System Design for MECHTRON 4TB6

Team 25, Formulate
Ahmed Nazir, nazira1
Stephen Oh, ohs9
Muhanad Sada, sadam
Tioluwalayomi Babayeju, babayejt

January 18, 2023

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
MECHTRON 4TB6	Explanation of program name

Contents

List of Tables

List of Figures

3 Introduction

The system design document establishes the group's development considerations for the Formulate system. The motivations which drove each aspect of the design were referenced back to the System Requirements Specification, Hazard Analysis, and Development Plan documents.

4 Purpose

Documentation of the Formulate system's design serves to improve the maintainability, reusability, and understandability of the project. This is accomplished through the system design, software architecture, and software detailed design documents by detailing how the design addressed the requirements outlined in the documents from this document's introduction.

5 Scope

This document in particular focuses on the considerations for the user interface, mechanical, electrical, and communication protocol aspects of the system. All relevant design decisions relating to the requirements were detailed by each aspect of the system, and any visual components used to aid the design were included at the end of the document by each aspect of the system.

5.1 System Context Diagram

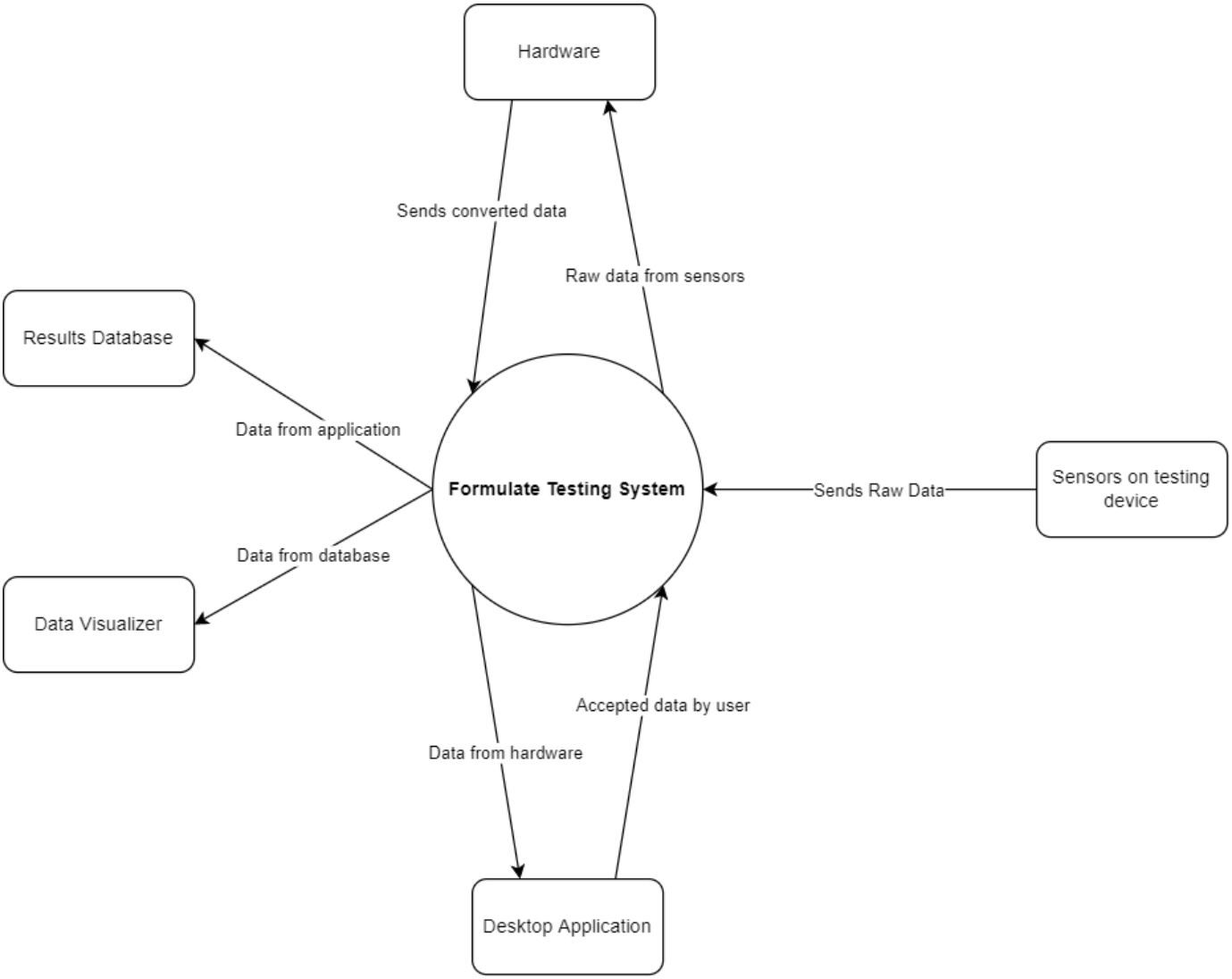

Figure 1: Formulate Context Diagram

6 Project Overview

6.1 Normal Behaviour

During normal operation the device should work as follows, the user has an option to either connect wired or wirelessly to our device, regardless of which method they choose the results will be consistent between the two. To connect to the device the user will open our desktop application and either create and account or login to an existing account, this will lead them to the home page where they user will select the connection method wireless or wired.

Before the user can start any testing they will need to secure our device on vehicle they sre testing. Our device has 2 methods of mounting, either the hose clamp holes can be used or the custom mount. After the device is secure the user will need to ensure that all the correct sensors are attached to the device.

To conduct any testing the user will first fill out the test parameters of the specific test they are conducting and they will also upload pictures of the current configuration of the car. The user will then go to the test page in our application and start the test, during testing our hardware will collect the sensor data and send it to the PC. When the user stops the test they will be able to preview all the raw data collected from the test and either decline the data or send it to the database. When the user sends data to the database all the sensor data will be sent along with the test parameters and pictures.

If the user wants to view previous test data they will be able to using a Power Bi dashboard, this dashboard will read all the data from the database and display it to the user in a more presentable way. The Power Bi dashboard will allow the user to do comparisons between tests and take a quick glance if the tests they conducted passed spec.

6.2 Undesired Event Handling

There might be cases where undesired events occur and to minimize any issues caused to the user we tried to account for them

6.2.1 Loss of Power/Connection

If at any point the connection between the PC and the device fails, i.e the power goes out or the device gets unplugged, we have implemented a back up system to ensure that no testing data is lost. Whenever data is sent to the PC it is also simultaneously sent to the local storage on the device its self. The on board MicroSD Card will have all the data saved and the user will be able to recover it if anything where to go wrong.

6.2.2 Excessive Vibration/Shaking

Since our device will be in an environment where there might be lots of vibration we decided to create a custom PCB instead of using normal jumper cables. This ensures thats regardless of the amount of motion the connections between the components inside our device will be fixed.

6.2.3 Loss of Data Packets

When data from our device is being sent to the PC there might be situations where some data is lost or the entire string of data is not sent correctly. To make sure that we are only reading data from complete bytestrings each bytestring start with 'B' and ends with 'E' and our python program checks to make sure that the data we are saving has both those values.

6.3 Component Diagram

6.4 Connection Between Requirements and Design

The following requirements are from the SRS document.

FR1,FR5: To measure vibration we chose to use an accelerometer which outputs in units of G's. For temperature we are using an LM35 temperature sensor which provides temperature in celsius and has an accuracy of +- 1 degree. To measure humidity we are using a DHT11 sensor which gives us the relative humidity in percent.

FR2: The device transmits data using the UART protocol to the PC when it is connected via a wire.

FR3,FR4: The device will have the start and stop button to control the tests inside the desktop application so the user can conduct tests remotely.

FR6: The user can preview the data after a test once they stop the test. A table will populate with all the raw data from the test on the testing page of the desktop application

FR7: After the user previews the test they will be able to decline or submit the test to our database. The table of test values will be sent to our Azure database once the user approves.

FR8: The Power Bi dashboard will connect to the Azure database and will be able to read all the test data

FR9:

FR10:

FR11: Our device is going to contain 4 AA batteries to power the entire thing, this allows for the user to be able to quickly swap out old batteries for new ones if they die.

FR12: Using an ESP8266 our device will wirelessley communicate with our desktop application through Wi-Fi and will be using TCP to send data back and forth.

7 System Variables

7.1 Monitored Variables

Monitored Variable	Type	Units	Description
m_vibration	Analog	V	A signal monitoring the vibration resistance of the motor
m_humidity	Analog	V	A signal monitoring the humidity of the motor's environment
m_temperature	Analog	V	A signal monitoring the temperature of the motor's environment
m_shock	Analog	V	A signal monitoring the shock resistance of the motor
m_conv_vibration	Digital	g	Converted vibration values that are in useful units
m_conv_humidity	Digital	%	Converted humidity values that are in useful units
m_conv_temperature	Digital	°C	Converted temperature values that are in useful units
m_conv_shock	Digital	g	Converted shock values that are in useful units
m_data_accepted	Digital	T/F	Determines if user has accepted the results and wants to send it to the database

Table 1: Monitored Variables

7.2 Controlled Variables

Controlled Variable	Type	Units	Description
c_green_light	Digital	1/0	Green LED light on testing device that indicates passed measurements
c_red_light	Digital	1/0	Red LED light on testing device that indicates failed measurements
c_sent_to_database	Digital	T/F	Determines if results displayed on the application are sent to the database

Table 2: Controlled Variables

7.3 Constants Variables

Constant	Units	Value	Description
k_temperature_range	°C	5-40	Acceptable ambient temperature values for a Formula Electric motor
k_humidity_range	%	5-85	Acceptable relative humidity values for a Formula electric motor
k_max_shock	g	100	Maximum shock resistance for a Formula Electric motor
k_max_vibration	g	20	Maximum vibration resistance for a Formula Electric motor

Table 3: Constants Variables

8 User Interfaces

8.1 Desktop Application

The user interface for the desktop application is designed through Qt designer, a software for designing and building GUIs through the Qt library. Qt designer generates UI files which can be converted to python scripts that build the static design and layout of the GUI. The desktop application is essentially multiple pages stacked on each other that change based on which buttons are clicked. The GUI is comprised of a left bar menu, top bar, and content pages being in the center, refer to figure 1 and 2 in the Appendix. Navigation through the application is done using the sidebar menu, where users can toggle the full menu and press on which page they want to go. The top bar will be used for extra functionality such as accessing user details, minimizing screen, etc. Users interact with the application using buttons to perform a variety of functions and form fields in which they can enter test/user information.

8.2 Hardware

The user will interface with the hardware as follows, they will mount a sensor to the top of our device and connect it via a terminal block which is on the side of the device. The device will then be mounted to the Formula vehicle and the rest of the operations will take place on the Desktop Application.

9 Design of Hardware

The hardware for our project will include a 3D printed chassis which will house all our electrical components. Our chassis was designed to meet the requirements outlined in our SRS document. The main requirements our chassis needs to meet is how easily it mounts to the Formula E car. We are currently implementing 2 methods of mounting the device onto the car, the first method consists of attaching a clickbond mount to the base of the car and out device will have a mechanism which connects to the mount. The second method is for attaching the device to any other surface of the vehicle, our chassis has slots at the bottom which hose clamps can fit though to allow the device to be clamped to any surface. A priliminary design can be found on Figure XX

10 Design of Electrical Components

The electrical components were selected to address the functional requirements regarding robust sensor connection points, wireless functionality, and backup data collection capabilities. These capabilities were enabled using hardware modules selected to interface with the embedded device.

The Arduino Uno R3 (Uno) was the choice electrical component for the devices embedded hardware. While other embedded devices were also capable of flexibly collecting data from a multitude of sensors, the Uno stood out as the optimal choice because of low monetary cost in hardware and the relatively high accessibility at large e-commerce platforms for the Formula Electric Team to purchase. In addition, the likelihood of the Formula Electric team using parts of the testing budget on the embedded hardware was minimized as many Formula Electric members already posessed an Uno board.

To support testing application flexibility in cases where no direct connections to power were available, four 1.5 volt batteries with a battery holder was used to provide the Uno with adequate power for on vehicle testing sessions.

A single pole, double throw (SPDT) power switch was used to easily connect and disconnect the four 1.5 volt batteries with the circuit connecting all electrical components. This gave the user to The common pin actuated to connect either the 9V battery to the circuit or the circuit directly to ground.

Although the Uno provided many functionalities, the standard Uno model did not natively support wireless communication capabilities. As a result, the group chose to integrate a hardware module capable of a Transmission Control Protocol / Internet Protocol (TCP/IP) stack through an ESP8266 Software On Chip (SOC). The electrical component containing the ESP8266 SOC that was selected was the Node MCU 1.0. The Node MCU 1.0 is a development board with the ESP8266 SOC already built onto the Node MCU's PCB in addition to an on-board voltage regulator for the development boards 3.3 volt power in requirement.

A single pole, single throw (SPST) switch packaged with four ports was also required to interface the Node MCU 1.0 module with the Uno during initial device comissioning. Specifically, three of the four ports were used to disconnect the 3.3 volt power, transmit (TX), and recieve (RX) signal between the wifi module and the embedded device when flashing the wifi module with firmware via a micro Universal Serial Bus (USB) port. When the one time firmware flash is complete, the three switches could be actuated to reconnect the power and signal connections between the two components. Despite the connect/disconnect functionality requiring only three of the four ports, a four port SPST switch was selected due to the high accesibility on large electronics e-commerce platforms relative to three port SPST switches.

Two diagnostic light-emitting diode's (LED) were used to provide the user with feedback on the live transmission status of the wifi module. The first diagnostic signal conveyed when a connection between the wifi module and the desktop application was established, showing the module was capable of transmitting data. This required the first LED to visually depict the active signal to the user. The second diagnostic signal conveyed when the wifi module was actively transmitting data live to the desktop application. This required the second LED to visually depict the active signal to the user.

Two resistors were also paired with the diagnostic LED's to primarily limit the current going through each LED, but also the luminosity of both LED's. Limiting the current through each LED was an important consideration as it supports a consistent luminoscity in the LED's and mitigates premature breakdown of the LED due to a high current burnout.

Backup data storage to local memory in the event of wireless communication error due to the wifi module's failure or device operation outside the wifi router's range necessitated the use of a local memory storage electrical component. A 32 GigaByte (GB) micro Secure Digital (SD) card paired with a micro SD card adapter was used to provide the Uno with local memory storage to concurrently write test data to the SD card while also sending data over wifi to the desktop application. SanDisk, the microSD card manufacturer, was chosen primarily due to their cost effectiveness against other microSD manufacturers as measured by GB/dollar. Similarly, Geek Story was used as the microSD card adapter manufacturer as a result of their cost-effectiveness.

Robust sensor connection components between the sensor and the Uno's input ports were required for tests in physically demanding scenario's as loose or broken connections from high vibration or shock compromised the reliability of the sensor readings and thus the test data. As a result, Phoenix Contact's through hole, 8 port terminal blocks were used because the terminal block style connections provided a stronger connective interface between the sensor conductors and the Uno's ports.

Robust connections between all components of the circuit such as the electrical connections between the Uno, wifi module, micro SD card adapter, switches, and LED's also required a more robust solution relative to jumper wires. As a result, the group chose to design and manufacture a Printed Circuit Board (PCB) onto which all electrical components outlined could be soldered onto the board for a higher strength connection.

The required electrical connections between the micro SD adapters pinouts and the Uno's pinouts were first outlined to organize the connection layout in the table below.

Pin Name	Pin Description	Arduino Port	Arduino Port Description
VCC	5 Volt	5 Volt	Power output
GND	Ground	Ground	Ground
MISO	SPI output from microSD	12	Digital I/O
MOSI	SPI input to microSD	11	Digital I/O
SCK	Synchronize data transmission via Arduino clock	13	Digital I/O
CS	Select slave device on SPI bus	10	Digital I/O

The required electrical connections between the wifi module's pinouts and the Uno's pinouts were also outlined to organize the connection layout in the table below.

Pin Name	Pin Description	Arduino Port	Arduino Port Description
3V3	3.3 Volt	3.3 Volt	Power output
GND	Ground	Ground	Ground
TX	Transmit	2	Digital I/O
RX	Recieve	3	Digital I/O

A final list of the required electrical components was shown below.

Component	Manufacturer	Part Number	Description	Quantity
Microcontroller	Arduino	Uno R3	System microcontroller	1
Wifi Module	NodeMCU	1.0	System wifi	1
Micro SD Adapter	Geek Story	N/A	Local memory interface	1
Micro SD Card	Sandisk	SDSQUAR-032G-GN6MA	System local memory	1
Battery	Duracell	4330206640	System power	4
SPDT Switch	E-Switch	100SPTITI1B4M2QE	System power switch	1
4 Port SPST Switch	TE	435640-2	Wifi signal control switch	1
Through Hole Terminal Block	Phoenix Contact	1715789	System Sensor ports	2
LED	Kingbright	WP7113ID5V	LED Resistor	2
Resistor	TE	CBT25J100R	LED Resistor	2
Custom PCB	JLCPCB	N/A	System PCB	1

The electrical schematic for the overall circuit containing all components was designed in Kicad and was shown in Appendix C. The PCB layout was also designed in Kicad and was shown in Appendix C. The PCB layout was then fabricated by the manufacturer JLCPCB.

11 Design of Communication Protocols

Our project mainly uses two ways to communicate. The user will either directly plug the device into their computer or they will wirelessly communicate with the device over Wi-Fi. Each of the sensors that are connected to the Arduino use different protocols to send data.

Device	Communication Protocol
MicroSD Card Module	SPI
Accelerometer (ADXL345)	I2C
Temperature Sensor (L535)	Single Bus
Humidity Sensor (DHT11)	Single Bus
Wi-Fi Module (ESP8266, NodeMCU 1.0)	UART

Table 4: Communication Protocols

To simplify the communication protocol, after the Arduino reads all the values from the sensors it formats them into bytestring to send to either the Wi-Fi module or the PC directly via USB. The bytestring takes the form

```
B<X-VAL>S<Y-VAL>S<Z-VAL>S<TEMP-VAL>S<HUMIDITY-VAL>E
```

The order in which the sensor values are sent are in the following order; Vibration in X, Vibration in Y, Vibration in Z, Temperature, Humidity. Each bytestring starts with 'B' and ends with 'E', after the data is sent to the PC our python program parses through the received bytestring and only stores values from complete bytestrings, it checks for the 'B' and the 'E'. When the device is directly plugged into the PC it sends the data over USB via serial and data is sent every second. The other way of receiving data is over Wi-Fi, the Arduino take the bytestring and sends it using the TX and RX pins to the NodeMCU which then relays the information to the PC via a TCP connection.

To connect the device to our PC wirelessly there are two main methods we have implemented. The first method is to make our device act as an access point, this means the PC will directly connect to the the device and they will exchange information via a TCP connection. The second method would be to connect our device to a central hub and also connect our PC to the same hub, this would require an ethernet connection to be present to plug the hub in. Using both methods our ESP8266 acts a relay device which passes the information from the Arduino to the PC via TCP.

12 Timeline

A Interface

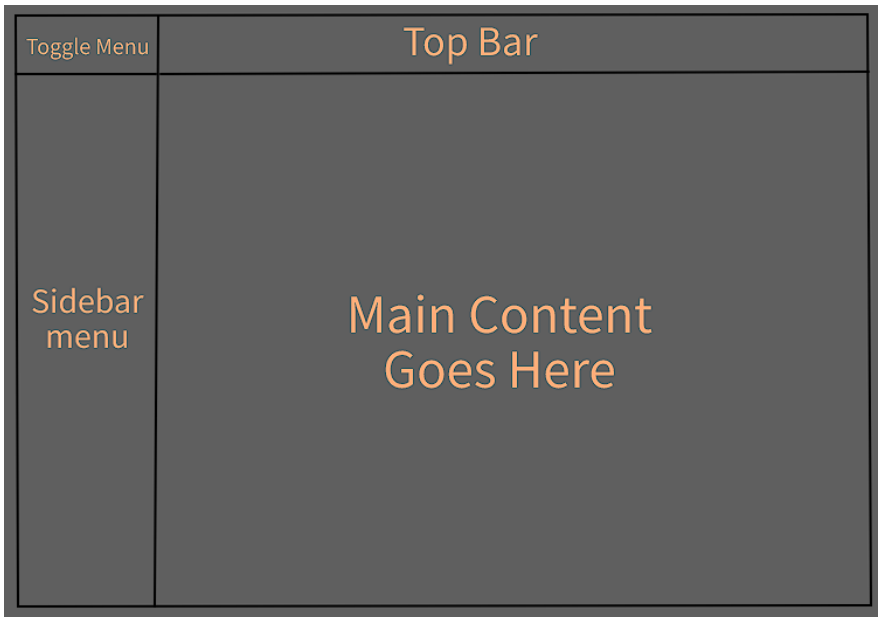

Figure 2: Basic layout of GUI

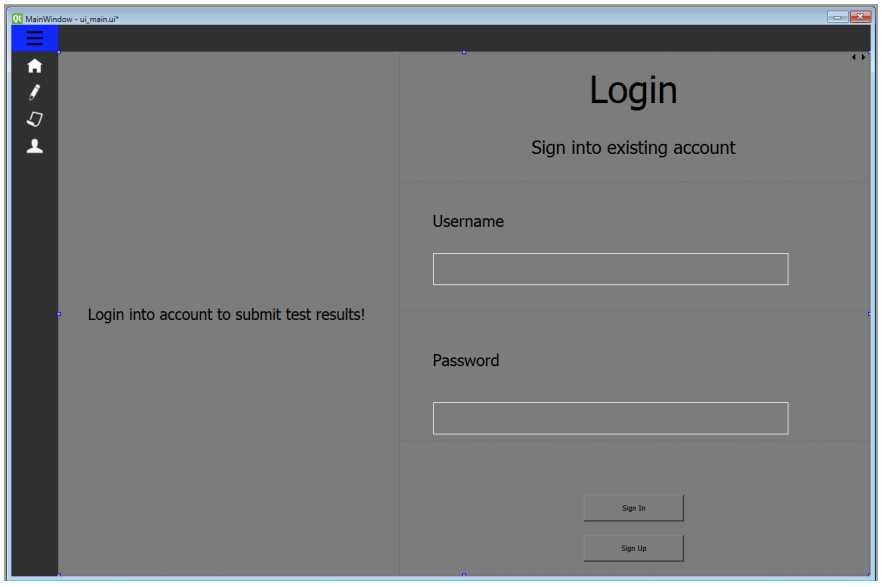

Figure 3: GUI design in Qt Designer

B Mechanical Hardware

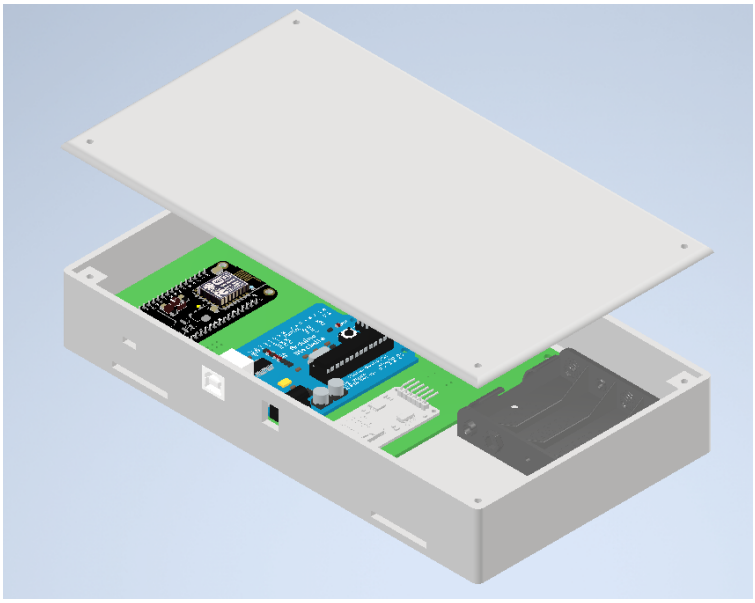

Figure 4: Chassis 3D Render

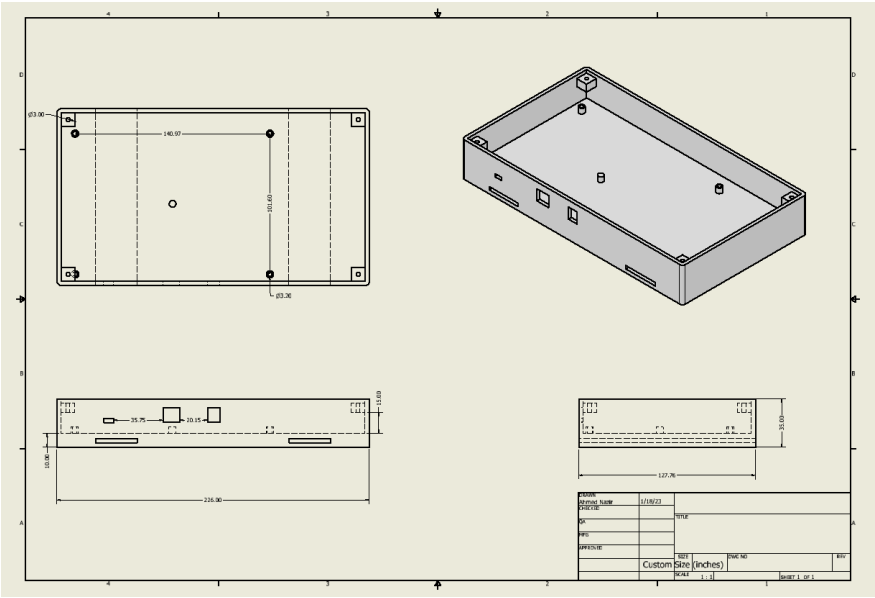

Figure 5: Chassis Drawing

C Electrical Components

C.1 Electrical Schematic

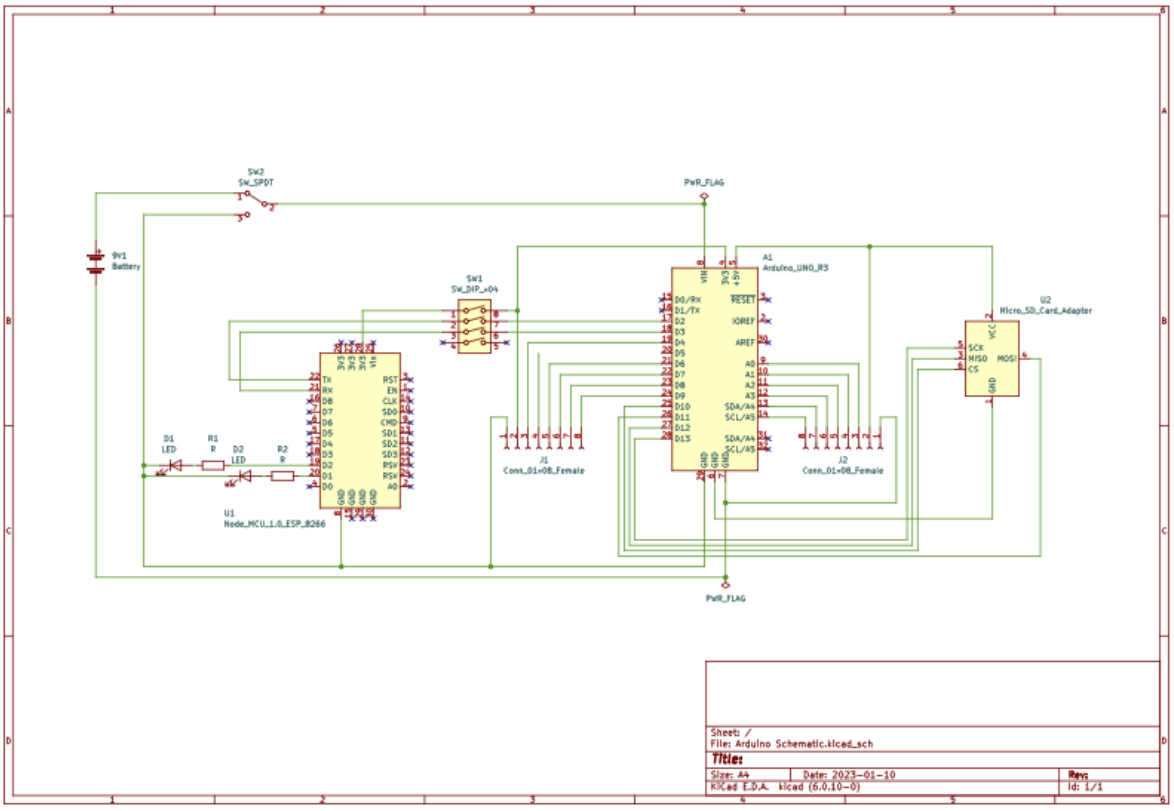

Figure 6: Electrical Schematic in Kicad

C.2 PCB Layout

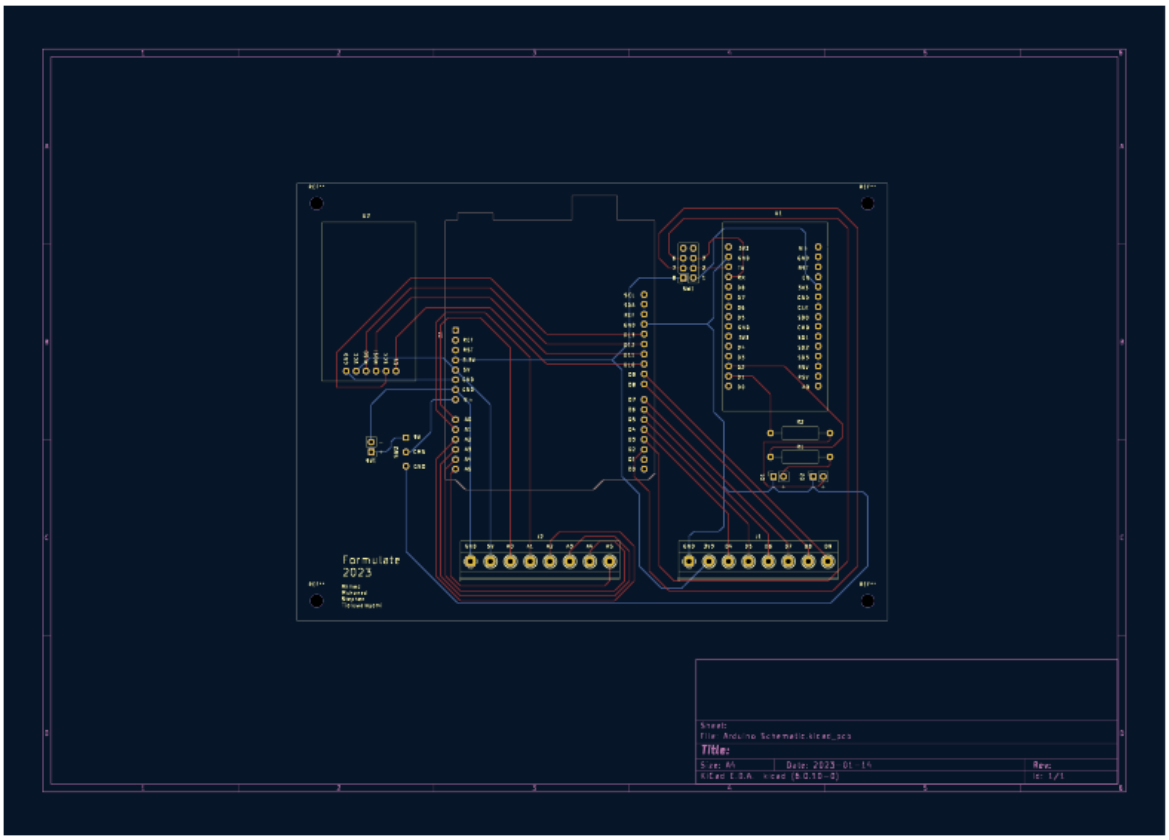

Figure 7: PCB Layout in Kicad

C.3 PCB CAD

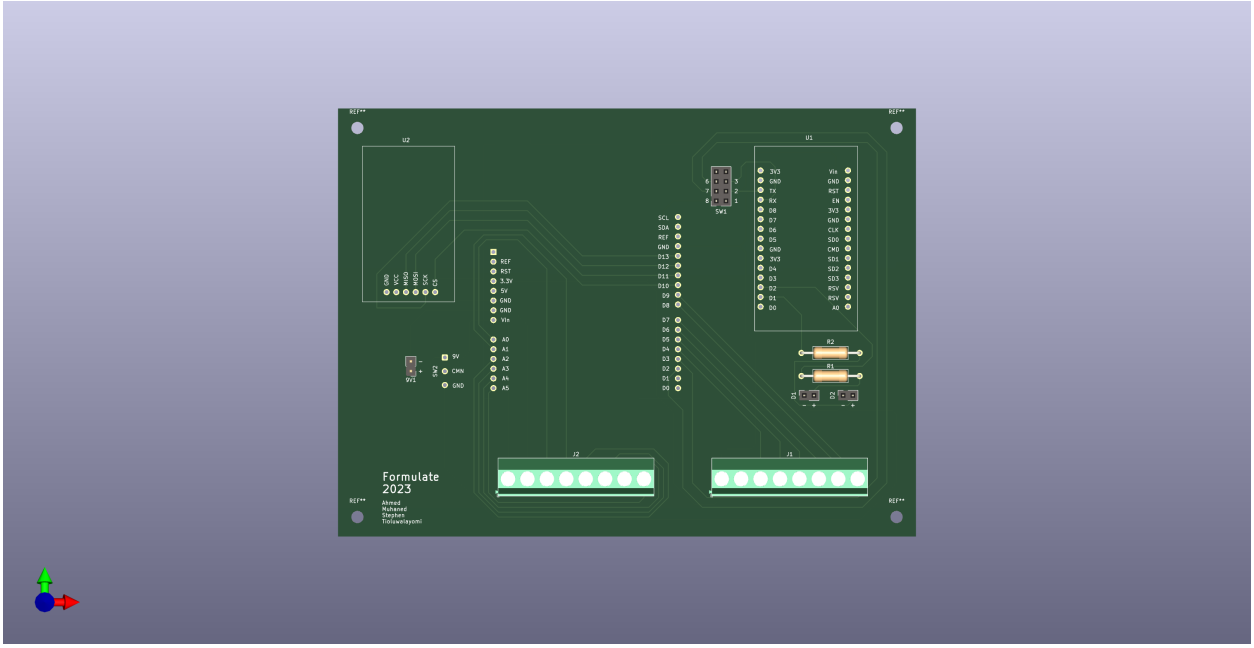

Figure 8: Top view of the PCB

C.4 PCB CAD

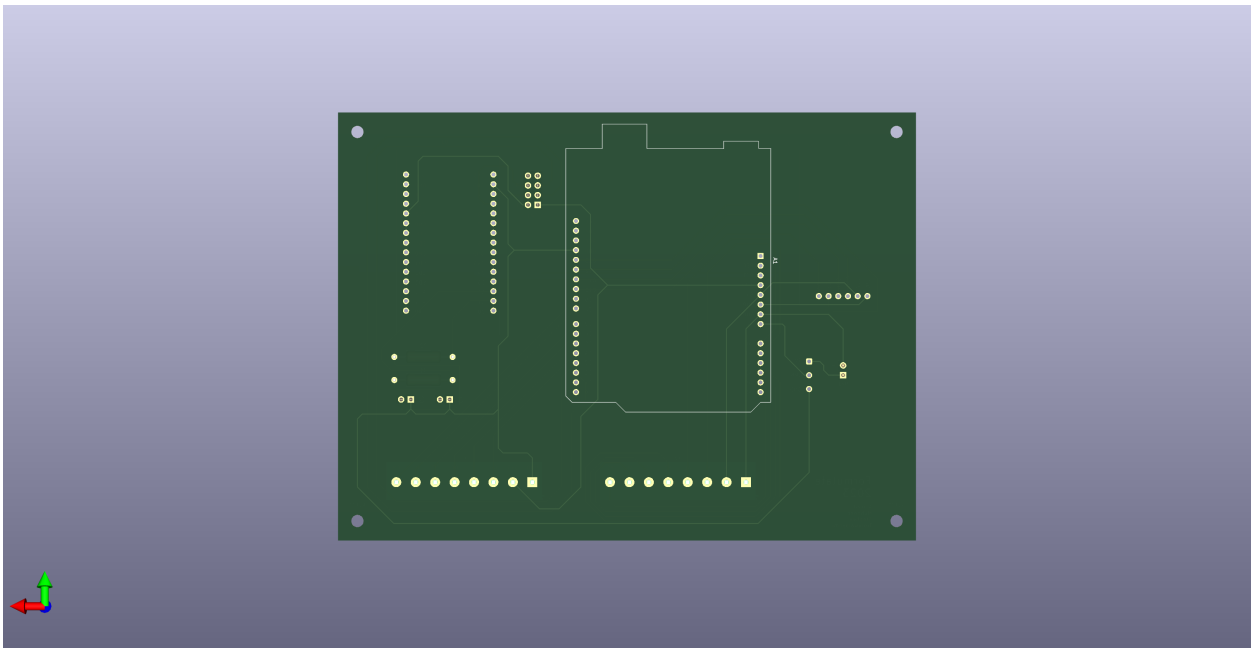

Figure 9: Bottom view of the PCB

D Communication Protocols

E Reflection

E.1 Wireless Communication

1. Some of the limitations of our device would be the range that our device can connect wirelessly. We are currently using an ESP8266 NodeMCU as the wireless module for our device, it acts as an access point and and broadcasts a network which our PC connects to. The NodeMCU operates on a 2.4GHz wifi network which does have long range but if we had unlimited resources we could have used another wireless protocol like LoRaWAN.

E.2 Application GUI

1. Designing GUIs through the Qt library has many advantages including having the Designer tool, which allows you to create the GUI visually instead of manually coding every aspect. Furthermore, it has a variety of classes and functions that allow for flexibility and advanced functionality when handling GUI events such as pressing a button, and have it do something on the application. However, Qt is a huge library that takes time to understand all of its possible features and details. Therefore, due to time constraints of the semester and other course loads, a polished GUI with advanced features may not be feasible since not all features of Qt can be explored.

2. All team members were proficient or at least had experience with python, therefore for programming the GUI, the Tkinter and PyQt libraries were considered. The advantages of using Tkinter were that it is a native python library meaning it was easy to incorporate and execute in a python script. In addition, the Tkinter's API is simple to understand allowing for fast creation and testing of GUIs. However, the simplicity of Tkinter was ultimately its downfall since PyQt allowed for more advanced animation and objects inside the GUI. Therefore, PyQt was chosen to build a more polished GUI that is presentable to a Formula E team.

E.3 PCB Layout

1. Minimal PCB optimization was made to the layout. Primarily due to the time limitation of a lengthy and costly design, manufacture, test cycle for each PCB iteration, it was not feasible to optimize important PCB characteristics such as the absolute minimal layout size, noise minimization, and maximum structural rigidity. As a result, the team focused on achieving the functional solution in the shortest amount of time which limited optimization considerations.

2. A breadboard circuit with jumper cables and pin header connections was a simpler alternative solution to a PCB design. The breadboard circuit had some benefits such as the inexpensive monetary cost to design, manufacture, and test, and the short time to complete a complete circuit iteration. With that said however, the breadboard

circuit lacked the ability to create robust wire connections between components and the ability to design a circuit with a smaller physical footprint. As a result, a PCB layout which functionally replaced the breadboard circuit was chosen as the ability to design a physically smaller circuit with rigid connections through soldered points and terminal blocks was possible.