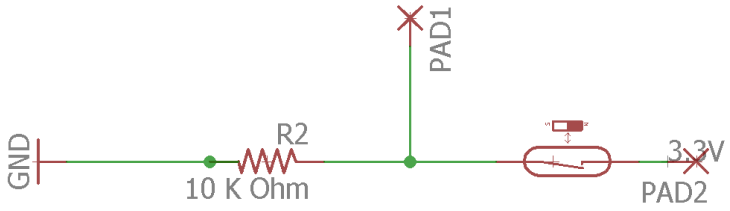

Figure 8: Magnetic Reed Switch Schematic

2.1.9 Speedometer Display

The speedometer display (SSD1306) is used to display the cyclist's current speed, which is calculated in Section 2.1.8. The display measures 0.96 inches diagonally and is a monochrome OLED display. An OLED display does not use a backlight and only outputs the desired pixels, thus reducing the overall power consumption and providing the best contrast for the displayed image [10]. This is important for our speedometer, which should be visible at a glance from a wide variety of angles.

Initially, we considered a hexadecimal digit display for our speedometer. After more research of the design, we determined that a hexadecimal display required too many microcontroller pins. A single digit on the hexadecimal display requires seven inputs. If the bicyclist travels over 9 mi/h, a hexadecimal display would not be possible with the amount of available microcontroller pins. Thus, we decided that an OLED display requiring only two data pins is much more practical.

2.1.10 Photocell

The photocell is a feature we have included in addition to the original design and requirements. For this reason, the photocell component is not included in Appendix A and will not be listed under Section 3 of this report.

The photocell (SEN-09088) detects a change of light in the environment and automatically powers on all LED turn signals when the environment is dark. We decided to add this feature because it would help conserve battery power during daylight hours. The SEN-09088 is inexpensive, compact, and consumes minimal power compared to other photocell sensors. Figure 9 shows the circuit schematic of the photocell

design. The photocell changes resistance based on the amount of light detected, and the 1000 Ω resistor is used as part of the voltage divider. A different resistor value could be used, but we would have to alter the threshold for automatically turning on the LED turn signals.

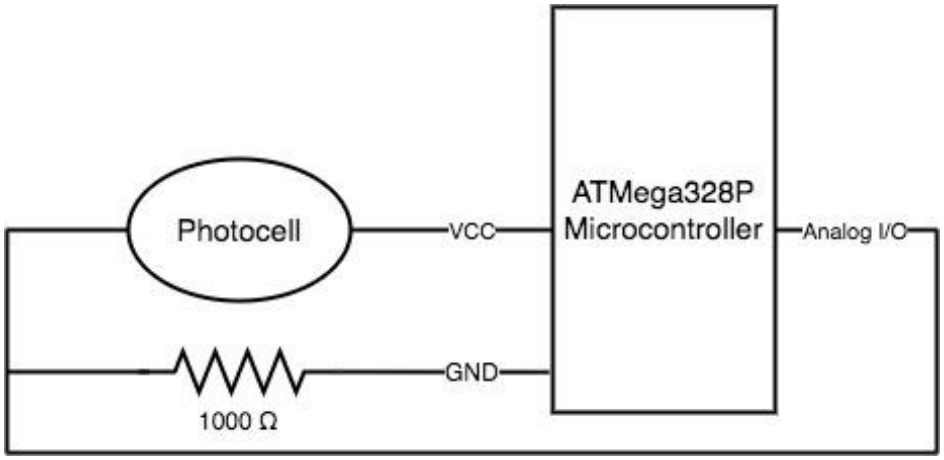

Figure 9: Photocell Circuit Schematic

3. Verification

Each subsection of Section 3 explains the testing and verification performed for each module. Refer to the requirements and verifications listed in Appendix A.

3.1 Power Supply

As mentioned in Section 2.1.1, the voltage regulator outputs a steady 3.3 V to the microcontroller. Each major component requires various amounts of current, which is drawn as needed. More specifically, the LEDs and speedometer require 20 mA to operate. When all components of our system are connected and powered on, the display and LED turn signals are on. The output voltage of the regulator is measured to be 3.31 V, which is the expected value. The fact that these two components are operational verifies that the power supply module is able to output a current of 20 mA.

3.2 Microcontroller

The microcontroller sends and receives all signals of our system. We verify it is able to simultaneously receive all the necessary signals by demonstrating the completely functional final product. During the final demonstration we were able to show that when a button is pressed, the corresponding LED turn signal blinks. At the same time, the turn signals can be automatically turned off because the microcontroller continuously analyzes the gyroscope data when a turn signal is blinking. Also, if the cyclist brakes at any time, the brake light turns on. Additionally, the speedometer always displays the current cyclist speed. Thus, the microcontroller can simultaneously receive and process up to four signals.

3.3 Turn Buttons

By designing our turn buttons to be active low, the buttons were responsive when pressed and could be verified by an LED. The button completes the circuit when pressed, and the pin reads a low voltage. When not pressed, the circuit is open. The voltage of the button pin reads a high voltage of 3.3 V. The LED state changed depending on the button state, which verifies that the turn buttons work properly as desired.

3.4 LED Turn Signals

The LED turn signal should blink after receiving a signal from the microcontroller, which is determined by the turn button state. We program the microcontroller to blink the proper LED turn signal after a turn button has been pressed. To verify the turn signals operate as desired, we connected a turn button and an LED to digital input/output (I/O) pins. After uploading the microcontroller code detailed in Appendix C, we were able to verify the LED blinks when the button is pressed and stops blinking when the button is pressed again.

It is important for the turn signals to be visible during both day and night from at least 10 ft away, which is approximately the length of one car. Shown using a tape measure, Figure 10 demonstrates the visibility of the lights measured 10 ft away.

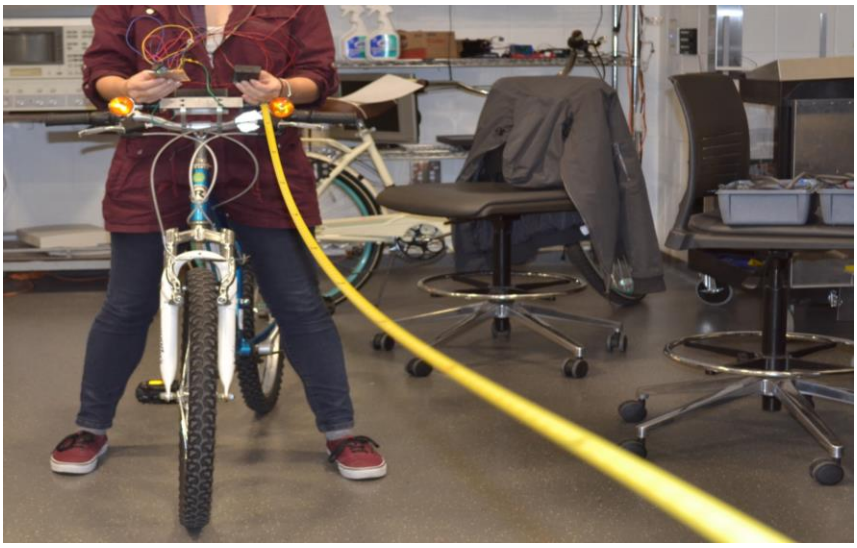

Figure 10: Verification for LED Turn Signals

3.5 Gyroscope

Just as in a car, the bicycle's automatic turn signals should only be triggered after a turn has been completed, and not while the cyclist is still attempting the turn. For this reason, we needed the gyroscope to discern between shallow turns and sharp turns alike. To ensure this, we needed to verify that the gyroscope could detect a turn of at least 2° from its starting position. Figure 11 shows the data captured while performing turn tests. Yaw data measured from -20 to 20 is a dead zone, which means the microcontroller does not perform any task when values are within this zone. This is to account for noise from erroneous readings. A value of ±40 on the y-axis corresponds approximately to a 25° turn from the normal of the handle bars. This verifies that our gyroscope is capable of detecting turns of at least 2° from its starting position.

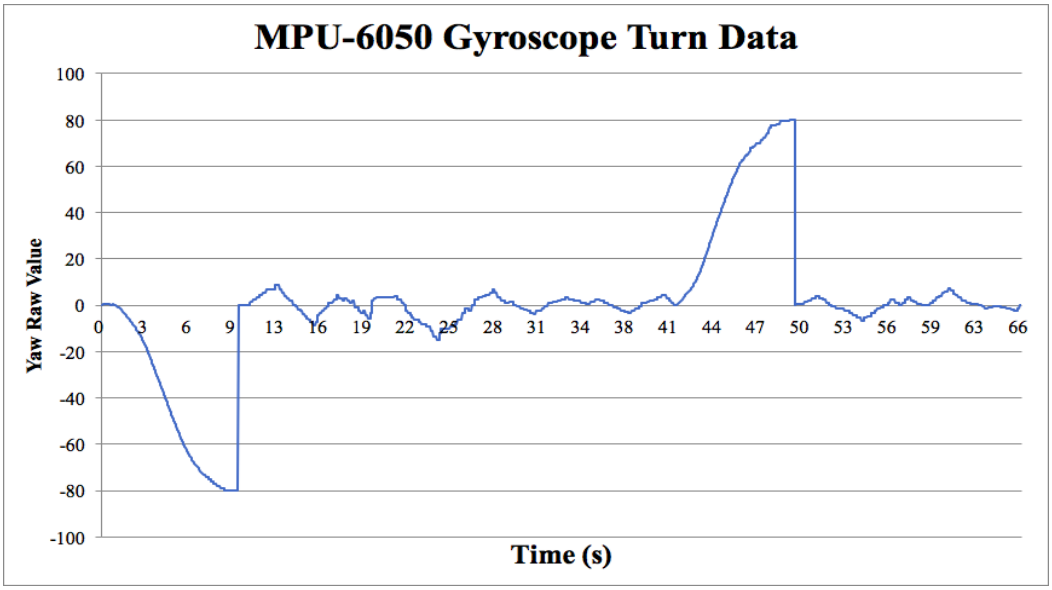

Figure 11: Gyroscope Turning Data Collection

3.6 Brake Light

In order to make our system safe for outside use, it was important to encase our electrical components within regulated casing. As the brake light was a component that would face moisture and splashing from the rain, we verified that the light could withstand being in contact with water for at least 30 minutes. Figure 12 shows the brake light being sprayed with water and verifies that it can function properly when turned on.

Figure 12: Verification for Brake Light

3.7 Accelerometer

As a cyclist's speed is never truly constant, we needed a way for the system to gauge that a definite change in speed has occurred so it can determine when the brake light should be on. As such, we require that our accelerometer is able to detect a speed change of at least ¼ mi/h. Figure 13 shows the accelerometer data captured during brake testing. The "Raw X-Values" are in units of meters per second, where 2 m/s equates to approximately 4.5 mi/h. This shows that the accelerometer can easily detect changes in acceleration that are at least ¼ mi/h.

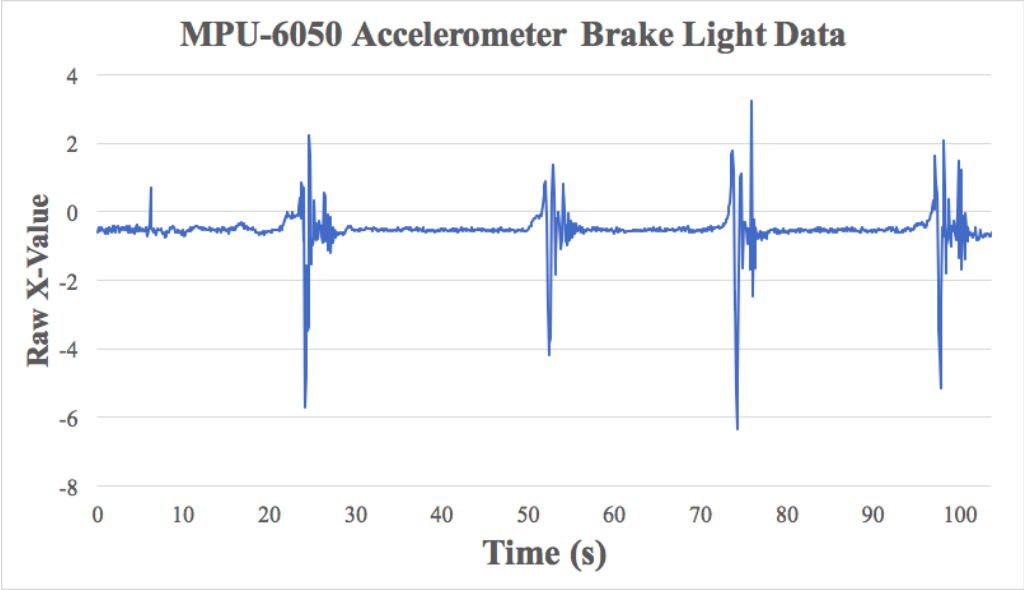

Figure 13: Accelerometer Brake Light Data Collection

3.8 Magnetic Reed Switch

To have an accurate and reliable reading for our speedometer, we needed to verify that it was accurate within ±1 mi/h. Figure 14 shows our speedometer on the left and compares it to the actual speed of the bicycle using the application “GPS Speedometer and Odometer” from the Google Play Store [11].

Figure 14: Verification of Magnetic Reed Switch

3.9 Speedometer Display

In order to assure the cyclist can focus solely on the road, the speedometer display should be visible from at least 18 in away. It should be visible and legible enough that the cyclist can quickly glance at it and know what speed they are traveling. Figure 15 shows our verification for the visibility of the speedometer display, which is placed 18 in away measured by a tape measure. It is clear the display is visible and readable.

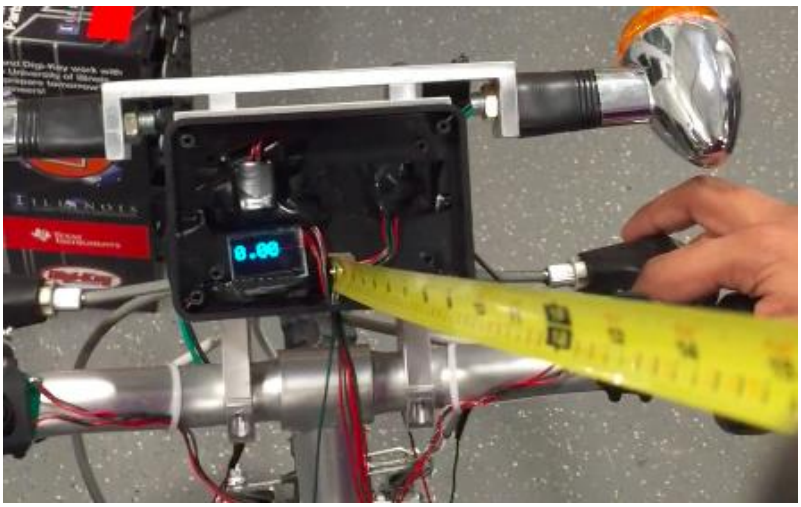

Figure 15: Verification of Speedometer Display

4. Costs

We estimate our labor costs to be $30 per hour for 10 hours a week. For 16 weeks with a 2.5 expense multiplier, Equation (6) calculates the total labor costs for three team members as

$$3 \times 2.5 \times \frac{\$30}{hour} \times \frac{10\ hour}{week} \times 16\ weeks = \$36{,}000 \tag{6}$$

The costs for the components needed are detailed in Appendix B. The total cost was $90.76.

5. Conclusion

5.1 Accomplishments

We have been successful in completing our original design of the bicycle notification system. We were able to verify all our requirements at both the high level, as well as for each module listed in Appendix A. All modules were completely integrated into a fully operational product.

Because we had satisfied our original requirements with additional time remaining, we were able to implement extra features including a photocell and battery gauge. The photocell increases the battery life of the system by automatically powering on all turn lights only when the environment is dark, and we were able to successfully incorporate the photocell into the final product. The battery gauge notifies the user of the current battery level by outputting the battery life to the display. We were able to create a prototype on a breadboard for testing; however, due to time constraints we were unable to incorporate the working battery gauge for the project demonstration.

5.2 Uncertainties

Our biggest uncertainty with this project is its effectiveness in all types of environments. As we tested our system in leveled hallways and smooth concrete, we are unsure of how the system may behave when used on other terrains that are more rocky or hilly. Further testing and possible re-calibration of sensor sensitivity would be needed to assure complete functionality for multi-environment use.

5.3 Ethical Considerations

We have based our ethical considerations on the IEEE Code of Ethics [12]. As our system is meant to be used outside on the road, our principal concern was the safety of the user. As such, during the initial design of our system we made the choice to have our casings fulfill the Ingress Protection Rating (IP67) guidelines. These requirements state that the device must be protected by errant dirt, dust, and sand-like objects, and be able to withstand being in contact with water for at least 30 minutes [13]. During the building of our project, it came to our attention that the IP67 standard requires the casing to be submerged in water for at least 30 minutes. As our system is meant to be used on land and will only come in contact with splattering of water from rain, we came to the conclusion that our system only needs to be held to IP54 standard. The IP54 guidelines state that the casing must be dust proof and protected from water projected from a nozzle [14]. After discussing the new design consideration with our teaching assistant (TA), we proceeded to hold our casings to the IP54 standard. After testing, the IP54 capabilities of our system's casings have been verified, and our system is safe for outside use.

5.4 Future Work

While we managed to complete the original design of our system, we still feel that there are ways to further improve our project. During our project build we came up with the idea to add in a photocell to control when the lights are on, as well as a battery gauge so the user would know when to replace the batteries. While we managed to integrate the photocell into our final product, we want to update it to be usable under any outside condition instead of just solid daylight or darkness. Our battery gauge needs further development as it currently consumes too much power and is not integratable into our current system. We also hope to add a global positioning system (GPS) alongside our speedometer, as well as make our system completely wireless. The desire for wireless capability is for easier user implementation as well as improved aesthetics.

Appendix A: Requirements and Verifications

Table 1: Requirements and Verifications

Module	Requirements	Verification
Power	**1)** The power system will supply an output current of 20 mA within a +/- 5% error. **(7 pts)**	**1a)** Connect an ammeter to the linear voltage regulator circuit and measure the output current. **1b)** Verify that the output is within an error range of +/- 5%.
Microcontroller	**1)** The microcontroller should be able to simultaneously receive all four sensor signals every operation cycle. **(2 pts for each signal, 8 pts total)**	**1a)** Create a program with the arduino processing tool to simulate the four different sensors connected to the microcontroller. **1b)** Direct program to send multiple inputs within the same operation cycle to microcontroller.
Turn Buttons	**1)** The left turn button should turn on/off the left LED signals when pressed, and the right turn button should turn on/off the right LED signals when pressed. **(3 pts)**	**1a)** Wire up a testing LED to the button and a power source. **1b)** Press the button and check if the LED lights up. If yes, that indicates a proper closed connection. **1c)** Release the button to check for the reversion back to an open circuit. The LED should turn off.
LED Turn Signals	**1)** The LEDs will turn on once a signal from the microcontroller is received. **(2 pts)** **2)** The LEDs will be visible from at least 10 ft away. **(2 pts)**	**1a)** Create a circuit with the LEDs connected to the microcontroller. **1b)** Program the microcontroller to send an alternating on/off signal every other second. **1c)** Verify that the LED's act accordingly **2a)** Measure out a distance of 10 ft. **2b)** Place the LEDs in their casing and connected to a power source at one end of the distance. Have it face the other end of the 10ft. **2c)** Go to the other end of the distance and make sure the LEDs in their casing are clearly visible.
Continued on next page		

Table 1: Requirements and Verifications (continued)

Module	Requirements	Verification
Gyroscope	**1)** The gyroscope should be able to detect an angle change equal or greater to 2° of the starting degree. **(6 pts)**	**1a)** Have the bike with attached gyroscope system stand upright with handlebars in forward resting position. **1b)** Measure angle of 2° from the front wheels position to the right. **1c)** Move the handlebars of the bike past the measured angle and check the value sent to the microcontroller. Verify the angle has changed. **1d**) Repeat step 1b and 1c, except measure an angle of 2° to the left, move the bike, and then verify. **1e**) To check for the change in the up direction, hold the bike so that the front wheel is in the air and measure an angle of 2° above the handle bars. **1f**) Move the handlebars of the bike past the measured angle and check the value sent to the microcontroller. Verify the angle has changed. **1g**) Repeat step 1e and 1f, except measure an angle of 2° below the handlebars, move the bike, and then verify.
Brake Light	**1)** The brake light casing must hold up to the IP54 requirement of being able to withstand water contact for 30 minutes and still work. (**2 points**)	**1a)** Attach brake light to bike. **1b)** From three feet away, spray brake light with medium pressure hose for 30 minutes. **1c)** Once the time limit is up plug brake light into circuit and verify that there is no moisture inside the casing.
Continued on next page		

Table 1: Requirements and Verifications (continued)

Module	Requirements	Verification
Accelerometer	**1)** The accelerometer should be able to detect an acceleration change of at least ¼ mi/h of the bikes actual speed. **(6 pts)**	**1a)** Connect the accelerometer and microcontroller circuit onto the bike. **1b)** While using the phone app Sensor Kinetics, have someone ride the bike, accelerating and decelerating at different times and at different speeds. The times of these changes in speeds, as well as the changes in speed, will be recorded by the app as well as by the microcontroller for the bike. **1c)** After the ride, the data from the microcontroller will be plotted in terms of acceleration/deceleration vs time. **1d**) Convert the data from the app into mi/h and re-plot it. **1e)** Verify using the graph and the converted data from the app, that the accelerometer can detect a change of at least ¼ mi/h.
Magnetic Reed Switch	**1)** The magnetic reed switch must be able to detect four magnet passes within one rotation of the wheel while going at various speeds. **(6 pts)**	**1a)** With reed switch, magnets, and microcontroller attached to the bike, move the bike and count the amount of rotations the back wheel goes through. Multiply this number by four to get the number of times the microcontroller should have "seen" a magnet. **1b)** Verify the microcontroller has read four separate magnet counts per rotation by taking the number of magnets read by the microcontroller and dividing it by the number of rotations that were counted. If this number matches the number calculated in step 1a, then it has been verified. **1c)** Repeat steps 1a and 1b three times, while having the bike move at different speeds. Use this to verify accuracy of the calculations regardless of speed.

Continued on next page

Table 1: Requirements and Verifications (continued)

Module	Requirements	Verification
Speedometer Display	**1)** The speedometer display value should be accurate within 1 mi/h of the actual speed of the bike. **(5 pts)**	**1a)** Time the bike going a distance of 100ft. Use this measurement to find speed in units of mi/h. **1b)** Compare to the value stored in the microcontroller. **1c)** Verify that both values are within 1 mi/h of one another.
	2) Speedometer display value should be seen from 18 inches away both in daytime and at night. **(3 pts)**	**2a)** Hold programmed display 18 inches away from face while outside during the day. **2b)** Verify that data can be read without moving closer, and while taking short glances from below eyeline. **2c)** Repeat outside at night. Verify that data can be read.

Appendix B: Component Cost Breakdown

Table 2: Component Costs

Module	Part	Part Number	Description / Manufacturer	Unit Cost	Quantity	Total
Power Supply	Battery	E91	E91 Alkaline Max AA Battery / Energizer	$1.23	3	$3.69
	Battery Holder	BK03	2-pack 3 x1.5V AA Battery Holder with On Off Switch Cap Lead Wires / Bluewo	$6.50	1	$6.50
	Linear Voltage Regulator	TLV71333PDBVR	IC REG LINEAR 3.3V 150MA SOT23-5 / Texas Instruments	$0.42	1	$0.42
Control Unit	Microcontroller	ATMEGA328P-PU	ATmega Microcontroller IC 8-Bit 20MHz 32KB (16K x 16) FLASH 28-PDIP / Microchip Technology	$2.18	1	$2.18
	Crystals 16 MHz	ABLS-16.000MHZ-B4-T	16.0000MHZ 18PF SMD / Abracon LLC	$0.25	1	$0.25
Sensors	Accelerometer/ Gyroscope	MPU-6050	HiLetgo GY-521 MPU-6050 MPU6050 3 Axis Accelerometer Gyroscope Module	$5.09	1	$5.09
	Reed Switch	59170-1-T-00-D	SWITCH REED SPST-NO 350MA 140V / Littelfuse Inc.	$0.95	1	$0.95

Continued on next page

Table 2: Component Costs (continued)

Module	Part	Part Number	Description / Manufacturer	Unit Cost	Quantity	Total
Sensors	Magnets	8019	MAGNET ROUND NDFEB AXIAL 0.250”Dia X0.250”H	$0.72	8	$5.49
	Photocell	SEN-09088	SparkFun Mini Photocell	$1.50	1	$1.50
Peripherals	LEDs	100F5T-YT-WH-WH	5mm White LED Diode Lights / Chanzon	$0.07	100	$6.67
	Speedometer Display	SSD1306	0.96” Inch Blue I2C IIC Serial 128x64 OLED LCD LED SSD1306 Module / SunFounder	$8.59	1	$8.59
User Interface	Buttons	KS-01Q-01	SWITCH PUSH SPST-NO 0.01A 35V / E-Switch	$0.53	4	$2.12
PCBs	Various	Various	PCBWay	$41	1	$41
Circuit Essentials	Resistors	CFR-25JB-52-10K	10 kOhms ±5% 0.25W, 1/4W Through Hole Resistor Axial Carbon Film / Yageo	$0.10	4	$0.40

Continued on next page

Table 2: Component Costs (continued)

Module	Part	Part Number	Description / Manufacturer	Unit Cost	Quantity	Total
Circuit Essentials	Resistors	CFR-25JB-52-220R	220 Ohms ±5% 0.25W, 1/4W Through Hole Resistor Axial Carbon Film / Yageo	$0.10	10	$1.00
	Capacitors	GRM155R71H103KA88D	10000pF ±10% 50V Ceramic Capacitor X7R 0402 / Murata	$0.10	1	$0.10
		GRM033R71C222KA88D	2200pF ±10% 16V Ceramic Capacitor X7R 0201 / Murata	$0.10	1	$0.10
		CC0603MRX5R5BB226	22µF ±20% 6.3V Ceramic Capacitor X5R 0603 / Yageo	$0.28	1	$0.28
		GRM1555C1H220JA01D	22pF ±5% 50V Ceramic Capacitor C0G, NP0 0402 / Murata	$0.10	2	$0.20
		02016D104KAT2A	0.1µF ±10% 6.3V Ceramic Capacitor X5R 0201 / AVX Corporation	$0.10	2	$0.20
						Total Cost: $90.76

Appendix C: Microcontroller Code

The microcontroller code utilizes several sources and libraries [15] – [18].

```
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>
#include "stdio.h"
#include <MPU6050.h>

#define OLED_RESET 8
Adafruit_SSD1306 display(OLED_RESET);
#define OLED_address 0x3c
MPU6050 mpu;

// Define Brake
const int BRAKE_LED = 7;
const long brakeinterval = 250;           // interval at which to blink (milliseconds)
unsigned long brakeMillis;
unsigned long previousbrakeMillis = 0;    // store last time LED was updated

// Define LEDs
const int ledPinL = 3;                    // Left LED - Pin 3
const int ledPinR = 5;                    // Right LED - Pin 5
int left_button = 2;                      // Left Button - Pin 2
int right_button = 4;                     // Right Button - Pin 4

// Pitch, Roll, & Yaw variables
float yaw = 0;
float yaw2 = 0;
float x = 0;
float y = 0;
float z = 0;
float timeStep = 0.01;                    // timestep used to get values of yaw

// interval at which to LED blink (ms)
const long interval = 500;

// variables for left LED
unsigned long previousMillisL = 0;        // will store last time LED was updated
bool blinking_flagL = 0;                  // 1 means blinking mode, 0 means not blinking
bool already_set_flagL = 0;               // tells if LED is in blinking mode or not:
                                          // 0=already reset blinking_flag, 1=not reset yet
int ledStateL = HIGH;
```

```
bool rollflagL = 0 ;

// variables for right LED
unsigned long previousMillisR = 0;
bool blinking_flagR = 0;
bool already_set_flagR = 0;
int ledStateR = HIGH;
bool rollflagR = 0 ;
int counter = 207;

// Photocell
const float VCC = 3.3;                      // Measured voltage of 3.3V line
const float R_DIV = 1000;                   // Measured resistance of resistor
const float DARK_THRESHOLD = 5000.0;        // Set threshold to turn on LED

const int LIGHT_PIN = A0;                   // Pin connected to photocell output

// Define Speedometer
const int reed = 9;                         // pin connected to read switch
int reedVal;
long mph;
float radius = 7;                           // tire radius (in)
int maxReedCounter = 100;                   // min time (ms) of one rotation for debouncing
int reedCounter;
float pi = 3.14159;
float angle = 90;
float velocity;
float convert = .0568181818;
float W;

void setup(){
   Serial.begin(115200);
   Serial.println("Initialize MPU6050");

   pinMode(ledPinL, OUTPUT);                // initialize led and buttons
   pinMode(left_button, INPUT);             // initialize led and buttons
   pinMode(ledPinR, OUTPUT);                // initialize led and buttons
   pinMode(right_button, INPUT);            // initialize led and buttons
   pinMode(BRAKE_LED, OUTPUT);              // initialize brake light
   pinMode(LIGHT_PIN, INPUT);               // photocell is analog input

   while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_16G))
                                            // assures gyroscope connected
   {
```

```
    Serial.println("MPU not found");
    delay(500);
  }

  mpu.setAccelOffsetX(-2370);                // Calibrate MPU so it provides proper readings
  mpu.setAccelOffsetY(-1954);
  mpu.setAccelOffsetZ(2240);
  mpu.setGyroOffsetX(-37);
  mpu.setGyroOffsetY(35);
  mpu.setGyroOffsetZ(120);
  mpu.setThreshold(3);                       // Set threshold sensitivity. Default 3.
                                             // assign the display the proper I2C location
  display.begin(SSD1306_SWITCHCAPVCC, 0x3c);
  display.clearDisplay();                    // clear screen buffer
  reedCounter = maxReedCounter;              // make sure the reed switch doesn't bounce
  circumference = (2*pi*radius)/4;
  pinMode(reed, INPUT);                      // setup reed switch

                                             // Set up timer so reed switch only reads
                                             // when we want it to
  cli();                                     // stop interrupts
                                             // Timer1 easiest to use
  TCCR1A = 0;                                // set entire TCCR1A register to 0
  TCCR1B = 0;

  TCNT1  = 0;                                // counter = 0
  OCR1A = 1999;                              // = (1/1000) / ((1/(16*10^6))*8) - 1
                                             // we want a 1kHz frequency for our timer
  TCCR1B |= (1 << WGM12);                    // set it to CTC mode
  TCCR1B |= (1 << CS11);                     // Use Prescaler 8
  TIMSK1 |= (1 << OCIE1A);                   // enable timer compare interrupt
  sei();                                     // allow interrupts
}

ISR(TIMER1_COMPA_vect) {                     // Interrupt at freq of 1 kHz to measure reed switch
  reedVal = digitalRead(reed);               // get val of reed switch

  if (reedVal){                              // if reed switch is closed
    if (reedCounter == 0){                   // min time between pulses has passed
                                             // calculate miles per hour
      W = (angle/float(timer));
        velocity = W * radius;
      mph  = velocity * convert;
      timer = 0;                    // reset timer
```

```
        reedCounter = maxReedCounter;          // reset reedCounter
      }
    }
    else{                                      // if reed switch is open
      if (reedCounter > 0){                    // don't let reedCounter go negative
        reedCounter -= 1;                      // decrement reedCounter
      }
    }
    if (timer > 1500)
      mph = 0;     //if no new pulses from reed switch- tire is still, set mph to 0
      velocity = 0;
          W = 0;
                                               // tire is still set mph to 0
}
else{
    timer += 1;                                // increment timer
}
}

void printBattery()
{
    Serial.print(mph);
    display.setTextColor(WHITE);               // set color of OLED, default is white
    display.setTextSize(4);                    // set text size on OLED
    display.setCursor(55,210);                 // position the text on the display
    display.print(mph);
    display.display();
    display.clearDisplay();
}

void getvalues(){                              // function to read the values of the gyroscope

    Vector norm = mpu.readNormalizeGyro();     // Read normalized values
    Vector normAccel = mpu.readNormalizeAccel();
    yaw = yaw + norm.ZAxis * timeStep;         // Don’t reset yaw if no new value has been read
    x = normAccel.XAxis;
    y = normAccel.YAxis;
    z = normAccel.ZAxis;
}
void loop(){
    printBattery();                            // call function to display speedometer values
    getvalues();
    Serial.println();
```

```
getvalues();                                    // get values multiple times incase gyroscope
                                                // changes while function is in its cycle

                                                // Read the ADC, and calculate voltage and
                                                // resistance from it: for Photocell
int lightADC = analogRead(LIGHT_PIN);
if (lightADC > 0)
{
                                                // Use the ADC reading to calculate
                                                // voltage and resistance

   float lightV = lightADC * VCC / 1023.0;
   float lightR = R_DIV * (VCC / lightV - 1.0);

   if (lightR >= DARK_THRESHOLD){
                                                // If resistance of photocell is greater than
                                                // the dark threshold setting, turn the LED on.
      ledStateL = HIGH;
   }
   else{

      ledStateL = LOW;
   }
}

// LEFT BUTTON
digitalWrite(ledPinL, ledStateL);               // turn LEDs ON
                                                // get button state

int button_stateL = digitalRead(left_button);
if(x < -3.85 && y> -2 && z > 10.1){
   digitalWrite(BRAKE_LED, HIGH);
}

brakeMillis = millis();
                                                // save the last time you blinked the LED

if (brakeMillis - previousbrakeMillis >= brakeinterval) {
   previousbrakeMillis = brakeMillis;
   digitalWrite(BRAKE_LED, LOW);
}

                                                // buttons when pressed goes low
```

```
if (  (yaw > 80 ) && (rollflagL == 1) ) {
   yaw = 0;
   yaw2 = 0;
   rollflagL = 0;
   float lightV = lightADC * VCC / 1023.0;
   float lightR = R_DIV * (VCC / lightV - 1.0);

                                          // if dark out, LEDs ON
   if (lightR >= DARK_THRESHOLD){
      ledStateL = HIGH;
   }
   else{
                                          // if light out, LEDs OFF (except for turn signaling)
      ledStateL = LOW;
   }


   if (already_set_flagL == 0 ){
      blinking_flagL = !blinking_flagL;
      already_set_flagL = 1;
   }
}

getvalues();                              // get values multiple times incase gyroscope
                                          // changes while function is in its cycle


if (button_stateL == LOW ){               // buttons when pressed goes low
   yaw = 0;
   yaw2 = 0;
   float lightV = lightADC * VCC / 1023.0;
   float lightR = R_DIV * (VCC / lightV - 1.0);

                                          // if dark out, LEDs ON
   if (lightR >= DARK_THRESHOLD){
      ledStateL = HIGH;
   }
   else{
                                          // if light out, LEDs OFF (except for turn signaling)
      ledStateL = LOW;
   }
   if (already_set_flagL == 0){
      blinking_flagL = !blinking_flagL;
      already_set_flagL = 1;
```

```
   }
}
                                                // buttons when pressed it goes low
if (  (yaw >= 30 ) && blinking_flagL == 1 )  {
   yaw2 = yaw;
   rollflagL = 1;
}

getvalues();                                    // get values multiple times incase gyroscope
                                                // changes while function is in its cycle

                                                // the already_set_flag has not been reset yet
if (button_stateL == HIGH && already_set_flagL == 1) {
   already_set_flagL = 0;
   if (blinking_flagL == 1){
                                          // Blinking Code
      unsigned long currentMillisL = millis();

      if (currentMillisL - previousMillisL >= interval){

                                                // save the last time you blinked the LED

         previousMillisL = currentMillisL;

         if (ledStateL == LOW){
            ledStateL = HIGH;
            digitalWrite(ledPinL, ledStateL);
         }
         else if (ledStateL == HIGH){
            ledStateL = LOW;
            digitalWrite(ledPinL, ledStateL);
         }
      }
   }
}

getvalues();                                    // get values multiple times incase gyroscope
                                                // changes while function is in its cycle

if (button_stateL == HIGH && already_set_flagL == 0){
   if (blinking_flagL == 1){
                                                // Blinking Code
      unsigned long currentMillisL = millis();
```

```
      if (currentMillisL - previousMillisL >= interval){
                                          // save the last time you blinked the LED

         previousMillisL = currentMillisL;
         if (ledStateL == LOW){
            ledStateL = HIGH;
            digitalWrite(ledPinL, ledStateL);
         }
         else if (ledStateL == HIGH){
            ledStateL = LOW;
            digitalWrite(ledPinL, ledStateL);
         }
      }
   }
}

getvalues();                              // get values multiple times incase gyroscope
                                          // changes while function is in its cycle

lightADC = analogRead(LIGHT_PIN);
if (lightADC > 0){
                                          // Use the ADC reading to calculate
                                          // voltage and resistance
   float lightV = lightADC * VCC / 1023.0;
   float lightR = R_DIV * (VCC / lightV - 1.0);

                                          // If resistance of photocell is greater than the
                                          // dark threshold setting, turn the LED on.
   if (lightR >= DARK_THRESHOLD){
      ledStateR = HIGH;
   }
   else{
      ledStateR = LOW;
   }
}

// RIGHT BUTTON
digitalWrite(ledPinR, ledStateR);                 // turn LEDs ON
                                          // get button state

int button_stateR = digitalRead(right_button);

                                          // buttons when pressed are low
if (  (yaw < -80 ) && rollflagR == 1 ){
```

```
   float  lightV = lightADC * VCC / 1023.0;
   float  lightR = R_DIV * (VCC / lightV - 1.0);
   yaw = 0;
   yaw2= 0;
   rollflagR = 0;

   if (lightR >= DARK_THRESHOLD){
      ledStateL = HIGH;
   }
   else{
      ledStateL = LOW;
   }
   if (already_set_flagR == 0 ){
      blinking_flagR = !blinking_flagR;
      already_set_flagR = 1;
   }
}

getvalues();                                  // get values multiple times incase gyroscope
                                              // changes while function is in its cycle

if (button_stateR == LOW){
   float lightV = lightADC * VCC / 1023.0;
   float lightR = R_DIV * (VCC / lightV - 1.0);
   yaw = 0 ;
   yaw2 = 0;

   if (lightR >= DARK_THRESHOLD){
      ledStateL = HIGH;
   }
   else{
      ledStateL = LOW;
   }
   if (already_set_flagR == 0){
      blinking_flagR = !blinking_flagR;
      already_set_flagR = 1;
   }
}


if(blinking_flagR == 1 && (yaw <= -30) ){
   yaw2 = yaw ;
   rollflagR = 1;
}
```

```
                                                // the already_set_flag has not been reset yet

if (button_stateR == HIGH && already_set_flagR == 1)
{
   already_set_flagR = 0;

   if (blinking_flagR == 1){
      unsigned long currentMillisR = millis();
      if (currentMillisR - previousMillisR >= interval){
                                                // save the last time you blinked the LED
         previousMillisR = currentMillisR;
         if (ledStateR == LOW){
            ledStateR = HIGH;
            digitalWrite(ledPinR, ledStateR);
         }
         else if (ledStateR == HIGH){
            ledStateR = LOW;
            digitalWrite(ledPinR, ledStateR);
         }
      }
   }
}

                                                // the already_set_flag has been reset
if (button_stateR == HIGH && already_set_flagR == 0)
{
   if (blinking_flagR == 1){
      unsigned long currentMillisR = millis();
      if (currentMillisR - previousMillisR >= interval){
                                                // save the last time you blinked the LED

         previousMillisR = currentMillisR;
         if (ledStateR == LOW){
            ledStateR = HIGH;
            digitalWrite(ledPinR, ledStateR);
         }
         else if (ledStateR == HIGH){
            ledStateR = LOW;
            digitalWrite(ledPinR, ledStateR);
         }
      }
   }
}
```

References

[1] N. Scarborough, “Number of cyclists in the USA 2017 | Statistic”, Statista, 2017. [Online]. Available: https://www.statista.com/statistics/227415/number-of-cyclists-andbike-riders-usa/

[2] C. Facts, “Topic: Cycling”, www.statista.com, 2017. [Online]. Available: https://www.statista.com/topics/1686/cycling/

[3] ENERGIZER NO. E91. (2017). [ebook] Eveready Data Co, pp.1-2. Available: http://www.rosebatteries.com/pdfs/e91.pdf

[4] "Capacitor-Free, 150-mA, LDO Regulator w/ Foldback Current Limit for Portable Dev (Rev. E)", *Ti.com*. [Online]. Available: http://www.ti.com/lit/ds/symlink/tlv713p.pdf

[5] "150 mA Low-Noise LDO Regulator", *Biakom.com*, [Online]. Available: http://www.biakom.com/pdf/mic5205-271941.pdf

[6] "ATMEL 8-Bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash Datasheet", *Atmel.com*, 2017. [Online]. Available: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

[7] "100pcs / pack 5mm White Ultra Bright 8000mcd Transparent 5mm Light Emitting Diode LED Lamp 5 mm (DIP 3 V 6000K Ultrabright)-in Diodes from Electronic Components & Supplies on Aliexpress.com | Alibaba Group", *aliexpress.com*. [Online]. Available: https://www.aliexpress.com/item/100PCS-White-Color-5mm-Transparent-Water-ClearRound-Super-Bright-LED-DIODE/32266372205.html?spm=2114.search0104.3.1.LHehKB&ws_ab_test=searchweb0_0,searchweb201602_3_10152_10065_10151_10130_10068_10344_5560015_10342_10343_10340_10341_10307_10060_10155_10154_10056_10055_10054_5370015_10059_10534_10533_10532_100031_10099_10338_10339_10103_10102_10052_10053_10107_10050_10142_10051_10324_10325_10084_513_10083_10080_10082_10081_10178_10110_5590015_10111_10112_10113_10114_143_10312_10313_10314_5570015_10078_10079_10073,searchweb201603_25,ppcSwitch_5&btsid=e35213fb-a177-4fd6-bb9f-8c6ed2011ee9&algo_expid=6ec7878b-6eb2-44eb-9c2a-84f743c2107b-0&algo_pvid=6ec7878b-6eb2-44eb-9c2a-84f743c2107b

[8] "MPU-6050 | TDK", *Invensense.com*, 2017. [Online]. Available: https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

[9] "224-5205-01 3M | Connectors, Interconnects | DigiKey", *Digikey.com*, 2017. [Online]. Available: https://www.digikey.com/product-detail/en/3m/224-5205-01/3M10827-ND/3133918

[10] Adafruit Industries, "Monochrome 0.96 128x64 OLED graphic display ID: 326", *Adafruit.com*, 2017. [Online]. Available: https://www.adafruit.com/product/326

[11] “GPS Speedometer and Odometer”, *Play.google.com.* [Online]. Available: https://play.google.com/store/apps/details?id=com.coolniks.niksgps&hl=en

[12] “IEEE IEEE Code of Ethics”, *ieee.org*, 2017. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html

[13] “IP67 vs IPG8: Waterproof IP ratings explained | Trusted Reviews”, *Trusted Reviews*, 2017 [Online]. Available: http://www.trustedreviews.com/opinion/what_is_1p69-ip-ratings_explained-2947135

[14] "IP Rated Enclosures Explained", *Enclosurecompany.com*. [Online]. Available: http://www.enclosurecompany.com/ip-ratings-explained.php

[15] "Arduino - Blink Without Delay", *Arduino.cc*, 2015. [Online]. Available: https://www.arduino.cc/en/tutorial/BlinkWithoutDelay

[16] "Photocell Hookup Guide", *Learn.sparkfun.com*, 2016. [Online]. Available: https://learn.sparkfun.com/tutorials/photocell-hookup-guide#example-program

[17] "MPU6050 Triple Axis Gyroscope & Accelerometer Arduino Library", *GitHub*, 2017. [Online]. Available: https://github.com/jarzebski/Arduino-MPU6050

[18] "adafrSSD1306 oled driver library for 'monochrome' 128x64 and 128x32 OLEDs!", *GitHub*, 2016. [Online]. Available: https://github.com/adafruit/Adafruit_SSD1306