

**ECEN404**

**Electrical Design Laboratory II – Spring 2017**

**Submission Date: March 27, 2017**

**Senior Design Project - Progress Report**

**Shell Eco Marathon**

**Team Members:**

**Mohammed Hallawa (ECEN)**

**Ahmad Mokahal (MEEN)**

**Mohab Hanbal (MEEN)**

**Tariq Atiyah (MEEN)**

**Jayson El Turk (MEEN)**

**Doaa Awad (MEEN)**

**Mohamed Mohamed (MEEN)**

**Jawad Yammine (MEEN)**

**Mentor: Supervisor:**

**Dr. Michael Schuller Dr. Ziyad Shafik**

**Mr. Muhammad Bin Yousaf**

**“On my honor, as an Aggie, I have neither given nor received**

**Unauthorized aid on this academic work”**

**Table of Content:**

Abstract …………………………………………………………………………………………………………… 2

Chapter 1 – Literature Review ……………………………………………………………………… 3

* 1.1: Introduction ……………………………………………………………………………… 3
* 1.2: Customer Need Study ………………………………………………………………. . 3
* 1.3: Semester Plan ……………………………………………………………………………. 5

Chapter 2 – Project Design and Development Process ………………………………………… 6

* 2.1: Project Benchmarking …………………………………………………………........ 6
* 2.2:Block Diagram …………………………………………………………………………….. 9
* 2.3, 2.4: Concept Generation and Selection…………………………………….. 10

Chapter 3 – Detailed System Design and Modeling ……………………………………………. 12

* 3.1, 3.2: System Design and Modeling/ Simulation Results ……………….. 12
* 3.3: Component List / Budget ……………………………………………………………. 17

Chapter 4 – Experimental Results ……………….……………………………………………… 17

Chapter 5 – Conclusion and Progress …………………………………………………………… 18

References ……………………………………………………………………………………………………… 19

Appendices ………………………………………………………………………………………………… 20

* Appendix A …………………………………………………………………………… 20
* Appendix B …………………………………………………………………………… 30
* Appendix C …………………………………………………………………………… 31
* Appendix D …………………………………………………………………………… 32
* Appendix E …………………………………………………………………………… 33
* Appendix F: ………………………………………………………………………… 34

**Abstract:**

The effects of energy fuels used daily by vehicles is a huge issue that environmentalists are stressing about. Therefore energy companies are trying their best to create a solution for this problem. Cleaner energy fuels are always an option when trying to reach a solution and this creates opportunities around the world. Each year around the world the Shell Eco marathon competition takes place, which will take place in Singapore for the year 2017. The aim of this competition is to encourage college students to create and design cars that are energy efficient. The main purpose of this project is to create a vehicle that would compete and cope with the changes happening in the world right now. This means that car manufacturers which are students in this case should start and focus on how to build a car that would use the least amount of fuel for the longest distance possible. In this project the group will concentrate on creating a vehicle that could adapt to a small amount of energy (fuel), by creating a car that is light weight, also has great body dimensions to be optimum when it comes to aerodynamics. The electrical part of this project is to design and implement all the electrical circuits the car needs, including all the auxiliary systems. In addition to that, to design and create a “Vehicle Monitoring System”, that will act as the first aid and the main safety tool for the driver during the competition.

Throughout this report a literature review will be included to explain how a customer need analysis/study was implemented and what affected the methodology approach. In addition to that a process of the project’s design and process will be talked about. A very detailed system design and modeling will also be included to explain how the monitoring system works, and what was included in the circuit as a whole.

**Chapter 1 – Literature Review:**

**1.1: Introduction:**

In this year’s competition Texas A&M Qatar was the only university to operate in the GTL diesel category. This was chosen to mime the technological approach Qatar is taking in this field and to stress on the importance of such fuel. When at the competition most of the technical inspection team were curious on why we have chosen this category and how the vehicle would perform. The vehicle passed all inspections and passed the electrical inspection, and most eyes were on the VMS installed in front of the driver.

**1.2: Customer Need Study:**

As mentioned before, in this project there are two main customers, one being Shell and including all their rules and regulations. The other important customer is the MEEN department at Texas A&M Qatar. The first customer needs are found in the rules and regulations handbook given by the committee of the Shell Eco Marathon. The second customer needs is based internally coming from the MEEN department which include the needs to build the optimal car for this year’s competition.

|  |  |
| --- | --- |
| For safety reasons, the maximum voltage on board of any vehicle at any point must not exceed 48 Volts nominal and 60 Volts max (this includes on-board batteries, external batteries, Super Capacitors, fuel cell stack, etc.). | Battery chosen:  12 Volts  36 Ampere |
| For all vehicles only one on-board battery is allowed | Which is what is implemented in the design |
| Both propulsion and accessory batteries must be installed outside of the Driver’s compartment behind the bulkhead. | Battery was placed at the back compartment next to the engine |
| The vehicle horn is required to be powered by a built-in battery. | A DC motor is directly connected to both the wipers and horn. |
| Electrical wiring should be in good condition, neat, clearly labelled, secured and not close to any moving parts | Implemented as shown in the schematic design shown in the appendences, in addition to using clearly labeled junction boxes |

The table above shows that the needs from the competition committee is met and is all shown in the schematic and car’s electrical wiring layout shown in the appendices. All wiring will be also labeled and avoid “spaghetti wiring” as mentioned in the requirements. All wiring followed a specific path in a conduit that was placed in the driver’s hub. The battery that was picked for the competition will be shown later on and the reasoning behind it. Due to only using one battery, all the lighting that was used was LED due to its low usage to power.

**MEEN Department Needs:**

|  |  |
| --- | --- |
| All Wires preferably should be done on the right side of the car to avoid area of driver. | This was implemented as shown in the wiring layout in the appendences. |
| The engine started should be electric. | Also as shown in the appendices, a wire connecting the battery to the engine would be used. Specification to this wire were chosen already, with bigger cross sectional area. |
| The engine should include a solenoid valve that cuts off the fuel from reaching the engine and it should be functioned electrically. | Already implemented and ordered from the electrical team, with three dead man switches close to the driver. |
| Reasoning behind why the electrical team needs the battery on the front side of the car. | Reasoning was given which will be included under the table. |
| All electrical equipment in the car should be light weight and consume power as small as possible from the battery. | LED lights, Raspberry PI and all systems use very little power to function. |

The main concern in the case of our project whether it is from the rules and regulations by Shell eco marathon and necessities by the MEEN department is safety. Safety of both the car in general and the driver in specific during the competition and race. The battery needs to be low voltage to avoid big incidents if the battery had a defect. The solenoid switch is needed to cut off the fuel supply to the engine in the case of a hazard. All the electrical wiring system is on the right side of the car to avoid any contact with the driver, coming in and out of the car. Battery was preferred to be in the front side for various reasons including:

1. Avoid the excessive heat produced by the engine
2. Avoid bumping into the mechanical team during the competition.
3. Closer to the fuse box and all the switches that will be implemented.
4. Mechanical team had the concern of power dissipation in the wire running from the engine to the battery for the starter, but due to the picking of a bigger cross sectional area of the wire, power dissipation or voltage drops is minimal or close to zero.

**1.3: Semester Plan:**

When it comes down to the plan during this semester, deadlines were met including finishing off the VMS, Implementing the circuit design on the vehicle and later on implementing the VMS controls all around the vehicle. Car was shipped to Singapore late February and participated in the competition from the 16th- 19th of March.

The VMS that was used during the race included the following,

* Touch Screen buttons to operate the electrical system around the vehicle.
* Speedometer that received its signals from a hall effect Sensor
* Timer, which indicates to the driver how much time he is taking around the track.

What would be added to the VMS after this point:

* Flow Meter Sensor and reading to show Fuel consumption.
* Temperature Sensor which would give its signals to a gauge on the screen
* Warning system, including over speeding and overheating.

Due to time constraints and priorities by the Mechanical team, the flow meter and temperature sensor were not included. I believe that any VMS should include these things especially a vehicle in a fuel efficiency race. Any system that operates for such a cause should also include a warning system which would be very vital in the safety category.

**Chapter 2 – Project Design and Development Process:**

**2.1: Project/product benchmarking**

In this part of the report the main concern is to view different type of products that would be used to achieve the overall final project. Different materials and equipment for both the electrical and mechanical aspects of the car will be shown, mentioned and discussed. Most of the equipment that was chosen had two main advantages, first it met with the calculations done and had the weight desired to keep the car light. The team’s main aim was to keep the car with the minimum weight as possible to have an advantage during the race in March.

**Car Battery:**

|  |  |  |
| --- | --- | --- |
|  | LFX 24L3 – BS12 | LFX 36L3 – BS12 |
| Volts | 12 V | 12 V |
| Amps | 24 A | 36 A |
| Weight | 1.80 Kg | 2.19 Kg |
| Length, Width, Height | 6.55 3.39 6.10  (inches) | 6.55 3.39 6.10  (inches) |

Looking at the table above, one could see that both batteries have the same voltage, same dimensions too, but the Ampere is different. The second battery has a higher ampere and after research one could notice that the higher ampere is much better when starting the engine. Higher cranking which improves the starting, and minimizes power dissipation. Choosing the second battery had a great advantage due to the fact that the starter motor placed on the engine used a lot of power allowing the voltage to drop and the current to increase. Having the higher Ampere allowed us to contain this process and not burn any fuses or breakers.

**Sensors:**

All Sensors being bought are shown in the appendences section which include:

1. Tire Pressure sensors. 4- Speed Sensor
2. Oil Temperature sensor. 5- Engine Bay temperature sensor
3. Fuel flow Meter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Engine | Weight | L48N | L70N | L100N |
| A | Mass | 0.22 | 42.23 | 32.96 | 25.26 |
| B | Cost | 0.11 | 38.58 | 34.72 | 26.71 |
| C | Power Output | 0.33 | 18.87 | 34.91 | 44.81 |
| D | Size | 0.11 | 44.05 | 31.90 | 24.05 |
| E | Reliability | 0.22 | 33.33 | 33.33 | 33.33 |
|  |  | Total | 32.26 | 33.77 | 33.60 |

All sensors bought can handle very high temperatures, to ensure none get damaged when used.

**Vehicle’s Engine:**

Looking at the table above, the engine in the middle came out to be the most suitable one due to its weight and power output. It is the engine that would produce the amount of torque needed by the calculations done (65 N.m), and at the same time the lightest in its category.

**Vehicle’s Transmission:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Criteria | Weight |  | | | |
| Continuously Variable transmission | 1 speed transmission | 5 speed manual transmission | 3 gear transmission (neutral/reverse/forward) |
| Mass | 0.286 | 20 | 50 | 10 | 20 |
| Cost | 0.190 | 30 | 35 | 15 | 20 |
| Operation difficulty | 0.048 | 40 | 40 | 5 | 15 |
| Installation difficulty | 0.048 | 25 | 30 | 20 | 25 |
|  |  |  |  |  |  |
| Reliability | 0.143 | 25 | 30 | 25 | 20 |
| Durability | 0.048 | 25 | 25 | 25 | 25 |
| Fuel efficiency | 0.238 | 35 | 20 | 35 | 10 |
| Total weight | 1 | 27.645 | 34.56 | 20 | 17.86 |

From the table below one can see that due to efficiency in weight and being able to move the car at the same time, the one wheel transmission was the safest option to go with.

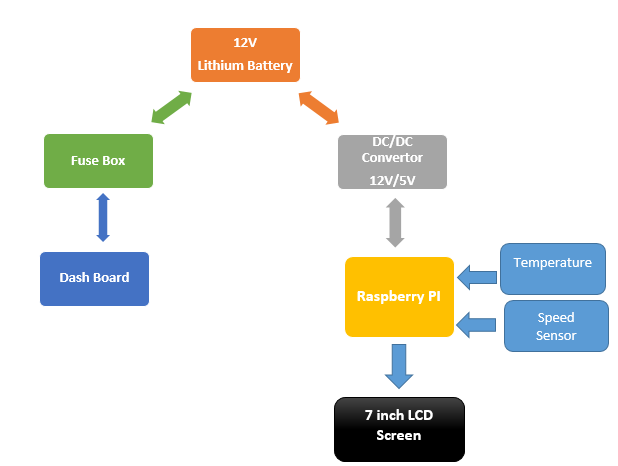
**Body Design Material:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Body Concepts** | **A** | **B** | **C** | **D** | **E** | **F** | **Total** | **Weight** |
| **A** | **Safety** | / | 1 | 1 | 1 | 1 | 1 | 5 | 0.53 |
| **B** | **Mass** | 0 | / | 1 | 1 | 1 | 1 | 4 | 0.27 |
| **C** | **Cost** | 1 | 1 | / | 1 | 1 |  | 3 | 0.20 |
| **D** | **Reliability** | 0 | 0 | 0 | / | 1 | 1 | 2 | 0.13 |
| **E** | **Ease of manufacturing** | 0 | 1 | 1 | 0 | / | 1 | 1 | 0.07 |
| **F** | **Factor** | 0 | 0 | 0 | 0 | 0 | / | 0 | 0 |
|  |  |  |  |  |  | Total |  | 15 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Weight** | **Carbon Fiber** | **Aluminum (6061)** | **Balsa Wood** | **Steel (1018)** |
| A | Mass | 0.33 | 40 | 20 | 30 | 10 |
| B | Cost | 0.13 | 10 | 30 | 35 | 25 |
| C | Ease of Manufacturing | 0.07 | 28 | 22 | 20 | 30 |
| D | Reliability | 0.20 | 30 | 30 | 25 | 15 |
| E | Safety | 0.27 | 25 | 25 | 20 | 30 |
|  |  |  | 29.2 | 24.8 | 26.3 | 19.7 |

Also from the above table, one could see that carbon fiber scored the lowest when it comes to cost but scored the highest in the other categories. This was not a problem though and carbon fiber came out to rank # 1 against other materials. Due to budget constraints, complexity of the Carbon Fiber design, and the availability of material, aluminum was used to build the body instead.

**2.2: Functional Block Diagram:**

****

The main power supply is stationed in the middle which is a lithium battery as discussed above which generates 12 volts and 36 Amps. This supply is used for two main functions, one being the auxiliary system of the car which is shown in the dashboard. It is used to power the engine, indication lights, lights, brake lights, horn and wiper. The engine had a thicker wire to make sure no voltage drop or power dissipation occurs. Before the switches there will be two fuse boxes, one located at the front of the car and the other at the back, first to have the connections between the buttons and their functions also to protect the wiring from external affects.

The second function is to power the vehicle monitoring system, which has to go through a DC to DC convertor first, this is implemented to drop down the voltage from 12 volts to 5 volts. The vehicle monitoring system is operated using raspberry pi 3. This is written using C-coding and will operate by receiving data from the inputs (sensors) shown above which will be later on displayed on an LCD (7 inch) screen as the outputs on the dashboard for the driver to view.

**2.3, 2.4: Concept generation and selection:**

|  |  |
| --- | --- |
| **Subsystem** | **Chosen Concept** |
| **Engine** | Yanmar L70N |
| **Fuel** | GTL Diesel |
| **Transmission** | One Speed Transmission |
| **Differential** | No Differential (One Wheel Drive) |
| **Body Shell Design** | Hybrid |
| **Body Shell Material** | Aluminum |
| **Frame** | Replace Aluminum Sheets with Balsa Wood |
| **Suspension** | No Suspension |
| **Brake** | Hydraulic Brake System |
| **Steering** | Mechanical Go Kart Steering |
| **Battery** | LFX36L3-BS12 (12 V, 36 A) |
| **VMS Programmer** | Raspberry PI 3 |
| **Sensors** | (As shown in the Appendices) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Complexity of design** | **Failure probability** | **Modification /innovation** | **Torque to weight** | **Experience/ competitiveness** | **Sponsor** | **Cost** | **Part availability** | **Total** |
| **Weight** | **0.139** | **0.222** | **0.056** | **0.056** | **0.056** | **0.19** | **0.17** | **0.111** | 1 |
| **GTL Diesel** | 40 | 40 | 33.33 | 40 | 37.5 | 33.33 | 35 | 50 | **38.47918** |
| **Gasoline** | 30 | 30 | 33.33 | 30 | 37.5 | 33.33 | 35 | 30 | 32.08918 |
| **Electric** | 30 | 30 | 33.33 | 30 | 25 | 33.33 | 30 | 20 | 29.42918 |

GTL diesel is a synthetic diesel substitute that is derived from natural gas. The state of Qatar is one of the leading countries that are rich with the amount of natural gas found. This type of fuel contains zero sulfur content and very low emissions, which makes this diesel both clean and friendly to the environment. The demand of energy is universally expanding and other alternatives are being developed to help keep up with the demand, therefore, the rise in use of GTL diesel all around the world.

Looking at the table above it explains why GTL diesel is the better option between its competitors. This is due to the reasons that it is the easiest to design, costs the lowest and has the best torque to weight conversion. This information can show that it is beneficial because of low cost, and the best fuel that converts power onto the wheels, and most efficient one.

When it comes to develop the interactive monitoring system, the system shall use Raspberry PI 3, as the processing unit to display the measured parameters on a 7 inch screen.

|  |  |
| --- | --- |
| Arduino | Raspberry PI 3 |
| Has limited libraries and difficult to implement in future upgrades | It is Linux based and offers various libraries, for future comp and upgrades |
| Limited processing Power (516 MHZ) | Big Processing Power (1.2 GHZ) |
| Storage (32 KB) | Storage (512KB) |
| N/A | HDMI Cable for output |

It was easy to choose between the two options above, the raspberry PI gives the user many more opportunities. This includes having larger space, operating at a much faster rate and also allows one to operate freely with no complications. This means that one can use the Raspberry PI to receive data from inputs and provide it on an output. In this case the Raspberry Pi would be reading the sensors and displaying the results on the 7 inch LCD screen that would be displayed in front of the driver during the race. Using the raspberry Pi allowed us to create an interface that includes, widgets (Buttons), Speedometer by including wheel radius, and a timer. Listed in the next chapter is the code that was used to design and create the GUI product also shown in the appendices.

**Chapter 3: Detailed System Design and Modeling:**

**3.1, 3.2: System Design and Modeling/Simulation Results.**

Taking a closer look on how the VMS was designed and created. Raspberry PI model 3 was used in the favor of its technological advantages. This included having an HDMI cable output which allowed us to view the GUI onto the LCD screen. Other advantages using this model included having various ports and a plug for an SD card.

The Vehicle monitoring system created is basically a setup and a group of codes taken from a library called Kivy. Kivy allows the user to access a global library which includes steps and codes on how to build a graphical user interface. Kivy allows the user to create a program that could be used with a touch screen. Later on the tools to create this software, such as installing images, widgets, labels, bars and toggle buttons were then imported from Kivy too. Most importantly the GPIO which are the pins located on the raspberry PI board were imported too.

This is shown in the set of codes shown below:

import kivy

kivy.require('1.0.6') # replace with your current kivy version !

from kivy.app import App

from kivy.uix.button import Button

from kivy.uix.togglebutton import ToggleButton

from kivy.properties import NumericProperty

from kivy.properties import StringProperty

from kivy.properties import ObjectProperty

from kivy.properties import BoundedNumericProperty

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.widget import Widget

from kivy.uix.scatter import Scatter

from kivy.uix.image import Image

from kivy.uix.label import Label

from kivy.uix.progressbar import ProgressBar

import os,inspect

from time import sleep

import time, math

from functools import partial

#from kivy.uix.anchorlayout import AnchorLayout

from kivy.uix.floatlayout import FloatLayout

from kivy.uix.image import Image

from kivy.uix.slider import Slider

from kivy.clock import Clock

from kivy.graphics import Color, Rectangle

import RPi.GPIO as GPIO

The second step is and setting up the touch buttons on the screen and adjusting its sizes and the labeling on the buttons. Creating the button had to be done by giving the sized required and its position (Coordinates) on the screen display. The step after that is allocating each button to the pin it will be connected to. Those pins are then connected to the relay board which gives the command to the outputs. This is shown below:

# Create the rest of the UI objects (and bind them to callbacks, if necessary):

                highbeam\_button = ToggleButton(text="High\nBeam\nLights", halign='center', size\_hint=(.14,.25),pos = (2,20), font\_size=25)

                highbeam\_button.bind(on\_press=press\_callback)

                lowbeam\_button = ToggleButton(text="Driving\nLights", halign='center' , size\_hint=(.14,.25),pos = (116,20), font\_size=25)

                lowbeam\_button.bind(on\_press=press\_callback)

                leftsig\_button = ToggleButton(text="Left\nSignal", group='signals', halign='center' , size\_hint=(.14,.25),pos = (230,20), font\_size=25)

                leftsig\_button.bind(on\_press=press\_callback)

                hazard\_button = ToggleButton(text="Hazard", group='signals',size\_hint=(.14,.25),pos = (344,20), font\_size=25)

                hazard\_button.bind(on\_press=press\_callback)

                rightsig\_button = ToggleButton(text="Right\nSignal", group='signals',halign='center',size\_hint=(.14,.25),pos = (458,20), font\_size=25)

                rightsig\_button.bind(on\_press=press\_callback)

                wiper\_button = ToggleButton(text="Wiper",size\_hint=(.14,.25),pos = (572,20), font\_size=25)

wiper\_button.bind(on\_press=press\_callback)

                horn\_button = Button(text="BEEP!",size\_hint=(.14,.25),pos = (686,20), font\_size=25)

                horn\_button.bind(on\_press=press\_callback)

shutdown\_button = Button(text="Bye :)",size\_hint=(.14,.14),pos = (686,410), font\_size=25)

                shutdown\_button.bind(on\_press=shutdown\_func)

                #wimg = Image(source='texas\_logo.jpeg', scale = 0.5, pos = (200,100))

# Add the UI elements to the layout:

                layout.add\_widget(mygauge)

layout.add\_widget(mystopwatch)

                layout.add\_widget(highbeam\_button)

                layout.add\_widget(lowbeam\_button)

                layout.add\_widget(leftsig\_button)

                layout.add\_widget(hazard\_button)

                layout.add\_widget(rightsig\_button)

                layout.add\_widget(wiper\_button)

                layout.add\_widget(horn\_button)

layout.add\_widget(shutdown\_button)

# Set up GPIO:

lowbeampin = 27

highbeampin = 22

hazardpin = 23

wiperpin = 24

hornpin = 25

leftsignalpin = 14

rightsignalpin = 15

GPIO.setmode(GPIO.BCM)

GPIO.setup(lowbeampin, GPIO.OUT)

GPIO.output(lowbeampin, GPIO.HIGH)

GPIO.setup(highbeampin, GPIO.OUT)

GPIO.output(highbeampin, GPIO.HIGH)

GPIO.setup(hazardpin, GPIO.OUT)

GPIO.output(hazardpin, GPIO.HIGH)

GPIO.setup(wiperpin, GPIO.OUT)

GPIO.output(wiperpin, GPIO.HIGH)

GPIO.setup(hornpin, GPIO.OUT)

GPIO.output(hornpin, GPIO.HIGH)

GPIO.setup(leftsignalpin, GPIO.OUT)

GPIO.output(leftsignalpin, GPIO.LOW)

GPIO.setup(rightsignalpin, GPIO.OUT)

GPIO.output(rightsignalpin, GPIO.LOW)

GPIO.setwarnings(False)

GPIO.setup(sensor,GPIO.IN,GPIO.PUD\_UP)

The main advantage of having the Raspberry PI in the car was to program the system to act as a relay for the signal lights and hazard. This was done to elimante the usage of physical timers, which in the past has always been a problem. Problems rised when the timers kept burning out and casued issues when going into technical inspections. This was created and is shown in the code below. The code works on having the light going on and off with a time delay of 0.75 seconds:

        if obj.text == 'Hazard':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(leftsig\_toggle)

Clock.unschedule(rightsig\_toggle)

                        GPIO.output(rightsignalpin, GPIO.HIGH)

GPIO.output(leftsignalpin, GPIO.HIGH)

Clock.schedule\_once(hazard\_toggle, 0.75)

else:

Clock.unschedule(hazard\_toggle)

GPIO.output(rightsignalpin, GPIO.LOW)

GPIO.output(leftsignalpin, GPIO.LOW)

        if obj.text == 'Left\nSignal':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(rightsig\_toggle)

Clock.unschedule(hazard\_toggle)

                        GPIO.output(leftsignalpin, GPIO.HIGH)

Clock.schedule\_once(leftsig\_toggle, 0.75)

else:

Clock.unschedule(leftsig\_toggle)

GPIO.output(leftsignalpin, GPIO.LOW)

        if obj.text == 'Right\nSignal':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(leftsig\_toggle)

Clock.unschedule(hazard\_toggle)

                        GPIO.output(rightsignalpin, GPIO.HIGH)

Clock.schedule\_once(rightsig\_toggle, 0.75)

else:

Clock.unschedule(rightsig\_toggle)

GPIO.output(rightsignalpin, GPIO.LOW)

After that came the designing of the speedometer, this is operated by creating the background with numbers as shown in the final image below. Then a needle was imported to indicate the speed, the needle moves from left to right on the shape of a semicircle. Its position on this semi-circle was an input from the speed sensor. This code is quite lengthy and can be seen in the Appendices.

Creating the speedometer depended on a Hallefect speed sensor, which operates using a magnet. The sensor was placed on the car chassis, while the magnet was places on the rotating wheel, once the sensor caught a magnetic field it would record it. A digital number indicating the speed was also placed in the center. Kivy had a code that asked for the radius of the wheel and was then integrated to our own as shown below:

# changed

#variables for speed calculations:

dist\_meas = 0.00

km\_per\_hour = 0

rpm = 0

elapse = 0

sensor = 17

pulse = 0

def update\_speed(dt):

value = calculate\_speed(25)

setgauge(0,value)

One of the most vital tools on the screen so far is the timer that lets the driver know the time he is taking to go around the track. Again, the code is very similar to the one for the buttons, allocating the size of the buttons and their location. How it functions is showed in the full code in the Appendices.

Due to time constraints and priorities for the competition, only the applications explained above were implemented. Things to be added include, a flow meter reading indicating to the driver and crew how much fuel is being consumed during the race. In addition to that a temperature sensor to give its signals to a temperature gauge on the screen to inform the driver of the temperature of the oil and lubricants running in the engine.

On top of that, warning messages will be added which will come up on the screen in such cases:

* Over speeding.
* Great increase in Temperature located in the Engine compartment
* Going over the time limit.
* Temperature increase in the oil running in the engine
* A noticeable drop in the battery voltage during the race

**3.3: Component List/Budget:**

|  |  |  |
| --- | --- | --- |
| Material | Quantity | Price |
| Speed/Distance Sensor | 2 | $5 |
| Oil Temperature Sensor | 2 | $30 |
| Fuel Flow Meter/ Solenoid Valve | 2 | $70 |
| Engine Chamber Temperature sensor | 3 | $35 |
| Battery Compartment temperature sensor | 2 |
| Tire Pressure Sensor | 8 | N/A |
| Raspberry PI 3 + GPS | 2 | $40 |
| LCD Screen + Case | 2 | $200 |
| LED Lights |  | $100 |
| **Total** | | **$500** |

**Chapter 4: Experimental Results:**

This is a type of project that is not waiting on results; but it is building a vehicle’s circuit which includes every small detail from the power system (Battery) all the way to the small pins on a raspberry PI board.

The first part of this project was to design and build the electrical circuit around the vehicle, this is shown in details in the car wiring layout in the appendices. Two fuse boxes were place in the vehicle, one being at the front side of the car and the other at the back. Those were used to make sure wires were correctly labeled and that things were organized. It also acted as a safety measure to ensure that no loose wires were found or exposed wires. In the technical inspection there was a category that included the checking of all the electrical wiring and it passed all safety measures and of course its functionality.

The second part of this project was the vehicle monitoring system, which had to be on because of its usage of timers, which functioned the signals and hazard. All eyes were on the 7 inch LCD screen, and many people were curious on why it was there and how it worked and why it was important. It received a thumbs up and a huge smile from the whole technical inspection team and they assured our team on how important this technology is for the team.

The aim of this project is to create a touch screen system that would act as the main tool in the driver’s hub. The driver would be able to function any of the auxiliary systems such as the lights, wiper or horn and at the same time make sure he is safe. This is done by having a speedometer, a temperature gauge and a timer.

Most of the drivers during the race were in contact using a phone with the driver telling them how fast they are going, how much time has passed and how much was left. Our team’s driver had all this information in front of his eyes, and did not need any means of communication with the pit crew. With adding the things left the driver will have a full system just the ones found in commercial cars.

Our vehicle finished a total of 10.17 Km, in the 25:08, 25 minutes and 8 seconds while having an efficiency of 70 km per liter of GTL diesel.

**Chapter 5: Conclusion**

Our vehicle passed all the strict technical inspections, and was the first car for TAMUQ to do so in the last four years. It preformed quite well around the track too, not only we participated in the race but also very close to accomplish all goals. The vehicle was only eight seconds late around the track and that was due to fault by another team on the track, when their vehicle didn’t start and therefore we were delayed. When comparing our fuel efficiency and fuel consumption, it was consuming a bit under what a commercial car would consume. That placed us in the middle averaging the ranges and numbers that were reached.

The electrical circuit worked with zero faults and was always functioning when needed, The VMS was very helpful to the driver and was always operating on point despite the very high temperatures and heavy vibrations produced by the engine. The VMS showed its importance when mostly all the technical team applauded the idea and its importance for such a race. Efficiency is very vital these days and monitoring it will indicate to the crew were the problem is allowing solutions to be found easily. With the few additions as mentioned above the VMS would be one of the best systems a driver could have in his/her vehicle.

**References:**

1. "FROM DESIGN TO PRODUCT - Solar Team Eindhoven". *Solar Team Eindhoven*. N.p., 2015. Web. 20 Oct. 2016..
2. "SAMAGAGA". *Samagaga.com. N* .p., 2016. Web. 20 Oct. 2016.
3. "Shell Eco-Marathon, Rules 2017". *shell.com. N* ..p., 2016. Web. 20 Oct. 2016.
4. "Home - Yanmar Marine". *Yanmarmarine.com*. N.p., 2016. Web. 20 Oct. 2016.
5. http://www.shell.com/energy-and-innovation/shell-ecomarathon/asia/news-and-highlights/\_jcr\_content/pagePromo/image.img.1000.jpeg/1456915443294/shell-eco-marathon-2015-track.jpeg
6. http://photos.prnewswire.com/prnvar/20120917/MM76045LOGO?max=1600
7. http://img.cdn2.vietnamnet.vn/Images/english/2015/02/13/16/20150213164336-eco-marathon.jpg
8. http://shellecomarathon.newsweaver.co.uk/files/1/31964/56654/5550191/83924f8fcc9b533967d2e880/sem\_newsweaver\_header\_asia\_banner2\_v1\_0.jpg
9. http://www.eventindustrynews.co.uk/wp-content/uploads/2015/06/SEM-Asia-2014-Track-Photo-2.jpg
10. http://www.shell.com/energy-and-innovation/shell-ecomarathon/about.html
11. http://www.zinc.uk.com/wp-content/uploads/2012/05/shell-eco-marathon-day-4-2012-5-19-8-20-38.jpg
12. https://www.google.com/search?q=raspberry+pi+screen+with+gauges&espv=2&biw=1366&bih=667&source=lnms&tbm=isch&sa=X&ved=0ahUKEwi-4JPcgpbPAhWD6RQKHVd8Cw4Q\_AUIBygC#imgrc=2LGikGzqZvVxPM%3A
13. http://www.shell.com/promos/download-the-shell-eco-marathon-global-rules/\_jcr\_content.stream/1472744744388/7b204129356de26d47c9f21e2c8f58e07f3e2e4f266ae7714c8e6b17bb7051e1/sem-2017-rules-chapter-1.pdf

**Appendices**

**Appendix A: (Vehicle Monitoring System Code):**

import kivy

kivy.require('1.0.6') # replace with your current kivy version !

from kivy.app import App

from kivy.uix.button import Button

from kivy.uix.togglebutton import ToggleButton

from kivy.properties import NumericProperty

from kivy.properties import StringProperty

from kivy.properties import ObjectProperty

from kivy.properties import BoundedNumericProperty

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.widget import Widget

from kivy.uix.scatter import Scatter

from kivy.uix.image import Image

from kivy.uix.label import Label

from kivy.uix.progressbar import ProgressBar

import os,inspect

from time import sleep

import time, math

from functools import partial

#from kivy.uix.anchorlayout import AnchorLayout

from kivy.uix.floatlayout import FloatLayout

from kivy.uix.image import Image

from kivy.uix.slider import Slider

from kivy.clock import Clock

from kivy.graphics import Color, Rectangle

import RPi.GPIO as GPIO

# changed

#variables for speed calculations:

dist\_meas = 0.00

km\_per\_hour = 0

rpm = 0

elapse = 0

sensor = 17

pulse = 0

start\_timer = time.time()

# Set up GPIO:

lowbeampin = 27

highbeampin = 22

hazardpin = 23

wiperpin = 24

hornpin = 25

leftsignalpin = 14

rightsignalpin = 15

GPIO.setmode(GPIO.BCM)

GPIO.setup(lowbeampin, GPIO.OUT)

GPIO.output(lowbeampin, GPIO.HIGH)

GPIO.setup(highbeampin, GPIO.OUT)

GPIO.output(highbeampin, GPIO.HIGH)

GPIO.setup(hazardpin, GPIO.OUT)

GPIO.output(hazardpin, GPIO.HIGH)

GPIO.setup(wiperpin, GPIO.OUT)

GPIO.output(wiperpin, GPIO.HIGH)

GPIO.setup(hornpin, GPIO.OUT)

GPIO.output(hornpin, GPIO.HIGH)

GPIO.setup(leftsignalpin, GPIO.OUT)

GPIO.output(leftsignalpin, GPIO.LOW)

GPIO.setup(rightsignalpin, GPIO.OUT)

GPIO.output(rightsignalpin, GPIO.LOW)

GPIO.setwarnings(False)

GPIO.setup(sensor,GPIO.IN,GPIO.PUD\_UP)

# Define some helper functions:

class StopWatch(Widget):

    def \_\_init\_\_(self, \*\*kwargs):

        super(StopWatch, self).\_\_init\_\_(\*\*kwargs)

        self.laplbl = Label(text="test",font\_size=25, markup=True)

        self.add\_widget(self.laplbl)

        self.timelbl = Label(text="test",font\_size=50, markup=True)

        self.add\_widget(self.timelbl)

        self.startbtn = Button(text="Start", size=(112,120), font\_size=25)

        self.add\_widget(self.startbtn)

        self.startbtn.bind(on\_press=self.start\_time)

        self.stopbtn = Button(text="Stop", size=(112,120), font\_size=25)

        self.add\_widget(self.stopbtn)

        self.stopbtn.bind(on\_press=self.stop\_time)

        self.resetbtn = Button(text="Reset", size=(112,120), font\_size=25)

        self.add\_widget(self.resetbtn)

        self.resetbtn.bind(on\_press=self.reset\_time)

        self.lapbtn = Button(text="Lap", size=(112,120), font\_size=25)

        self.add\_widget(self.lapbtn)

        self.lapbtn.bind(on\_press=self.lap\_time)

        self.laplbl.pos = (525,235+100)

        self.laplbl.max\_lines=5

        self.timelbl.pos = (525,235)

        self.startbtn.pos = (344,142)

        self.stopbtn.pos = (458,142)

        self.resetbtn.pos = (572,142)

        self.lapbtn.pos = (686,142)

    #gloabl variables

    seconds = 0

    minuets = 0

    mili = 0

    \_mili = 0

    time\_track = StringProperty("00:00:00")

    laps = []

    running = False

    resetting = True

    addLap = False

    def printLaps(self):

        lap\_string = ""

        count = len(self.laps)

        for lap in reversed(self.laps):

            lap\_string += "Lap " + str(count) +": " + lap + "\n"

            count -=1

        return lap\_string

    def update(self,dt):

        self.timelbl.text = str(self.time\_track)

        if not self.laps:

            self.laplbl.text = ""

        else:

            self.laplbl.text = self.printLaps()

        if(self.running):

            if(self.seconds < 10 and self.minuets < 10 and self.mili < 10):

                self.time\_track = "0" + str(self.minuets) + ":0" + str(self.seconds) + ":0" + str(self.mili)

            elif(self.seconds < 10 and self.minuets < 10 and self.mili >= 10):

                self.time\_track = "0" + str(self.minuets) + ":0" + str(self.seconds) + ":" + str(self.mili)

            elif(self.seconds >= 10 and self.minuets < 10 and self.mili < 10 ):

                self.time\_track = "0" + str(self.minuets) + ":" + str(self.seconds) + ":0" + str(self.mili)

            elif(self.seconds >= 10 and self.minuets < 10 and self.mili >= 10):

                self.time\_track = "0" + str(self.minuets) + ":" + str(self.seconds) + ":" + str(self.mili)

            elif(self.seconds >= 10 and self.minuets >= 10 and self.mili < 10):

                self.time\_track = str(self.minuets) + ":" + str(self.seconds) + ":0" + str(self.mili)

            elif(self.seconds < 10 and self.minuets >= 10 and self.mili >= 10):

                self.time\_track = "0" + str(self.minuets) + ":" + str(self.seconds) + ":" + str(self.mili)

            elif(self.seconds < 10 and self.minuets >= 10 and self.mili < 10):

                self.time\_track = str(self.minuets) + ":0" + str(self.seconds) + ":0" + str(self.mili)

            elif(self.seconds >= 10 and self.minuets >= 10 and self.mili >= 10):

                self.time\_track = str(self.minuets) + ":" + str(self.seconds) + ":" + str(self.mili)

            if(self.seconds >= 60):

                self.minuets += 1

                self.seconds = 0

            if(self.mili >= 99):

                self.seconds+=1

                self.mili = 0

                self.\_mili = 0

            self.\_mili += dt\*100

            self.mili = int(round(self.\_mili, 0))

        if(self.resetting == True):

            self.minuets = 0

            self.seconds = 0

            self.mili = 0

            self.time\_track = str(self.minuets) + "0:0" + str(self.seconds) + ":0" + str(self.mili)

            self.laps = []

            self.resetting = False

            self.running = False

        if(self.addLap == True):

            self.laps.append(self.time\_track)

            self.laplbl.text = str(self.time\_track)

            self.addLap = False

    #Starts the timer

    def start\_time(self,dt):

        print("START TIME!")

        self.running = True

    #Stops the timer

    def stop\_time(self,dt):

        print("STOP TIME!")

        self.running = False

    #Reset the timer

    def reset\_time(self,dt):

        print("RESET TIME!")

        self.running = False

        self.resetting = True

    def lap\_time(self,dt):

        print("LAP!")

        self.addLap = True

        print(self.time\_track)

class DummyClass: pass

class Gauge(Widget):

    '''

Gauge class

'''

    dummy = DummyClass

    unit = NumericProperty(1.8)

    value = BoundedNumericProperty(0, min=0, max=100, errorvalue=0)

    mypath = os.path.dirname(os.path.abspath(inspect.getsourcefile(dummy)))

    file\_gauge = StringProperty(mypath + os.sep + "cadran.png")

    file\_needle = StringProperty(mypath + os.sep + "needle.png")

    size\_gauge = BoundedNumericProperty(128, min=128, max=256, errorvalue=128)

    size\_text = NumericProperty(10)

    def \_\_init\_\_(self, \*\*kwargs):

        super(Gauge, self).\_\_init\_\_(\*\*kwargs)

        self.\_gauge = Scatter(

            size=(self.size\_gauge, self.size\_gauge),

            do\_rotate=False,

            do\_scale=False,

            do\_translation=False,

scale=1.6

            )

        \_img\_gauge = Image(source=self.file\_gauge, size=(self.size\_gauge,

            self.size\_gauge))

        self.\_needle = Scatter(

            size=(self.size\_gauge, self.size\_gauge),

            do\_rotate=False,

            do\_scale=False,

            do\_translation=False,

scale=1.6

            )

        \_img\_needle = Image(source=self.file\_needle, size=(self.size\_gauge,

            self.size\_gauge))

        self.\_glab = Label(font\_size=self.size\_text, markup=True)

        self.\_progress = ProgressBar(max=100, height=100, value=self.value)

        self.\_gauge.add\_widget(\_img\_gauge)

        self.\_needle.add\_widget(\_img\_needle)

        self.add\_widget(self.\_gauge)

        self.add\_widget(self.\_needle)

        self.add\_widget(self.\_glab)

        self.add\_widget(self.\_progress)

        self.bind(pos=self.\_update)

        self.bind(size=self.\_update)

        self.bind(value=self.\_turn)

    def \_update(self, \*args):

        '''

Update gauge and needle positions after sizing or positioning.

'''

        self.\_gauge.pos = self.pos

        self.\_needle.pos = (self.x, self.y)

        self.\_needle.center = self.\_gauge.center

        self.\_glab.center\_x = self.\_gauge.center\_x

        self.\_glab.center\_y = self.\_gauge.center\_y + (self.size\_gauge/4)

        self.\_progress.x = self.\_gauge.x+75

        self.\_progress.y = self.\_gauge.y + (self.size\_gauge / 4)

        self.\_progress.width = self.size\_gauge

    def \_turn(self, \*args):

        '''

Turn needle, 1 degree = 1 unit, 0 degree point start on 50 value.

'''

        self.\_needle.center\_x = self.\_gauge.center\_x

        self.\_needle.center\_y = self.\_gauge.center\_y

        self.\_needle.rotation = (50 \* self.unit) - (self.value \* self.unit)

        self.\_glab.text = "[b]{0:.0f}[/b]".format(self.value)

        self.\_progress.value = self.value

dirflag = 1

value = 0

# This callback will be bound to the LED toggle and Beep button:

def press\_callback(obj):

print("Button pressed,", obj.text)

if obj.text == 'BEEP!':

# turn on the beeper:

GPIO.output(hornpin, GPIO.LOW)

GPIO.output(hornpin, GPIO.HIGH)

GPIO.output(hornpin, GPIO.LOW)

# schedule it to turn off:

Clock.schedule\_once(horn\_off, .1)

if obj.text == 'Driving\nLights':

if obj.state == "down":

print ("button on")

GPIO.output(lowbeampin, GPIO.LOW)

else:

print ("button off")

GPIO.output(lowbeampin, GPIO.HIGH)

        if obj.text == 'High\nBeam\nLights':

                if obj.state == "down":

                        print ("button on")

                        GPIO.output(highbeampin, GPIO.LOW)

                else:

                        print ("button off")

                        GPIO.output(highbeampin, GPIO.HIGH)

        if obj.text == 'Wiper':

                if obj.state == "down":

                        print ("button on")

                        GPIO.output(wiperpin, GPIO.LOW)

                else:

                        print ("button off")

                        GPIO.output(wiperpin, GPIO.HIGH)

        if obj.text == 'Hazard':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(leftsig\_toggle)

Clock.unschedule(rightsig\_toggle)

                        GPIO.output(rightsignalpin, GPIO.HIGH)

GPIO.output(leftsignalpin, GPIO.HIGH)

Clock.schedule\_once(hazard\_toggle, 0.75)

else:

Clock.unschedule(hazard\_toggle)

GPIO.output(rightsignalpin, GPIO.LOW)

GPIO.output(leftsignalpin, GPIO.LOW)

        if obj.text == 'Left\nSignal':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(rightsig\_toggle)

Clock.unschedule(hazard\_toggle)

                        GPIO.output(leftsignalpin, GPIO.HIGH)

Clock.schedule\_once(leftsig\_toggle, 0.75)

else:

Clock.unschedule(leftsig\_toggle)

GPIO.output(leftsignalpin, GPIO.LOW)

        if obj.text == 'Right\nSignal':

                if obj.state == "down":

                        print ("button on")

Clock.unschedule(leftsig\_toggle)

Clock.unschedule(hazard\_toggle)

                        GPIO.output(rightsignalpin, GPIO.HIGH)

Clock.schedule\_once(rightsig\_toggle, 0.75)

else:

Clock.unschedule(rightsig\_toggle)

GPIO.output(rightsignalpin, GPIO.LOW)

#def setgauge(sender,value):

# mygauge.value = value

#def update\_speed(dt):

# value = calculate\_speed(20)

# setgauge(0,value)

def shutdown\_func(obj):

    os.system("sudo shutdown -h now")

def horn\_off(dt):

GPIO.output(hornpin, GPIO.HIGH)

def hazard\_toggle(dt):

GPIO.output(rightsignalpin, not GPIO.input(rightsignalpin))

GPIO.output(leftsignalpin, not GPIO.input(leftsignalpin))

Clock.schedule\_once(hazard\_toggle, 0.75)

def rightsig\_toggle(dt):

GPIO.output(rightsignalpin, not GPIO.input(rightsignalpin))

Clock.schedule\_once(rightsig\_toggle, 0.75)

def leftsig\_toggle(dt):

GPIO.output(leftsignalpin, not GPIO.input(leftsignalpin))

Clock.schedule\_once(leftsig\_toggle, 0.75)

def calculate\_elapse(channel): # callback function

global pulse, start\_timer, elapse

pulse+=1 # increase pulse by 1 whenever interrupt occurred

elapse = time.time() - start\_timer # elapse for every 1 complete rotation made!

start\_timer = time.time() # let current time equals to start\_timer

def calculate\_speed(r\_cm):

global pulse,elapse,rpm,dist\_km,dist\_meas,km\_per\_sec,km\_per\_hour

if elapse !=0: # to avoid DivisionByZero error

rpm = 1/elapse \* 60

circ\_cm = (2\*math.pi)\*r\_cm # calculate wheel circumference in CM

dist\_km = circ\_cm/100000 # convert cm to km

km\_per\_sec = dist\_km / elapse # calculate KM/sec

km\_per\_hour = km\_per\_sec \* 3600 # calculate KM/h

dist\_meas = (dist\_km\*pulse)\*1000 # measure distance traverse in meter

print('rpm:{0:.0f}-RPM kmh:{1:.0f}-KMH dist\_meas:{2:.2f}m pulse:{3}'.format(rpm,km\_per\_hour,dist\_meas,pulse))

return km\_per\_hour

def init\_interrupt():

   GPIO.add\_event\_detect(sensor, GPIO.FALLING, callback = calculate\_elapse, bouncetime = 20)

class MyApp(App):

def build(self):

# Set up the layout:

# layout = GridLayout(cols=5, spacing=30, padding=30, row\_default\_height=150)

layout = FloatLayout(size=(800,600))

mystopwatch = StopWatch()

         Clock.schedule\_interval(mystopwatch.update, 1/100)

# Make the background gray:

#with layout.canvas.before:

# Color(.2,.2,.2,1)

# self.rect = Rectangle(size=(800,600), pos=layout.pos)

# from kivy.clock import Clock

# from functools import partial

# from kivy.uix.slider import Slider

def update\_speed(dt):

value = calculate\_speed(25)

setgauge(0,value)

def setgauge(sender, value):

mygauge.value = value

def incgauge(dt):

global value

global temp\_val

for temp\_val in range(0,100):

temp\_val += 1

value += 1

setgauge(0,value)

print(value)

for temp\_var in range(0,100):

temp\_val += 1

value -= 1

setgauge(0,value)

print(value)

mygauge = Gauge(value=0, size\_gauge=256, size\_text=60, pos=(0,84))

Clock.schedule\_interval(update\_speed, 0.5)

Clock.schedule\_once(incgauge,1)

# Create the rest of the UI objects (and bind them to callbacks, if necessary):

                highbeam\_button = ToggleButton(text="High\nBeam\nLights", halign='center', size\_hint=(.14,.25),pos = (2,20), font\_size=25)

                highbeam\_button.bind(on\_press=press\_callback)

                lowbeam\_button = ToggleButton(text="Driving\nLights", halign='center' , size\_hint=(.14,.25),pos = (116,20), font\_size=25)

                lowbeam\_button.bind(on\_press=press\_callback)

                leftsig\_button = ToggleButton(text="Left\nSignal", group='signals', halign='center' , size\_hint=(.14,.25),pos = (230,20), font\_size=25)

                leftsig\_button.bind(on\_press=press\_callback)

                hazard\_button = ToggleButton(text="Hazard", group='signals',size\_hint=(.14,.25),pos = (344,20), font\_size=25)

                hazard\_button.bind(on\_press=press\_callback)

                rightsig\_button = ToggleButton(text="Right\nSignal", group='signals',halign='center',size\_hint=(.14,.25),pos = (458,20), font\_size=25)

                rightsig\_button.bind(on\_press=press\_callback)

                wiper\_button = ToggleButton(text="Wiper",size\_hint=(.14,.25),pos = (572,20), font\_size=25)

                wiper\_button.bind(on\_press=press\_callback)

                horn\_button = Button(text="BEEP!",size\_hint=(.14,.25),pos = (686,20), font\_size=25)

                horn\_button.bind(on\_press=press\_callback)

shutdown\_button = Button(text="Bye :)",size\_hint=(.14,.14),pos = (686,410), font\_size=25)

                shutdown\_button.bind(on\_press=shutdown\_func)

                #wimg = Image(source='texas\_logo.jpeg', scale = 0.5, pos = (200,100))

# Add the UI elements to the layout:

                layout.add\_widget(mygauge)

layout.add\_widget(mystopwatch)

                layout.add\_widget(highbeam\_button)

                layout.add\_widget(lowbeam\_button)

                layout.add\_widget(leftsig\_button)

                layout.add\_widget(hazard\_button)

                layout.add\_widget(rightsig\_button)

                layout.add\_widget(wiper\_button)

                layout.add\_widget(horn\_button)

layout.add\_widget(shutdown\_button)

# Clock.schedule\_once(incgauge,5)

         return layout

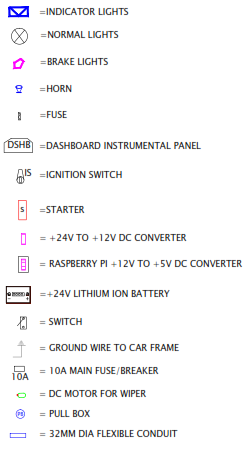
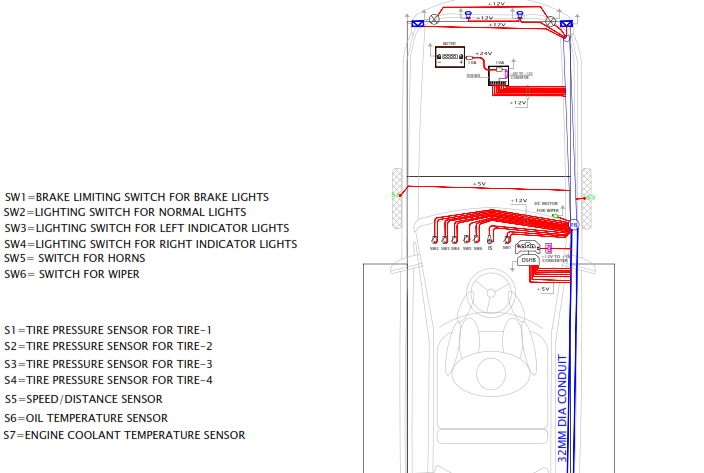
if \_\_name\_\_ == '\_\_main\_\_':

    init\_interrupt()

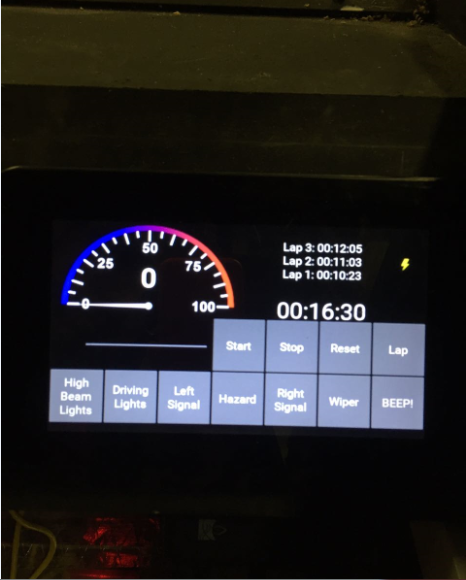
# Clock.schedule\_interval(update\_speed, 0.5)

    MyApp().run()

**Appendix B: (Car Wiring Layout):**

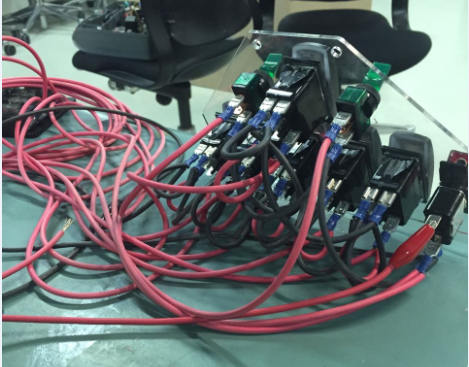
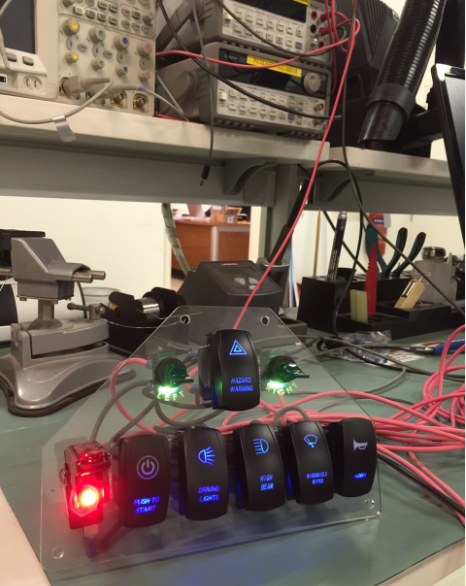


**Appendix C: (Picture of the VMS in Singapore):**



**Appendix D: (Picture of Dashboard built and process):**



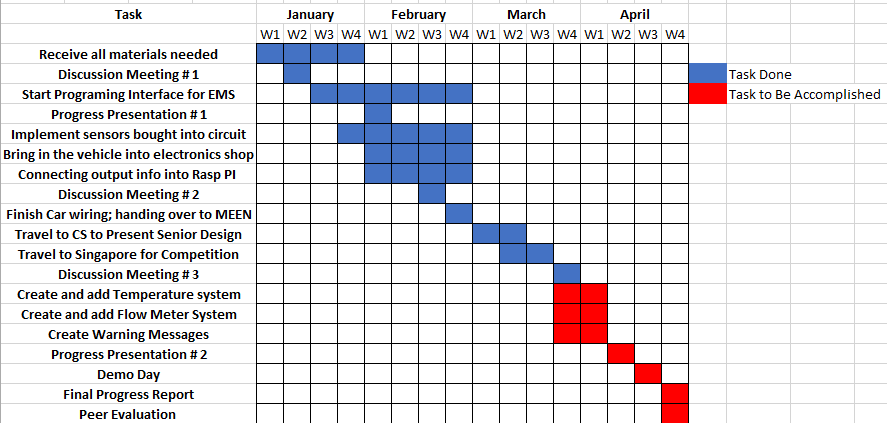




**Appendix E: (Team Photo):**



**Appendix F: (Timeline):**

****