XF04: Smart Bike Using Regenerative Energy Harvesting

by

X

Computer/Electrical Engineering X Design Project

X, 2024

Acknowledgements

We would like to acknowledge our faculty and specifically our faculty lab coordinator, - for his assistance and mentorship throughout this project. Additionally, we would like to thank the - team for providing its facilities for storage, manufacturing and testing of our project. Finally, we would like to thank - for 3D printing services and for providing access to various components used in our project.

Certification of Authorship

We hereby certify that we are the authors of this document and that any assistance we received in its preparation is fully acknowledged and disclosed in the document. All sources used throughout the document have been cited in section X - References of the report. This report complies with the rules set out by Policy 60 of -.

Table of Contents

Abstract

This project focuses on designing an energy-harvesting smart bike with advanced safety and health monitoring features. The bike harnesses kinetic energy from pedaling, and uses it to power its components. Key features include: speed measurement, inclination detection, rear vehicle proximity alerts, vital signs monitoring, Bluetooth communication, GPS tracking, and turn-indicating LEDs. Furthermore, collected data is transmitted wirelessly to a companion mobile app for real-time performance and location monitoring.

This report outlines the specific motivations and objectives of the project. The theory supporting design decisions and reasoning for the design itself are also discussed. Design decisions are made based on overall team consultation to ensure that each subsystem works with one another and does not interfere. Availability of components and the ability to successfully complete the requirements are also considered when making decisions. Alternate solutions that have been considered, but did not become final choices due to justified reasoning are also discussed.

The manufacturing process involved several redesigns of some subsystems of the project, but the final result complied with the requirements of the project. The materials and components used to incorporate each subsystem have been tabulated, quantized and priced to have a clear understanding of the resources required to implement this project.

In order to assess the performance of the subsystems individually, and the project as a whole, various repeatable testing procedures have been developed and performed. The results of these tests determined if the performance of the systems tested were satisfactory and subsequent modifications were made if required. The performance of the systems is assessed and any undesirable results are also discussed.

It can be concluded that the project performed up to the expectations and met the requirements. All subsystems functioned as a single product capable of being a Smart bike that assists and provides convenience to the user. The combination of energy efficiency, safety features, health monitoring, and connectivity through the mobile app creates a holistic and innovative solution for modern urban biking within the constraints of sustainability and safety.

I - Introduction & Background

This report serves the purpose of defining objectives and background, explaining the design process, theory, testing procedures and accomplishments of the XF04: Smart Bike Using Regenerative Energy Harvesting capstone project.

In recent years, the popularity of electric bicycles has transformed the landscape of urban commuting, offering riders an array of benefits and features. Electric bikes, or e-bikes, have become synonymous with innovation, leveraging sophisticated systems to monitor, record, and enhance the overall riding experience. The market of such products has grown significantly, expected to be worth around $120 billion globally by 2030 [1]. As such, developments in this field offer much potential for opportunities in creative designs and solutions.

The XF04 project is positioned at the forefront of this evolution, aiming to replicate the advanced features of modern e-bikes and integrate them seamlessly onto a conventional bicycle. Central to this endeavor is incorporating systems that encompass the monitoring of various parameters through sensors, GPS technology, Bluetooth connectivity, phone integration, and regenerative energy harvesting. By converging these technologies onto a standard bicycle, the objective is to create a Smart Bike that mirrors the functionalities of its electric counterparts and enhances the cycling experience by introducing a blend of connectivity, safety, and environmental consciousness.

Throughout the fall semester, the focus of the project was mostly directed towards the development of the designs in the project. Different design options were analyzed and preferable solutions were determined. From the start of the winter semester the actual manufacturing and implementation phase of the project was initiated. At the start, a set of milestones was developed and each had to be met by certain points of the project. At each milestone the completed progress was demonstrated to the FLC to ensure that the project was on track and that the required work was being completed. By the third milestone, the majority of the requirements were functional with final changes implemented by the last milestone and demonstration.

Work on the project was distributed across the four members of the team. Student A had the task of working on the energy harvesting system (EHS), Student B worked on the various sensors and their interfacing to the phone app, Student C implemented the GPS functionality through a sensor and on the phone app, and Student D implemented the Bluetooth communication and UI design of the phone app. The clear distribution of responsibilities allowed for each member to work independently while communicating with the rest of the team on weekly meetings to determine completed progress, issues, and future steps.

This report will delve into the intricacies of the project, offering insights into the design choices, methodologies, and the completed product. By aligning the team's efforts with the transformative trends in urban mobility, the XF04 project endeavors to set new benchmarks in smart, sustainable transportation.

II - Objectives

This chapter will outline the objectives of the XF04 capstone project. The overall project objectives and specific requirements will be outlined.

Overall Project Objectives:

To design and implement an energy-harvesting smart bike with advanced safety and health monitoring features. The team is to collaborate and distribute the workload to meet all the design requirements within the project timeline. The project should be managed in a professional manner with effective communication between project managers, the FLC and the rest of the team. Throughout the project, weekly team meetings should be attended and meeting minutes submitted. Likewise, milestones are to be presented to the FLC to demonstrate team progress and to ensure that the project is on track.

Design Specifications:

1. Harvest energy from the kinetic energy of pedaling the wheels.
2. Measure the speed of the biker and the amount of inclination.
3. Alert the rider about any approaching vehicle from behind
4. Keep track of the user's vitals with sensors.
5. Incorporate a Bluetooth phone speaker for hands-free communication.
6. Have a GPS module to keep track of the travel log.
7. Have left/right turn indicating LEDs
8. Data collected by the sensors to be then transmitted wirelessly to a companion mobile app which shall display real time information about the bike's performance and location.

III - Theory and Design

This chapter will outline the theory behind various design decisions and justifications, along with how the various systems on the project are implemented in order to meet the requirements. There are five subsections included, describing the energy harvesting system (EHS), sensors, GPS, Bluetooth and app development.

Energy Harvesting System (EHS):

The requirements were determined based on the requirements of the overall project and the requirements of the other systems on the bicycle. The requirements of the energy harvesting system are outlined as the following:

- Ability to provide enough power to charge the battery on the bicycle
- Provide constant and stable power to the other systems on the bicycle
- Output the voltages that are required by the other systems (3.3V and 5V)
- Enable charging of the battery when the bicycle is in motion and use battery power when insufficient power supplied from the wheel rotation
- Communicate charge and operating state data to the main controller

Based on the requirements, an overall concept was developed. The main block of the system was chosen as a 6V SLA battery, as it is capable of providing the required power to all the operating systems and is quite affordable. Additionally, a lead-acid battery is quite forgiving to different variations in charging, which cannot be said about other types of batteries. This battery would constantly power the systems of the bicycle and get charged when power from the wheels gets supplied. Electric power would be generated by rotating a DC motor with a set of gears connected to the rear wheel of the bicycle. A 12V motor was chosen for its compatibility , and as it meets all the requirements. This motor also offered convenient mounting options and was available for a good price. Buck and boost converters were used throughout the energy harvesting system to have stable voltage levels at outputs, even if the inputs would vary. Finally, the controller of the energy harvesting system was chosen as an Arduino Nano that would monitor the output voltage from the motor/generator and enable the connection to the battery to allow charging once sufficient voltage is present. The various connections would be created through MOSFETs that would be controlled by the Arduino through an OP-AMP buffer.

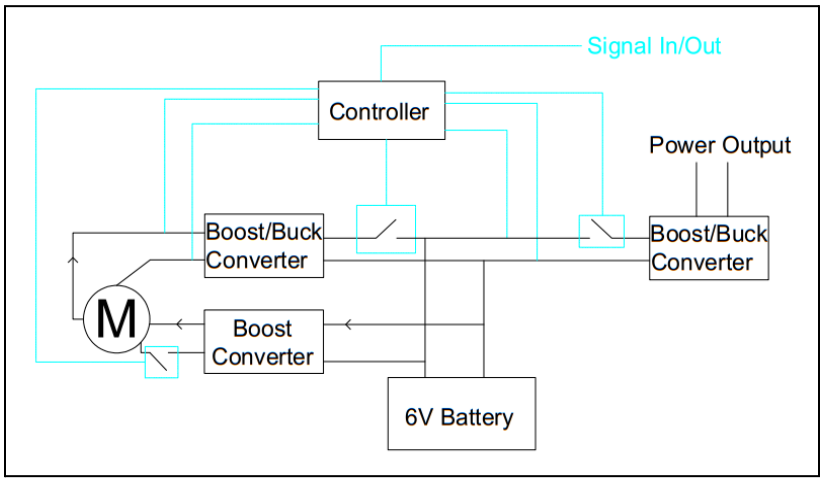


Figure 3.1: Energy Harvesting System Layout.

A large portion of the design process for the EHS was the design and fabrication of the required mounting and gearing solutions. Once the bicycle was selected, measurements of it were taken to design the required parts. The size of the wheel was measured to determine how fast it would rotate at maximum speed and the gearing was sized accordingly. Gearing calculations can be found at the end of the report in Appendix A. Mounts for the motor, the gears, and the gears themselves were designed in Solidworks (shown in Figure 3.2) and will be fabricated through 3D printing.

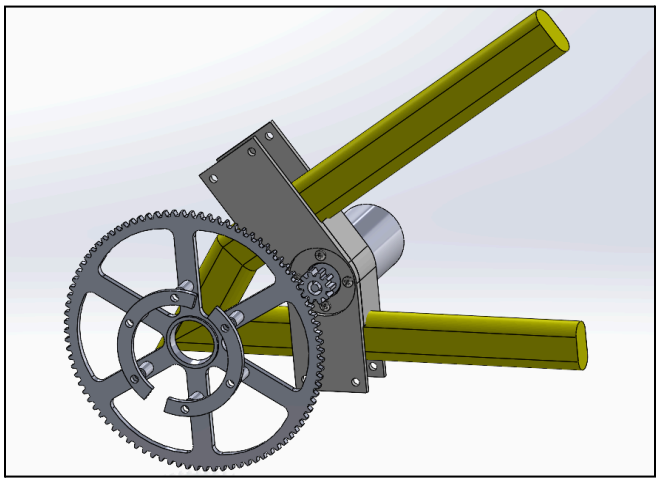
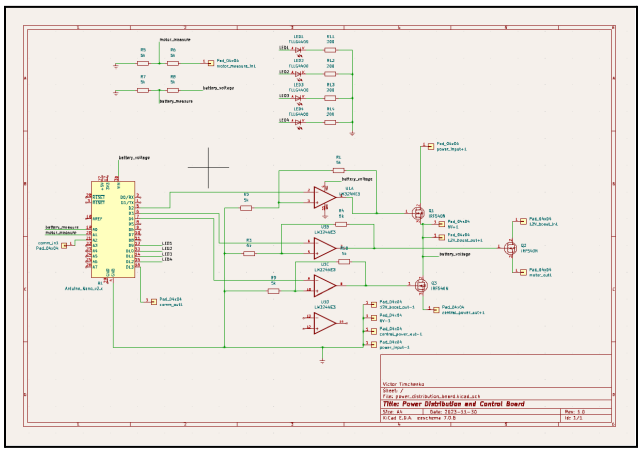


Figure 3.2: Solidworks Model of the Gearing and Mounting System.

The controller was designed using components that met the requirements and that were easily available. These involved MOSFETs, an OP-AMP, resistors and an Arduino Nano. A schematic in Figure 3.3 was designed in KiCAD to have a clear understanding of how all the components interact.



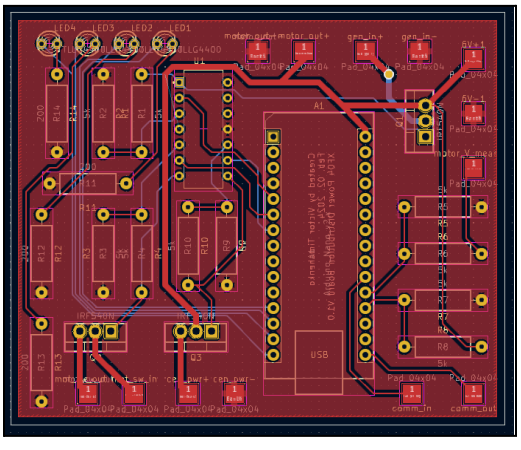


Figure 3.3: Power Distribution and Control Board Schematic and PCB Layout.

Sensors Components:

Hardware

The main component used in this project is the Arduino UNO board with Arduino-based sensors. The 5V pin from the Arduino provides about 1 Amps. The main components include:

- WS2812B (neo pixel) with a 470-ohm resistor to protect the LEDs if something were to happen to the input signal pin. (digital pin 9)
- The A3144 hall effect sensor gives a digital output that is low when a magnetic field is near, otherwise, it outputs a high. The sensor also needs a 10k ohm resistor for more accurate measurements. (digital pin 2)
- 3 HC-SR04 ultrasonic sonars. (digital pins from 3-8, 11)
- the temperature sensor TMP36G which connects to analog input A0.
- 2 Tact switches with 100-ohm resistors (digital pin 6, and pin 10)
- a pulse sensor(analog pin A1)
- MPU6050, a gyroscope that outputs roll pitch and yaw angles (with pins SCL, and SCA)) [2][3][4][5]

Software

The provided Arduino code controls a sophisticated smart bike system by seamlessly integrating various sensors and functionalities. The majority of the sensors have a library that makes it easier to use. Specifically, it incorporates a Pulse Sensor to monitor the user's heart rate, NeoPixels for LED turn signaling, ultrasonic sensors to measure distances, and an analog temperature sensor for environmental sensing. In the setup section, A0 is declared as input for the temperature sensor followed by the trig and echo pins being output and input. The strip object was created using the library detailing the number of pixels, pins, and type which is preset. The pulse sensor pin is set up using the library as well as a threshold. The temperature sensor reads the data and adjusts such that it maps from 0-1023(its input) to -40- 125C. The oncoming vehicles can be detected by using 3 ultrasonic sensors. With trig and echo pins that calculate the distance based on how far it is using sound to bounce off the vehicle. Specifically, the ultrasonic sensors work by sending a signal using the trig pin, and the echo pin will return the time it took to go and bounce back. Dividing this value by 29.155 us/cm and 2 gives us the distance in centimeters. The same procedure was done for the other 2 sensors. The pulse works by calling the get beats per minute function. Two push buttons simulate left and right turn signals, triggering corresponding NeoPixel indicators. To show the LED the *fill* function was used, along with a show function. Additionally, the code calculates the bike's revolutions per minute (RPM) using interrupt-based programming. Furthermore, an MPU6050 will use a premade library that works with a 3-axis accelerometer and a 3-axis gyroscope for orientation data. The turn signals use 40 programmable RGB neopixels. Moreover, the left and right buttons determine the left, right, or stationary signals. The buttons are configured such that if pressed it sends a 0 or LOW. This is used in the if statement such that the right half (signals) turns on and off 5 times using an indicator function. The pixels will take about 1 A and 470 Ohm resistors. The diagram below (Figure 3.4) shows the pin configurations for almost all the sensors.

Overall, the basic examples provided by the library helped set out the basics except for the hall effect sensor. The Hall effect sensor works by measuring the number of times a magnet moves in a circle along the circumference of the bike tire. The speed can be measured by angular

velocity times the circumference or in coding terms, the number of times it detects within the time period (1 second), times the circumference. The interrupt is then enabled and the pin is attached to the function that increases the count. A timer function millis is used to calculate the elapsed time.

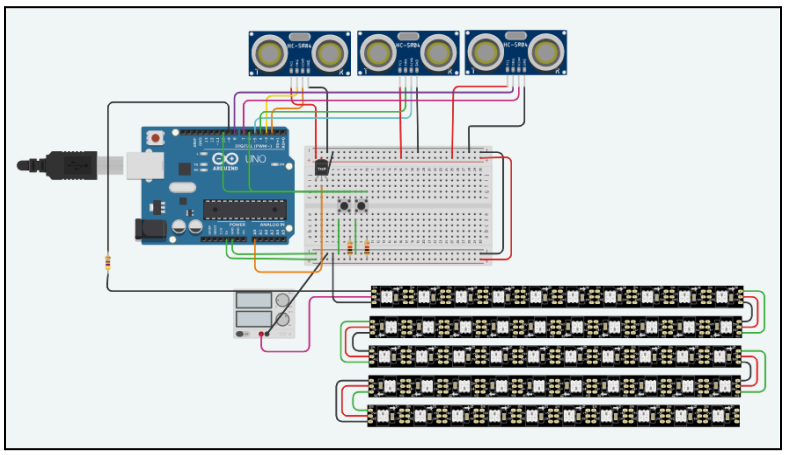


Figure 3.4: Arduino Sensor Simulation in TinkerCAD. [6]

Global Positioning System(GPS):

A requirement of the smart bike project includes the addition of a global positioning system (GPS) for determining position as the bike is transported around by an individual. The system had some requirements that were necessary for integration with other teammates' components and the microcontroller that was chosen (Arduino UNO). The GPS system module requirements and considerations follow:

- Compatibility with Arduino UNO
- 5V or 3.3V input voltage
- TX and RX pins
- Ground pin
- Basic GPS functionality (latitude and longitude)
- Reasonable price
- Live location updates
- Route planning and ability to readout directions to the user for hands free navigation

The development methods for the GPS device involve the following steps: Students begin with researching the basics of GPS technology such as understanding how GPS receivers work, and satellites (GPS and GLONASS, etc). Identification of the specific requirements for integrating GPS into the project continues (accuracy, update rate, power consumption). A clear and detailed GPS specification document that outlines the required functionalities, performance metrics, and any restrictions is developed. Next, based on the research and requirements, the team makes a decision on the GPS hardware that will be used. The component is then ordered and prototyping the system starts with the creation of a code for the Arduino. Expected results include the printing out of current latitude and longitude.

The GPS device for the project is designed to meet the specifications mentioned above to ensure accurate and reliable GPS tracking functionality. A couple of different options are considered for use. The first option includes the GPS module by Ublox called the NEO-6M. The

second option includes the use of the built-in GPS functionality of the android phone using cellular or Wi-Fi connectivity.

Key Features

Starting with key design specifications for the NEO-6M. These include utilizing the NEO-6M GPS module, chosen for its compatibility with the Arduino UNO and its proven reliability in providing accurate location data. This is one of three different GPS modules considered and is chosen due to being a solid middle ground in terms of reliability and price. The microcontroller platform selected for implementation is the Arduino UNO. Its versatility and easy integration make it a great choice for use with the NEO-6M GPS module. This integration involved considerations for physical mounting on the bike, weather resistance, and a minimal impact on the overall bike design. Also as stated before, the primary function of the device is to enable GPS tracking, allowing users to monitor and record the bike's location during rides accurately. Finally, by considering the limitations of power sources on a bike and from the Arduino, the design must prioritize power efficiency to ensure the module works as intended and to prolong its use for as long as possible.

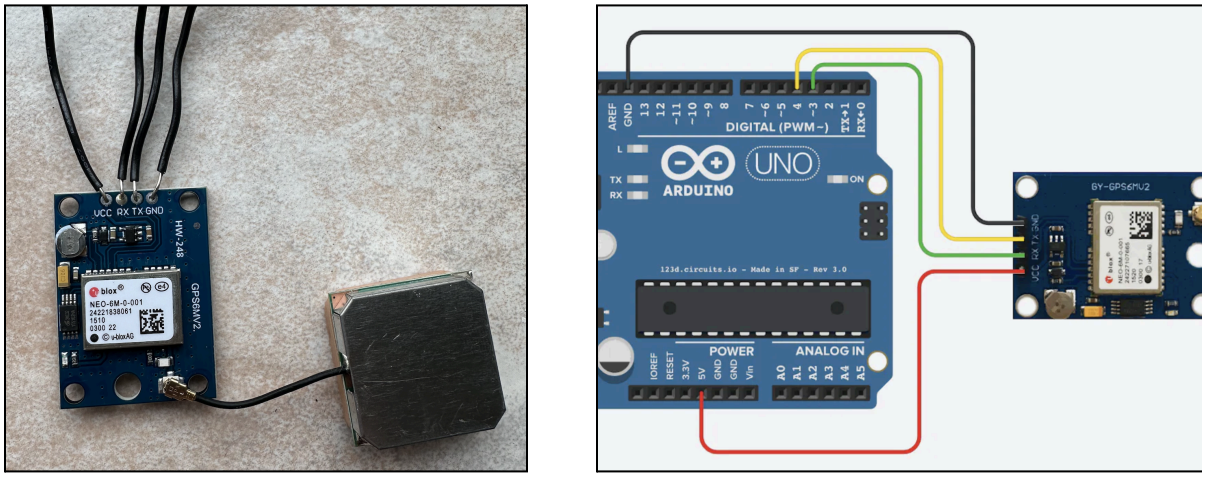


Figure 3.5: NEO-6M Module (Left) and Connection to Arduino UNO (Right).

The second option utilizes the Android mobile phone that would eventually house the app. Majority of smartphones in today's world are able to access their location using wifi or cellular networks. Using these, the maps feature would be implemented using the android phone of choice, a Google Pixel.

Using the Google Pixel for all required GPS functionality would provide the group with accurate and most importantly, consistent location and route planning. To get this function working properly, only app related coding and operations were required. A new *DirectionsActivity.java* and *activity_directions.xml* files would need to be implemented which in turn created a button to access the 'Linked Map' which directs the user to a screen depicting a map feature that automatically plots current location and displacement in live time with a red marker and then uses current location as the starting point for nay directions required by the user. Location was accessed through the phone's wifi. The only operation the user must do is type in their choice of destination. The app will redirect to Google Maps route planning and the app will

direct the user to the destination on-screen and via voice through ‘Google Assistant’ for hands free directions.

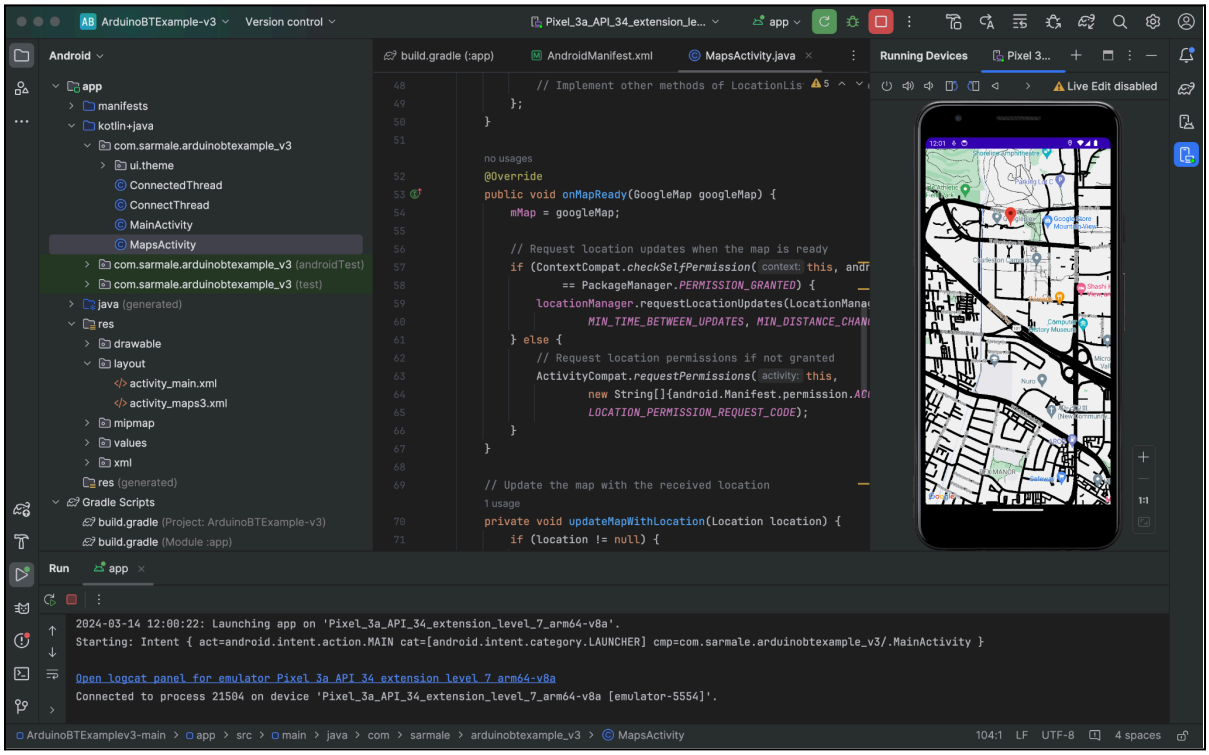


Figure 3.6: Creation of Location Updates on App.

By following this methodology, the development process targets a robust and reliable GPS system that easily integrates with a smart bike, as well as meeting the specified design requirements and overcoming the challenges that were to arise.

Bluetooth Connection:

The HC-05 Bluetooth module is essential in this smart bicycle project because it enables wireless communication between the Arduino and Android devices. By using the HC-05, all sensors on the bicycle can transmit real-time data, such as heartbeat, temperature, and humidity, to the Android app, allowing for dynamic monitoring and analysis. This wireless connectivity enhances user experience by eliminating the need for physical connections and enabling seamless integration of sensor data with the mobile app, providing a more flexible and user-friendly smart bicycle system.

Connection between HC-05 and Arduino UNO board/ power supply:

- VCC to 3.3V power Voltage Supply
- GND to Common Ground
- TXD and RXD to RX and TX of Arduino UNO board
- EN to VCC (to enable the module)

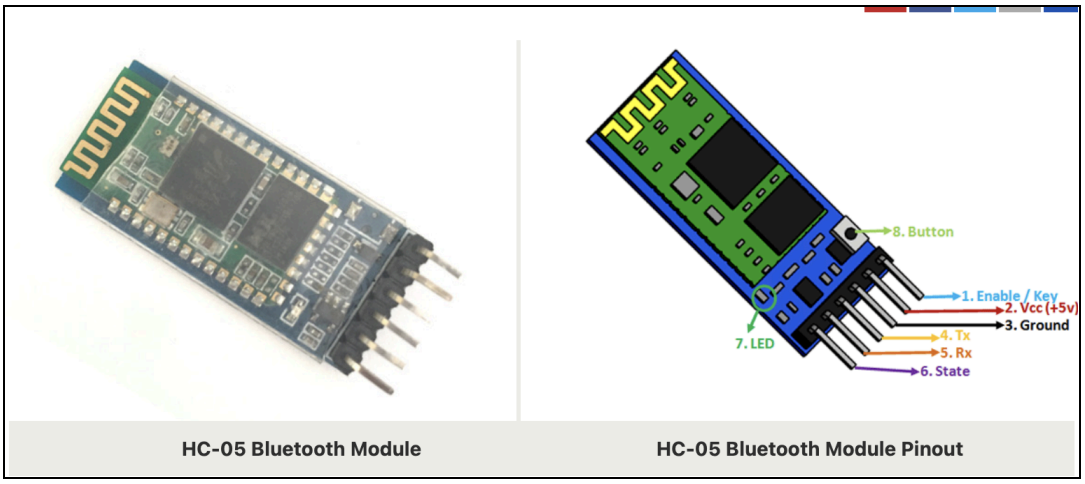


Figure 3.7 : HC-05 Wireless Bluetooth Model.

Configuration:

- The HC-05 can be configured using the AT command. Set the communication mode, baud rate, and other parameters using Arduino sketch or a dedicated configuration tool.

Key Features:

After gathering all the requirements from all the components including sensors and GPS module, it was in the best interest to choose the HC-05 as the component to communicate with the mobile device because of the following features:

- Bluetooth version: 2.0 - suitable for various applications, especially for the simple app builder that is planned to be built through Android Studio.
- Serial Communication: it provides a simple serial communication interface, allowing devices to communicate using UART (Universal Asynchronous Receiver - Transmitter) protocol.
- Range: it offers a range for Bluetooth communication, typically around 10 meters or more, depending on the environment, and since the mobile phone will be mounted on the bike, it is safe to say that HC-05 can offer a strong signal connection between the Arduino UNO board and the mobile phone application.

The intended physical implementation of the Bluetooth module is shown below in Figure 3.8. The design was developed and simulated using online simulation software.

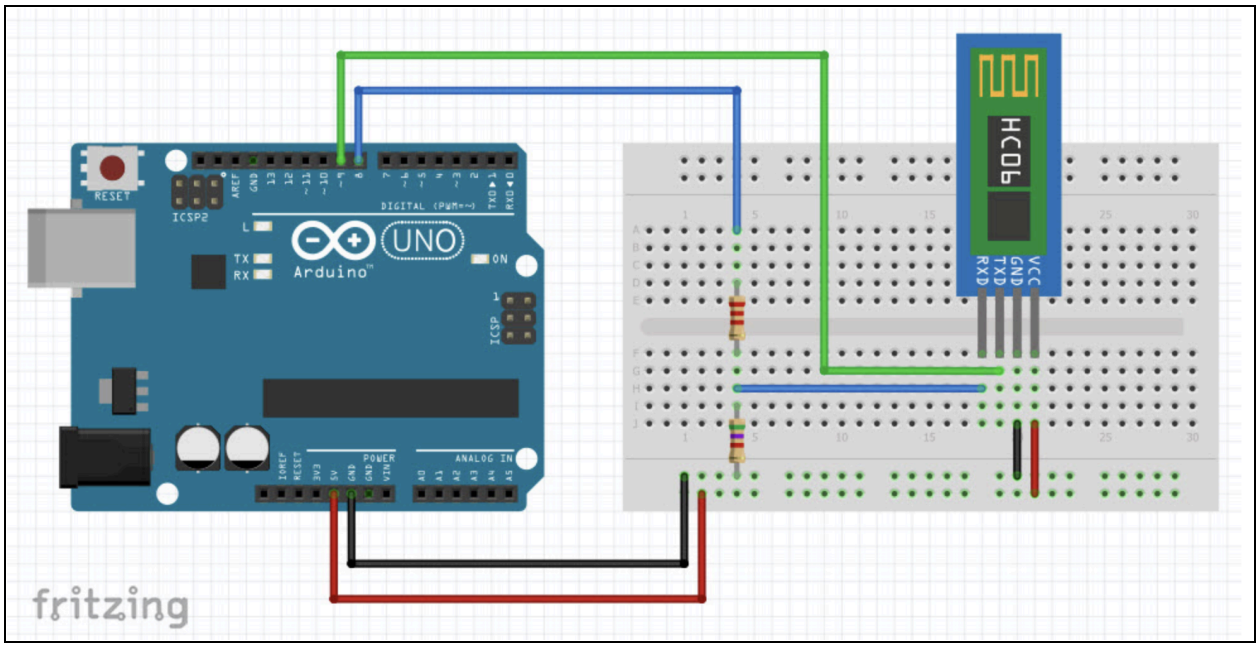


Figure 3.8: Physical Implementation and connection with the Arduino UNO board. [6]

APP (Android App Development):

The app will process information by setting up the Bluetooth connection. This process includes editing the manifest file with specific permissions. The main file will call the connect thread which will connect to the HC-05 bluetooth. This socket will then be used later in the code. The main code then calls *ConnectThread* and *ConnectedThread* to update the UI. The connect and connected threads simply handle the connection and the other errors or exceptions. The structure can be seen in the below figure. Note that the majority of the code involves creating catch statements and protocols to properly establish a connection. Moreover, there are specific UUID and password requirements for the Bluetooth module. The HC-05 name is used to search and connect to it from the main thread.

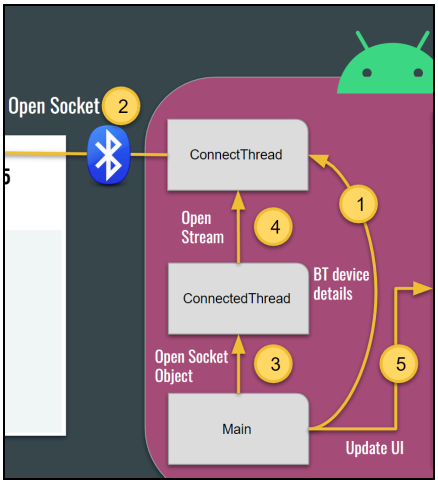


Figure 3.9: App Development.

The main function begins by initializing UI elements and Bluetooth-related objects in the onCreate() method. The application utilizes a Handler to update the UI, Observable from RxAndroid for asynchronous Bluetooth connection handling, and separate threads (*ConnectThread* and *ConnectedThread*) for establishing and managing the Bluetooth connection. Buttons trigger actions such as connecting to the Arduino device and searching for paired Bluetooth devices. Other interactive UI elements use the same strategy. The majority of the text UI uses the input stream to get access to the messages.

The code employs RxAndroid to emit data (to UI in the main thread) received from the Arduino asynchronously, ensuring smooth interaction with the UI. Additionally, it checks for Bluetooth permissions and requests them if necessary. The app can also read multiple data and display it simultaneously. This is done by processing all the information and separating them by the start of the messages. Using simple Java string if statements the separation can be made. Furthermore, a while loop exists within the input stream that periodically updates the data. Within the while loop are the if statements that call the handler with an identifier to separate the messages in the main thread where ui is an updated case statements that handle the different cases and make the appropriate text changes. This is done using a handler instead of the RxAndroid library.

Figure 3.10 shows the general layout which will be built on top of it. The features include a google map for GPS, a UI that shows speed, and warning signs for tilt angle. There is also an indicator of incoming vehicles from behind the bike. Note that the actual text will be directly taken from the Arduino serial print line.

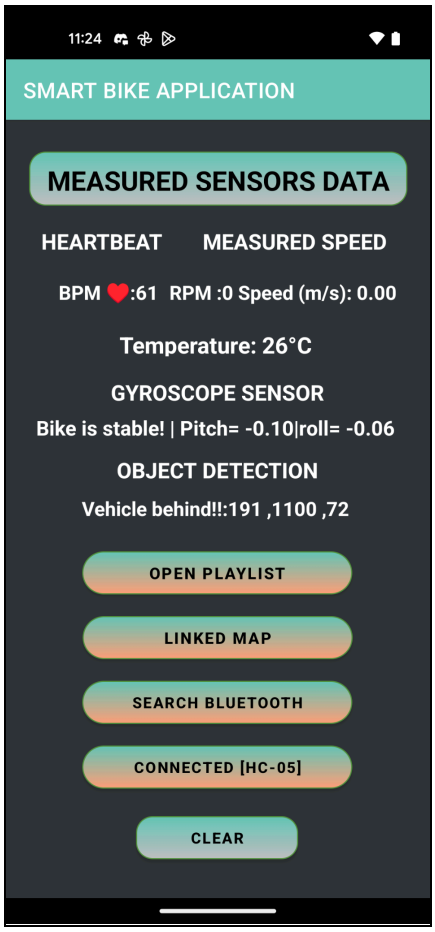


Figure 3.10: App Interface.

The current chapter outlined the theory considerations and design choices involved in the creation of this project. Each section outlined a major portion of the design.

IV - Alternative Designs

This chapter outlines alternate designs that have been considered as different options for the current solutions. Reasons why these designs were not chosen are also discussed. There are five subsections, discussing the energy harvesting system, sensors, GPS, Bluetooth and app development.

Energy Harvesting System:

Several different approaches were assessed in various parts of the energy harvesting system. Beginning with the energy generation, at the start of the course, a system consisting of magnets and copper coils was investigated as an alternative to the current design. This system would involve attaching magnets with alternating polarities to the spokes of the bicycle wheel and having several coil windings attached next to the wheel. The magnets would spin past the coils and induce an alternating electric current that would then be used to supply power to the other systems of the bicycle [7]. A benefit that this system would provide over the current setup is that a physical connection would not be required for this system to work, thus it would be quieter and cleaner, compared to the current gearing solution. Unfortunately, this design would also be more complicated, expensive and difficult to implement. All the additional magnets and coils would be more expensive than the current DC motor. Additionally, significant testing, and almost certainly redesigns would need to be done in order to create a setup that would work and supply sufficient power This would also take considerable resources that may not be available. Finally, a rectification circuit would need to be incorporated in order to stabilize the current that would be outputted into a constant state that could be used to charge the battery. This design could have worked, but its incorporation would be significantly more challenging and uncertain than the current setup.

An alternate approach was also assessed for the gearing required to provide the rotational force for the motor/generator. There are examples of old school bicycles using a small generator that used the wheel tread as a point of contact to generate rotational force [8]. Something similar could have been implemented in the project as well, and this would have required less gearing than the current setup. The reason this design was not pursued further is that it was decided that there might not be enough grip between the bicycle tire and the generator gear. This would also have limited the additional driving ability that is currently implemented on the bike.

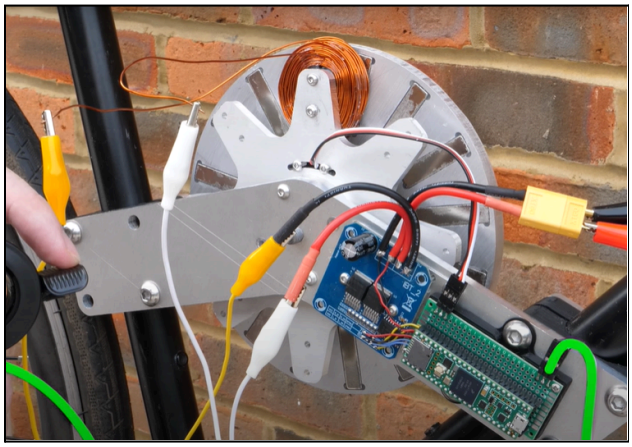


Figure 4.1: Potential Alternate EHS Concept [6] and Gearing Setups [7].

Different types of motors were also looked at when selecting a viable option. Key characteristics that were reviewed were the sizing, type of electric motor, operating parameters and physical dimensions. An induction motor could have been selected but this would have required additional circuitry in order to convert the developed AC into DC, which would complicate the design and increase the cost. DC motors with other voltage ratings were reviewed, but out of the available options, only the 12V variants seemed to be the best choice, as they provided a high enough voltage to charge the battery and had lower speed ratings. A lot of the other options also had quite limited mounting capabilities, not having any mounting holes like those available in the selected option. Finally, the output shaft of the motor had to be large enough to attach a self-designed gear to, and most alternate variants did not meet this requirement. The selected motor met all the requirements outlined previously.

In terms of the control portion of the EHS, it was considered to design the EHS control circuit to only meet the requirements for the project; to only support energy generation. This would have made the circuit slightly simpler, cheaper and it would perform as required. Driving capability was incorporated into it because this addition required only minimal changes to the actual circuit and would result in a drastic increase in functionality, allowing for both charging and driving capabilities.

Sensors:

Many sensors are incorporated into the Arduino board to measure speed, vehicles, vitals, inclination of the bike, as well as left/right turn signals. The design difference comes from different ways to implement the sensors. Primarily, the gyroscope (MPU6050) initially incorporated a manual calculation of all the data and used the wire library to extract raw data. It uses filters to mark out certain errors along with other errors caused by setup. Majority of the number used in the calculation comes from factory made settings [9].

The design then had to be changed since manually setting the error values took too long. A more stable approach was found using a premade library that does the calculation and the calibrating automatically.

The hall effect sensor had to be revised as the interrupt pin was set incorrectly as only pins 2 and 3 support interrupts. The code also had to be revised and debugged according to the actual circuit. The initial design involved using a capacitor to create a rising circuit where it failed to perform accurately. The change was made to be a falling circuit with only the 10k resistor. This resulted in a more responsive system.

The LED strip also changed from the initial plan of 50 pixels to 40 pixels to decrease the amount of current needed. The Arduino program was also modified to be clearer, by only using 2 if statements to signal. Introducing the fill command in the setup section and within the if statements reduces the computing power of the Arduino. Meaning it would be redundant to call a function that fills the entire strip with red even if it was already full.

Global Positioning System (GPS):

An alternative to the groups' original option for location data and route planning came in the form of implementing the Android phones’ built in location data using a cellular connection or wifi. Due to inconsistency in location readings from the NEO-6M GPS module in the downtown core, the team decided on the use of the Google Pixel to read location data instead.

Since the location data is already implemented within the phone's settings, the need was to find a way to access the data within the Android Studio app. This was entirely done by means of coding. The app was given the ability to pull from the phone's data to meet all the requirements of the maps feature. The only adversity the group faced in this part of the project included whether to use cellular or wifi in addition to simple UI changes to make the map easy for the user. That being said, the use of cellular data was the source that was decided on as it is more likely the phone has cellular connection to wifi in outdoor environments.

Another advantage of using this system included the fact that it reduced the amount of electronics connected to the main power and bread board with all of the sensors. Additionally, without the need for an external specific GPS module, shielding from the weather conditions becomes easier.

Bluetooth Connection:

Alternative ways to implement wireless communication between the Arduino UNO (main circuit board) and the mobile app beside using HC-05 or HC-06 were:

1. ESP8266/ESP32: These are popular Wi-Fi modules that can be used to establish a wireless connection between the Arduino UNO and a mobile device app over a Wi-Fi network. They offer more advanced features and higher data transfer rates compared to Bluetooth modules.
2. Bluetooth Low Energy (BLE) Modules: BLE modules like the HM-10 or HM-11 offer low-power wireless communication between the Arduino UNO and a mobile device app. They are suitable for applications requiring energy efficiency and can be used for transmitting sensor data or control commands.
3. XBee Modules: XBee modules provide robust wireless communication using Zigbee or other protocols. They offer longer range and can be used in mesh networking configurations for more complex projects.
4. NRF24L01+ Modules: These are low-cost RF transceiver modules that operate on the 2.4GHz frequency band. They are commonly used for short-range communication between Arduino boards or between an Arduino and a mobile device with a compatible receiver.
5. LoRa Modules: LoRa (Long Range) modules enable long-range wireless communication using the LoRaWAN protocol. They are suitable for applications requiring communication over several kilometers and are often used in IoT projects.

APP (Android App Development):

Developing an application was one of the main parts of the project, and while Android Studio was a primary choice for building the “Smart Bike App”, there were alternative solutions available to serve different needs:

1. Swift: Serves as powerful software for creating a Smart Bike App tailored for iOS and Apple OS devices. Using Swift would widen the potential user base and provide an alternative avenue for app creators to reach their intended audience.
2. MIT App Inventor: Another tool for developing Android applications without the necessity of traditional programming expertise. Its intuitive interface makes it accessible to beginners, ensuring that virtually anyone can bring their app ideas to fruition.

In terms of the actual code development, the main software came from multiple sources and rigorous testing methods were employed to establish a connection, and display multiple messages at the same time.

This section outlined alternative designs that could have been chosen instead of the current solutions, and discussed reasons as to why they were not selected. There were five subsections, discussing the energy harvesting system, sensors, GPS, Bluetooth and app development.

V - Material/Component list

This chapter contains tables listing all the materials and components, their quantities and prices, that are used in the XF04 project. Materials are grouped according to the subsystem in which they are used.

Energy Harvesting System:

Table 5.1: Energy Harvesting System Material and Component List.

Material/Component	Quantity	Price
12V DC Motor (Buhler Motor, 1.13.044.284.50)	1	20.00$
6V SLA Battery (Mighty Max ML4-6)	1	26.12$
Motor Shaft Gear	1	0.40$
Wheel Gear	1	1.60$
Motor Mounting Bracket	2	4.00$
PCB Mounting Plate	1	1.49$
Buck Converter (LM2596)	1	2.80$
Boost Converter (XL6009E1)	2	3.17$
EHS Circuit: - PCB - Mosfet (Infineon IRF540N) - Various Resistors - Arduino Nano - 5mm LEDs - Operational Amplifier (GS GL324)	 1 3 14 1 4 1	25.00$
22 AWG Wire		19.89$ / 19.6ft
22-18 AWG Insulated Male Quick Connectors (Powerfist 8234809)	8	7.99$ / 25pcs
22-18 AWG Insulated Female Quick Connectors (Powerfist 8234817)	10	5.99$ / 25pcs
Zip-ties	10	-
6 x 3/8 Self-Tapping Screws (Paulin 846-580)	10	4.99$
4mm x 6mm Phillips Head Screws (Mode Electronics Ltd. 54-435-100)	4	5.99$
100K Rotary Linear Potentiometer	1	2.99$
Total		132.42$

Electrical Sensors:

Table 5.2: Electrical Sensors Material and Component List.

Material/Component	Quantity	Price
(HC-SR04) Ultrasonic sensor	3	5.50$ / 1pcs
(WS2812B neo pixel) LED strip	60 pixels (1M)	35.00$
(A3144) hall effect sensor	1	10.00$ / 10pcs
Temperature sensor (TMP36G)	1	3.00$
Pulse sensor	1	19.95$
7mm Tactile push button	2	1.00$ / 1pcs
MPU6050	1	27.00$
470, 10k ohm resistors	1	-
100-ohm resistors	2	-
LED mounting bracket	1	-
Mounting bracket for eBox	1	58.99 $
Total		172.44$

Bluetooth Connection Components:

Table 5.3: Bluetooth Material and Component List.

Material/Component	Quantity	Price
Module 3.3V HC-05 Bluetooth	1	$17.18/pc
Arduino UNO R3 DIP Edition	2	$42.99/pc
Total		$60.17

Global Positioning System(GPS):

Table 5.4: GPS Material and Component List.

Material/Component	Quantity	Price
Neo-6m GPS Module	1	$23.04
Active Antenna	1	$13.99
High Gain Antenna	1	$15.52
Total		$52.55

Other Components:

Table 5.5: Other Material and Component List.

Material/Component	Quantity	Price
Phone Mount	1	$15.99
Speaker	1	$99.98
Total		$115.97

The materials listed in Tables 5.1-5.5 contain only the materials and components required to implement all of the requirements as outlined in section II - Objectives. This does not include that actual bicycle and the phone, as the developed solutions should have the ability to be incorporated onto any bicycle. Using the tabulated costs from the above tables, the total cost of the design that was developed and manufactured as part of this project, comes to $533.55.

This chapter contained the tables listing all the materials and components, their quantities and prices, that were used in the XF04 project. Materials were grouped according to the subsystem in which they were used.

VI - Measurement and Testing Procedures

This chapter outlines various measurement and testing procedures that were used to assess the performance of the project. Each test outlines its purpose and the required procedure. There are five subsections, listing tests for the energy harvesting system, sensors, GPS, Bluetooth, and app development.

Energy Harvesting System:

Test 1.1 - Motor Rotation

The purpose of this test was to assess the motor operation. Since the motor was not purchased new, it had to be determined if it actually worked, met the requirements and what direction of rotation provided the required polarity.

Procedure:

1. Connect motor terminals to an oscilloscope or multimeter.
2. Connect a drill to the shaft of the motor.
3. Run the setup and record the voltage output of the motor.

Test 1.2 - Gearing Fitments

The purpose of this procedure was to determine the sizing of the bicycle gears and to ensure that they fit properly once they are manufactured.

Procedure (measurement):

1. Using bicycle wheel size, motor specifications, potential speed, determine the required gearing ratio to spin the motor at rated speed when the bicycle is traveling at maximum speed.
2. Using a Vernier caliper, measure the size and shape of the shaft of the motor and the mounting surface of the bicycle.

Procedure (testing):

1. Attempt to fit the gears together to ensure that the teeth align and that they are able to spin properly. Note down any issues.
2. Attempt to fit the bicycle gear onto the wheel and assess fitment. Note down any fitment issues.
3. Attempt to fit the motor gear onto the motor by heating the shaft of the motor and forcing the gear onto it to ensure snug fitment. Note down any fitment issues.
4. Implement any required modifications and repeat steps 1-3 until fitments satisfactory.

Test 1.3 - Motor Mount Fitments

The purpose of this procedure was to gather the required measurements, and to fit the motor mounts to the bicycle.

Procedure (measurement):

1. Using the motor specifications, determine the compatible screw size, mounting hole locations, and size of the motor.
2. Using a Vernier caliper, measure the size and shape of the bicycle frame and translate the shape into a drawing.

Procedure (testing):

1. Attempt to attach the motor to the motor mounts using predetermined screws. Note down any issues.
2. Attempt to fit the assembled motor and mount element onto the bicycle frame. Note down any issues.
3. Attempt to fit the gearing and the motor mounts at the same time. Note down any issues.
4. Implement any required modifications and repeat steps 1-3 until fitments satisfactory.

Test 1.4 - Battery Charging Through Buck/Boost Converters and PSU or Motor

The purpose of this test was to determine if a battery can be charged using the stable output voltage from the buck converters, using either the motor or a PSU variable input voltage.

Procedure:

1. Connect output of the PSU to the input of a boost converter, output of boost converter to input of buck converter, output of the buck converter to a multimeter.
2. Turn on PSU and adjust the boost/buck converters to achieve a 7V output. Vary input voltage and determine acceptable input voltage range.
3. Measure initial battery voltage.
4. Connect output of the buck converter to the battery.
5. Turn on the PSU for 30 seconds. Measure the voltage of the battery. Charging should have occurred.
6. Repeat steps 1-5, but replace the PSU with the DC motor. The DC motor should be spun using the rotational force of the bicycle.

Test 1.5 - Testing the EHS Circuit on a Breadboard

The purpose of this test was to test if the control circuit is capable of creating a connection to charge the battery once significant voltage is present.

Procedure:

1. Construct the EHS circuit on a PCB.
2. Measure the voltage of the battery.
3. Connect a PSU to the input of the circuit and increase the voltage until above the charging threshold. Hold for 30 seconds.
4. Measure the voltage of the battery, charging should have occurred.
5. Repeat steps 1-4, but replace the PSU with the DC motor. The DC motor should be spun using the rotational force of the bicycle.

Test 1.6 - Testing the Operation of the EHS PCB

The purpose of this procedure was to determine if the EHS PCB was manufactured and designed correctly and to test its functionality to ensure proper operation.

Procedure:

1. Use a multimeter to check the continuity of various points on the board.
2. Connect the Arduino Nano and upload the code.
3. Check the operation of the board.
4. Connect a battery or a PSU to the input of the board. Using a multimeter, test the operation of the board.

5. Vary the input voltage and the potentiometer input, and measure the outputs.
6. Note down any issues and modify the code to implement proper operation and fix any issues.

Test 1.7 - Testing Operation of the Full EHS System on the Bicycle

The purpose of this procedure was to tune various parameters in the EHS PCB and to test the operation of the entire system.

Procedure:

1. Connect all the components of the EHS system on the bicycle, but the battery.
2. Connect to the Arduino to measure various parameters and readings. Turn the potentiometer fully counterclockwise.
3. Connect the 6V battery to the system and measure its voltage.
4. Rotate the wheel of the bicycle and check that once the threshold is reached the PCB goes into the charging state (the first blue LED should turn on).
5. Continue for 30s. Measure the voltage of the battery, it should have increased.
6. Begin to turn the potentiometer clockwise. The rear wheel should begin to rotate and the second blue LED should light up.
7. Measure the voltage at the output terminals of the board that will be supplied to the other systems of the bicycle. It should read almost the same as the battery voltage.

Test 1.8 - Measuring Charging Ability

The purpose of this procedure was to measure the current that is generated by the generator in relation to the rotational speed of the motor and bicycle wheel.

Procedure:

1. Connect the circuit coming from the generator to an ammeter, and make a mark on the bicycle wheel.
2. Begin recording a video of the ammeter and the bicycle wheel in the same shot.
3. Gradually increase the speed of the bicycle wheel, until reaching maximum speed.
4. Using the video, measure the time required for one wheel rotation and relate the speed to the measured current at that point in time.
5. Repeat for the entire speed range, until a current-speed relationship is established.

Sensors:

Test 2.1 - Gyroscope(MPU6050)

The purpose of this procedure was to test the gyroscope sensor.

Procedure:

1. Connect the module to the Arduino.
2. Access and calibrate the module using the Arduino library.
3. Once the stable ground and reference are determined move the module in the x and y directions(yaw and pitch).
4. The change should be observed in the serial monitor.

Test 2.2 - Ultrasonic sensors

The purpose of this procedure was to test the ultrasonic sensors.

Procedure:

1. Connect the 3 sensors to the Arduino with the corresponding trig and echo pins.
2. Observe the changes as the 3 sensors display various distances.
3. The changes can be done by simply blocking the path with other objects and individual functionality will be checked.
4. Below 100 cm the code automatically changes the display to a “Vehicle Behind” message.

Test 2.3 - Button and LED turn lights

The purpose of this procedure was to test the functionality of the turn signals.

Procedure:

1. Connect the two buttons and the neopixel to the Arduino.
2. Long press the button and observe the change in the neopixel.
3. The left and right signals should both be observed.

Test 2.4 - Temperature sensor

The temperature sensor has a simple function of observing the temperature to provide some safety to the rider.

Procedure:

1. Connect the sensor to the Arduino and power it up.
2. Using the mathematical function the correct temperature can be seen.
3. Confirm the correct temperature using other sensors.

Test 2.5 - Pulse sensor

The sensor should display the correct heartbeat.

Procedure:

1. Connect the sensor to the Arduino using the A1 analog pin.
2. By gently placing your thumb on the sensor the heartbeat can be seen on the serial monitor.

Test 2.6 - Hall effect sensor for speed measurement

The hall effect sensor should correctly measure the speed and rpm.

Procedure:

1. Secure the sensor onto the bike.
2. Set the magnet to the spoke of the wheel.
3. Measure the approximate radius of the rotation.
4. Record the RPM, and calculate the resulting speed and approximately compare it to the true rotation.

Global Positioning System(GPS):

Test 3.1 - NEO-6M

The module must print out current latitude and longitude in live time

Procedure:

1. Connect the module to the Arduino and antenna to the module.
2. Implement necessary Arduino Libraries (TinyGPS++).
3. Implement the code in the Arduino IDE.
4. Go outside and wait until the module connects with the satellite.
5. The serial monitor will print ‘Loop’ until connection is secured and latitude and longitude will be printed out.

Bluetooth Connection:

Test 4.1: Bluetooth Connectivity

A simple circuit is used to test out the connectivity of the bluetooth connection by itself.

Procedure:

1. Assemble the hardware components by connecting the resistors and the HC-05 Bluetooth module to the Arduino board.
2. Install the "Serial Bluetooth Terminal" app on the mobile device.
3. Compile and upload a Bluetooth sketch to the Arduino using the Arduino IDE software.
4. On the mobile device, navigate to the Bluetooth settings and search for the HC-05 module. Once found, pair and connect to it.
5. Open the Serial Monitor in the Arduino IDE software.
6. Send a simple text string from the Serial Monitor.
7. Switch to the "Serial Bluetooth Terminal" app on the mobile device. If the connection is stable and successful, the text string should appear in the terminal on the screen without any noticeable delay.

Test 4.2: Multiple Bluetooth devices connection

In this project, a Bluetooth speaker is used for hand-free communication as well as to assist in hearing. This test was conducted to ensure that the chosen mobile device can seamlessly function with multiple bluetooth connections at the same time.

Procedure:

1. Establish the connection between the main circuit board and the mobile device using the HC-05 module as required.
2. Connect the Bluetooth speaker to the mobile device.
3. Launch the "Smart Bike App" on the mobile device.
4. Navigate to the "Open Playlist" option within the app.
5. Test the connectivity of the Bluetooth speaker by playing any songs from the playlist.

APP (Android App Development):

Test 5.1 - Final Iteration of Maps Feature on App

The Maps feature on the app must track location in live time and implement route planning

Procedure:

1. Open the app, mount the phone onto the bike and press the ‘Linked Map’ button.
2. Start riding. New locations should be marked in red on the map.
3. Next test route planning. Make sure the latitude and longitude of current location is automatically inputted into the first text box.
4. Type in destination location into the second text box and press ‘open navigation’.
5. The Maps screen should open with a route between the current and final location.
6. Press start. Should receive directions by audio with Google Assistant as well as displayed directions.

Test 5.2: App Functionality and Stability

Follow up with GPS, sensor compatibility testing with the app. The whole final version of the app should also be tested for stability and functionality.

Procedure:

1. Connect all sensors, and the Bluetooth module to the Arduino UNO board, ensuring proper wiring and connections.
2. Download and install the "Smart Bike App" APK on your mobile device.
3. Alternatively use developer mode (in phone settings) and connect straight to your computer.
4. Compile and upload the sketch file to the Arduino UNO board using the Arduino IDE software (disconnect the bluetooth module when uploading).
5. On your mobile device, access the Bluetooth settings and search for the HC-05 module. Pair and establish a connection with it.
6. Make sure to grant all permissions in your setting since some devices disable them automatically.
7. Open the "Smart Bike App" on your mobile device. If the connection is successful, the app should display all the measurements received from the sensors, indicating proper signal communication between the Arduino UNO board and the mobile app.

Test 5.3: UI Interface Testing/ Device Compatibility

The app will be implemented in different mobile devices with different Androids OS as well as screen size. This test is crucial to ensure that the app has great compatibility for most devices.

Procedure:

1. Download the "Smart Bike App" on multiple mobile devices with varying screen sizes and different versions of the Android operating system.
2. Launch the app on each device.
3. Thoroughly examine the app's display for any errors, inconsistencies, or layout issues across different screen sizes.
4. Test the functionality of the app on each device, ensuring that all features work as intended.
5. Verify the signal communication between the app and any connected devices or sensors, such as the Arduino UNO board with the Bluetooth module.
6. Document any display errors, functionality issues, or communication failures encountered during the testing process.

This chapter outlined various measurement and testing procedures that were used to assess the performance of the project. Each test outlined its purpose and the required procedure. There were five subsections, listing tests for the energy harvesting system, sensors, GPS, Bluetooth, and app development.

VII - Performance Measurement Results

This chapter discusses the results obtained from completing the test procedures outlined in the previous chapter. There are five subsections, discussing the results from tests related to the energy harvesting system, sensors, GPS, Bluetooth, and app development.

Energy Harvesting System:

Test 1.1 - Motor Rotation

As the rotational speed of the motor was increased, the voltage at the output terminals of the motor also increased, signifying that the motor operated properly. The voltage increased linearly with the speed of the motor, as a DC motor should. Additionally, the polarity of the voltage was measured to determine the orientation of the future connection to the rest of the system.

Test 1.2 - Gearing Fitments

The gear mounting surfaces were measured and transferred into Solidworks. The first design iteration of the gear fit perfectly, both onto the bicycle and the motor. Additionally, the teeth of the gears mated well. No further adjustments were required.

Test 1.3 - Motor Mount Fitments

The motor mounts required several iterations as the geometry of the bicycle frame was quite complex. Several attempts were made to determine a rough sizing for the mounts. Once this was achieved, fiberglass filler was used to mold some of the parts for secure fitment. The fiberglass filler was shaved down and final measurements were taken that were used in the last iteration of the mounts. The mounting holes for the motor did not cause any issue as the documentation acquired was accurate.

Test 1.4 - Battery Charging Through Buck/Boost Converters and PSU or Motor

The battery was charged and the achieved results met expectations. This was true when using the PSU and the motor, in both cases the battery was charged.

Test 1.5 - Testing the EHS Circuit on a Breadboard

This procedure resulted in some modifications that had to be implemented into the PCB design of the EHS circuit. The Arduino was capable of measuring the voltage and activating switches when required but the MOSFETs were incapable of actually turning on. The reason behind this was that the voltage from the Arduino was not higher than the source voltage of the MOSFET, thus it would not turn on. This was easily resolved by using the OP-AMP as an amplifier instead of simply a buffer, since a higher voltage was already available in the circuit. Once this was done, the circuit behaved as required and met the requirements.

Test 1.6 - Testing the Operation of the EHS PCB

The continuity test determined that the routing of the board was correct. Basic testing of the EHS PCB also demonstrated that it operates as required and makes all the required connections. Unfortunately, there was a design flaw in the circuit that was not noticed by pure coincidence and only came up when doing Test 1.7.

Test 1.7 - Testing Operation of the Full EHS System on the Bicycle

Initially, the circuit operated as required and there were no noticeable issues. It was later noticed that one of the MOSFETs gets quite hot for no apparent reason and the board exhibits some odd behavior. The cause of this issue was determined to be that during PCB design the OP-AMP was configured to operate under positive feedback instead of negative feedback. By pure coincidence all other operations worked and the issues were easily resolved by modifying the code of the Arduino. Once this was complete, the circuit was tuned to enable connections once voltage thresholds were reached and it operated as required. Another issue that was potentially the result of the incorrect circuit setup and carelessness during testing is that at some point terminal connections were switched that resulted in damage to the circuit. This was also fixed, but better safety features in the design could have prevented this.

Test 1.8 - Measuring Charging Ability

Based on the measurements, it was determined that sufficient voltage for charging is present once the speed of the bicycle reaches above 12 km/h. Additionally, there is a clear relationship between the speed and generated current, as speed increased, current also increased. Though as the speed increased the amount of current disproportionately increased, increasing less at higher speeds than at lower speeds. Peak generated current occurred at a maximum speed of 50 km/h where 1.76A was fed from the motor to the boost/buck converters.

Sensors:

Test 2.1 - Gyroscope(MPU6050)

The pins had to be changed to SCL and SDA pins for it to measure correctly. The code needed to set up the initial reference point every time the code is loaded. An Arduino library had to be used and changed such that it no longer uses an interrupt function as pin2 was already being used. The sensor first sets up its reference point from which the angle is recorded and measured.

```
Bike is stable! | Pitch= -0.03|roll= 0.04
```

Figure 7.1:The Base Pitch and Roll.

The pitch dictates the stability in terms of along the bike and the roll is the side to side. The below figure displays the change.

```
Bike is stable! | Pitch= -1.58|roll= 4.32
```

```
Bike is stable! | Pitch= -0.25|roll= 6.03
```

Figure 7.2: Varied Roll and Pitch.

Setting the bike back to its reference point resets the pitch and roll to near 0.

Test 2.2 - Ultrasonic sensors

The 3 ultrasonic sensors were set and soldered into the correct formation to maximize its range. The following figure showcases the change in measurement in cm. Below a certain value, the message changes to "Vehicle behind!!".

```
Vehicle behind!!:1097 ,8 ,4
```

Figure 7.3: Message display when Vehicle detected behind.

Test 2.3 - 2 Button and LED turning lights

The left and right buttons were tested extensively and the buttons had to be soldered and mounted to the handle.

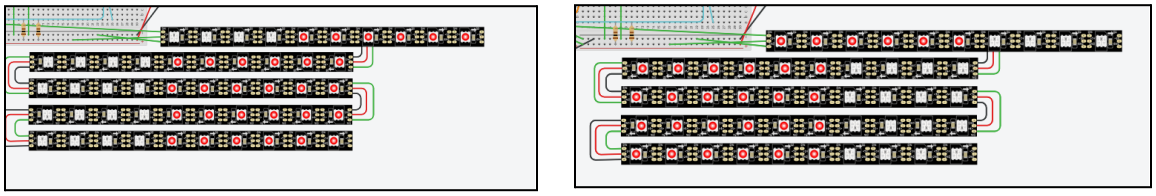


Figure 7.4: Simulation of the Left and Right Turn Signal.

Test 2.4 - Temperature sensor

The temperature sensor was the easiest to implement and multiple data and changes were observed.

```
Temperature: 29°C
```

Figure 7.5: Temperature Display Text String.

Test 2.5 - Pulse sensor

The heartbeat was mounted on the bike handle as well and various beats were recorded and displayed.

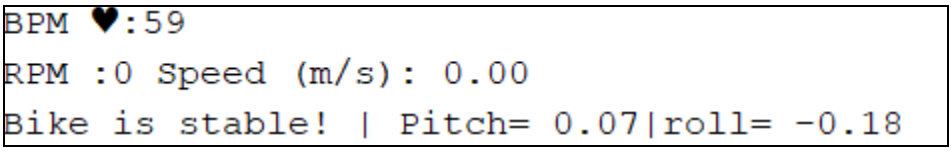


Figure 7.6: Pulse and Speed Sensor Display Text String.

There were a few fluctuations in the measurement and for certain people, the proper heartbeat was not recorded. Otherwise, the result closely matched the Apple watch's heartbeat.

Test 2.6 - Hall effect sensor for speed measurement

The hall effect sensor had to be set up in a certain direction such that the magnet could be detected. It was then soldered and mounted to the bike along with a powerful magnet.

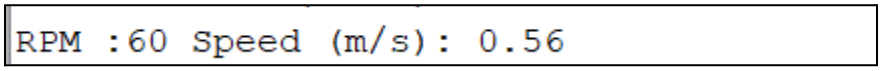


Figure 7.7: Hall Effect and Speed Sensor Display Text String.

The RPM displayed only multiples of 60 and when it sped up the rpm was increased accordingly.

Global Positioning System(GPS):

Test 3.1: NEO-6M

By following the testing procedure above for the NEO-6M GPS module, students were able to see that once the module connects to a total of five satellites, latitude, longitude and altitude measurements of current location are printed out on the arduino serial monitor. The code prints 'Loop' until the module connects to the satellite and then prints out location and altitude. This procedure worked, although not consistently and so alternative designs were then considered.

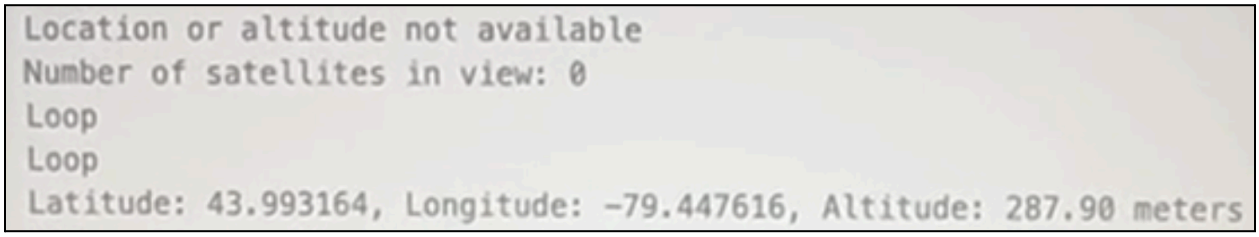


Figure 7.8: Testing Results of NEO-6M on Arduino Serial Monitor.

Bluetooth Connection:

Test 4.1: Bluetooth Connectivity

The expected outcome of the test between the Serial Bluetooth Monitor and the Arduino IDE is a seamless and accurate exchange of data(text string). Upon establishing a Bluetooth

connection, the Serial Monitor of the Arduino IDE should promptly display any information transmitted from the Serial Bluetooth Monitor app. Similarly, data sent from the Arduino IDE's Serial Monitor should be efficiently received and processed by the Serial Bluetooth Monitor app. This bidirectional communication ensures real-time monitoring and interaction with the connected hardware, facilitating effective testing and debugging processes. Overall, the successful execution of this test confirms the reliability and stability of the communication between the two platforms.

APP (Android App Development):

Test 5.1: Final Iteration of Maps Feature on App

Through the testing procedure outlined in the above section the final iteration of the maps feature was tested. The button ‘Linked Map’ redirected the user to a map screen with two text boxes underneath. The map screen accurately displayed the location of the bike as it moved around, each new location marked with a red marker. As this occurs, the first text box automatically updates with the current location of the bike, ready to be used in route planning. When the user types in the destination location and presses the ‘Open Navigation’ button, the app redirects to a route planning screen with directions to the location. This feature was also tested to make sure Google Assistant was able to read out the directions for hands-free use, which it does successfully.

Test 5.2: App Functionality and Stability

Through GPS and sensor compatibility testing, as well as evaluation of the app's overall stability and functionality, it is ensured that the final version of the app performs optimally. Following the established procedure, which involves connecting sensors and the Bluetooth module to the Arduino UNO board, installing the app on a mobile device, and verifying signal communication, the expected result is a seamless integration between hardware and software components. Upon successful execution, the app accurately displays all measurements received from the sensors, validating proper signal communication between the Arduino UNO board and the mobile app.

Test 5.3: UI Interface Testing/ Device Compatibility

Ensuring the "Smart Bike App" maintains optimal compatibility across a spectrum of Android devices, this test evaluates its performance on varying screen sizes and operating system versions. Subsequently, the functionality of all app features is thoroughly tested on each device to confirm seamless operation as intended. Moreover, signal communication between the app and connected devices, like the Arduino UNO board with the Bluetooth module, is verified for reliability. Any encountered display discrepancies, functionality issues, or communication setbacks are documented to be resolved.

This chapter discussed the results obtained from completing the test procedures outlined in the previous chapter. There were five subsections, discussing the results from tests related to the energy harvesting system, sensors, GPS, Bluetooth, and app development.

VIII - Analysis of Performance

This chapter analyzes the performance of each of the subsystems on the project based on the results obtained from completing the test procedures outlined in the chapter VI - Measurement and Testing. There are five subsections, analyzing the performance from tests related to the energy harvesting system, sensors, GPS, Bluetooth, and app development.

Energy Harvesting System:

This system performed up to the requirements outlined for this project. The EHS is capable of charging a battery, providing power to other systems on the bicycle and has "drive-assist" functionality that was added as an additional feature. The output of the boost/buck converters was set to be around 7V, which is a voltage sufficiently large enough to charge the 6V SLA battery on the bicycle that has an average voltage of around 6.45V. The boost/buck converters are capable of providing a sufficient voltage once the DC motor is spun fast enough to generate at least 3V. The circuit of the EHS reads the voltage at the terminals of the motor and once it is above 3V, the connection gets established and the battery begins to get charged. This correlates to a bicycle speed of around 12 km/h. Generated current varies from around 0.4A at the lowest speeds, and up to 1.76A when the bicycle is traveling at around 50 km/h and the generator is rotating at rated speed. Additionally, a threshold speed needs to be reached to begin charging of the battery, meaning that at very low bicycle speeds charging will not happen. The speed-current relationship is plotted below in Figure 8.2.

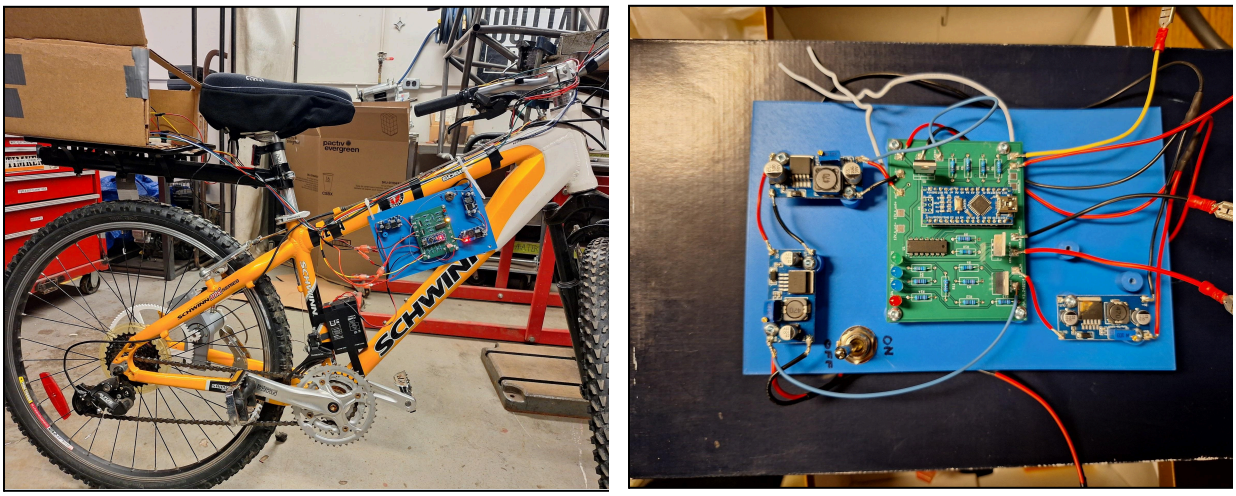


Figure 8.1: Energy Harvesting System Mounted to the Bicycle and the Control Circuit.

After testing the EHS circuit functioned as required but several shortfalls have also been determined that can degrade the performance of the system. The main aspect that could provide improvement would be to add several safety features to the circuit such as fusing and proper connectors. Fusing would be very beneficial as current overload protection would save the circuit if there would be a fault during operation. During testing/tuning of the circuit, a software version was uploaded that resulted in significant current consumption and an increase in temperature of various components that could have been damaged. A fuse would have prevented this from occurring by blowing and signifying that an issue had occurred. Proper connectors would also improve the safety and reliability of the circuit as they would prevent any potential mistakes to

be made while connecting up the circuit. During testing two wires were accidentally switched, resulting in damage to the board and it needing to be partially replaced. Connectors that allow for only one way to connect wires would have prevented this. Finally, thermal additions would also benefit the circuit. Due to high current sometimes flowing through some of the transistors, they heat up and may get damaged if the temperature gets too high. Adding heat sinks would minimize this effect and would improve the performance of the circuit.

Bicycle gearing performed as required, but could have been improved. The gears are made out of PLA plastic that is used by the 3D printers at the DME lab. This allows them to be cheaply and quickly manufactured but also leads to the gears being quite weak. They handled the load so far, but it is questionable how well they would perform over a very long period of time. Most likely the teeth would wear down or break due to excessive wear. Making the gears out of a stronger material, such as metal or a stronger plastic, would mitigate this concern but would also significantly increase the cost of the design.

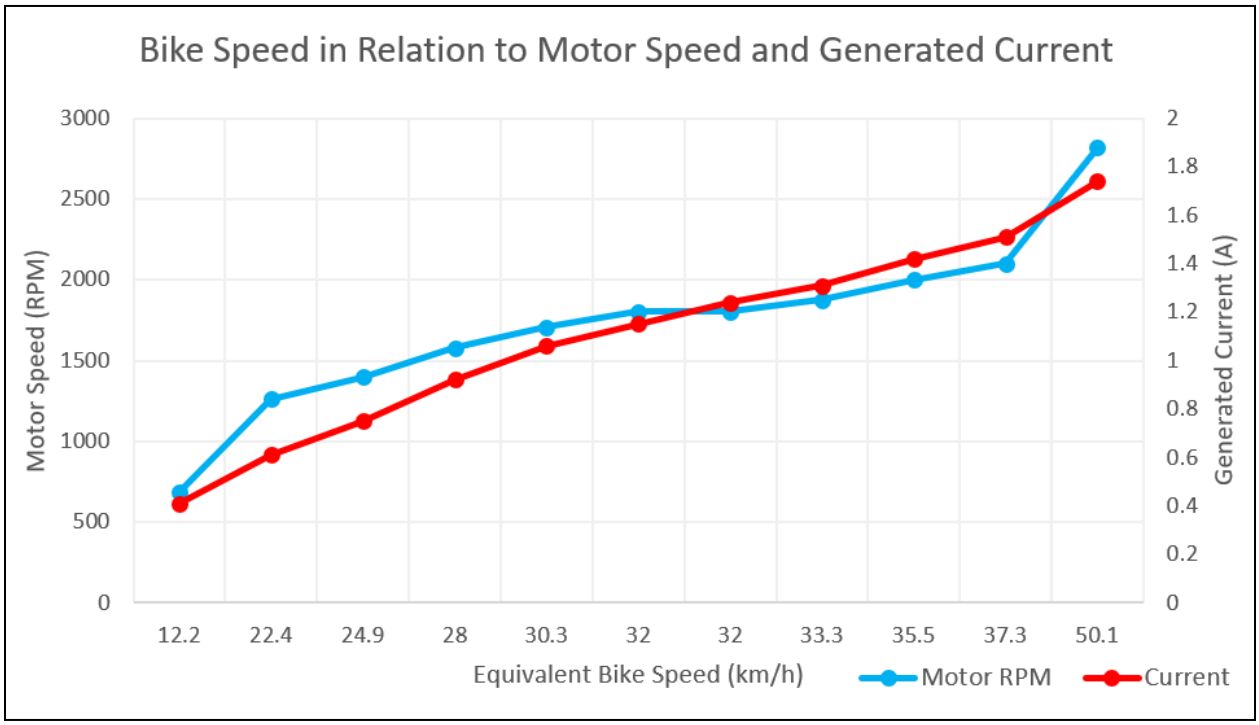


Figure 8.2: Plot of the Speed-Current Relationship

In conclusion, the EHS performed up to the requirements of this project, but there can be improvements made in the design to improve performance, reliability and safety of the system.

Sensors:

In terms of the Gyroscope(MPU6050) the actual angle of the sensor is not accurate but with the reference, it displays a number that is pretty close to the actual angle. The main functionality comes from the math that stabilizes the readings as well as various values that stabilize the pitch and roll. Various changes can also be recorded with a response time of about

0.2s. The majority of the delay comes from other sensors that impact the performance. Playing with the threshold value provided an accurate sense of where the bike is in relation.

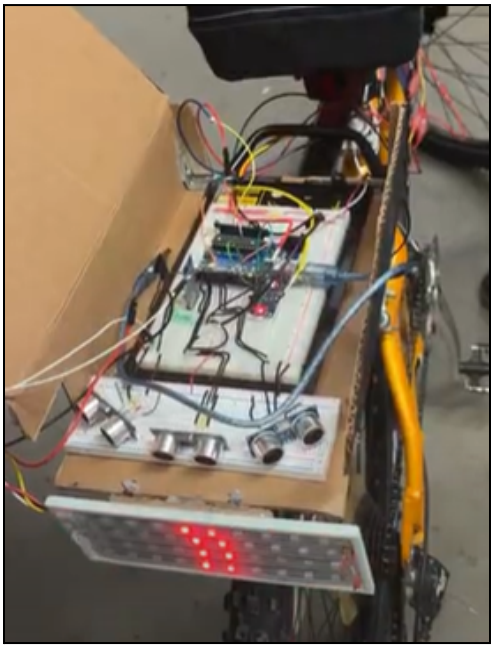


Figure 8.3: The Main Circuit.

The ultrasonic sensors have a functional range of 2-400 cm however a default of 1000 cm is displayed when no object is within the range. The results are responsive with an accuracy of about 3mm. The buttons work as intended however the response time is lengthy as buttons do not work as effectively for turn signals. This results in having to press the buttons for about 0.5-1s to register the signal. This is mainly due to other sensors occupying the processing time. Setting the button as interrupts fails as there are no more supporting interrupts within the Arduino.

The temperature sensor works as intended and matches the sensors on the apple watch. It also changes accordingly when applied heat via touch.

The pulse sensor works with a few incorrect measurements. There are also problems relating to thresholds that are specific for each person. In order to correctly measure the pulse the threshold then needs to be changed. Otherwise the sensor matches with the apple watch.

Using hall effect sensors, results in less than accurate results as the interrupt does not function as expected. The exact rpm could not be determined however within reason and is about 70% accurate. The delay is substantial however it is also measuring the instant speed.

Global Positioning System (GPS):

Several inconsistencies and challenges posed a problem when it came to the maps feature. The main problem was the NEO-6M ability to consistently provide location information while the antenna was in some way blocked from receiving satellites. Due to the nature of the testing ground this was a serious consideration. Performance was not up to par when this module

was being used. This ensured the group had to consider an alternative design to solve this problem.

Out of all the challenges taken into account, it was found that the environment itself posed the greatest challenge. Due to the area where testing of the bike occurred being heavily surrounded with large concrete buildings, characteristic of the downtown core, the antenna from the GPS module was not powerful enough to connect to the five satellites required for secure and accurate location readings. To combat this issue the group tried and tested high gain and active antennas to determine if a secure reading would be accessed in the downtown core. Unfortunately this only provided a very limited and insecure connection where only sometimes location data would be transferred to the arduino serial monitor for viewing and verification. Due to this issue, it was decided to change the platform on which the location data was gathered and displayed, with support from the rest of the group. This resulted in the use of an android phone, the Google Pixel.

After initial testing the Google Pixel was able to accurately display location data on a consistent basis. By using the phone's connection to cellular data, the group reduced the risk of inconsistent results. Cellular was chosen over wifi due to the fact wifi is not readily available most of the time while one is riding a bike outside. This gives the user a huge range, both in urban and rural areas in which they can determine their own location and then use it to gather directions to their destination of choice. Once location data was accurate a route planning feature, similar to that of Google Maps was implemented. Current location was then always used as the source for the directions. The user then types in the destination for a route to be shown. Once ‘Open Navigation’ is pressed, a route will show. The below figure shows examples of both from Google headquarters (GooglePlex), an arbitrary location, to Toronto, Ontario, Canada.

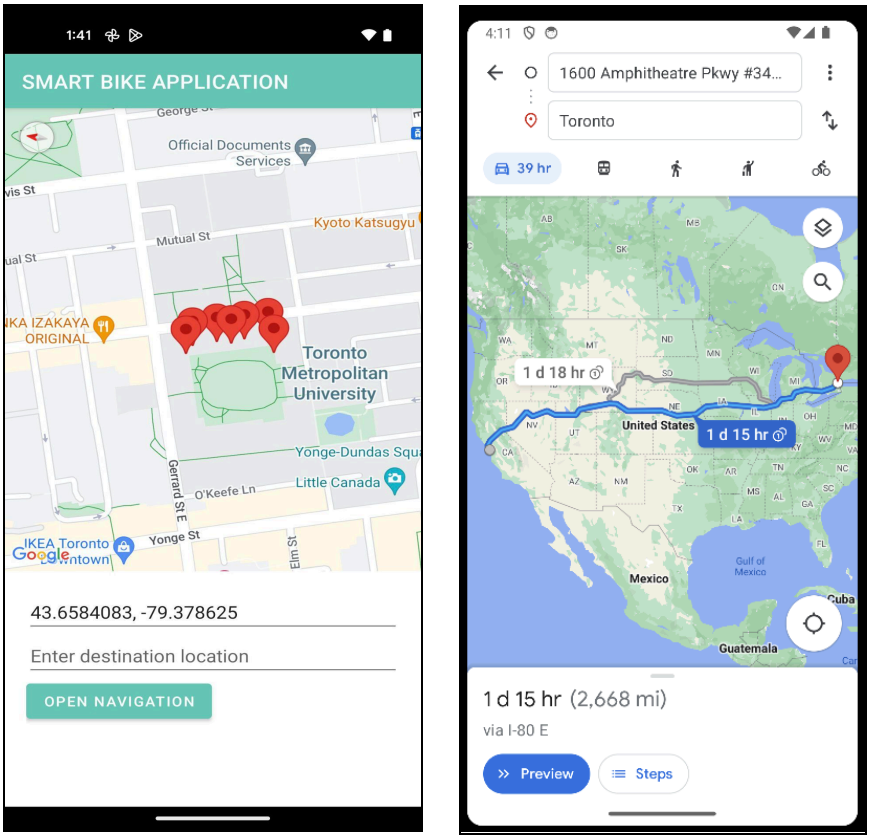


Figure 8.4: Final Iteration of Maps Screen on App and Route Planning.

Through all of these changes, all of the group's challenges were addressed and all required functions of the Maps feature were implemented. The app has a user friendly design when it comes to the maps feature and it is very simple to use and display while open. Thus, it is clear that the performance of the maps feature has no problems and meets all requirements for the project.

Bluetooth Connection:

In assessing the Bluetooth connection's performance within the project, several key factors are carefully examined. Initially, the stability of the link between the app and the Arduino UNO board, featuring the HC-05 module, is examined to ensure consistent data transmission without interruptions. Simultaneously, the integrity of data exchange via this connection is verified to maintain communication accuracy and reliability. Furthermore, the project evaluates the latency and response time of Bluetooth communication to optimize real-time interaction by minimizing delays. Additionally, the strength of the Bluetooth signal undergoes assessment to pinpoint potential connectivity issues and enhance transmission efficiency. Notably, the stability and reliability of Bluetooth connections between multiple devices, including the Bluetooth module and speaker, are also tested across varied environments. By addressing these considerations, the analysis aims to elevate the Bluetooth connection's reliability, speed, and overall performance within the "Smart Bike App," ultimately enriching the user experience.

APP (Android App Development):

The analysis of the "Smart Bike App" performance covers several critical aspects derived from multiple testing. Firstly, the app's compatibility is assessed across a spectrum of Android devices, spanning varying screen sizes and operating system versions. This evaluation identifies any display errors or layout inconsistencies and evaluates the app's adaptability to diverse device specifications. Secondly, functionality is thoroughly examined to ensure seamless operation of all features, including GPS tracking and sensor data reception, across different devices. The app continues to update the sensors periodically with little to no delay time. Moreover, the reliability of signal communication between the app and external hardware components, such as the Arduino UNO board with the Bluetooth module, is verified. Documentation of encountered issues provides valuable insights for further refinement, guiding recommendations aimed at optimizing compatibility, functionality, and signal communication reliability to enhance the overall user experience.

This chapter analyzed the performance of each of the subsystems on the project based on the results obtained from completing the test procedures outlined in the chapter VI - Measurement and Testing. There were five subsections analyzing the performance from tests related to the energy harvesting system, sensors, GPS, Bluetooth, and app development.

IX - Conclusions

This chapter summarizes the entire report and discusses any conclusions that have been made based on the performance of the project. Each of the subsystems is discussed, with positive and negative aspects of performance briefly summarized. Challenges are also listed, along with any potential future steps.

The integration of various components within the smart bike system represents a diverse project aimed at enhancing the safety, convenience, and overall experience of cyclists. From the Energy Harvesting System (EHS) to the many sensors, the Global Positioning System (GPS), Bluetooth connectivity, and the android app, each play a role in the project's requirements. Conclusions are drawn from the assessment of each subsystem, finding recommendations to further enhance the functionality and efficiency of the smart bike system.

The Energy Harvesting System (EHS) performed as intended, completing the requirements set for the project. It effectively charges the battery and provides power to other systems allowing for greater efficiency. Some areas for improvement were identified in terms of safety features and component durability. Implementing fusing, adequate connectors, and thermal management could enhance the system's reliability and prevent damage during operation. Overall, while the EHS met project objectives, introducing these additions could optimize its performance and longevity.

The sensors demonstrated satisfactory performance, with each sensor fulfilling its intended function within acceptable parameters. However, some sensors exhibited minor issues such as response time delays and random inaccuracies. Improving interrupt handling for buttons and refining threshold settings for pulse sensors could boost performance. Despite this, the sensors overall contribute to the functionality of the bike system, providing data for rider safety and convenience.

Challenges encountered with the NEO-6M GPS module included limitations in satellite connectivity, particularly in urban environments with obstructed line-of-sight to satellites. Moving to a smartphone-based GPS solution was effective by using cellular data connectivity to ensure consistent and reliable location tracking. This method addressed the initial issues and also enhanced user convenience by integrating seamlessly with existing android functionality. The implementation of route planning further aids the use of the GPS and maps feature.

The Bluetooth connection allowed for smooth communication between the mobile app and external components. Testing confirmed reliable data exchange and minimal latency, allowing for real time observation by the user. Overall, the Bluetooth device met expectations, providing an interface for integrating the smart bike system with multiple external devices.

The "Smart Bike App" demonstrated its functionality through testing. It identified and addressed display errors, layout inconsistencies and functionality issues, ensuring an easy user experience. The app's ability to update sensor data with a small delay continues to enhance its use for riders, providing accurate real-time information for quick decision-making.

In conclusion, while each component of the smart bike system displayed strengths and areas for improvement, overall, they form a functional system that enhances rider safety, convenience, and overall experience.

This chapter summarized the entire report and discussed any conclusions that have been made based on the performance of the project. Each of the subsystems was discussed, with

positive and negative aspects of performance briefly summarized. Challenges were also listed, along with any potential future steps.

X - References

This section lists all the sources used for reference during the creation of this report. The references are listed in the order that they appear in the report.

[1] "Electric Bike Market Size, Share | E-Bike Industry Report 2026," www.fortunebusinessinsights.com, 2022. https://www.fortunebusinessinsights.com/electric-e-bike-market-102022

[2] Betancor, Javier. "Impulse." *Hackaday.Io*, hackaday.io/project/159305/files. Accessed 4 Dec. 2023.

[3] Tokala, Vikram. "Smart Bicycle." *Hackster.Io*, 16 May 2021, www.hackster.io/vikram-viky/smart-bicycle-4905b9#comments.

[4] *Arduino® Uno R3*, docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf. Accessed 5 Dec. 2023.

[5] Chris, et al. "Arduino and MPU6050 Accelerometer and Gyroscope Tutorial." *How To Mechatronics*, 18 Feb. 2022, howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

[6] Bob, "Adding Bluetooth to Your Arduino Projects," Jan. 30, 2021. https://bytesnbits.co.uk/adding-bluetooth-to-your-arduino-projects/

[7] Tom Stanton. (2021, September 27). *Super Capacitor bike* [Video]. YouTube. https://www.youtube.com/watch?v=V_f8Q2_Q_J0

[8] Generalissimo_II, "Bicycle wheel generator for a headlight," Jun. 10, 2021. https://www.reddit.com/r/nostalgia/comments/nx0f06/bicycle_wheel_generator_for_a_headlight/ (accessed Apr. 11, 2024).

[9] Dejan, "Arduino and MPU6050 Accelerometer and Gyroscope Tutorial - HowToMechatronics," HowToMechatronics, Apr. 09, 2019. https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

[10] The-Frugal-Engineer. (n.d.). *The-frugal-engineer/arduinobtexamplev3: Arduinobtexamplev3*. GitHub. https://github.com/The-Frugal-Engineer/ArduinoBTExamplev3

XI - Appendices

Motor Rated Speed = 3000rpm
Motor Rated Voltage = 12V
Wheel Diameter = 26in (0.6604m)
Assumed max. speed of bicycle = 25km/h

$$Distance\ traveled\ per\ wheel\ revolution\ = \pi \cdot d$$
$$= \pi \cdot 0.6604m$$
$$= 2.075m$$

$$Number\ of\ Wheel\ Revolutions\ per\ Minute\ = Speed/Distance\ traveled\ per\ wheel\ revo$$
$$= \frac{25km/h}{2.075m}$$
$$= \frac{416m/minute}{2.075m}$$
$$= 200\ wheel\ revolutions\ per\ minute$$

At max bike speed the motor should rotate at rated speed therefore at 200 wheel rpm the motor should rotate at 3000 rpm. Thus the gear ratio between the wheel gear and motor gear should be 3000:200 or 15:1.

Figure 11.1: Gearing Calculations

```
// NOTES:

// It's crazy but only now did I realize that for pins 2-4 LOW output
// makes the connection and HIGH output blocks it... IS the OPAMP acting
// as an inverting OPAMP??? Why?

int potentiometer_input = 0;  // A3
int motor_voltage_input = 0;  // A1
int battery_voltage = 0;      // A0

int comm_in = 0;              // A2

int counter = 0;

// STATES:
// 0 = not charging, not driving
// 1 = charging
// 2 = driving
int state = 0;

bool low_voltage = 0;
// 6.37V/2 = 652
// sufficient voltage present at output once voltage is higher than 3V (300 analog eq.)


int battery_voltage_threshold = 550; // update to whatever upon testing
int motor_voltage_threashold = 300;  // update to whatever upon testing
// inpuy from bucks is 716
```

```
void setup() {
  Serial.begin(9600);

  // STATUS LEDs
  pinMode(9, OUTPUT);   // ON
  pinMode(10, OUTPUT);  // Charging
  pinMode(11, OUTPUT);  // Driving
  pinMode(12, OUTPUT);  // Low battery

  // OPAMP PINS
  pinMode(2, OUTPUT);   // Charging switch
  pinMode(3, OUTPUT);   // PWM driving switch
  pinMode(4, OUTPUT);   // Output connection enable switch

  // disable all switches
  digitalWrite(2, HIGH);
  analogWrite(3, 1023);
  digitalWrite(4, HIGH);

}

void loop() {
  digitalWrite(9, HIGH);  // Board is ON


  battery_voltage = analogRead(A0); // Read battery voltage

  if (battery_voltage > battery_voltage_threshold) { // If sufficient battery voltage is present

    digitalWrite(4, LOW);  // enable power output


    digitalWrite(12, LOW);  // Low battery LED OFF

    low_voltage = 0;

  }
  else { // If low voltage

    digitalWrite(4, HIGH);   // disable power output
    digitalWrite(3, 1023);  // disable motor output

    digitalWrite(12, HIGH);  // Low battery LED ON

    low_voltage = 1;
  }


  potentiometer_input = analogRead(A3);
  motor_voltage_input = analogRead(A1);


  if ((potentiometer_input > 50) && (low_voltage != 1)) {     // Driving Mode
    state = 2;
    digitalWrite(11, HIGH);            // UPDATE LEDS
    digitalWrite(10, LOW);
    digitalWrite(2, HIGH);          // Disable charging
    //PWM output
    analogWrite(3, (1023-(potentiometer_input/4)));
    //analogWrite(3, 0);
```

```
  }

  else if ((motor_voltage_input > motor_voltage_threashold) && (potentiometer_input < 10)){    //Charging Mode
    state = 1;

    digitalWrite(11, LOW);        // UPDATE LEDs
    digitalWrite(10, HIGH);

    digitalWrite(2, LOW);          // Enable charging
    analogWrite(3, 1023);
  }

  else {  // insufficient charging voltage, and not driving
    state = 0;
    digitalWrite(11, LOW);        // UPDATE LEDs
    digitalWrite(10, LOW);
    //analogWrite(3, 0);
    digitalWrite(2, HIGH);

  }


  // STATUS OUTPUT
  counter++;
  if (counter > 300) { // 3 seconds between each print



    counter = 0;
    Serial.print("Current state: ");
    Serial.println(state);


    if (state == 2) {
      Serial.print("  Current motor output: ");
      Serial.print(potentiometer_input);
      Serial.println(" ");
    }
    else if (state ==1) {
      Serial.print("Motor Voltage Input: ");
      Serial.println(motor_voltage_input);
    }
    else {
      Serial.println("Not driving and insufficnent voltage present for charging.");
    }

    Serial.print("Battery Voltage: ");
    Serial.println(battery_voltage);

    Serial.print("Potentiometer: ");
    Serial.println(potentiometer_input);

    Serial.print("Motor Voltage Input: ");
    Serial.println(motor_voltage_input);
    Serial.println("-----------------------------------------\n\n\n");

  }
  delay(10);
}
```

Figure 11.2: Arduino Code Used in the EHS Control Circuit

```
 //////////The actual code below//////////////////
#include <Adafruit_NeoPixel.h>
#include <Wire.h>
#define USE_ARDUINO_INTERRUPTS true    // Set-up low-level interrupts for most acurate BPM math.
#include <PulseSensorPlayground.h>
PulseSensorPlayground pulseSensor;   // Creates an instance of the PulseSensorPlayground object called "pulseSensor"
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
//////////////////////
//Variables for Gyroscope
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif

MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL

//#define INTERRUPT_PIN 12  // use pin 2 on Arduino UNO & most boards

bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
float angle_pitch_output, angle_roll_output;
// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
```

```
void dmpDataReady() {
   mpuInterrupt = true;
}

//////////pulse sensor measurements////////////////*/
float revolutions=0;
int rpm=0; // max value 32,767 16 bit
long  startTime=0;
long  passedTime;



//NEO PIXAL
#define LED_PIN    9

// How many NeoPixels are attached to the Arduino?
#define LED_COUNT 50

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
// Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
////////////////



int celsius = 0;
int fahrenheit = 0;



int trigPin1=3;
int echoPin1=11;    //note was changed

int trigPin2=4;
int echoPin2=5;

int trigPin3=8;
int echoPin3=7;

int const PulseWire = 1;   // 'S' Signal pin connected to A1
int Signal;                // Store incoming ADC data. Value can range from 0-1024
int Threshold = 550;       // Determine which Signal to "count as a beat" and which to ignore.

int chk;
float hum;


const int buttonPin = 6;  // the number of the pushbutton pin
```

```
const int buttonPin2 = 10;  // the number of the pushbutton pin
// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status
int buttonState2 = 0;
volatile int countL = 1; //Count for LI
volatile int countR = 1; //Count for RI
uint32_t red = strip.Color(255, 0, 0);

//GPS SYSTEM


void setup() {
  Serial.begin(9600);    //38400 bit for hc-05

  pinMode(A0, INPUT);

  /////


  pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);
  pinMode(trigPin3, OUTPUT);
  pinMode(echoPin3, INPUT);
  //neo pixal setup

  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
  strip.setBrightness(20); // Set BRIGHTNESS to about 1/5 (max = 255)- may cause issues

  // Configure the PulseSensor object, by assigning our variables to it.
  pulseSensor.analogInput(PulseWire);
  pulseSensor.setThreshold(Threshold);

  //may need /n to send data
  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
      //Serial.println("We created a pulseSensor Object !");   //This prints one time at Arduino power-up,   or on
Arduino reset.
   }
  //rpm sensor
  //pinMode(2, INPUT_PULLUP);           // set pin to input
  //attachInterrupt(digitalPinToInterrupt(2),interruptFunction,FALLING);

  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin2, INPUT);
  strip.fill(red, 0);
  strip.show();
  //MPU6050 sensor
  mpusetup();

}
```

```
void loop() {

 // measure temperature in Celsius
 celsius = map(((analogRead(A0) - 20) * 3.04), 0, 1023, -40, 125);
 // convert to Fahrenheit
 //fahrenheit = ((celsius * 9) / 5 + 32);
 String message = (String) "Temperature: " +celsius+"°C";
 Serial.println(message);

 // Check for button 1 press
 if (digitalRead(buttonPin) == LOW) {
  //Serial.println("LEFT");
  buttonPressed();
 }

 // Check for button 2 press
 if (digitalRead(buttonPin2) == LOW) {
  //Serial.println("RIGHT");
  buttonPressed2();
 }

 //code for ultrasonic sensors
 long duration1, distance1;
 digitalWrite(trigPin1, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin1, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin1, LOW);
 duration1 = pulseIn(echoPin1, HIGH);
 distance1 = (duration1/2) / 29.1;


 long duration2, distance2;
 digitalWrite(trigPin2, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin2, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin2, LOW);
 duration2 = pulseIn(echoPin2, HIGH);
 distance2= (duration2/2) / 29.1;

 long duration3, distance3;
 digitalWrite(trigPin3, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin3, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin3, LOW);
 duration3 = pulseIn(echoPin3, HIGH);
 distance3= (duration3/2) / 29.1;
```

```
if (distance1 < 100 || distance2 < 100 || distance3 < 100)
{
  //Serial.println("Vehicle behind!!");
  String outMsg = String("Vehicle behind!!:") + distance1+ String(" ,")+distance2 +String(" ,") +distance3;
  Serial.println(outMsg);
}
else {
  String outMsg = String("Vehicle:") + distance1+ String(" ,")+distance2 +String(" ,") +distance3;
  Serial.println(outMsg);
}

  ////////////////////////////////

  //pulse sensor
   if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".
int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.
 //Serial.print("♥  A HeartBeat Happened ! "); // If test is "true", print a message "a heartbeat happened".
 //Serial.print("BPM: ");                        // Print phrase "BPM: "
 //Serial.println(myBPM);                        // Print the value inside of myBPM.
  String outMsg = String("BPM ♥:") + myBPM;
  Serial.println(outMsg);
}
  //neopixal
  // read the state of the pushbutton value:
  // Check for button 1 press
  if (digitalRead(buttonPin) == LOW) {
    //Serial.println("LEFT");
    buttonPressed();
  }

  // Check for button 2 press
  if (digitalRead(buttonPin2) == LOW) {
    //Serial.println("RIGHT");
    buttonPressed2();
  }


  //rainbow(10);             // Flowing rainbow cycle along the whole strip


  //calculate revolution
  revolutions=0; rpm=0;
  startTime=millis();
  attachInterrupt(digitalPinToInterrupt(2),interruptFunction,FALLING);
  delay(1000);
  detachInterrupt(2);

  //now let's see how many counts we've had from the hall effect sensor and calc the RPM
  passedTime=millis()-startTime;     //finds the time, should be very close to 1 sec

  if(revolutions>0)
  {
    rpm=(max(1, revolutions) * 60000) / passedTime;        //calculates rpm
```

```
 }

 float diameterInMeters = 0.1778;//diameter

 // Calculate speed in m/s
 float speedInMetersPerSecond = (PI * diameterInMeters / 60.0) * rpm;

 // Print speed to Serial Monitor
 String speedMsg = String("RPM :") + rpm+ String(" Speed (m/s): ") + speedInMetersPerSecond;
 Serial.println(speedMsg);


//MPU6050 data
 // if programming failed, don't try to do anything
  if (!dmpReady) return;
  // read a packet from FIFO
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet


     #ifdef OUTPUT_READABLE_YAWPITCHROLL
        // display Euler angles in degrees
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
        //Serial.print("ypr\t");
        //Serial.print(ypr[0] * 180/M_PI);
        //Serial.print("\t");
        //Serial.print(ypr[1] * 180/M_PI);
        angle_pitch_output=ypr[1] * 180/M_PI;
        //Serial.print("\t");
        //Serial.println(ypr[2] * 180/M_PI);
        angle_roll_output =ypr[2] * 180/M_PI;
     #endif

  }

 // Thresholds for instability detection (adjust as needed)
 float pitchThreshold = 7.0; // in degrees
 float rollThreshold = 7.0;  // in degrees

 // Check for instability
 if (abs(angle_pitch_output) > pitchThreshold || abs(angle_roll_output) > rollThreshold) {
  Serial.println("Bike is on unstable ground!");
  // Add your code to handle instability,
 }
 else {
       String out1Msg = String("Bike is stable! | Pitch= ") + angle_pitch_output + String("|roll= ")+
angle_roll_output;
  Serial.println(out1Msg);
 }

}

//End of indicatorOn*/
void buttonPressed() {
```

```
  // turn LED on Right:
     for (int i = 0; i < 5; i++) {
    indicatorOn(0, 1);
  }
  strip.fill(red, 0);
  strip.show();
}

void buttonPressed2() {
  for (int i = 0; i < 5; i++) {
    indicatorOn(1,-1);
    }

  strip.fill(red, 0);
  strip.show();
}

//indicatorOn
void indicatorOn(int z,int j) {

  for (int i = 0; i<=5;i++){

    //Rear Lights
    strip.setPixelColor(((5-z)-i*j),red);
    strip.setPixelColor(((14+z)+i*j),red);
    strip.setPixelColor(((25-z)-i*j),red);
    strip.setPixelColor(((34+z)+i*j),red);
    strip.setPixelColor(((45-z)-i*j),red);
  strip.show();

    delay (70);
  }
  delay(200);
  strip.clear();
}
//End of indicatorOn*/
void interruptFunction() //interrupt service routine
{
  revolutions++;
}

//////////////////////////////////////////////////////////////////////////

//////////////////
void mpusetup() {

  // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
```

```
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)

    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    //pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
        Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));


    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        //Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        //attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;
```

```
        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    //pinMode(LED_PIN, OUTPUT);
}


//////////
// Rainbow cycle along whole strip. Pass delay time (in ms) between frames.
void rainbow(int wait) {
  // Hue of first pixel runs 5 complete loops through the color wheel.
  // Color wheel has a range of 65536 but it's OK if we roll over, so
  // just count from 0 to 5*65536. Adding 256 to firstPixelHue each time
  // means we'll make 5*65536/256 = 1280 passes through this loop:
  for(long firstPixelHue = 0; firstPixelHue < 5*65536; firstPixelHue += 256) {
    // strip.rainbow() can take a single argument (first pixel hue) or
    // optionally a few extras: number of rainbow repetitions (default 1),
    // saturation and value (brightness) (both 0-255, similar to the
    // ColorHSV() function, default 255), and a true/false flag for whether
    // to apply gamma correction to provide 'truer' colors (default true).
    strip.rainbow(firstPixelHue);
    // Above line is equivalent to:
    // strip.rainbow(firstPixelHue, 1, 255, 255, true);
    strip.show(); // Update strip with new contents
    delay(wait);  // Pause for a moment
  }
}
```

Figure 11.3: Arduino Sensors Code