Telemetry

Table of Contents

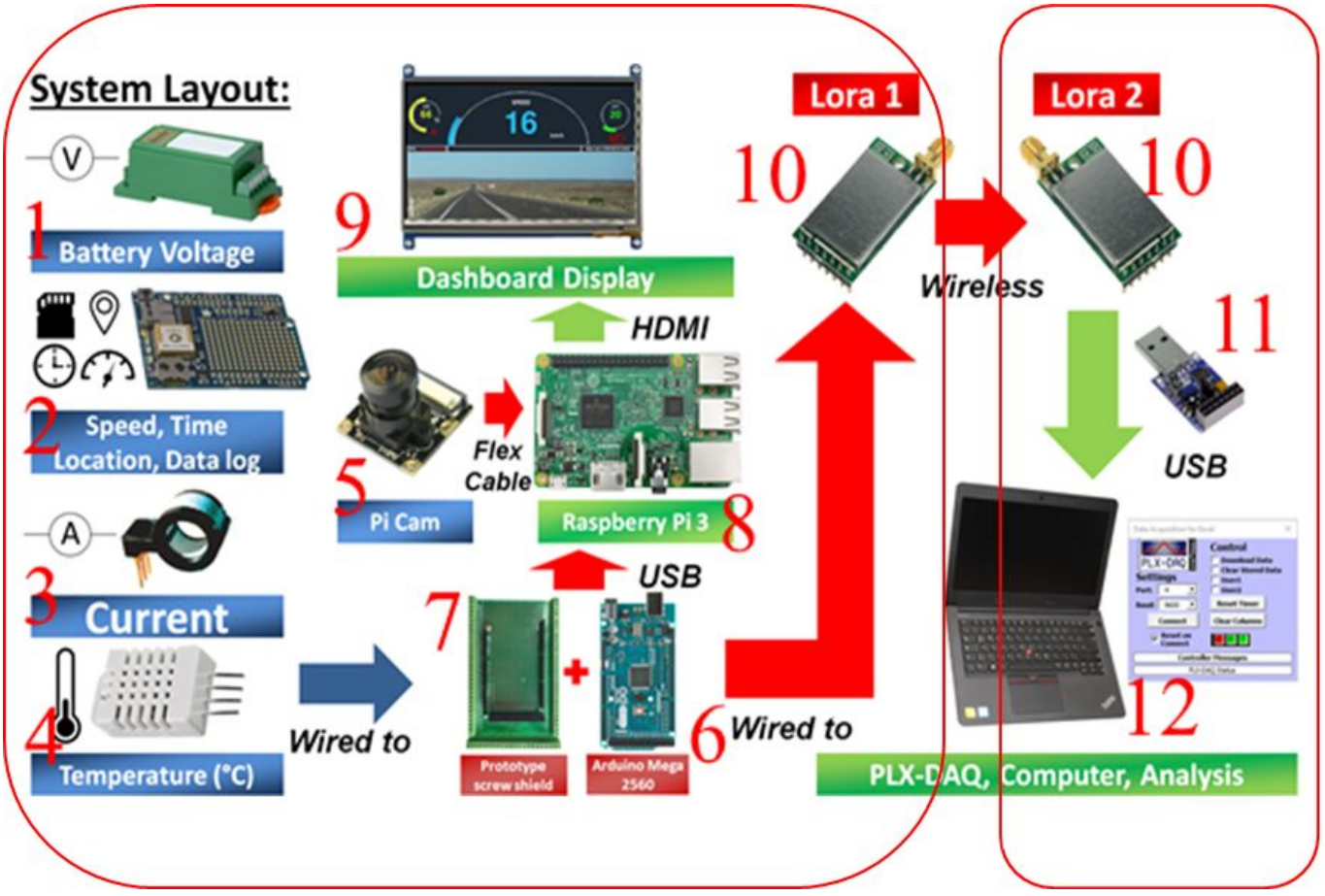
Notations used in this report

EEE - Electrical and Electronics Engineering
SunSPEC 6 - 6th generation of Singapore Polytechnic Solar Car
PLX-DAQ - Parallax Data Acquisition tool
VCC - "5" Volts
Ground - "0" volts
HDMI - High Definition Multimedia interface
AI - Analog Input
DI - Digital Input
IPS - In plane switching
CSI - Camera Serial Interface (CSI)
RTOS - Real Time Operating System
OS - Operating System
LED - Light Emitting Diode
GPS - Global Positioning System
LoRa - Long Range
FPGA - Field Programmable Gate Arrays
NI - National Instruments
SMA - SubMiniature version A
LCD - Liquid Crystal Display
RF - Radio Frequency
SBC - Single Board Computer

List of illustrations for telemetry

Section 1
TELEMETRY HARDWARE

SunSPEC 5 Telemetry Architecture



The following table shows the telemetry hardware

No.	Name	Objective
1	CR5310-150 voltage transducer	To collect voltage values
2	Ultimate GPS shield	To collect speed, time and location. data logger
3	AMP 25 open loop hall effect current sensor	To collect current values
4	DHT22 temperature and humidity sensor	To collect temperature values
5	Raspberry Pi Camera Module	Rear view vision
6	Arduino Mega 2560	Collects and processes sensor data

7	Prototype screw shield	Easy and wide access to many VCC,GND and IO pins
8	Raspberry Pi 3 Model B+	1.4GHz Single Board Computer
9	7 inch IPS TFT LCD screen	Displays the dashboard GUI to the driver
10	LoRa UM402 433 MHz RF transceiver	Transmits data from solar vehicle to chase vehicle
11	USB to UART converter board (USB LoRa tester)	Interfaces LoRa and computer
12	General Computer	Runs PLX-DAQ software and displays a dashboard of the data collected from the solar vehicle

Improved version of Telemetry Architecture for SunSPEC 6

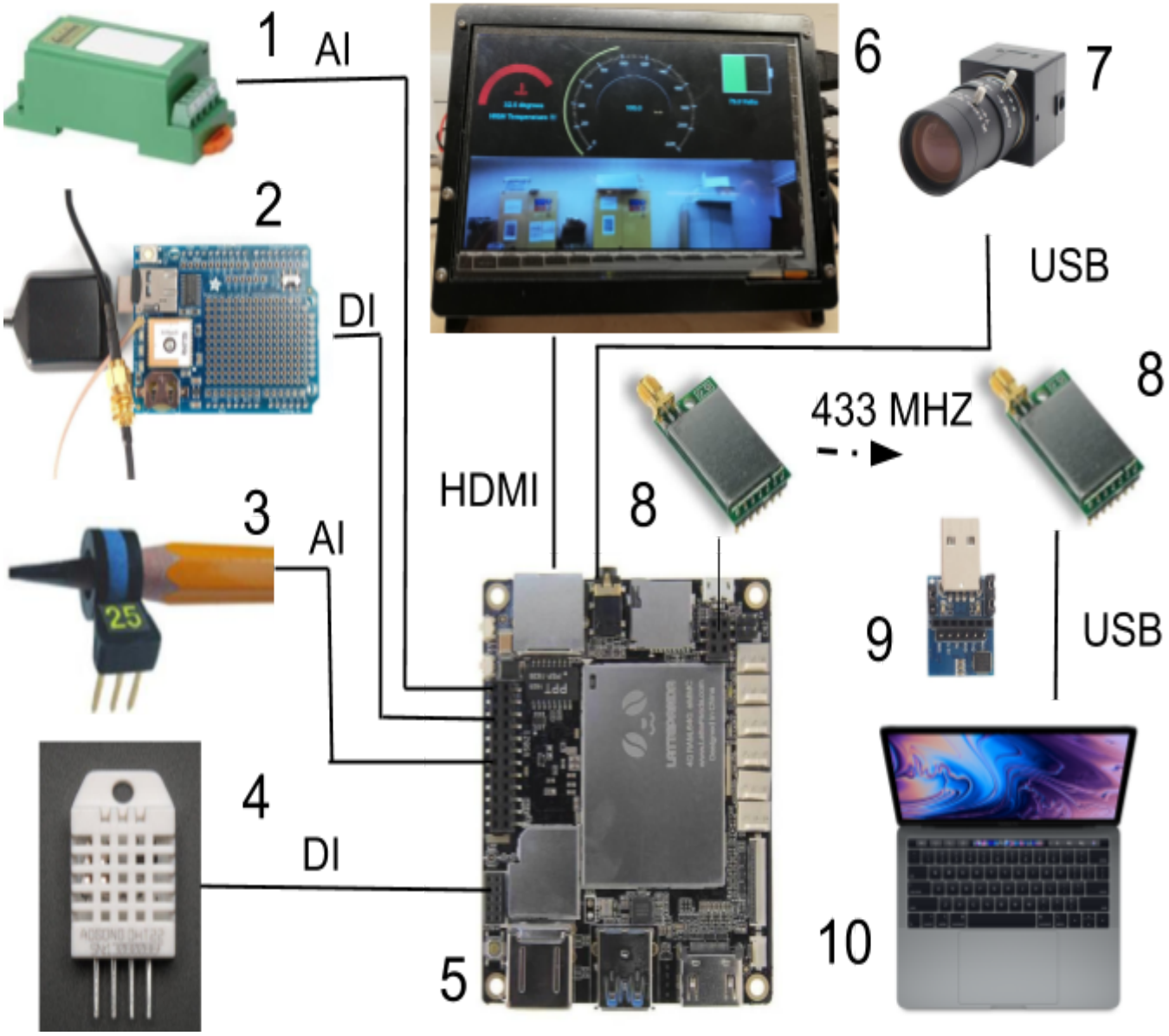


Figure 1 : Telemetry System Architecture for Lattepanda

This is the new and updated SunSPEC telemetry setup and it consists of 10 hardware components instead of 12 for the dashboard display, communication and data collection.

The following table shows a description of the above hardware and its functions in the telemetry system.

The following table shows the telemetry hardware

No.	Name	Objective

1	CR5310-150 voltage transducer	To collect voltage values
2	Ultimate GPS shield	To collect speed, time and location. data logger
3	AMP 25 open loop hall effect current sensor	To collect current values
4	DHT22 temperature and humidity sensor	To collect temperature values
5	Lattepanda - a Windows 10 Single Board Computer	Processes sensor data and runs the graphical user interface (GUI) in the dashboard
6	7 inch IPS TFT LCD screen	Displays the dashboard GUI to the driver
7	8mp usb camera	Rear view vision
8	LoRa UM402 433 MHz RF transceiver	Transmits data from solar vehicle to chase vehicle
9	USB to UART converter board (USB LoRa tester)	Interfaces LoRa and computer
10	Computer	Runs PLX-DAQ software

2.1 World Solar Challenge 2019 regulations

According to the World Solar Challenge 2019 regulations, the required information must be provided to the driver at all times while driving.

"2.26.1 The following information must be provided to the driver at all times while driving:
• the speed of the solar car
• whether the direction indicators are operating
• whether the hazard lights are operating
• energy storage system warnings
• electronic rear vision images (if fitted)." (Selwood AM and Pudney, 2019)

2.2 CR 5310-150 DC Voltage Transducer

Figure 3 : DC Voltage Transducer (CR5310-150) (Cr Magnetics.com, 2019)

As stated by the World Solar Challenge 2019 regulations, a DC Voltage Transducer is used to monitor the energy storage system and place warnings where applicable in the driver's dashboard. This sensor provides an output DC signal (5V) that is linearly proportional to the input DC voltage (150V). The output DC signal is used to feed either an Arduino 2560 or lattepanda. This sensor effectively steps down from a high

voltage to a low voltage (buck converter) which enables microcontrollers and Single Board Computers (SBC) to measure the voltage of the bus bar.
Thus, these devices are focused to provide an easy solution to DC voltage instrumentation needs.

Basic Accuracy:	1.0 %	Supply Voltage:	24 VDC ±10%
Linearity:	10% to 100% FS	Frequency Range:	DC only
Thermal Drift:	500 PPM/°C	Output Load:	4-20 mADC - 0 to 300 Ω
Operating Temperature:	0°C to +50°C		0-5 VDC - 2K Ω or Greater
Installation Category:	CAT II	Relative Humidity:	80% for temperatures up to
Vibration Tested To:	IEC 60068-2-6,1995		31°C and decreasing linearly
Pollution Degree:	2		to 50% at 40°C
Altitude:	2000 meter max.	Supply Current:	
Response Time:	250 ms. max.	CR5310:	Typical 35mA Max 40mA
Insulation Voltage:	2500 VDC	CR5320:	Typical 35mA Max 40mA
Cleaning:	Water-dampened cloth	Torque Specs.:	3.0 inch lbs. (0.4Nm)
		Weight:	0.5 lbs.

Figure 4 Technical Specifications of a CR5320 DC Voltage Transducers

2.3 Ultimate Global Positioning System (GPS) Shield

Figure 5 Adafruit GPS shield (ada and Cooper, 2019)
The GPS Shield is of utmost importance to the Telemetry Architecture. The data gathered from this sensor are speed (km/h), date, time-stamp, longitude, latitude and altitude. The GPS shield features an external coin cell as shown in Figure 5. However, the World Solar Challenge regulations allow under the clause 2.5.10

From the World Solar Challenge regulations (Selwood AM and Pudney, 2019), "Batteries used only to

• power a real-time clock when the solar car is turned off; or
• retain data when the solar car is turned off; or
• power wireless tyre pressure monitors
are not considered to be part of the energy storage system, provided that the total energy
capacity does not exceed 2.0 Wh.”

The technical specifications are as follows

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- Low power module - only 20mA current draw, half of most GPS's
- Assembled and tested shield for Arduino Uno/Duemilanove/Diecimila/Leonardo
- MicroSD card slot for data logging onto a removable card
- RTC battery included, for up to 7 years backup
- Built-in data logging to flash
- PPS output on fix
- Internal patch antenna + u.FL connector for external active antenna
- Power, Pin #13 and Fix status LED
- Big prototyping area

This sensor module provides fast updates and is highly reliable and capable of updating the graphical user interface (GUI) quickly. Since this sensor is going to be inside an enclosure, it has an external antenna support for greater reliability.

2.3.1 GPS Antenna

Figure 6 GPS External Active Antenna

Figure 7 SMA to uFL RF Adapter Cable

The built in patch antenna provides -165 dBm sensitivity. The external active antenna gives an additional 28 dB of gain. This will enable the GPS shield to provide a better accuracy and an improved connectivity to the satellites. Since the GPS shield is placed in an enclosure we need more sensitivity. Active antennas draw current, so they do provide more gain but at a power cost. The current draw is usually around 10-20 mA. The uFL connector is lightweight and small. Since the GPS shield has a uFL (miniature coaxial RF connector) connector, we need to connect to a SMA (SubMiniature version A) (Industries, 2019) connector in the GPS antenna shown in Figure 6. Thus, a SMA to uFL RF (Radio Frequency) adapter cable is required. The Ultimate GPS shield will automatically detect an external active antenna and switch over without the need to send any commands.

2.3.2 LEDs

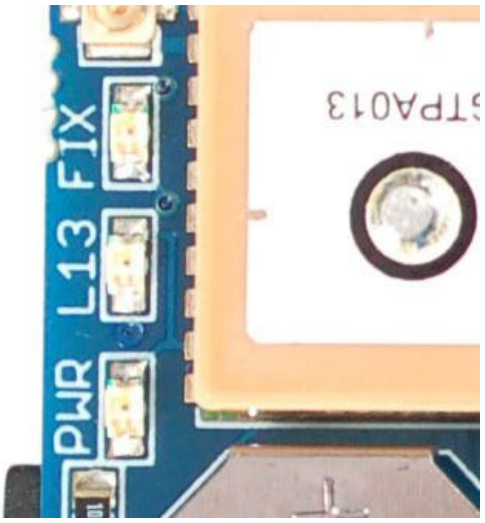


Figure 8 LEDs in GPS Shield

There are three LEDs on board to help with debugging and status updates.

- Green PWR LED indicates that there is a good 3V power supply. If this isn't on, there's a serious problem with the power supply, perhaps the battery died.

- Yellow L13 (and SD card access) LED is connected to digital 13, this is handy for telling when the Arduino is bootloading and also will flicker whenever the SD card is accessed.

- Red FIX LED is connected to the GPS's fix output. When this is turning on/off once a second it does not have a fix. When it blinks once every 15 seconds, the GPS has a fix.

2.4 Current Sensor

The current sensor goes hand in hand with the CR 5310-150 DC Voltage Transducer mentioned above. Since Power = Voltage * Current, current sensors are needed to measure the power of the SunSPEC solar car. Thus, AMPLOC 50 hall effect current sensors were used in the World Solar Challenge 2017. However, this sensor is still inaccurate and does not provide real time readings. Thus, we have switched to CR Magnetics Current Transducer for reliable and precise readings.

The hall effect theory states that when a current carrying conductor is placed in a magnetic field, a voltage will be generated in perpendicular to the direction of the magnetic field and the flow of current due to Faraday's law of induction. The current in the wire can have two possible directions.The direction of the current flow can be determined by Fleming’s right hand rule. (En.wikipedia.org, 2019)

The interface circuit shown in figure 9 will be most useful for our purpose. This is an optional circuitry.

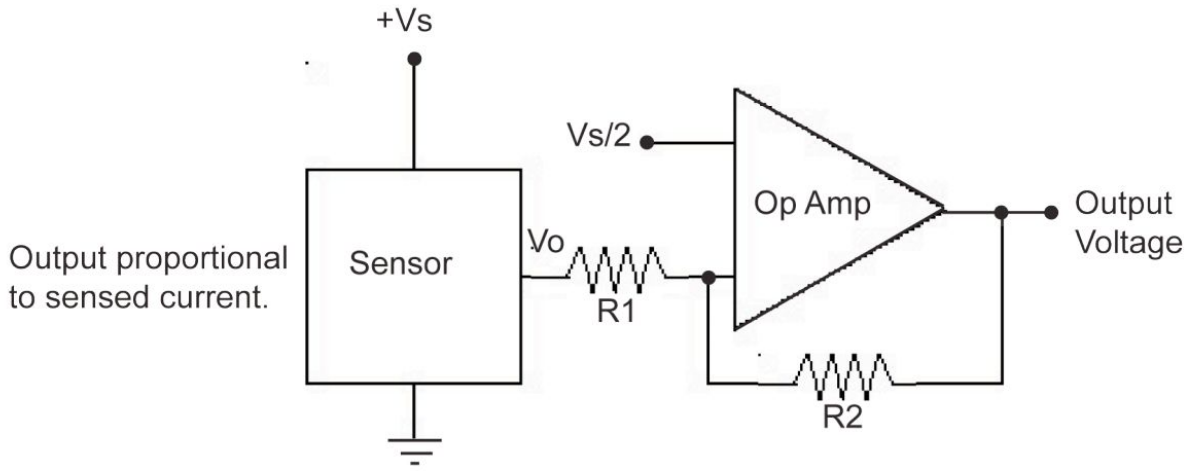
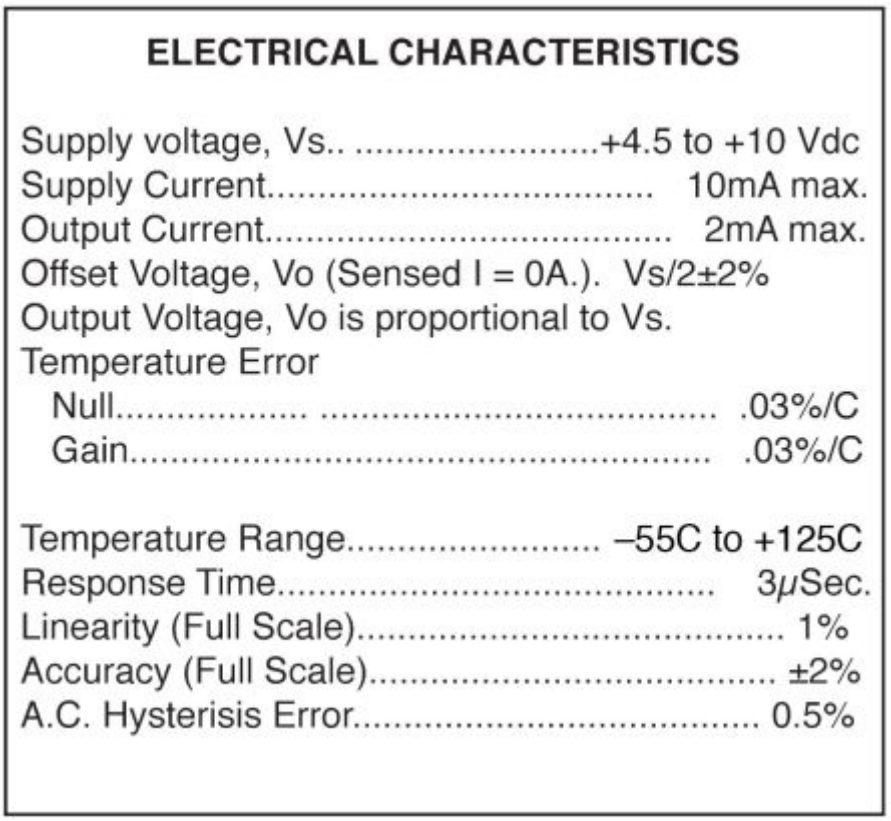


Figure 9 Interface Circuit for AMPLOC current sensor

The AMPLOC linear hall effect sensors have the following advantages,

1. AMPLOC current sensors provide a galvanic isolation (ensures better safety) and are capable of measuring ac, dc and other complex waveforms.
2. No unnecessary cable terminations are required for current measurements.
3. It has a wide temperature range from -55℃ to 125℃, are encapsulated in tough polypropylene and are impervious to battery chemical attacks.
4. They offer a great value in terms of cost and performance. They are also light-weight.

Figure 10 AMPLOC 50 linear hall effect sensor



ELECTRICAL CHARACTERISTICS

Supply voltage, Vs.. .....+4.5 to +10 Vdc
Supply Current..... 10mA max.
Output Current..... 2mA max.
Offset Voltage, Vo (Sensed I = 0A.). Vs/2±2%
Output Voltage, Vo is proportional to Vs.
Temperature Error
Null..... ..... .03%/C
Gain..... .03%/C

Temperature Range..... –55C to +125C
Response Time..... 3μSec.
Linearity (Full Scale)..... 1%
Accuracy (Full Scale)..... ±2%
A.C. Hysterisis Error..... 0.5%

Figure 11 Electrical Characteristics of AMPLOC 50 linear hall effect sensor

2.4.1 CR Magnetics Current Transducer

Figure 12 CR5200 DC Current Transducer (Digikey.com, 2019)

The CR5200 DC Current Transducer is designed to provide a DC signal which is proportional to a DC sensed current. These devices are designed for direct current only. Current ranging from 2 to 10 Amp utilize an advanced magnetic modulator technology while current 20 amps and above utilize hall effect technology. This device is great for troubleshooting and changing the orientation of the transducer.

Features :

- Closed loop sensing for accuracy
- 35mm DIN rail or panel mount
- Available with ±5 VDC, ±10 VDC or 4 - 20 mADC outputs
- Non-contact DC current sensing
- Connection diagram printed on case

2.5 Temperature and Humidity sensor (DHT 22)

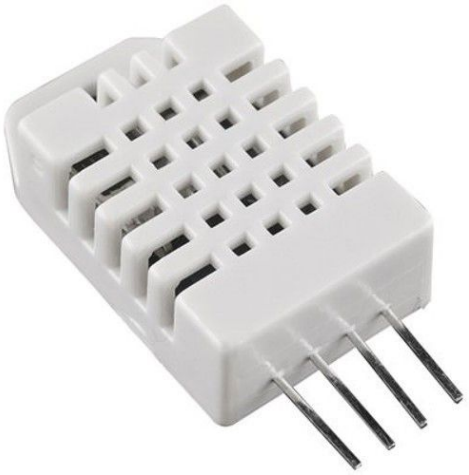


Figure 13 DHT22 Temperature and Humidity sensor

The DHT22 (also named as AM2302) output is a calibrated digital signal which measures relative humidity and temperature. It utilizes exclusive digital signal (no analog pins are needed) collecting technique and humidity sensing technology. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air.

The features of the sensor are:

- More precise, more accurate and works in a bigger range of temperature/humidity than DHT 11
- Uses a capacitive humidity sensor and a thermistor
- Calibrated digital signals
- Low power consumption
- No external components are required
- Long-term Stability

Technical Specifications of a DHT22 sensor

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

2.6 Rear View Vision

Figure 14 Raspberry Pi Camera Module v2

The Raspberry Pi Camera Module v2 is a replacement for a rear view mirror in traditional vehicles to allow the driver to see rearward view and provides vision for blind spots. This is compulsory as stated in the 2019 Bridgestone World Solar Challenge Regulations under clause 2.18.2. The camera is directly connected to the raspberry pi via a ribbon cable. The camera view is display as part of the dashboard.

Technical details

Image Sensor	Sony IMX219 8-megapixel sensor
Video Modes	Supports 1080p30, 720p60 and VGA90 video modes
APIs (Application Programming Interface)	MMAL and V4L APIs
Libraries	Picamera Python library
Uses	Still photographs, high-definition video, time-lapse, slow-motion, and others

2.6.1 ELP USB Camera

Figure 15 ELP USB Camera

This model camera replaces the previous raspberry pi camera module v2 because the Lattepanda does not feature a Camera Serial Interface (CSI) port. Therefore, a USB camera is suitable to connect to the lattepanda board.

Technical Specifications

Model	ELP-USB8MP02G-L75
Sensor	SONY IMX179
Lens Size	1/3.2inch
Image area	6.18mm x 5.85mm
Max. Resolution	3264(H) X 2448(V) 8 Megapixel
Power supply	USB BUS POWER 4P-2.0mm socket
Operating Voltage	DC 5V
Working temperature	-10 ～70℃
Storage temperature	-20 ～ 85℃
Resolution & frame	3264X2448 @ 15fps / 2592X1944@ 20fps 2048X1536 @ 20fps / 1600X1200@ 20fps 1280X960 @ 20fps / 1024X768@ 30fps 800X600 @ 30fps / 640X480@ 30fps

2.7 Arduino Mega development board (ATMega 2560 microcontroller)

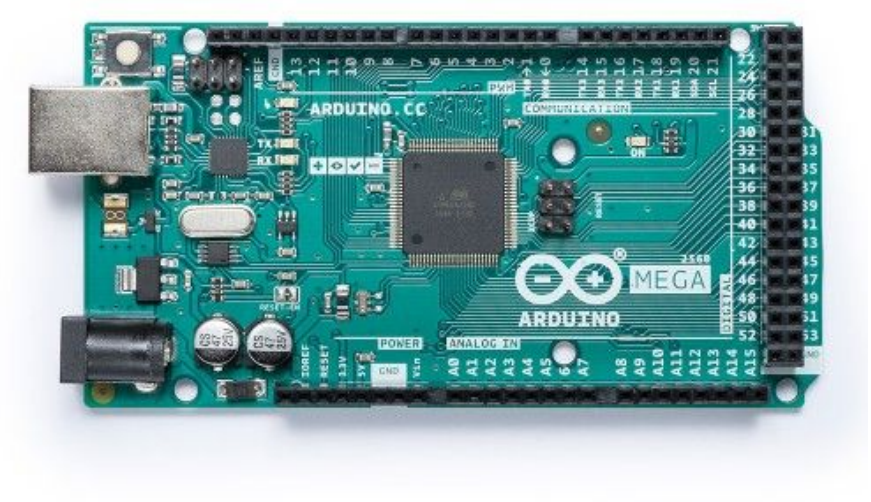


Figure 16 Arduino Mega 2560 (Store.arduino.cc, 2019)

Figure 17 AT Mega 2560 microcontroller

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. A microcontroller is a small computer on a single integrated circuit. It consists on or more of CPUs (processor cores) along with memory and programmable input/output peripherals.(En.wikipedia.org, 2019) It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It encompasses all components required to assist the microcontroller.. (Store.arduino.cc, 2019)

The microcontroller is the interacts directly to the sensors to acquire data, process the raw data and send them to the dashboard and to the chase vehicle.

The AT Mega 2650 has 4 serial ports which are used for communication between the Arduino board and a computer or other devices. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB.

Serial : Pins 0 (RX) and 1 (TX) are called as soft serial because it is a virtual interface between the computer and Arduino.
Serial 1 : Pins 19 (RX) and 18 (TX) are called as hardware serial because it requires an additional USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground.
Serial 2 : Pins 17 (RX) and 16 (TX) Hardware serial
Serial 3 : Pins 15 (RX) and 14 (TX) Hardware serial

The serial port (also known as a UART or USART) communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, these cannot also use pins 0 and 1 for digital input or output. We have to unplug the pins attached to 0 and 1 before uploading any C program and attach them back in to the pins 0 and 1.

Keep in mind that these pins are used for communication between the Arduino board and a computer or other devices. Thus, it is not advised to use these pins as general purpose input and output pins (GPIO).

The advantages of Arduino are that

1. The C programs are highly abstracted and are able to run on any Arduino specific boards with minimal reconfiguration.
2. Furthermore, the Arduino board is great for prototyping and immediate testing.
3. There are many open source libraries available.

The disadvantages of Arduino boards are numerous such as follows

1. Arduino boards hide away a lot of complexity which is unsuitable for tweaking different parameters for maximum performance which gives a major disadvantage to the hardware engineer.
2. The Arduino libraries are not very efficient in certain parts and waste RAM and CPU cycles. Therefore, one has to write their own C library which is more efficient and is less error prone.
3. Arduino doesn't support the data type double which is an important factor to consider for real time monitoring of data such as from GPS.
4. The Arduino IDE does not support hardware debugging.

Microcontrollers such as MSP430 Launchpad, STM32 MCU Nucleo and the Motorola 68HC11 are more efficient and more powerful than Arduino boards.

Arduino Mega is chosen due to its simplicity, more IO pins and it has a bigger flash memory of 256 KB than other Arduinos such as Arduino Uno. Digital and Analog pins are used to interact with the sensors.

Technical Specifications

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm
Weight	37 g

2.7.1 Arduino Mega Shield

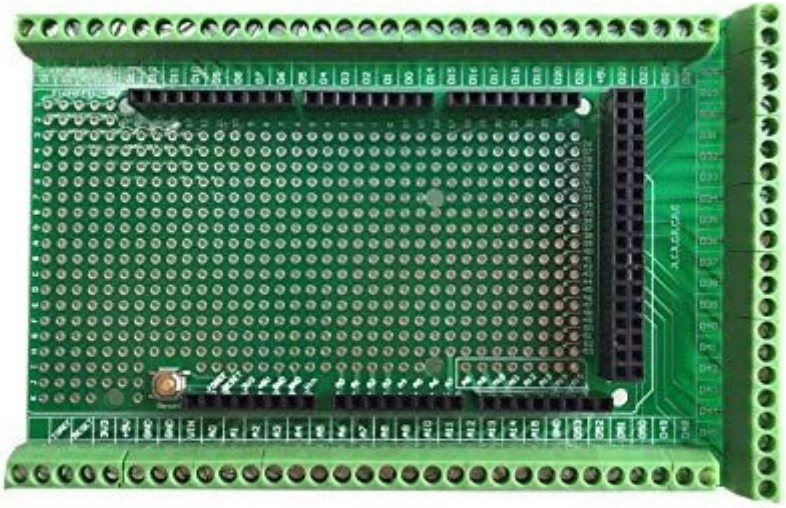


Figure 18 Prototype Screw / Terminal Block Shield Board Kit For Arduino MEGA 2560

Figure 19 Screw Terminals

The telemetry system has to ensure that it is reliable and rugged all the time. The physical hardware connections need to be firm due to coarse road and bumpy conditions. Thus, the terminal block shield ensures that connections remain secured and is immune to vibrations and other external factors.

2.8 LCD Screen

Figure 20 7 inch IPS TFT LCD Screen

The 7 inch IPS TFT LCD Screen is connected to either the raspberry pi or lattepanda via a HDMI (High-Definition Multimedia Interface) cable. The power supply is given from a 12V to 5V buck converter.

2.9 Raspberry Pi

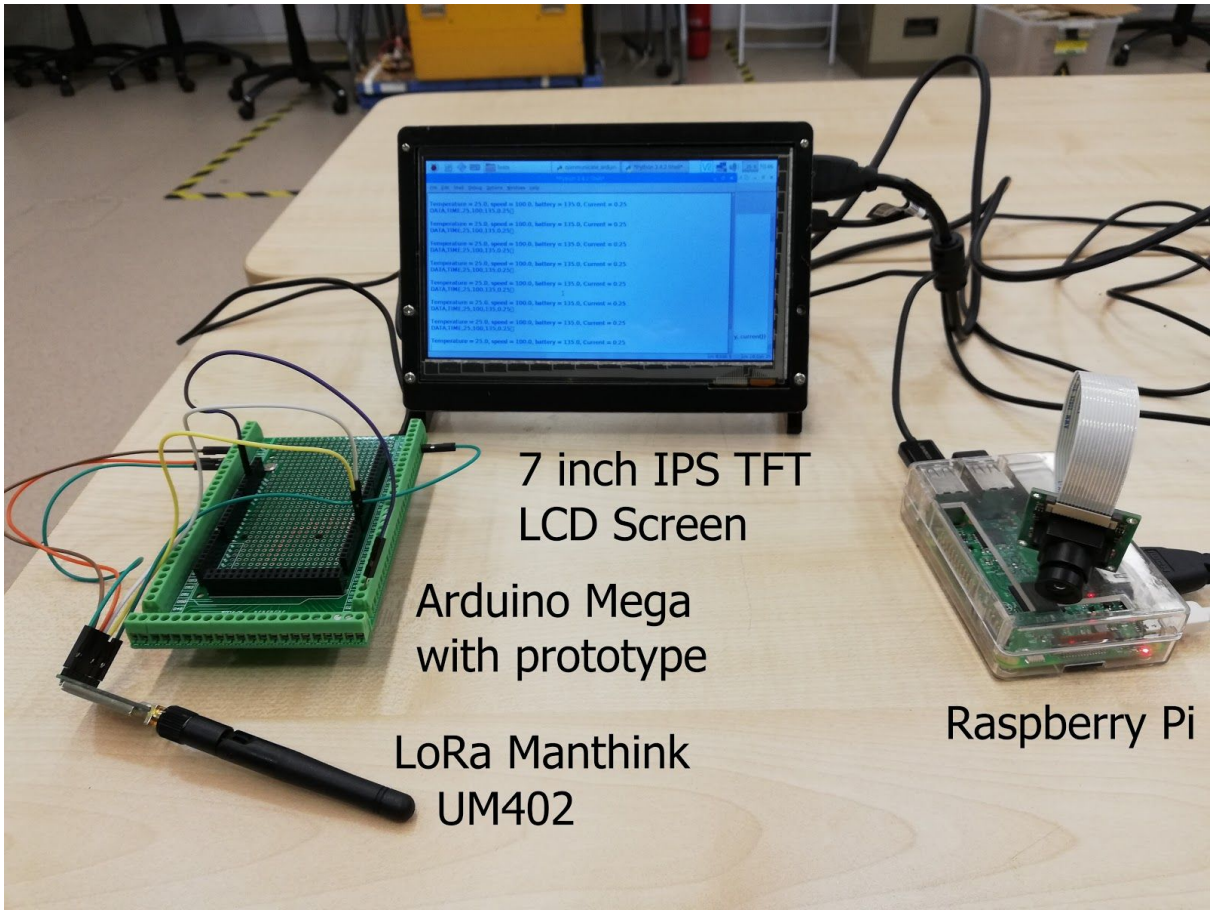


Figure 21 Raspberry Pi setup

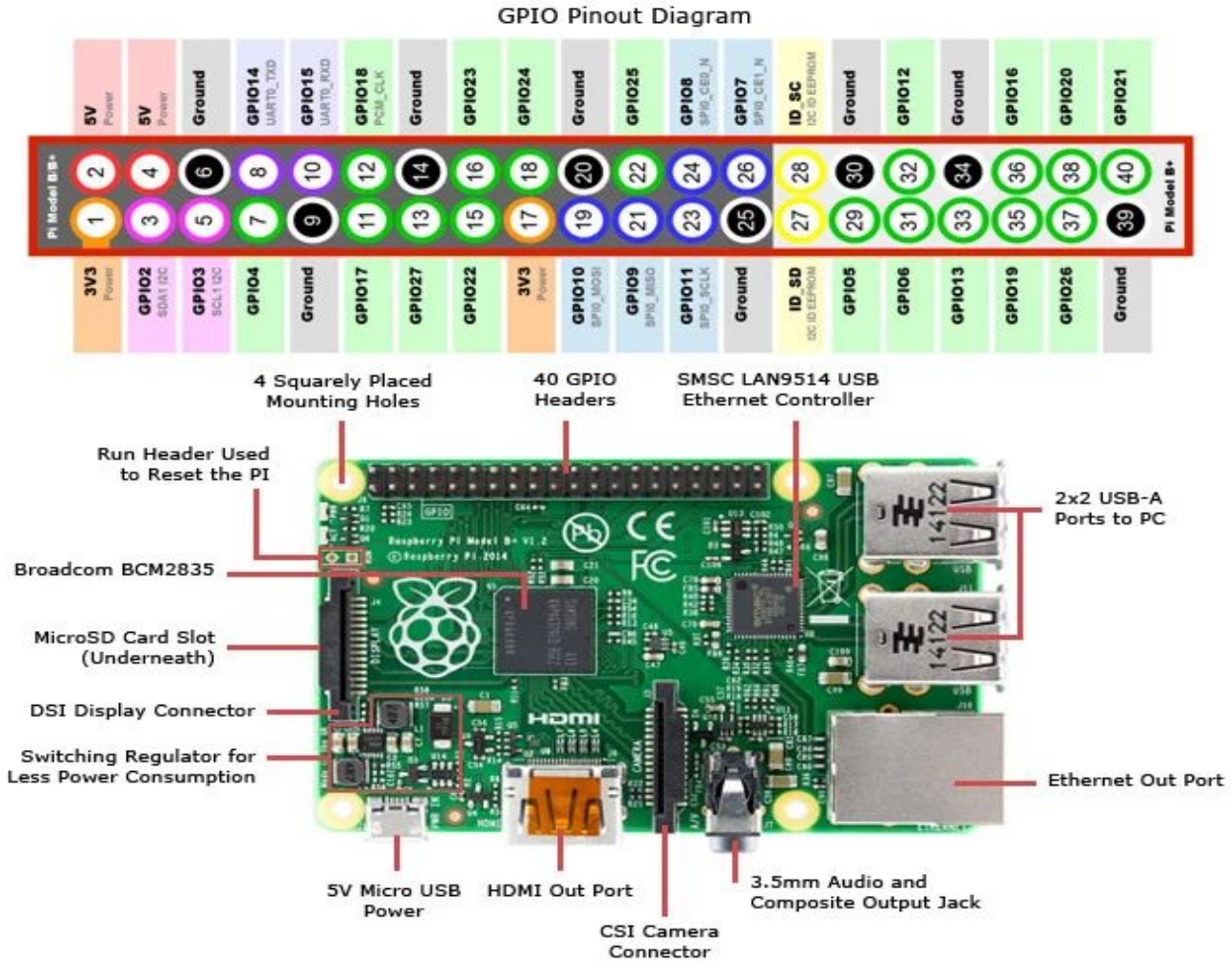


Figure 22 Raspberry Pi Pinout Diagram

The Raspberry Pi is a single-board computer (SBC) is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) and other features required of a functional computer. These are used as embedded computer controllers to display complex graphical user interfaces such as for the dashboard.

The Raspberry Pi provides us many tools and complex software to program the user interface in any programming language desired. Python is used as it is versatile, forgiving and easy to implement.

Technical Specifications

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input

3.0 Lattepanda

Figure 23 Lattepanda

The Lattepanda board runs a full version of Windows 10. It is integrated with an Arduino Leonardo board featuring a ATmega32u4 microcontroller that enables us to receive data from external sensors, process them and display in a 7 inch TFT LCD screen.

LattePanda brings single board computers to a whole new level of power and performance. It features an Intel Quad Core 1.8GHz processor, 2-4GB RAM and 32-64GB onboard flash memory, LattePanda can easily carry out image recognition, real-time CNC control and more!

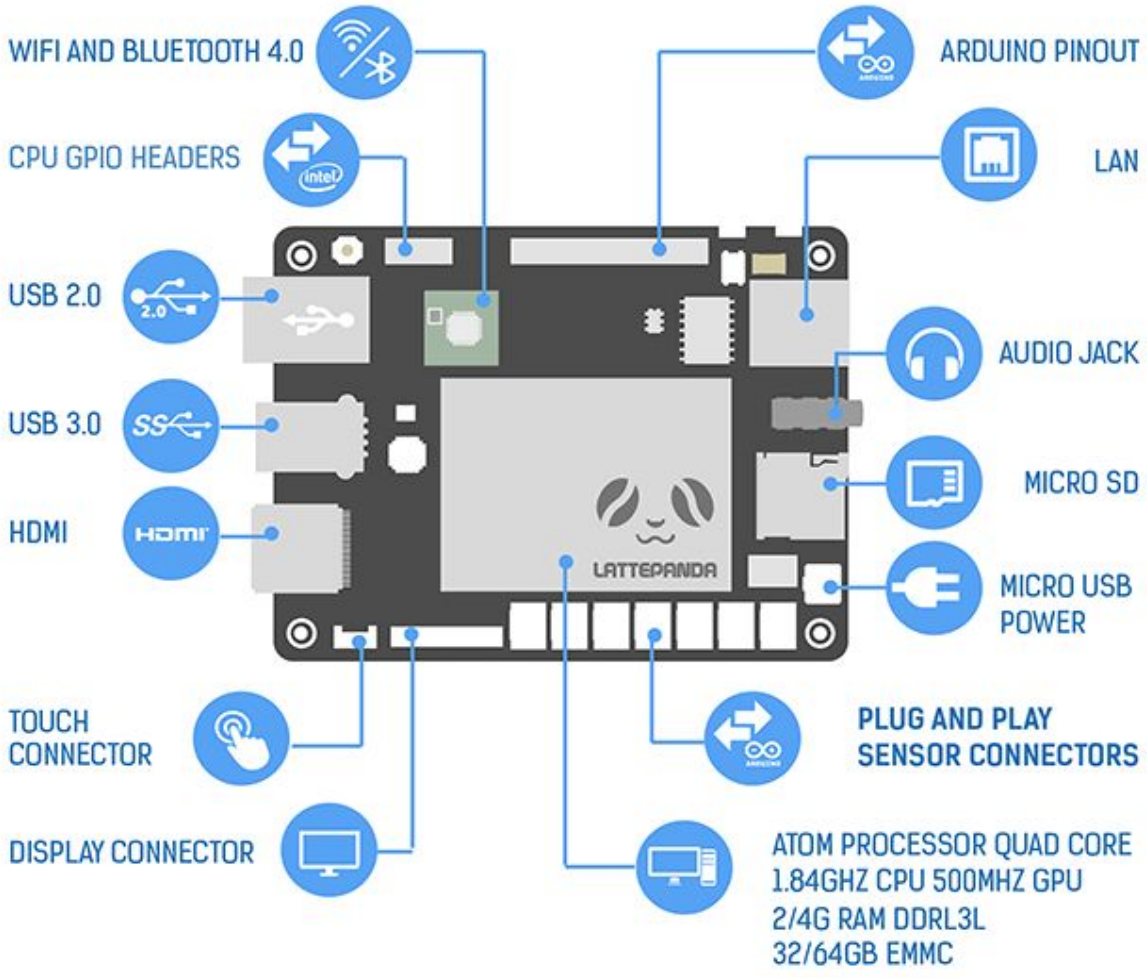


Figure 24 Lattepanda pinout diagram

The Lattepanda is more faster and powerful than Raspberry Pi, thus it is capable of acquiring more accurate telemetry data than Raspberry Pi. This lets us to do a better analysis of data and allows us to run fault prediction algorithms.

3.1 LoRa Module

Figure 25 LoRa Manthink UM 402 433/470 MHz Low power with long range RF module with UART interface

Features

- Transmission distance > 5000 m
- Frequency 410 MHz to 510 MHz
- High sensitivity -144.1dBm@50.78bps
- Maximum transmission power 20dBm
- LoRa modem
- Low sleep current 1.3uA
- Receiving mode current 13mA
- Max current 120mA@20dBm
- Super small size 34.5mm*20mm*6mm
- Single needle interface
- Support wireless awakening function which could low the average power.
- Provide four working modes could further reduce the power consumption; sleep current is 1.3uA, continuous receiving current is around 13 mA.
- Support on-line modification (by MCU) and local modification (using
- RF Module Manager) to modify module parameters.
- Up to 255 bytes in packet once.

UM402 is a highly integrated low-power half-duplex Radio Frequency (RF) transceiver module embedding high-speed low-power microcontroller (MCU) and high-performance RF chip with LoRa modem which is capable of achieving significant longer range than existing RF transceiver based on Frequency Shift Keying (FSK) or On-off keying (OOK) modulation. UM402 module provides multiple channel choices, which users could modify the serial baud rate, transmission power, radio frequency and other parameters in a software program.

This module uses LoRa modem to improve the sensitivity up to -144.1dBm, significantly extending the transmission distance under a low power. So there is no need for repeater and complex communication infrastructure. Since the transmission distance is better than before, users could significantly reduce the usage of repeater, simplify the system design and reduce the production cost.

UM402 module supplies the voltage 2.6-3.6 V with consuming only 13 mA at the receiving mode. UM402 module has four work modes and each one could be switched free. Under the power saving mode, the module consumption is only 1.3uA. The lithium battery with 3.6 V / 3.6 AH could work for several years, which is very suitable for a battery-powered system.

3.1.1 UM402 has four working modes

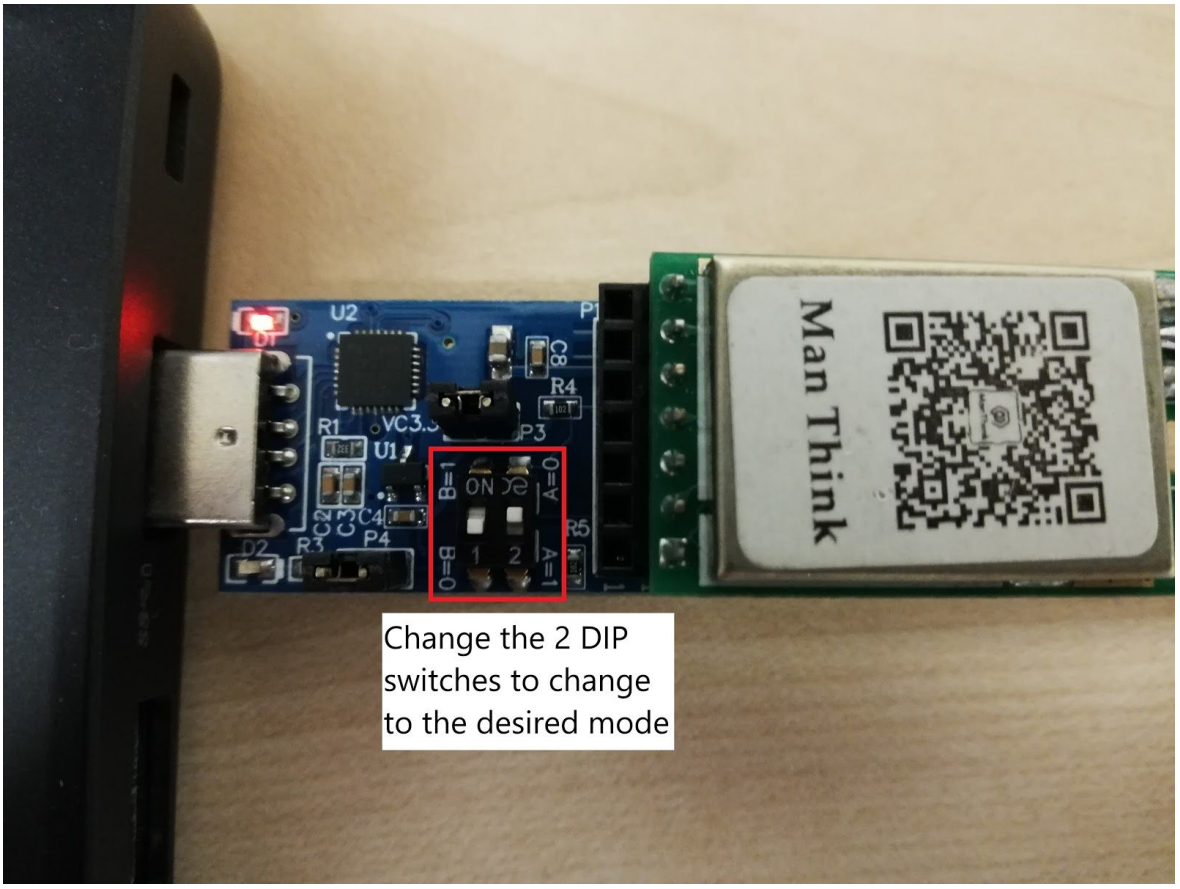


Figure 26 LoRa's 2 DIP switches to set any 4 modes

- Mode 1 – normal (SETA=0,SETB=0)

- Mode 2 – awaken (SETA=0，SETB=1)
- Mode 3 - low power (SETA=1，SETB=0)
- Model 4 - set (SETA=1，SETB=1)

For our purposes, we only use 2 modes mainly Mode 1 and 4. The normal mode is used in operation while the set mode is used to configure settings of the LoRa module. A software program called RFModule is required to set the configuration settings of the 2 LoRa modules.
We will start the process by setting the parameters for the 2 LoRa modules.

3.1.2 RF Module Manager

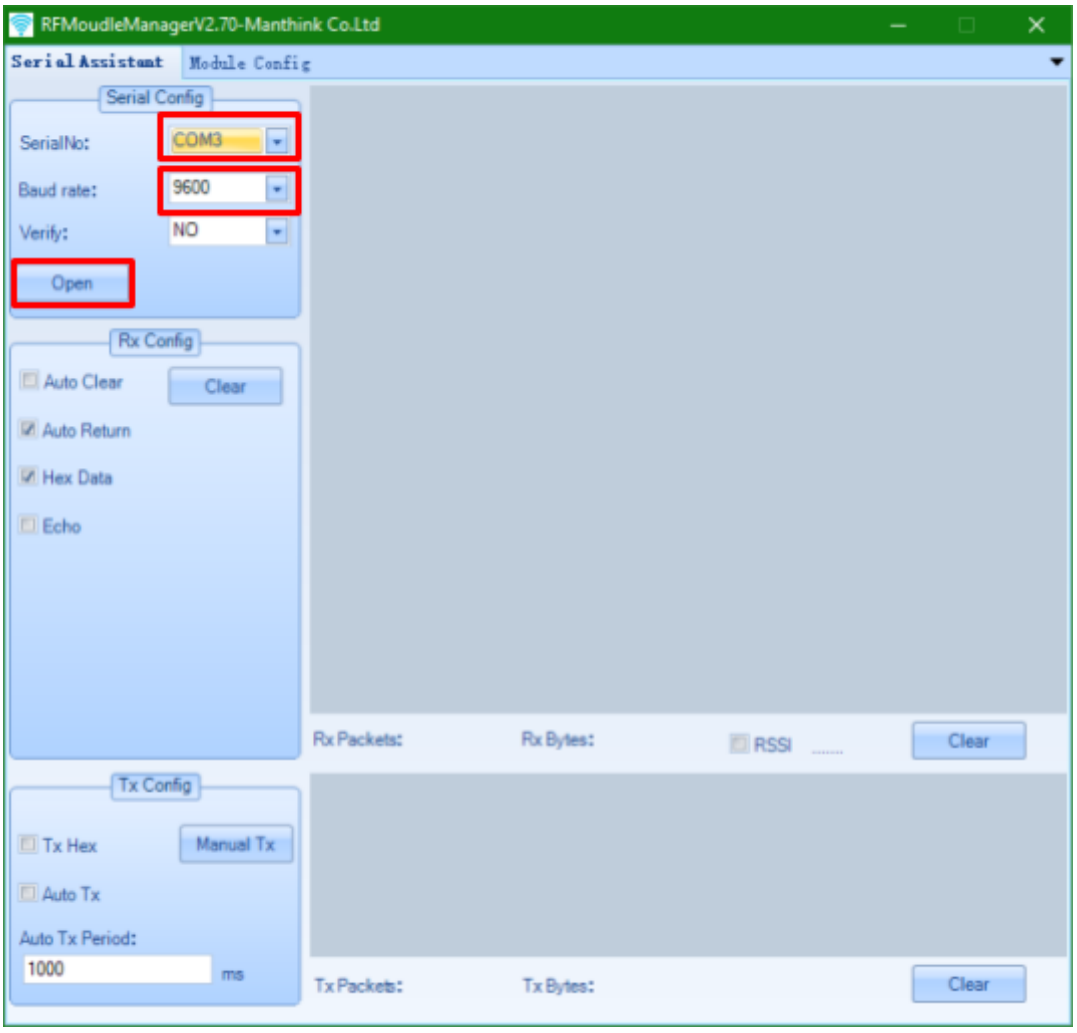


Figure 27 RF Module Manager's initial welcome screen

Choose the COM (Communication port) of the connected LoRa. If you do not know which port it is connected, you need to go to the Device Manager and under Ports (COM & LPT), search for the LoRa port number. Remove other devices to isolate the LoRa port for better identification of the port number. Then, click the Open button.

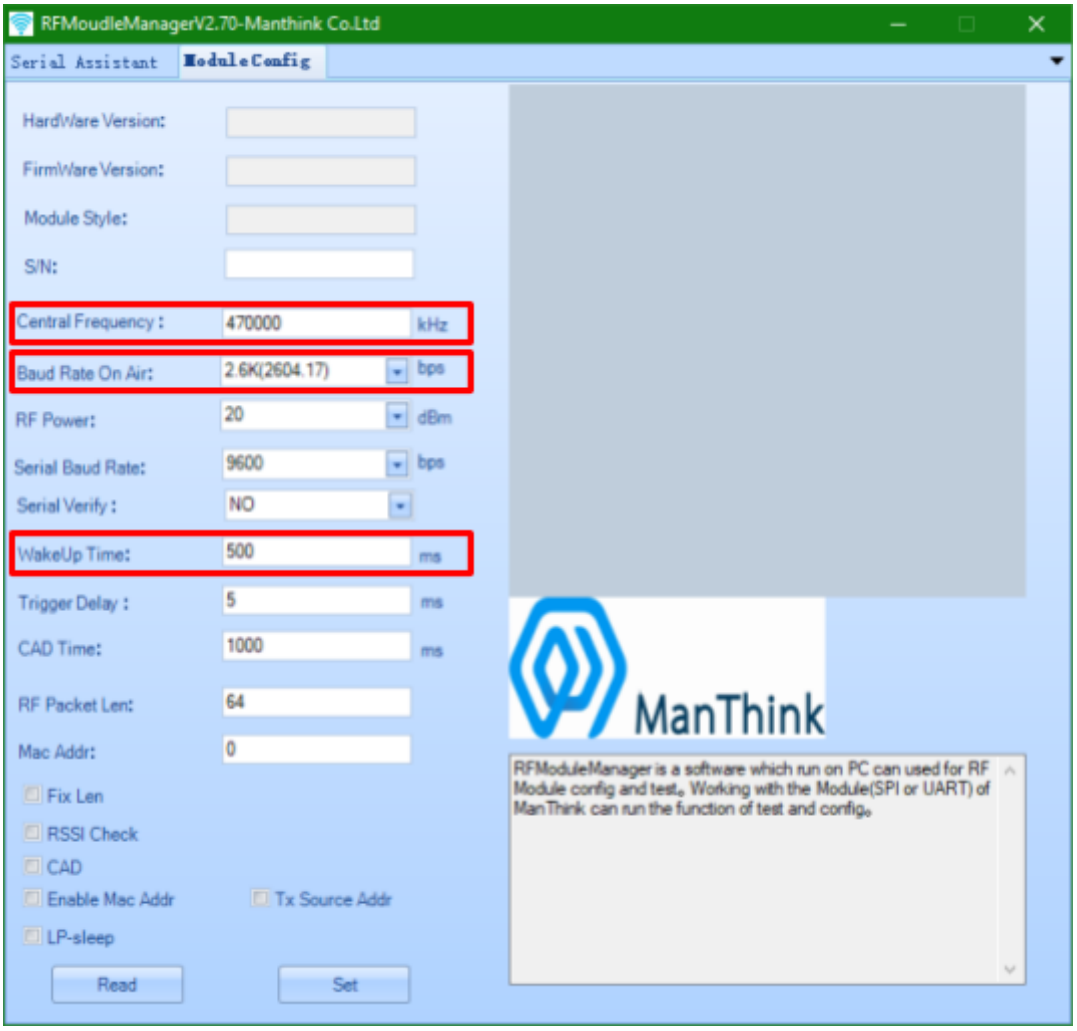


Figure 28 RF Module Manager Setting Screen

The above 3 settings are crucial to the LoRa device and must be the same for the other pair, or else it may fail to communicate.

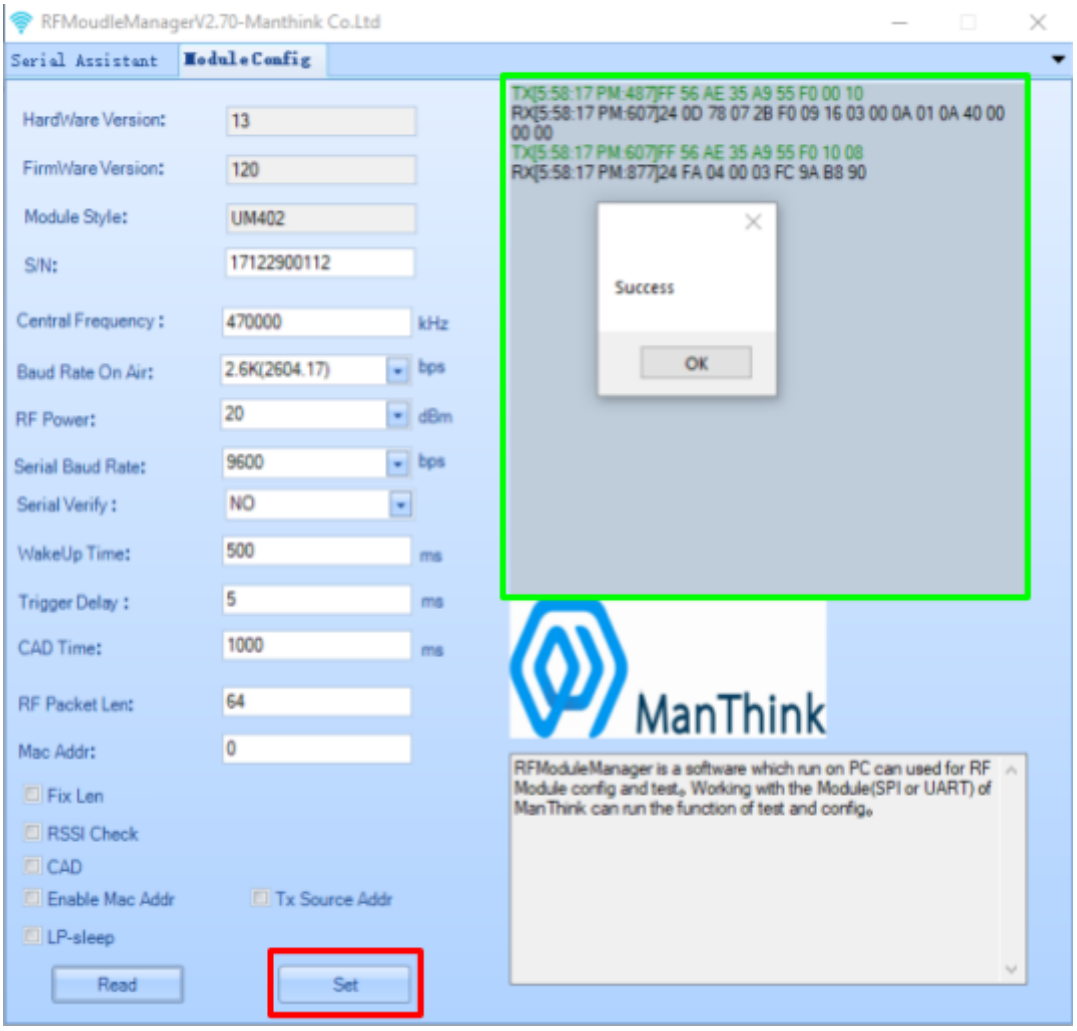


Figure 29 RF Module Manager Setting Screen

After we have selected the required settings, we need to press the Set button to download to the LoRa device. The same process has to be repeated for the other LoRa device as well.

Things to take note of :

- The configuration settings of the LoRa modules must be the same for each pair.
- There are a minor changes to the microcontroller code for different microcontrollers.

Pin Definition

Pin	Name	Direction	State
1	GND	-	GND
2	VCC	-	3.3V

3	SETA	Input (weak pull-up)	GND or 3.3V depending on the mode chosen (Normally GND)
4	RXD	Input (weak pull-up)	TX
5	TXD	output	RX
6	AUX	output	NC
7	SETB	Input (pull-down)	GND or 3.3V depending on the mode chosen (Normally GND)
8	NC	-	Grounded
9	NC	-	Grounded

3.1.3 Comparison of LoRa Module VS nRF24 VS Generic RF Module; Transmission Range & Power consumption test

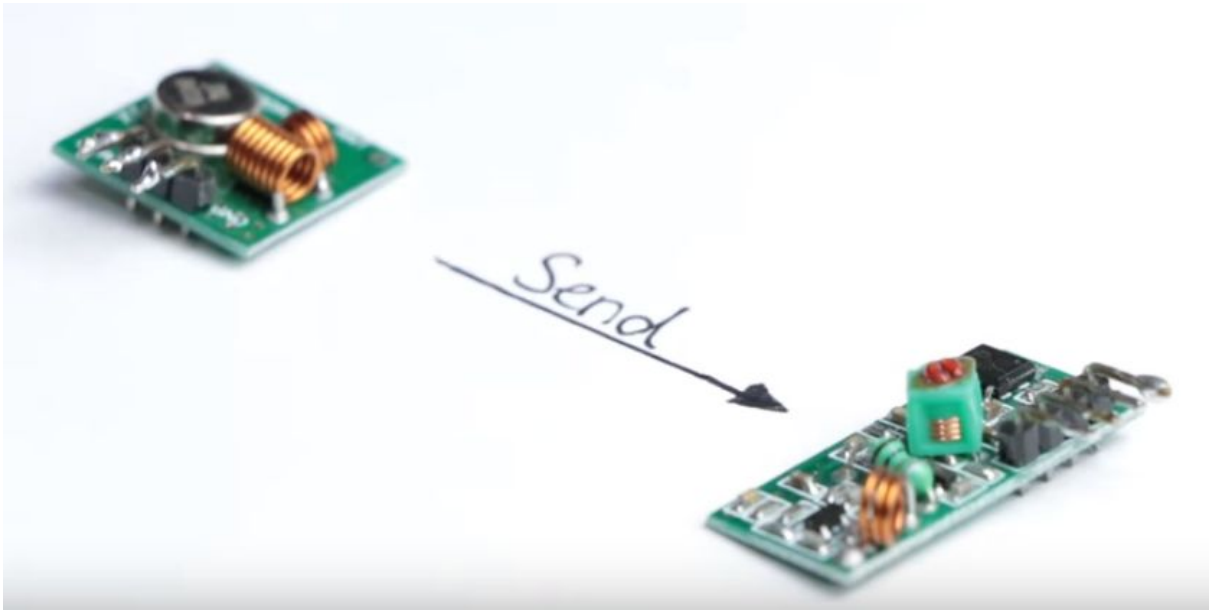


Figure 30 Generic 433Mhz Rf Transmitter and Receiver Module

Generic 433Mhz Rf Transmitter and Receiver Module

- Simple and easy to use
- Simplex transmission
- It transmits at a frequency of 433 MHz
- Range tested is 10 meters (different voltage, different results)

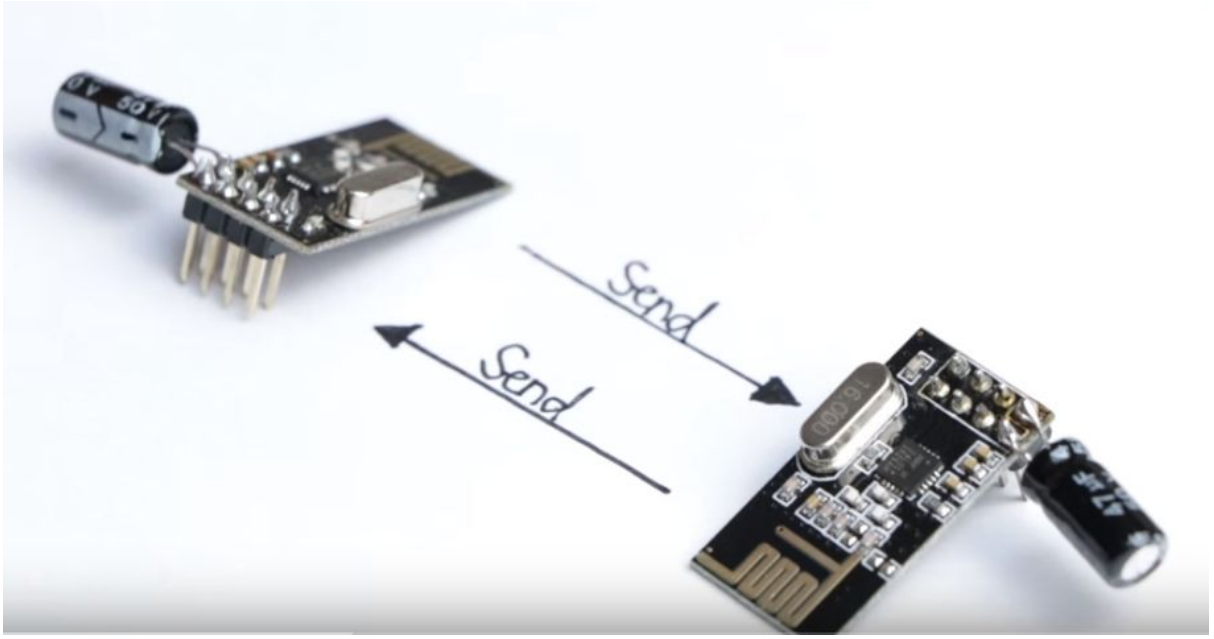


Figure 31 nRF24L01 Rf Transmitter and Receiver Module

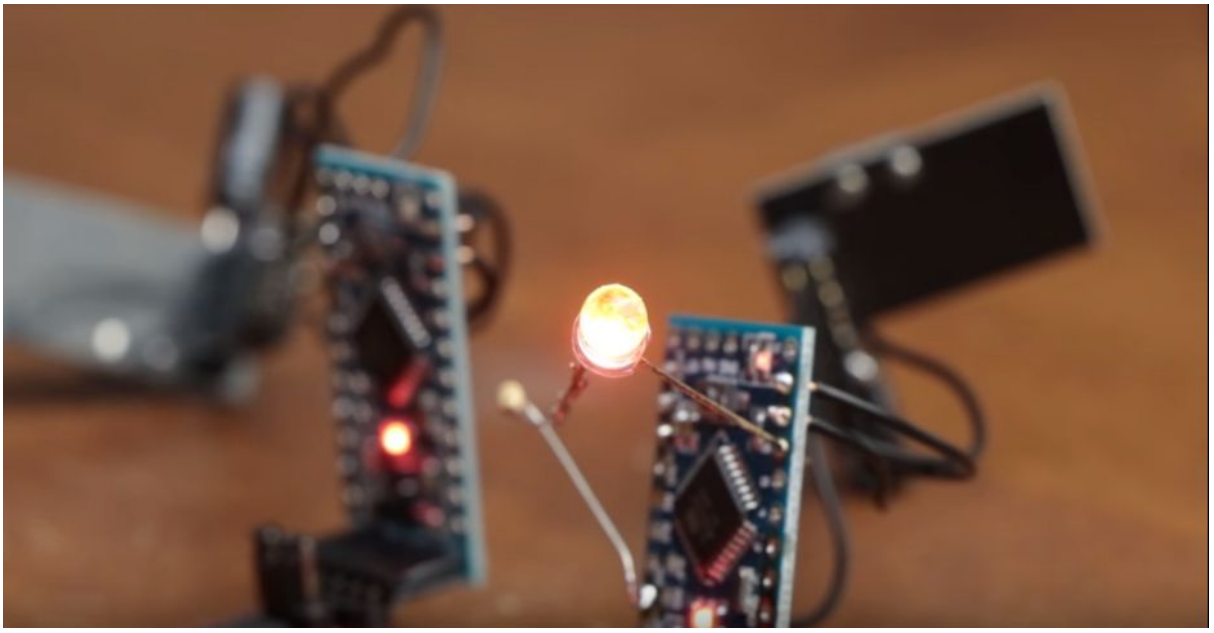
nRF24L01

- Cheap hardware
- They offer more setting options than the generic 433Mhz Rf Transmitter and Receiver Module
- They allow (half or full) duplex transmission between both modules

- It transmits at a frequency of 2.4 GHz
- Traffic capturing for network with known parameters (channel, baud rate, base address)
- Analysis can be done using Wireshark network protocol analyzer (able to capture data) (Yveaux.blogspot.com, 2019)
- Possible to analyze protocols which use the nRF24 for data transport in the network
- Uses TMRh20 library (GitHub, 2019)
- Range is 5 meters

LoRa Module

- A half-duplex (HDX) system provides communication in both directions, but only one direction at a time (not simultaneously)
- It transmits at a frequency at 433/470 MHz
- Range is more than 300 meters



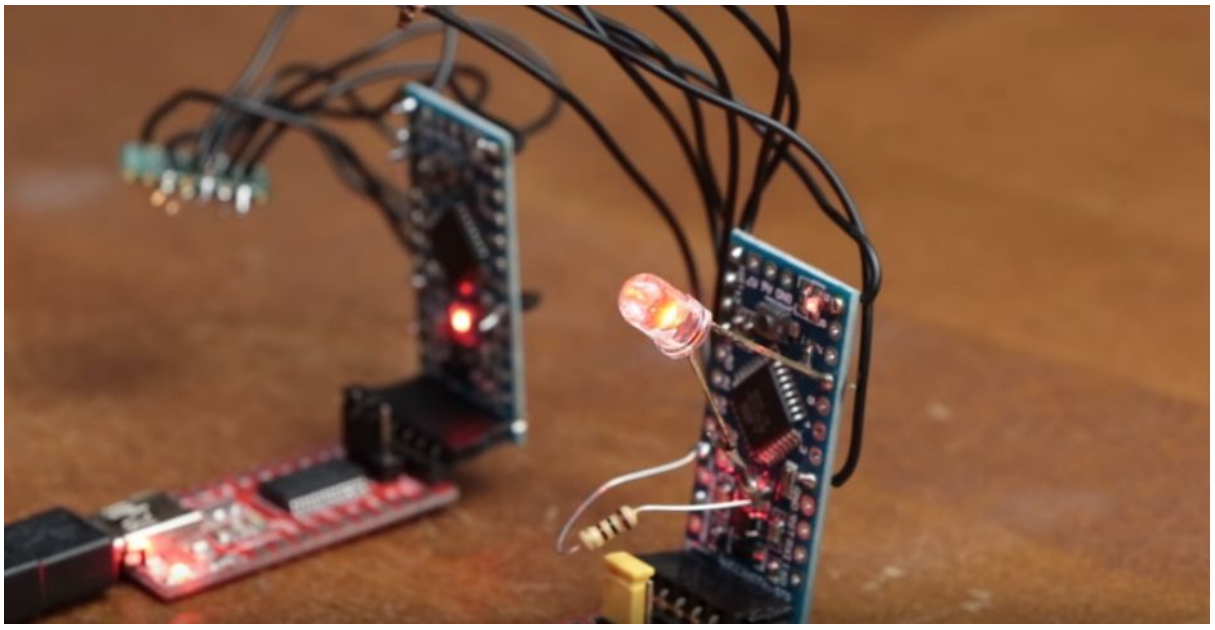


Figure 32 and 33 LoRa modules Rf Transmitter and Receiver Module

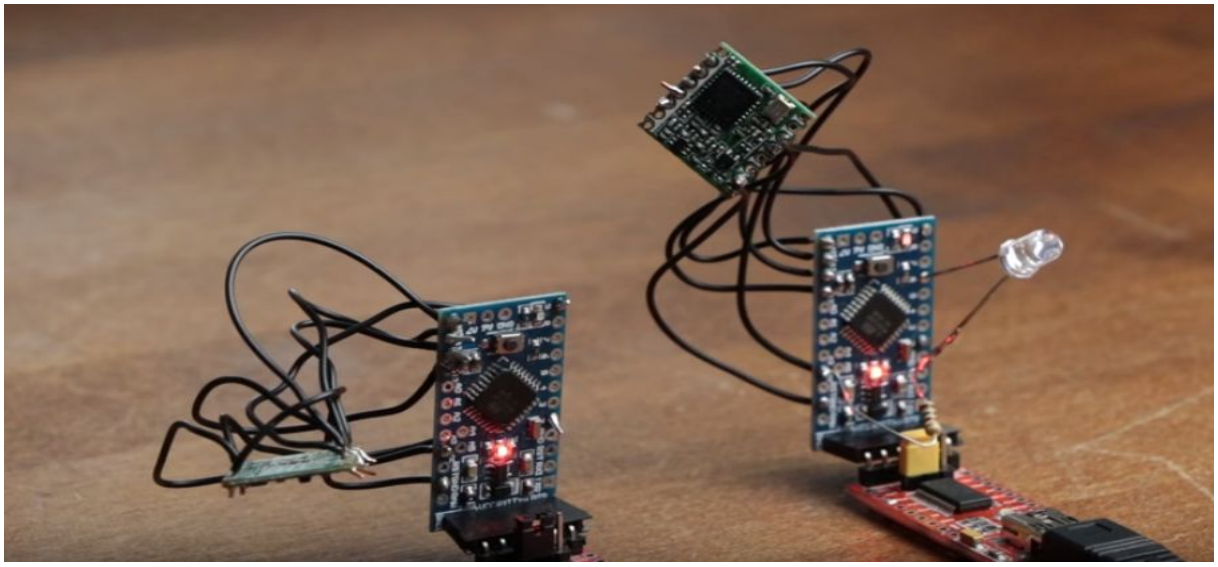


Figure 34 LoRa modules Rf Transmitter and Receiver Module

Summary of different communication modules

	Generic RF module	nRF24L01+ module	LoRa module
Data transfer Rate	4 KB/S	2 MB/S	37.5 KB/S
Range	20 - 200 meters	5 meters	> 300 meters
Frequency	433 MHz	2.4 GHz	433/470

3.1.4 PLX-DAQ Software

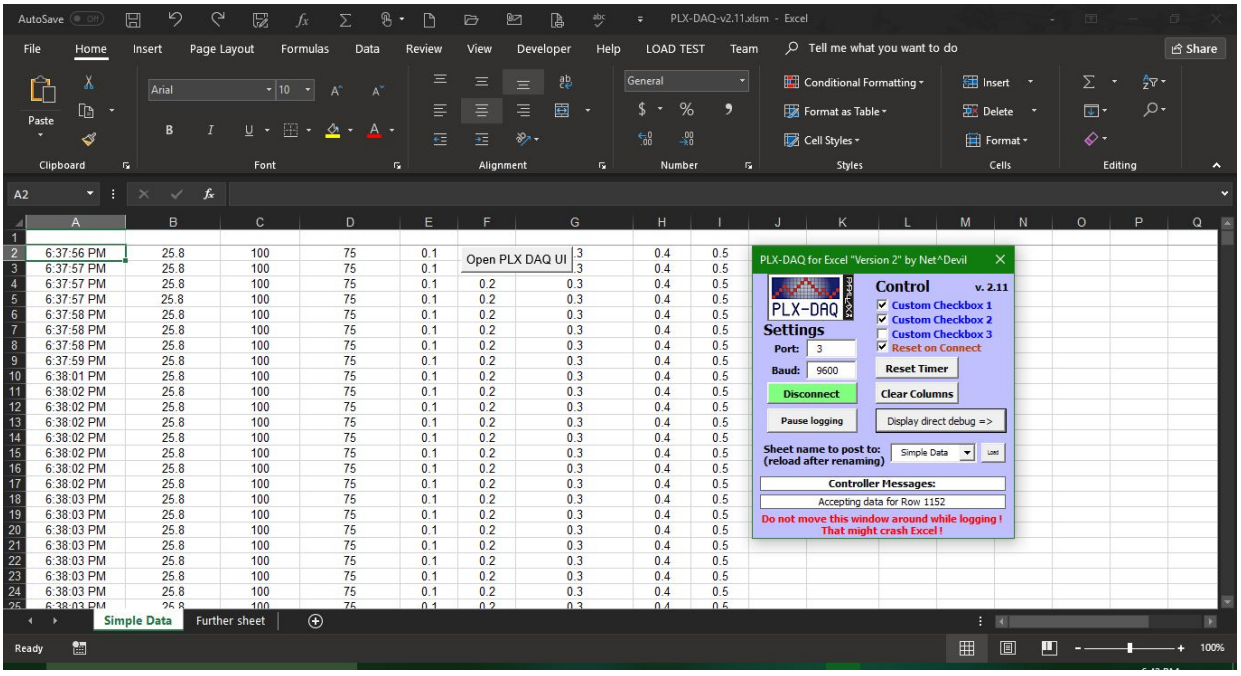


Figure 35 PLX-DAQ plugin for a Spreadsheet

PLX-DAQ is a Parallax microcontroller data acquisition add-on tool for Microsoft Excel which acquires up to 26 channels of data from any Parallax microcontrollers and drops the numbers into columns as they arrive.. Any of microcontrollers connected to any sensor and the serial port of a PC can now send data directly into Excel. PLX-DAQ provides easy spreadsheet analysis of data collected in the field, laboratory analysis of sensors and real-time equipment monitoring.

PLX-DAQ has the following features:

- Plot or graph data as it arrives in real-time using Microsoft Excel

- Record up to 26 columns of data
- Mark data with real-time (hh:mm:ss) or seconds since reset
- Read/Write any cell on a worksheet
- Read/Set any of 4 checkboxes on control the interface
- Example code for the BS2, SX (SX/B) and Propeller available
- Baud rates up to 128K
- Supports Com1-15

The baud rate in the PLX-DAQ software and the microcontroller baud should be the same else it will not work.

```
void setup(){
     /* the baud rate below should be the same baud rate as in the PLX-DAQ
software. Here it is 9600. Therefore, 9600 should also be set in the PLX-DAQ
software */
        Serial.begin(9600);
}

void loop(){
  Serial.print("DATA");
  Serial.print(",");
  Serial.print("TIME");
  Serial.print(",");
  Serial.print(temperature_value);
  Serial.print(",");
  Serial.print(speedometer_value);
  Serial.print(",");
  Serial.print(battery_value);
  Serial.print(",");
  Serial.print(current_value_1);
  Serial.print(",");
  Serial.print(current_value_2);
  Serial.print(",");
  Serial.print(current_value_3);
  Serial.print(",");
  Serial.print(current_value_4);
  Serial.print(",");
  Serial.println(current_value_5);
}
```

As shown above, Serial.print(",") is used to separate all the received variables into distinct columns. As shown above, 9 distinct columns will be created for 9 separate variables.
All the variables up to Serial.println(current_value_5); will be printed in every row in the excel file.
All of the received raw data should be processed before data analysis to be carried out.

Section 2
Telemetry Software

3.2 Software Tools

3.2.1 Visual Studio Code

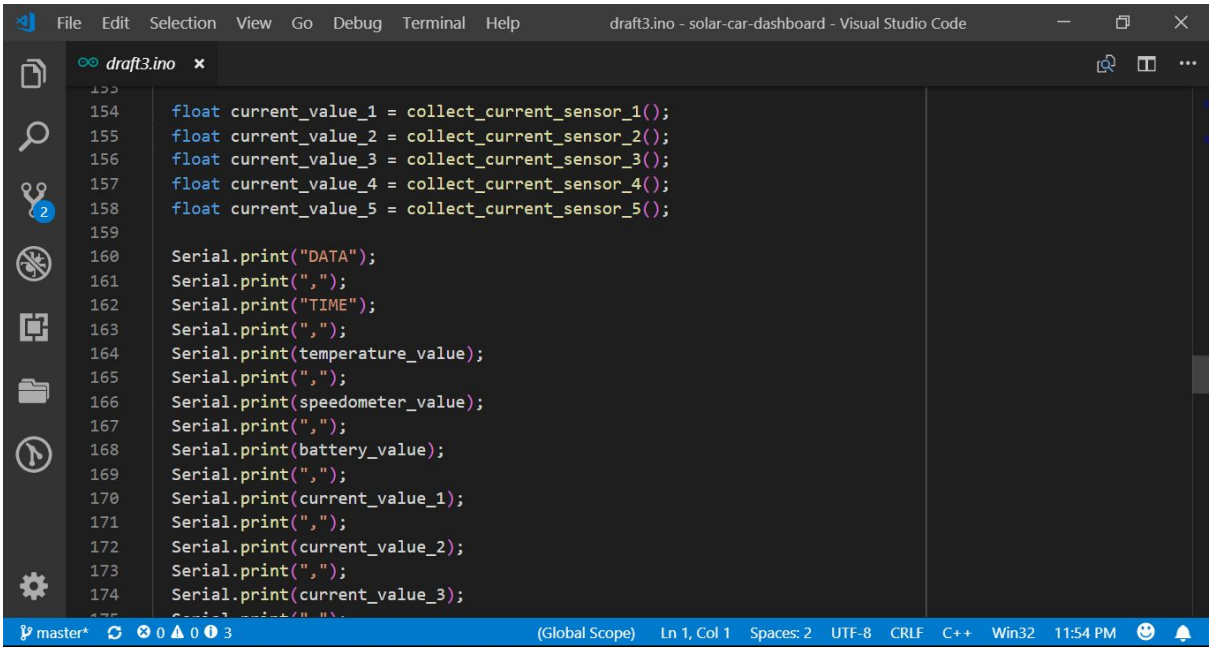


Figure 36 Visual Studio Code Editor

Visual Studio Code combines the simplicity of a source code editor with a powerful developer tooling, like IntelliSense code completion and debugging.

First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment, and more time executing on your ideas.

VS Code is also cross platform. Visual Studio Code features a lightning fast source code editor, perfect for day-to-day use. With support for hundreds of languages, VS Code helps to be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more. Intuitive keyboard shortcuts, easy customization and community-contributed keyboard shortcut mappings let you navigate your code with ease.

Visual Studio Code includes built-in support for IntelliSense code completion, rich semantic code understanding and navigation, and code refactoring.It features a robust and an extensible architecture. Architecturally, Visual Studio Code combines the best of web, native, and language-specific technologies. Using Electron, VS Code combines web technologies such as JavaScript and Node.js with the speed and flexibility of native apps.

3.2.2 Arduino Integrated Development Environment

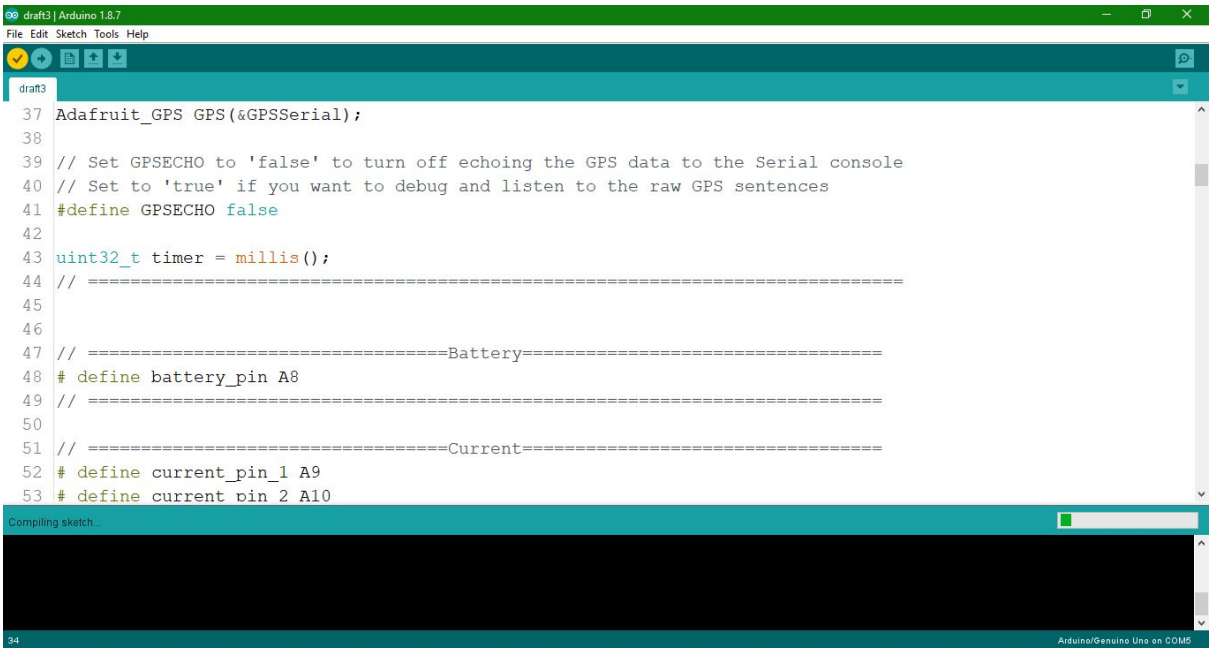


Figure 37 Arduino IDE

The Arduino IDE is a very basic platform to write code and upload it to the board. It is also cross-platform. The environment is written in Java and based on Processing and other open-source software. This software does not support hardware debugging and software debugging which is very essential for our telemetry purpose and is very limited for electrical engineers. Nevertheless, it enables a beginner to get started very easily.

Libraries

- DHT Sensor Library 1.2.3 version
- GPS Library Latest version

3.2.3 Atmel Studio 7

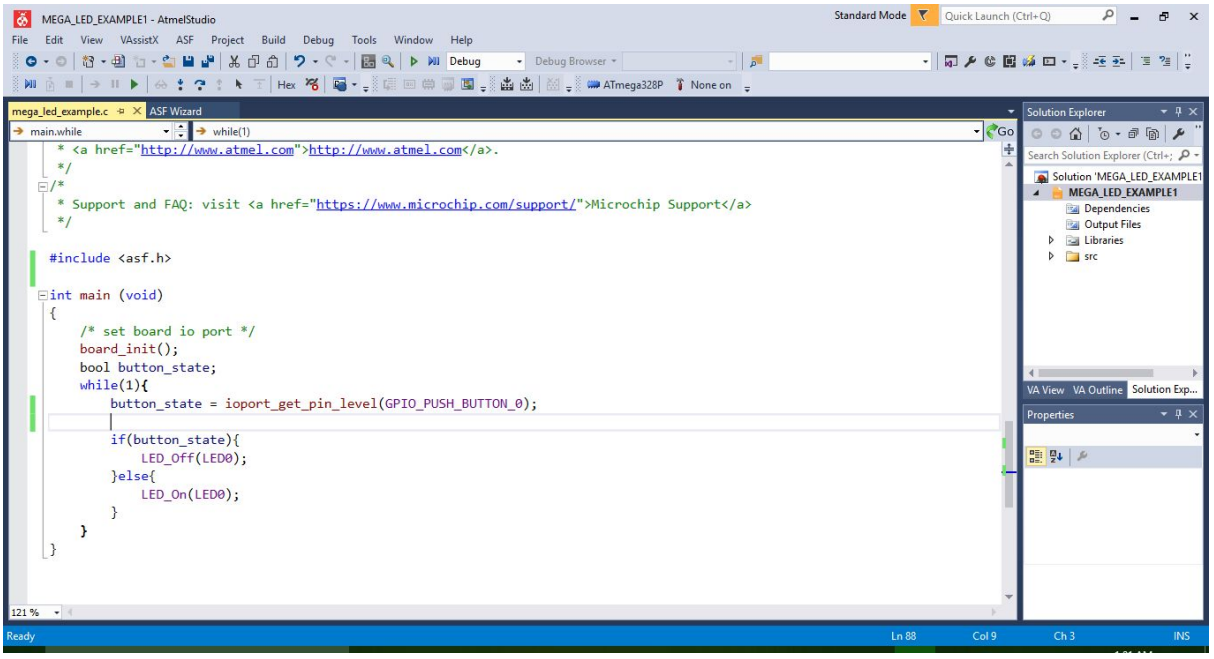


Figure 38 Atmel Studio 7

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR and SAM microcontroller applications. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. It also connects seamlessly to the debuggers, programmers and development kits that support AVR and SAM devices. The Atmel Studio has many advantages over the Arduino IDE. For example, Atmel studio

- Supports for 500+ AVR and SAM devices
- Has a vast source code library, including drivers, communication stacks, 1,600+ project examples with source code, graphic services and touch functionality through Advanced Software Framework (ASF)
- Has an advanced debugging features include complex data breakpoints, nonintrusive trace support (SAM3 and SAM4 devices), statistical code profiling, interrupt trace/monitoring, polled data tracing (Cortex-M0+ devices), real-time variable tracking with optional timestamping.
- Has a full chip simulation for an accurate model of CPU, interrupts, peripherals, and external stimuli and
- Can create transparent debug views into CPU and peripherals for easy code development and debugging

3.3 AT Mega 328p microcontroller C++ flowchart

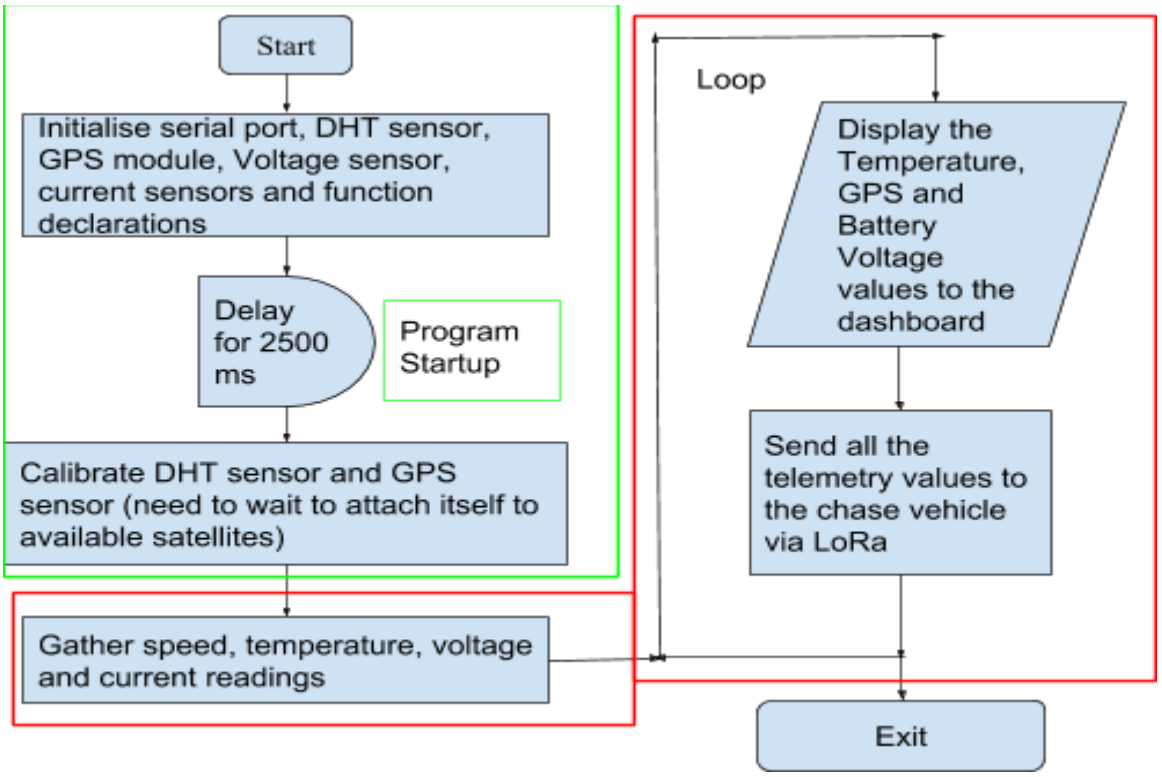


Figure 39 AT Mega 2560 flowchart

3.4 AT Mega 2560 C and C++ program for Raspberry Pi

```
#include "DHT.h"
#define DHTPIN 31     // what digital pin we're connected to
#define DHTTYPE DHT22   // DHT22  (AM2302), AM2321

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors.  This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);
```

The above code will initialise the DHT22 sensor and will configure all the required parameters such as the pin number and the DHT sensor type.

```
#include <Adafruit_GPS.h>

// what's the name of the hardware serial port?
#define GPSSerial Serial1

// Connect to the GPS on the hardware port
Adafruit_GPS GPS(&GPSSerial);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences
#define GPSECHO false

uint32_t timer = millis();
```

The above code includes the Adafruit GPS library and initialises all the required parameters. The hardware serial port can be any Serial port in the Arduino Mega development board. We are defining GPSECHO to false because we do not need to other extra information, however it is used only for debugging purposes.

```
// ============================Battery
Sensor============================
# define battery_pin A8

// ====================================Current
Sensor==================
# define current_pin_1 A9
# define current_pin_2 A10
# define current_pin_3 A11
# define current_pin_4 A12
# define current_pin_5 A13
```

The battery and current sensors are assigned to the respective pin numbers as shown above.

```
// lambda declarations
float collect_temperature();
float collect_speed();
float collect_battery();

float collect_current_sensor_1();
float collect_current_sensor_2();
```

```
float collect_current_sensor_3();
float collect_current_sensor_4();
float collect_current_sensor_5();
```

A function declaration tells the AVR-GCC compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function. Here functions are used to make testing and debugging easier. Functions also increase readability, as well.
The 2 most important uses of functions are :

- Reusability. Once a function is defined, it can be used over and over and over again. We can invoke the same function many times in our program, which saves us many problems.
- Abstraction. We do not need to know the internal details of how a function works. We only need to know the name of the function, what the function does, what arguments must given to the function and what kind of result the function returns.

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

 // ===============DHT
22==============================
 dht.begin();
 // ============End of DHT 22============================


// ======================Ultimate GPS shield==========
//while (!Serial);  // uncomment to have the sketch wait until Serial is ready

 // connect at 115200 so we can read the GPS fast enough and echo without
dropping chars
 // also spit it out
 // Serial.begin(9600);
 // Serial.println("Adafruit GPS library basic test!");

 // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
 GPS.begin(9600);
 // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
data) including altitude
 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
 // uncomment this line to turn on only the "minimum recommended" data
 //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
 // For parsing data, we don't suggest using anything but either RMC only or
RMC+GGA since
 // the parser doesn't care about other sentences at this time
 // Set the update rate
 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 1 Hz update rate
```

```
 // For the parsing code to work nicely and have time to sort thru the data, and
 // print it out we don't suggest using anything higher than 1 Hz

 // Request updates on antenna status, comment out to keep quiet
 GPS.sendCommand(PGCMD_ANTENNA);

 delay(1000);

// Ask for firmware version
// GPSSerial.println(PMTK_Q_RELEASE);
// ========================End of Ultimate GPS
shield==================

}
```

The above setup() function initialises the baud rate which means the number of time signal changes its state.When the signal is binary then baud rate and bit rate are the same. The setup() function is called when a C program is being executed . It initializes variables, pin modes, starts using the required libraries defined earlier. The setup() function will only run once, after each power up or reset of the Arduino development board.

```
void loop() {

 //  testing purposes
 //  float temperature_value = 25;
 //  float speedometer_value = 100;
 //  float battery_value = 75;
 //  float current_value_1 = 0.10;
 //  float current_value_2 = 0.20;
 //  float current_value_3 = 0.30;
 //  float current_value_4 = 0.40;
 //  float current_value_5 = 0.50;

 float temperature_value = collect_temperature();
 float speedometer_value = collect_speed();
 float battery_value = collect_battery();

 float current_value_1 = collect_current_sensor_1();
 float current_value_2 = collect_current_sensor_2();
 float current_value_3 = collect_current_sensor_3();
 float current_value_4 = collect_current_sensor_4();
 float current_value_5 = collect_current_sensor_5();
```

```
 Serial.print("DATA");
 Serial.print(",");
 Serial.print("TIME");
 Serial.print(",");
 Serial.print(temperature_value);
 Serial.print(",");
 Serial.print(speedometer_value);
 Serial.print(",");
 Serial.print(battery_value);
 Serial.print(",");
 Serial.print(current_value_1);
 Serial.print(",");
 Serial.print(current_value_2);
 Serial.print(",");
 Serial.print(current_value_3);
 Serial.print(",");
 Serial.print(current_value_4);
 Serial.print(",");
 Serial.println(current_value_5);

}
```

The `loop()` function loops sequentially, allowing the C program update its sensor values. The sensor values are updated via the respective functions. `Serial.print();` prints data to the serial port as an ASCII text. We will read the serial port from our python program to extract the values to show in the dashboard. The `Serial.print();` is also used to send telemetry values to the chase vehicle via LoRa using the Tx and the Rx pins of the Arduino Mega development board.

```
float collect_temperature() {
 // Wait a few seconds between measurements.
 delay(250);

 // Reading temperature or humidity takes about 250 milliseconds!
 // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
 // Read temperature as Celsius (the default)
 float t = dht.readTemperature();
 // Read temperature as Fahrenheit (isFahrenheit = true)
 // float f = dht.readTemperature(true);

 // Check if any reads failed and exit early (to try again).
 if (isnan(t)) {
   Serial.println("Failed to read from DHT sensor!");
   return;
 }
 return (t);
}
```

The collect_temperature() function collects the temperature value from the DHT sensor. If there is an error reading the temperature, an error message is printed out.

```
float collect_speed() {

 // read data from the GPS in the 'main loop'
 char c = GPS.read();
 // if you want to debug, this is a good time to do it!
 if (GPSECHO)
   if (c) Serial.print(c);
 // if a sentence is received, we can check the checksum, parse it...
 if (GPS.newNMEAreceived()) {
   // a tricky thing here is if we print the NMEA sentence, or data
   // we end up not listening and catching other sentences!
   // so be very wary if using OUTPUT_ALLDATA and trying to print out data
   // Serial.println(GPS.lastNMEA()); // this also sets the newNMEAreceived() flag
to false
   if (!GPS.parse(GPS.lastNMEA())) // this also sets the newNMEAreceived() flag to
false
     return; // we can fail to parse a sentence in which case we should just wait for
another
 }
 // if millis() or timer wraps around, we'll just reset it
 if (timer > millis()) timer = millis();

 // approximately every 2 seconds or so, print out the current stats
 if (millis() - timer > 2000) {
   timer = millis(); // reset the timer

   // Serial.print("\nTime: ");
   // Serial.print(GPS.hour, DEC); Serial.print(':');
   // Serial.print(GPS.minute, DEC); Serial.print(':');
   // Serial.print(GPS.seconds, DEC); Serial.print('.');
   // Serial.println(GPS.milliseconds);
   // Serial.print("Date: ");
   // Serial.print(GPS.day, DEC); Serial.print('/');
   // Serial.print(GPS.month, DEC); Serial.print("/20");
   // Serial.println(GPS.year, DEC);
   // Serial.print("Fix: "); Serial.print((int)GPS.fix);
   // Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
   if (GPS.fix) {
     // Serial.print("Location: ");
     // Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
     // Serial.print(", ");
     // Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);
     // Serial.print("Speed (knots): ");
     return (GPS.speed * 1.852);
```

```
    // Serial.print("Angle: "); Serial.println(GPS.angle);
    // Serial.print("Altitude: "); Serial.println(GPS.altitude);
    // Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
  }
 }
}
```

Hardware Serial parsing is used to get the GPS speed value.

```
// If using hardware serial (e.g. Arduino Mega):
// Connect the GPS TX (transmit) pin to Arduino RX1, RX2 or RX3
// Connect the GPS RX (receive) pin to matching TX1, TX2 or TX3

// If using hardware serial (e.g. Arduino Mega), comment out the
// above SoftwareSerial line, and enable this line instead
// (you can change the Serial number to match your wiring):

HardwareSerial mySerial = Serial1;
```

We comment out only the unnecessary portions to get the speed from the GPS module.
Here, we multiply by 1.852 because the original speed is in knots, therefore we need to convert from knots to km/h to show this value in the dashboard.

```
float collect_battery() {
 // read the input on analog pin 0:
 //  float batteryValue = analogRead(batteryPin); // Read OUTPUT = 0 V to 5 V.
 //  float value = (batteryValue / 1023) * 5; //
 //  float realValue = value * 30;

 return (((analogRead(battery_pin) / 1023) * 5) * 30);
}
```

The analogRead(A0) function returns an integer value between 0 and 1023. The AT Mega 2560 contains a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V) into integer values between 0 and 1023. This yields a resolution of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit.

As the resolution of the ADC converter increases, the accuracy also increases too. The AT Mega 2560 takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second which is not industrial standard.

We multiply with 30 because the 1V measured in the arduino mega development board corresponds to 30V in the battery system and 5V measured in the arduino mega development board corresponds to 150V in the battery system.

```
float collect_current_sensor_1() {
 float output_voltage_value_1 = analogRead(current_pin_1) * (5.0 / 1023.0);
 float current_value_1 = (output_voltage_value_1 - (5.0 / 2.0)) / 0.037;
 return current_value_1;
}
```

$$Vo(output\ voltage) = m(0.037) \times X(Ip\ aka\ .sensed\ current) + C(2.5V)$$

A linear regression equation $(Y = m \times X + C)$ is used to collect the current values.

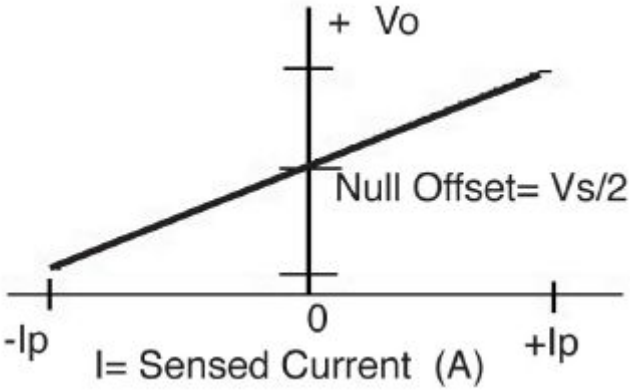


Figure 40 Linear Curve for AMPLOC 25 current sensor

Based on the above figure, the y-intercept is $Vs/2$ (supply voltage / 2 = 2.5V). The Y is Vo (output voltage). The gradient is given in the datasheet as sensitivity in mV/A which is 37mV/A for AMP 25 sensors. Now, the only thing that we need to find is the X value. Rearranging the equation, we get $\frac{Y-c}{m}$ = X (I = Sensed Current (A)) , which is the result we require for our telemetry system.

3.5 AT Mega 2560 C and C++ program for Lattepanda

There are only 2 major differences when porting the above code to the lattepanda board.

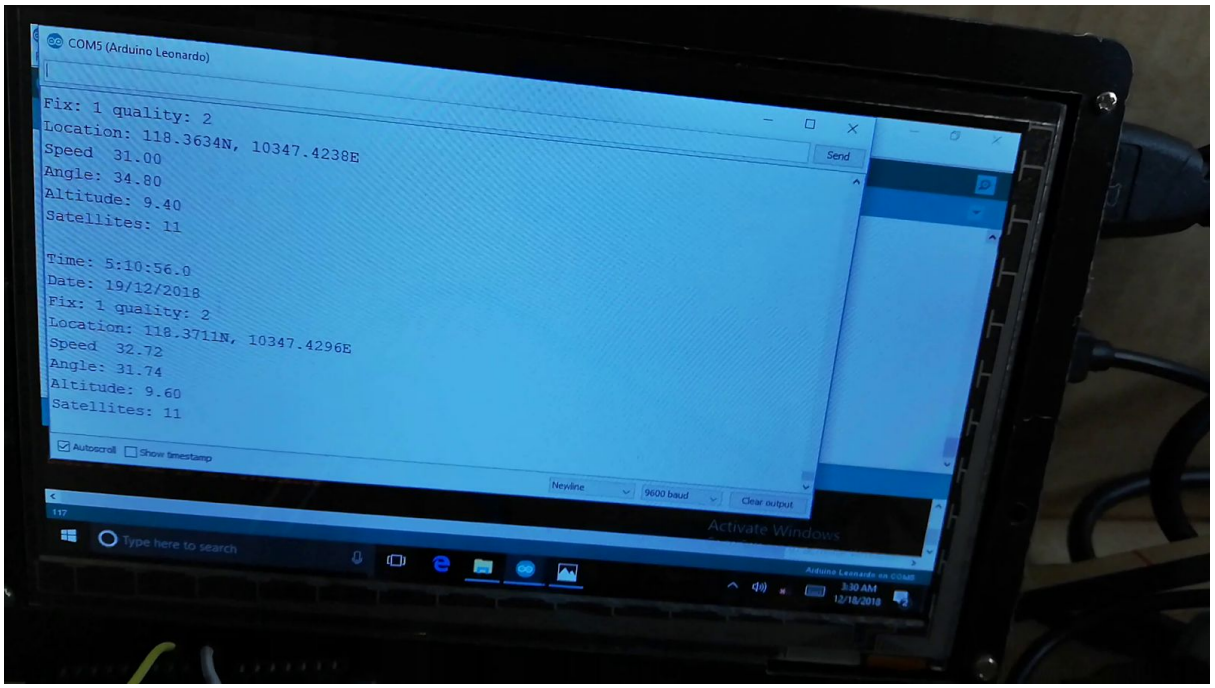


Figure 41 Testing GPS for Lattepanda in a vehicle

The first major difference is the GPS code.

We need to use software serial parsing to listen to the GPS module in an interrupt based way which allows the program to have more 'freedom'. We just parse when a new NMEA sentence is available! Then, we access data when desired.

Since we are using Adafruit GPS shield, we need to change the following

```
#include <SoftwareSerial.h>
// DO not use SoftwareSerial mySerial(3, 2); ->
SoftwareSerial mySerial(8, 7);
// and make sure the switch is set to SoftSerial

// Ask for firmware version -> We do need it, so comment out
// mySerial.println(PMTK_Q_RELEASE);
```

The last major change is LoRa code, where we will use `Serial1.print("sensorValue");` to allow values to be sent to the LoRa. This is possible because Leonardo uses an ATmega32u4 which is a multi-serial micro-controller. USB connection works on default serial communication which can be used via `Serial.print("somevalue")`. On the other hand, RX/TX on pins 0, 1 which are actually RXD1 and TXD1. You can use USB and RX/TX (hardware serial) at the same time. This is how the Lattepanda is able to use LoRa and GPS.

```
// Send to LoRa
  Serial1.print("DATA");
  Serial1.print(",");
  Serial1.print("TIME");
  Serial1.print(",");
  Serial1.print(temperature_value);
  Serial1.print(",");
  Serial1.print(speedometer_value);
```

```
Serial1.print(",");
Serial1.print(battery_value);
Serial1.print(",");
Serial1.print(current_value_1);
Serial1.print(",");
Serial1.print(current_value_2);
Serial1.print(",");
Serial1.print(current_value_3);
Serial1.print(",");
Serial1.print(current_value_4);
Serial1.print(",");
Serial1.println(current_value_5);
```

3.6 SunSPEC Dashboard's Graphical User Interface in Raspberry Pi and Lattepanda

Raspberry Pi

The Raspberry Pi model 3 runs on Raspbian operating system. However, it is **inefficient** to use the raspbian operating system as it is not a true real-time operating system. To understand a real time operating system, we need to understand some fundamental concepts,

General purpose / Embedded Linux operating systems (Freertos.org, 2019):
A "soft" real time operating system's typical latencies are in the order of tens or hundreds of microseconds, whereas a typical RTOS real-time kernel can achieve from latencies from zero to a few microseconds. Embedded Linux needs significant CPU resources just for booting up the system(which may take several seconds). A embedded linux scheduling algorithm does not guarantee a deterministic behaviour and timely response events and interrupts.

Real Time Operating Systems (RTOS): (Freertos.org, 2019)
The scheduler in an RTOS is designed to provide a expected (normally described as deterministic) execution pattern. This is particularly of interest to SunSPEC telemetry as the telemetry system have real time requirements. A real time requirement is one that defines that the telemetry embedded system must respond to a certain event within a strictly defined time (the deadline).
The characteristic of the RTOS scheduler is deterministic if it guarantees to meet real time requirements.The user is allowed to assign a priority to each thread of execution to achieve determinism in a traditional real time schedulers ,such as the scheduler used in FreeRTOS.

An object oriented programming design methodology is chosen because it is apt for SunSPEC's telemetry system. There are many programming paradigms like Functional programming (FP), Object-oriented programming (OOP) and Procedural programming

What is object oriented programming (OOP)?
Object-Oriented Programming is a programming methodology or paradigm to design a program using classes and objects.

Advantages of Object-Oriented Programming (Burleson, 2019):

1. Improved software maintainability, reliability and flexibility
    a. The primary goal of object-oriented programing is the assurance that the telemetry system will enjoy a longer life while having far smaller maintenance work.
    b. Object oriented software is also easier to maintain. If a part of a telemetry system needs to be updated, changed or fixed in case of any issues, only that particular section is affected, unaffecting the rest of the program. Large scale changes do not take place, thus ensuring reliability and flexibility. This is possible due to the fact that the design is modular in OOP.
2. Faster development:
    i. The reuse of code enables faster development.
    ii. Object-oriented programming languages such as python come with a rich libraries of objects such as numpy, opencv and pygame. Thus, the code developed for the telemetry system is also reusable in future projects.
3. Real-World Modeling
    a. Objects and classes model the real world better than structured programming.. The model not based on data and processing, it is based on objects. An object is an instance of a class that has access to all the attributes and methods of that class.
4. Lower cost of development
    a. The reuse of telemetry code also lowers the cost of development.
5. Improved software-development productivity
    a. Object-oriented programming is modular, as it provides separation of duties in an object-based programming development.
    b. Objects can be extended to include new attributes and behaviors. Objects can also be reused across any applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques such as structured programming.

Disadvantages of OOP:

1. Steep learning curve
    a. The design process involved in object-oriented programming may not be intuitive for some people, and it can take time to get used to it.
    b. It is difficult to create programs based on interaction of objects. Some of the key programming techniques, such as inheritance, polymorphism, and encapsulation can be tricky to understand.
2. Not suitable for all types of problems:
    a. There are problems that that are more suitable to functional-programming style, logic-programming style, or procedure-based programming style, and applying object-oriented programming will not result in efficient programs.

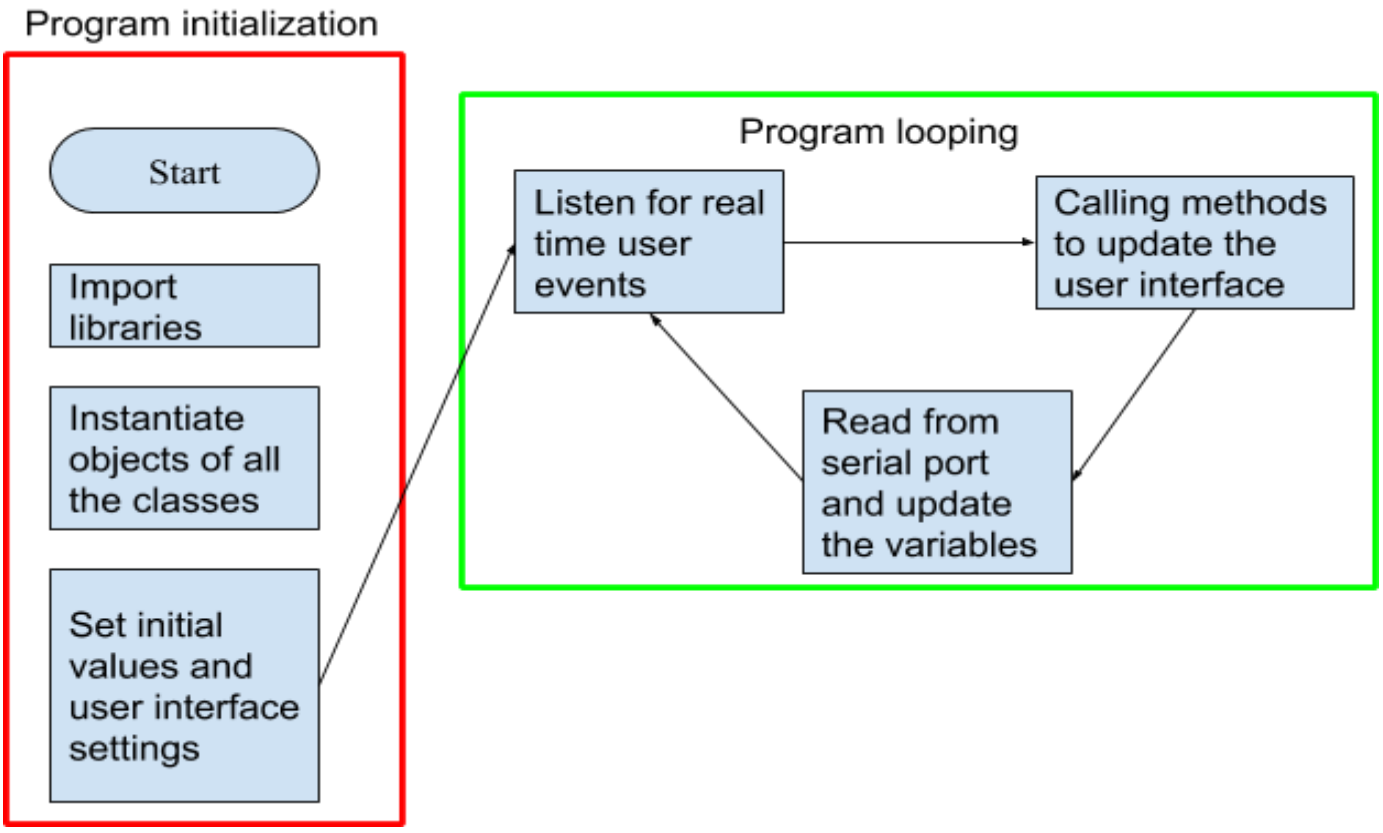


Figure 42.1 : Flowchart of the python program

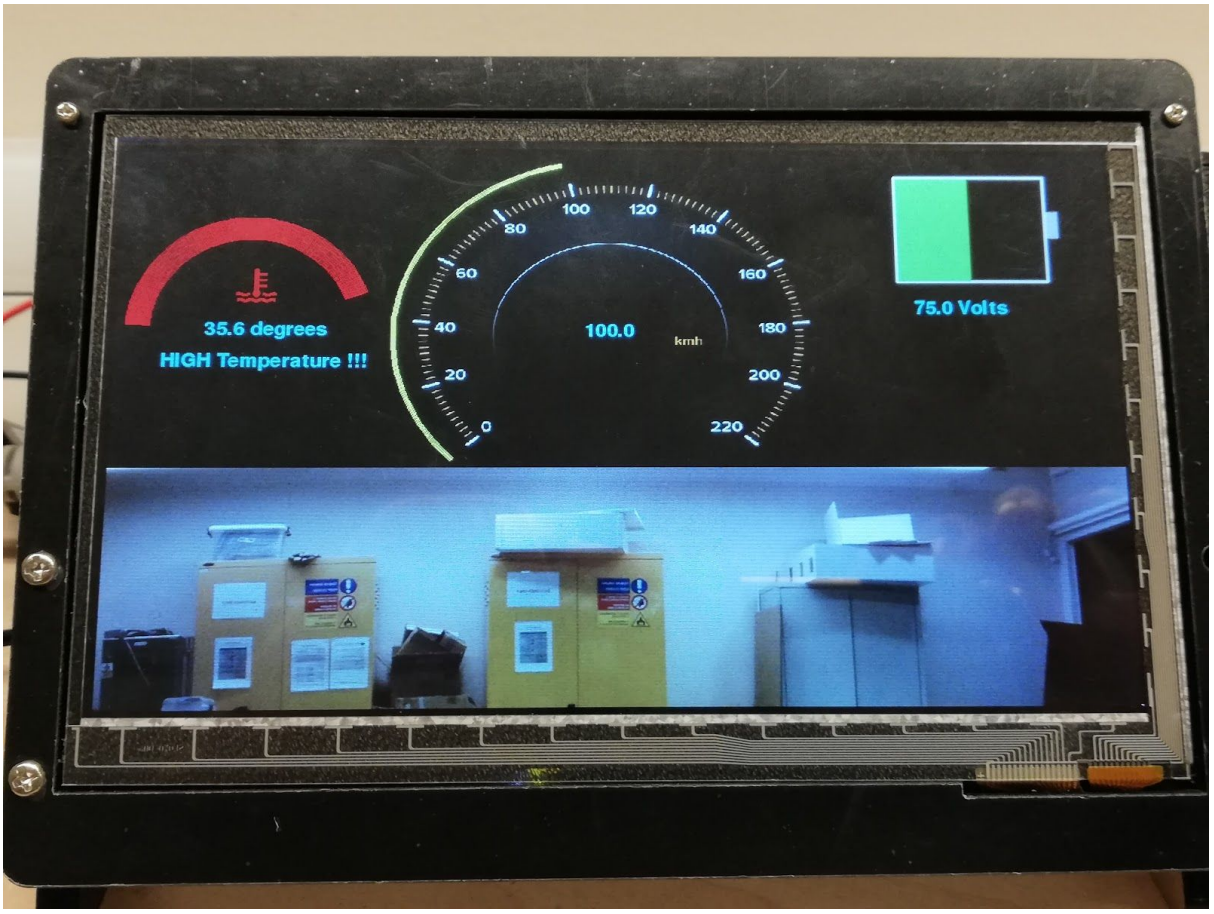


Figure 42.2 Dashboard created using python

```
try:
   import pygame
   import math
   import random
   import picamera
   from pygame.locals import *
  import serial
   import time
except ImportError as ImpErr:
   print("Check your imports !")
   print(str(ImpErr))
```

The libraries shown above are used to provide a collection of functions and methods that allows us to perform lots of functions without writing our own code.

Name of the Library	Description
pygame	pygame is a Free and Open Source python programming language library for making multimedia applications like games built on top of the excellent SDL library. pygame is highly portable and

	runs on nearly every platform and operating system.
math	It provides access to the mathematical functions defined by the C standard.
random	This module implements pseudo-random number generators for various distributions.
picamera	This package provides a pure Python interface to the Raspberry Pi camera module.
pyserial	This module encapsulates the access for the serial port. This library gives access to the port settings through python properties. It supports for different byte sizes, stop bits, parity and flow control with RTS/CTS and/or Xon/Xoff. This library works with or without receive timeout and supports file like API with "read" and "write" ("readline" etc. also supported).
time	This module provides various time-related functions.

How to install a python library - Windows installation

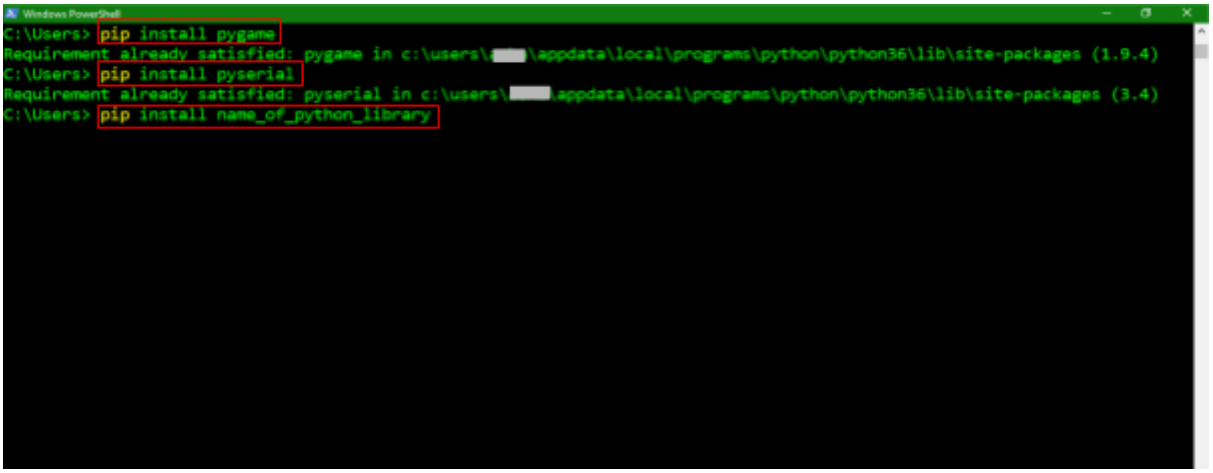


Figure 43 Windows pip installation

To install a python library, use pip install pyserial , the pyserial is the name of the python library name.

Unix / Linux Installation

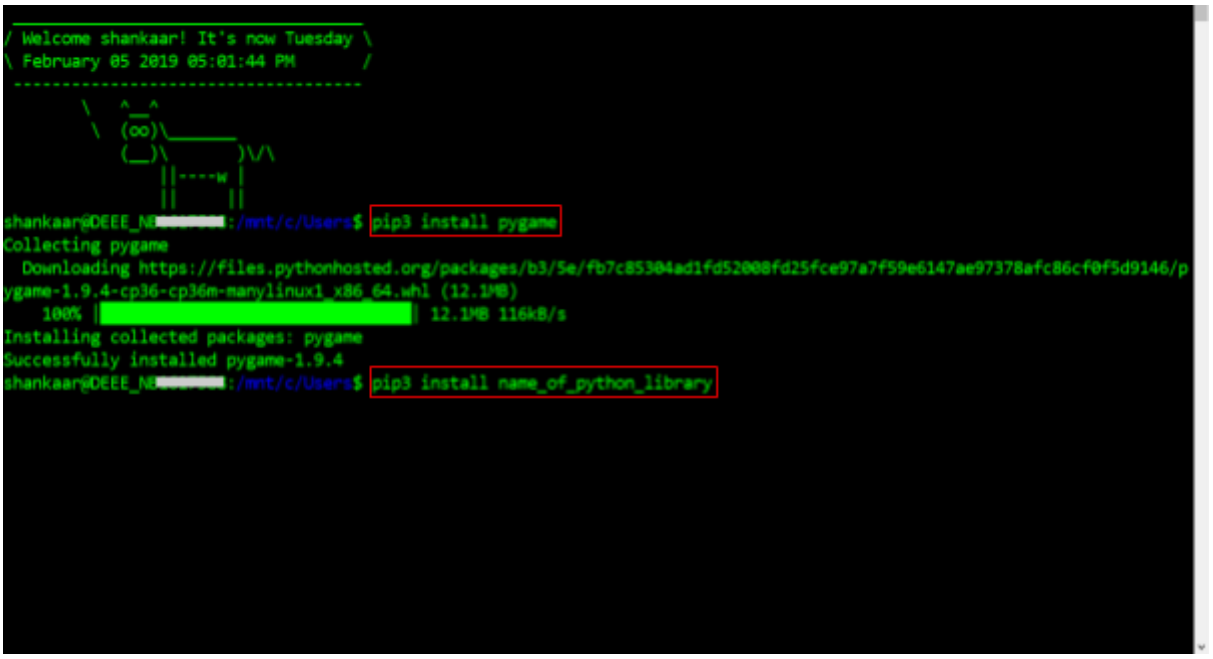


Figure 44 Unix/Linux pip installation

To install a python library, use pip3 install pyserial , pip3 refers to python version 3 and pip refers to python version 2. Python version 3 is the future and should be used for all projects. Python 2 is going to be deprecated in year 2020.

Note : The python PATH environment variable must be set properly or else you will not be able to install the libraries using pip.

```
class Initialise(object):
  # Colour definitions (Red ,Green, Blue)
  # All the below code are class variables
  BLACK = (0, 0, 0)
  WHITE = (255, 255, 255)
  RED = (255, 0, 0)
  GREEN = (0, 255, 0)
  BLUE = (0, 0, 255)
  YELLOW = (255, 255, 0)
  CYAN = (0, 255, 255)
  BROWN = (83, 91, 36)
  SILVER = (192, 192, 192)

  pygame.init()
  pygame.display.set_caption("Dashboard")
  clock = pygame.time.Clock()
  resolution = (1024, 600)
  screen = pygame.display.set_mode(resolution, pygame.RESIZABLE)
  ser = serial.Serial('/dev/ttyACM0', 9600, 8, 'N', 1, timeout=5)
```

The Initialise class declares all the necessary details to initialise the dashboard for example setting the name, screen size, starting up the pygame and opening the serial port to listen the data from the Arduino. All of the functions' parameters and its description are excellently documented in the official pygame website. https://www.pygame.org/docs/tut/newbieguide.html (Clark, 2019)

```
class Text(Initialise):
    def message_display(self, text, x_position, y_position):
        largeText = pygame.font.Font("freesansbold.ttf", 20)
        # The text is inside a rectangle and can be referenced by a rectangle.
        textSurface = largeText.render(text, True, Initialise.CYAN)

        # TextSurface, TextRect = text_objects(text, largeText)
        TextRect = textSurface.get_rect()
        TextRect.center = (x_position, y_position)
        Initialise.screen.blit(textSurface, TextRect)
```

The Text class is used to create messages in the dashboard.

```
class Battery(Text):

    def draw_rect(self, battery_value):
        pygame.draw.rect(Initialise.screen, Battery.SILVER, (956, 65, 15, 30))
        pygame.draw.rect(Initialise.screen, Battery.WHITE,
                         (800, 25, 158, 120), 3)

        # pygame.draw.rect(screen, color, (x,y,width,height), thickness)
        # pygame.draw.rect(self.screen, Battery.GREEN, (802, 27, 145, 97))
        if (battery_value > 70 and battery_value <= 140):
            pygame.draw.rect(Initialise.screen, Initialise.GREEN,
                             (802, 27, battery_value, 117))
        elif ((battery_value > 42) and (battery_value <= 70)):
            pygame.draw.rect(Initialise.screen, Battery.YELLOW,
                             (802, 27, battery_value, 117))
        elif (battery_value >= 0 and battery_value <= 42):
            pygame.draw.rect(Initialise.screen, Initialise.RED,
                             (802, 27, battery_value, 117))
            self.bolt_image = pygame.image.load("bolted.png")
            self.bolt_image = pygame.transform.scale(self.bolt_image,
                                                     (100, 110))
            Initialise.screen.blit(self.bolt_image, (830, 33))
            Text.message_display(self, text="LOW Battery !!!", x_position=865,
                                 y_position=200)
```

The Battery class draws 3 rectangles in total. The first 2 rectangles are for the battery exterior and the battery terminal. The third rectangle is drawn dynamically to show the actual battery percentage in the screen. Depending on the actual battery

percentage, the third rectangle is either shrunk or stretched, different colours are shown for different battery values and warning messages and symbols are shown where appropriate.

```
class Speedometer(Initialise):

    def load_image(self):
        self.speedometer_image = pygame.image.load("speed.png")
        self.speedometer_image = pygame.transform.scale(self.speedometer_image,
                                                        (700, 450))
        Initialise.screen.blit(self.speedometer_image, (155, -40))

    def draw_arc(self, speed_value):
        # pygame.draw.arc()
        # Good example arc below ->
        # pygame.draw.arc(self.screen, Speedometer.YELLOW,
        # (235, 75, 525, 525), math.radians(-42), math.radians(223), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
        """ Draws an elliptical arc on the Surface.
        The rect argument is the area that the ellipse will fill.
        The two angle arguments are the initial and final
        angle in radians, with the zero on the right. The width argument is the
        thickness to draw the outer edge.

        TAKE NOTE: <Worth mentioning>
        the initial angle must be less than the final angle;
        otherwise it will draw the full ellipse. """
        pygame.draw.arc(Initialise.screen, Speedometer.YELLOW,
                        (277, 5, 450, 400), math.radians(speed_value),
                        math.radians(224), 5)
```

The speedometer class loads a speed.png image and this image is used as a base to draw the speed arc. The speed_value variable is then used to draw the arc from 0 km/h to 220 km/h.

```
class Temperature(Text):

    def draw_arc(self, temperature_value):

        # Good example arc below ->
        # pygame.draw.arc(Initialise.screen, Initialise.YELLOW, (50, 75,
        # 200, 200), math.radians(0), math.radians(180), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
```

```
""" Draws an elliptical arc on the Surface.
The rect argument is the area that the ellipse will fill.
The two angle arguments are the initial and final angle in radians,
with the zero on the right.
The width argument is the thickness to draw the outer edge.

TAKE NOTE: <Worth mentioning> the initial angle must be less
than the final angle; otherwise it will draw the full ellipse. """
if (temperature_value >= 0 and temperature_value < 45):
    pygame.draw.arc(Initialise.screen, Initialise.RED,
                    (10, 75, 250, 250),
                    math.radians(temperature_value), math.radians(180),
                    25)
    Text.message_display(self, text="HIGH Temperature !!!",
                         x_position=150, y_position=235)

    self.high_temperature_image = pygame.image.load("temperature.jpg")
    self.high_temperature_image = pygame.transform.scale(
        self.high_temperature_image, (50, 50))
    Initialise.screen.blit(self.high_temperature_image, (115, 125))

elif (temperature_value >= 45 and temperature_value < 135):
    pygame.draw.arc(Initialise.screen, Initialise.YELLOW,
                    (10, 75, 250, 250),
                    math.radians(temperature_value), math.radians(180),
                    25)
elif (temperature_value >= 135):
    pygame.draw.arc(Initialise.screen, Initialise.GREEN,
                    (10, 75, 250, 250),
                    math.radians(temperature_value), math.radians(180),
                    25)
    Text.message_display(self, text="LOW Temperature !!!",
                         x_position=150, y_position=250)
```

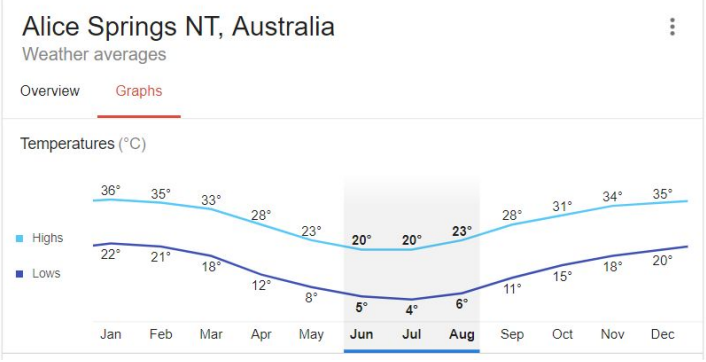


Figure 45 Average Temperature in alice springs

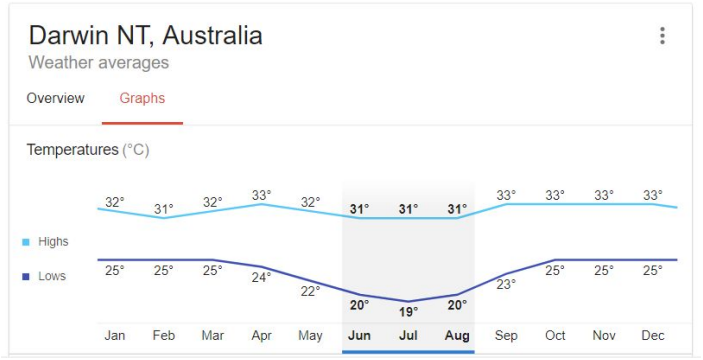


Figure 46 Average temperature in Darwin

Some research has been done to find out the minimum, average and the maximum temperatures from Darwin to Adelaide.

The Temperature class draws an elliptical arc dynamically in real time to show the temperature in the dashboard. The Temperature arc will change the colour of the arc and show warning messages and symbols depending on the temperature value.

```
class Camera(Initialise):
    # The below are declared as class variables
    camera_position = (0, 300, 1024, 300)
    camera_block = pygame.Surface((1024, 300))

    def __init__(self):
        self.camera = picamera.PiCamera()
        self.camera.preview_fullscreen = False
        self.camera.preview_window = (0, 350, 1024, 300)
        self.camera.resolution = (1024, 300)

    def use_camera(self):
        # If your preview is upside-down, you can rotate it with the following
        # code
        # self.camera.rotation = 180
        self.camera.start_preview()
        Initialise.screen.blit(
            Camera.camera_block,
            (Camera.camera_position[0], Camera.camera_position[1]))

    def stop_camera(self):
        self.camera.stop_preview()
```

The Camera class is used to set the camera's position on the screen, initialise the picamera library and show in the display. It is also responsible for other methods such as `use_camera(self)` and `stop_camera(self)` . The `stop_camera(self)` is used to deactivate the camera.

```
class UpdateValues(Initialise):
```

```
def __init__(self):
    self.temperature_value = 4.5
    self.temperature_value_original = 39

   self.speed_value = 30.11
    self.speed_value_original = 25

    self.battery_value = 135
    self.battery_value_original = 75

def transferValues(self):
    if (Initialise.ser.in_waiting):
        try:
            values = Initialise.ser.readline()
            print('values', values)
            decoded = values.decode("utf-8")
            print('Decoded', decoded)
            decoded = str(decoded)
            split_decoded = decoded.split(",")

            self.data = split_decoded[0].strip()
            self.time = split_decoded[1].strip()
            self.temperature_value_original = float(
                split_decoded[2].strip())
            self.speed_value_original = float(split_decoded[3].strip())
            self.battery_value_original = float(split_decoded[4].strip())
            self.current_1 = float(split_decoded[5].strip())
            self.current_2 = float(split_decoded[6].strip())
            self.current_3 = float(split_decoded[7].strip())
            self.current_4 = float(split_decoded[8].strip())
            self.current_5 = float(split_decoded[9].strip())

            #   data = split_decoded[0].strip()
            #   time = split_decoded[1].strip()
            #   temperature = float(split_decoded[2].strip())
            #   speed = float(split_decoded[3].strip())
            #   battery = float(split_decoded[4].strip())
            #   current = float(split_decoded[5].strip())

            self.temperature_value = (-9 / 2 *
                                      self.temperature_value_original) + 180
            self.speed_value = (-133 / 110 *
                                self.speed_value_original) + 224
            self.battery_value = self.battery_value_original

            print("Temperature = {}, speed = {}, battery = {}".format(
                self.temperature_value_original, self.speed_value_original,
                self.battery_value_original))
```

```
        except:
            print(
                "I cannot convert string (Failed to read from DHT sensor) "
                "to float")
```

This UpdateValues class is responsible for reading values from the Arduino serial port. The __init__ method sets some arbitrary values in the dashboard. This is done to confirm the speedometer, battery and temperature are working perfectly.

```
values = Initialise.ser.readline()
```

The values are read in from the Arduino serial port as shown above and decoded using UTF-8 standard. After that, the values are converted to string. Since the values are separated by comma, we can get the individual values using `split_decoded =decoded.split(",")` .
The split() method returns a **list** of **strings** after breaking the given string (`decoded`)by the specified separator `","`. The `split_decoded` is of a list type. (Docs.python.org, 2019).

```
self.data = split_decoded[0].strip()
self.time = split_decoded[1].strip()
self.temperature_value_original = float(split_decoded[2].strip())
self.speed_value_original = float(split_decoded[3].strip())
self.battery_value_original = float(split_decoded[4].strip())
self.current_1 = float(split_decoded[5].strip())
self.current_2 = float(split_decoded[6].strip())
self.current_3 = float(split_decoded[7].strip())
self.current_4 = float(split_decoded[8].strip())
self.current_5 = float(split_decoded[9].strip())
```

The above code is the most crucial part. Since the `split_decoded` is a list type, we will access elements from a list using list index. We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 10 elements will have index from 0 to 9.

Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.The strip() method returns a copy of the string with the leading trailing characters removed.

```
self.data = split_decoded[0].strip()
self.time = split_decoded[1].strip()
```

The above code serves no purpose in the dashboard, but we need to include them here because the DATA and TIME are included in the Serial.print("DATA"); Serial.print("TIME") in the Arduino Lattepanda C code.

```
self.temperature_value_original = float(split_decoded[2].strip())
self.speed_value_original = float(split_decoded[3].strip())
self.battery_value_original = float(split_decoded[4].strip())
self.current_1 = float(split_decoded[5].strip())
self.current_2 = float(split_decoded[6].strip())
self.current_3 = float(split_decoded[7].strip())
self.current_4 = float(split_decoded[8].strip())
self.current_5 = float(split_decoded[9].strip())
```

As shown above, we need to use an explicit type conversion to get a more accurate reading. In explicit type conversion, convert the data type of an object to a required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.This type of conversion is also called typecasting because we cast (change) the data type of the objects. Typecasting can be done by assigning the required data type function to the expression.

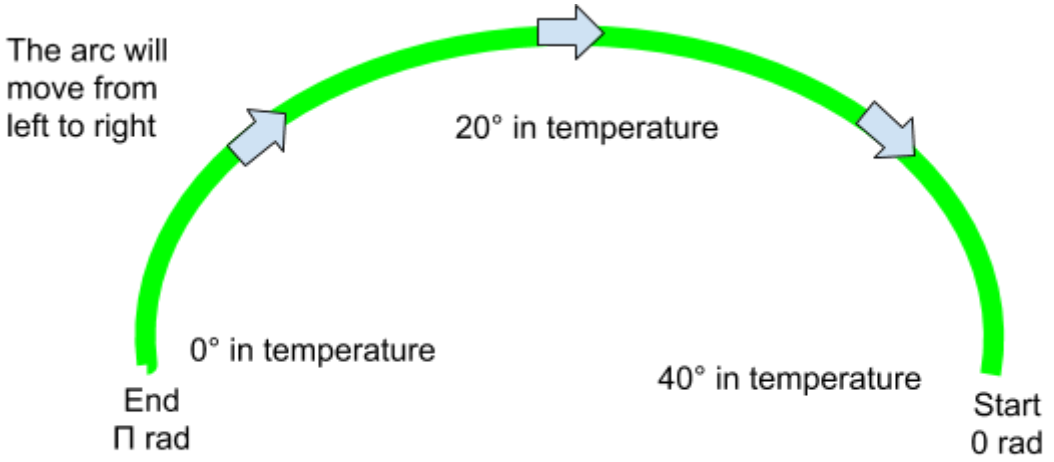
```
self.temperature_value = (-9 / 2 *self.temperature_value_original) + 180
self.speed_value = (-133 / 110 *self.speed_value_original) + 224
```

We use linear regression to draw the speedometer arc and the temperature arc.

Temperature Arc

The temperature arc can be modelled using an equation of a straight line.

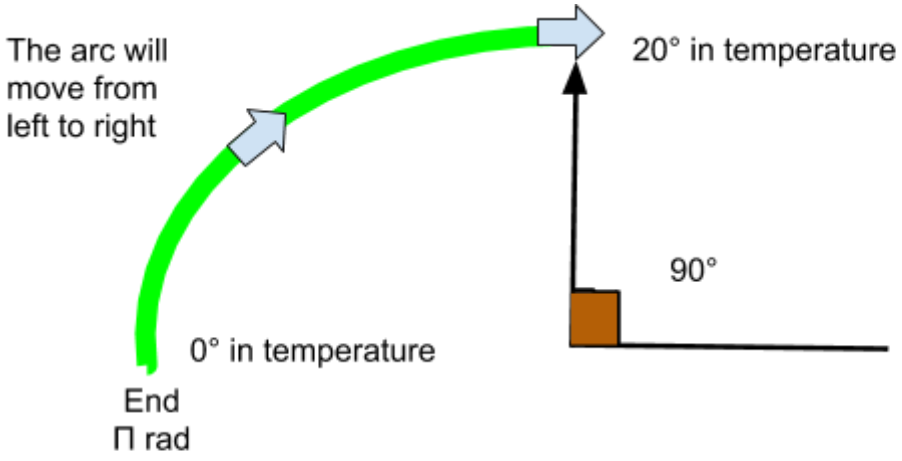
$$Y = mX + C$$



The arc will be in the left side if it is from 0° to 20° temperature

The arc will be in the right side if it is from 20° to 40° temperature

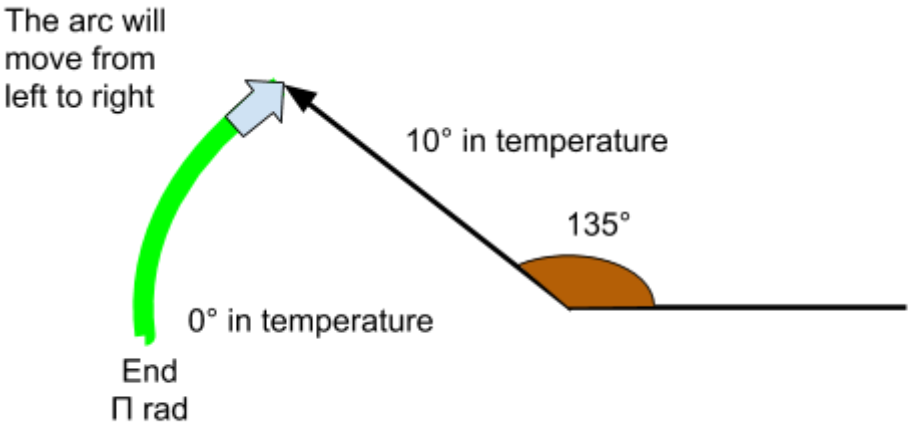
Thus, we only have to vary the Start position of the arc from 0 rad (meaning the temperature is 40 degrees celsius) to Π rad (meaning the temperature is 0 degrees celsius) to show the arc moving in various temperatures.



The arc will be in the left side if it is from 0° to 20° temperature

The arc will be in the right side if it is from 20° to 40° temperature

Here, we make the starting position to be π/2 radians or 90° instead of 0° or 0 radian, this gives the illusion of the temperature at 20° celsius.



The arc will be in the left side if it is from 0° to 20° temperature

The arc will be in the right side if it is from 20° to 40° temperature

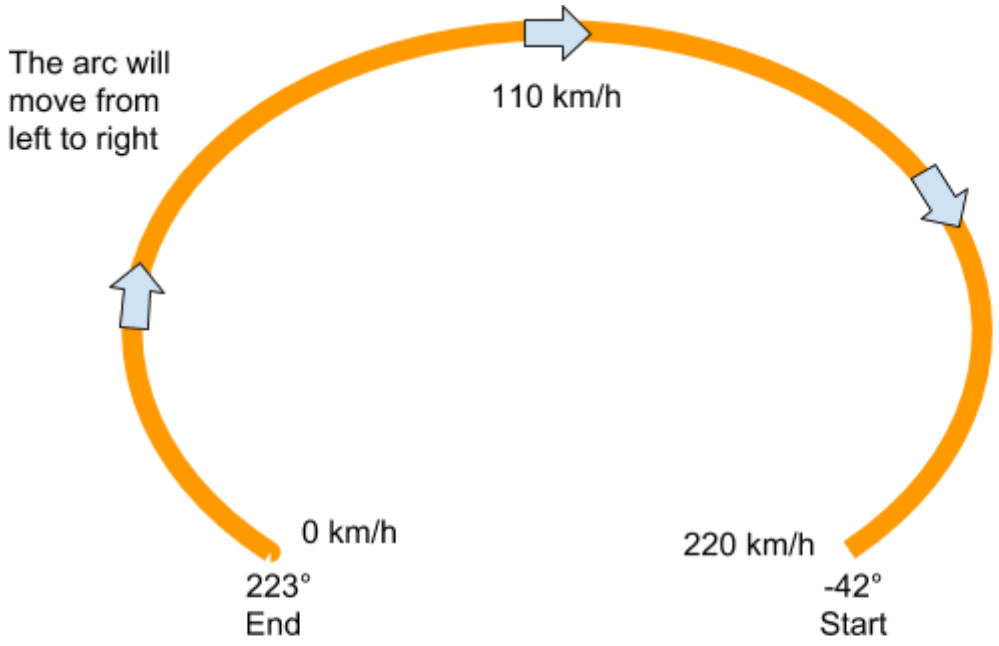
Here, we make the starting position to be 2.35619 radians or 135° instead of 0° or 0 radian, this gives the illusion of the temperature at 10° celsius.

0 degrees celsius corresponds to π radians
40 degrees celsius corresponds to 0 radian
The independent variable is the temperature and the dependent variable is the arc in radians.
(x1, y1) = (0 degrees celsius, π radians / 180°)
(x2, y2) = (40 degrees celsius, 0 radians / 0°)

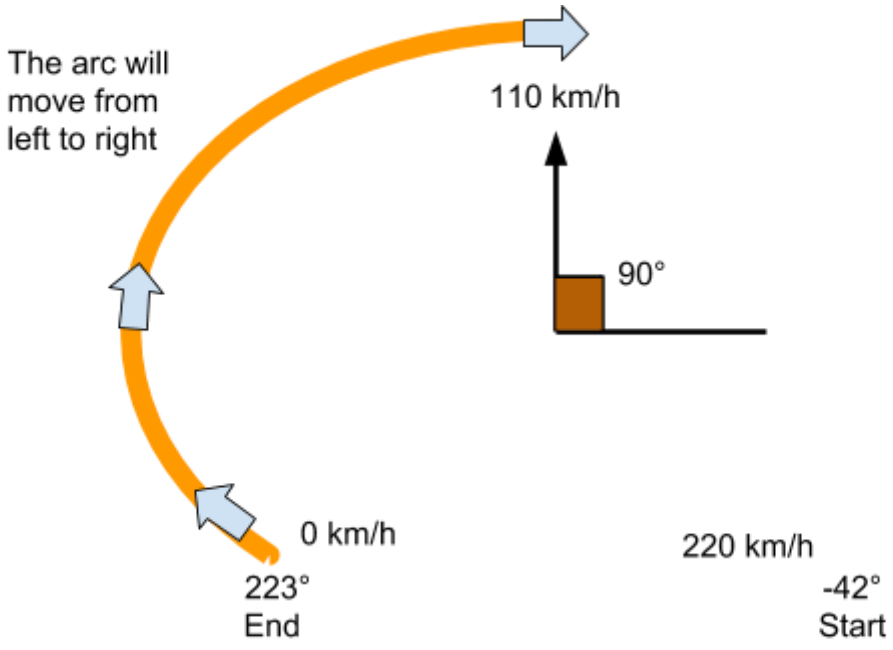
Thus, $m = \frac{y2 - y1}{x2 - x1} = \frac{0° - 180°}{40\ degrees\ celsius - 0\ degrees\ celsius} = -\frac{180}{40}$
$= -\frac{9}{2}$.

To find c , y Intercept, we substitute one of the coordinates into the equation.
$0 = -\frac{9}{2}(40) + c$. Rearranging the equation, we get the c as 180.
Thus, the straight line equation is $y = -\frac{9}{2}x + 180$

Speedometer Arc

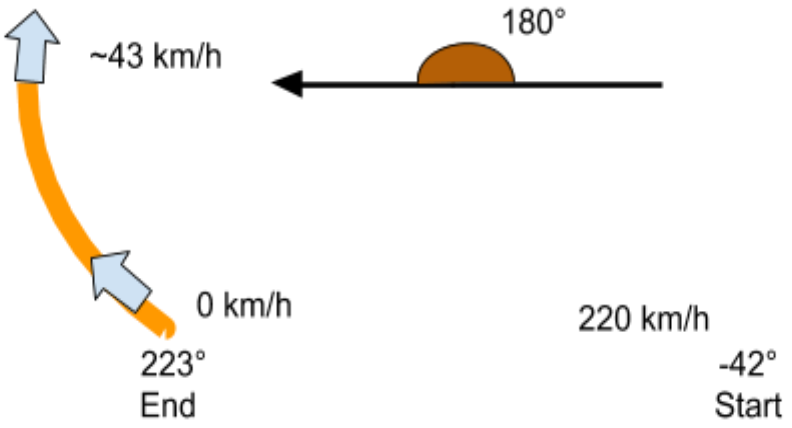


Thus, we only have to vary the Start position of the arc from -42° (meaning the speed is 220 km/h) to 223° (meaning the speed is 0 km/h) to show the arc moving in different speeds.



Here, we make the starting position to be 90° instead of -42°, this gives the illusion of the speed at 110 km/h.

The arc will move from left to right



Here, we make the starting position to be 180° instead of -42°, this gives the illusion of the speed at ~43 km/h.

To form the straight line equation, we need 2 points.
(x1, y1) = (0 km/h, 224°)
(x2, y2) = (220 km/h, -42°)

$$Y\ (start\ of\ the\ arc\ in\ degrees) = (gradient)m \times X(speed\ from\ GPS\ in\ km/h) + c$$

$$m = \frac{y2 - y1}{x2 - x1} = \frac{-42° - 224°}{220\ km/h - 0\ km/h} = -\frac{113}{110} = 1.027$$

Next, we have to find C (y - intercept),
Substituting (x1, y1), we have,

$$224 = -\frac{113}{110}(0) + c$$

∴, c = 224.

Thus, we have the following speed equation,

$$y = -\frac{113}{110}(x) + 224$$

```
if __name__ == "__main__":

    initialise = Initialise()

    battery = Battery()

    speedometer = Speedometer()
    speedometer.load_image()

    temperature = Temperature()

    text = Text()

    camera = Camera()

    updatevalues = UpdateValues()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_f:
                    initialise.resolution = (1024, 600)
                    initialise.screen = pygame.display.set_mode(
                        initialise.resolution, pygame.FULLSCREEN)
                elif event.key == pygame.K_g:
                    initialise.resolution = (1024, 600)
                    initialise.screen = pygame.display.set_mode(
                        initialise.resolution, pygame.RESIZABLE)
                elif event.key == pygame.K_c:
                    camera.use_camera()
                elif event.key == pygame.K_v:
                    camera.stop_camera()

        initialise.screen.fill(initialise.BLACK)

        speedometer.load_image()
        speedometer.draw_arc(updatevalues.speed_value)

        battery.draw_rect(updatevalues.battery_value_original)

        temperature.draw_arc(updatevalues.temperature_value)

        text.message_display(text="{} Volts".format(
            updatevalues.battery_value_original), x_position=865,
            y_position=175)
```

```
        text.message_display(text="{}".format(
            updatevalues.speed_value_original), x_position=500, y_position=200)

        text.message_display(text="{} degrees".format(
            updatevalues.temperature_value_original), x_position=150,
            y_position=200)

        updatevalues.transferValues()

        initialise.clock.tick(60)
        pygame.display.update()

pygame.quit()
quit()
```

The above code will create a new, unique instance for all the above defined classes. The while loop runs the whole program ensuring the dashboard GUI can receive values, update the values, change the battery, speed and temperature graphics, output text and pictures to the screen when appropriate and respond to user input. The pygame.quit() will terminate all pygame modules that have previously been initialised.

3.6.1 Lattepanda

All the information mentioned above applies to the Lattepanda board as well. However, there are only 2 issues we need to take care of running the GUI program. The first issue is the serial port.

```
    # raspberry pi code
    # ser = serial.Serial('COM5', 9600, 8, 'N', 1, timeout=5)

    # lattepanda code
    ser = serial.Serial('COM5')
```

We need to change the port to COM5 to communicate with the Arduino serial port.

The second issue is the camera. The picamera module does not work with the lattepanda because raspberry pi has a specific hardware dedicated for a camera and is unavailable in lattepanda. Thus, we have to use opencv to utilise the camera module.

```
class Camera(Initialise):
    def __init__(self):
        # NOTE -> tutorial link :
        # https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui
        # /py_video_display/py_video_display.html
```

```
        self.cap = cv2.VideoCapture(0)

        # set the width and height
        self.cap.set(3, 1000)
        self.cap.set(4, 250)

        # Check if camera is opened successfully
        if (self.cap.isOpened() == False):
            print("Error opening video stream or file")

    def use_camera(self):
        # Capture frame-by-frame
        ret, frame = self.cap.read()
        frame = np.rot90(frame)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = pygame.surfarray.make_surface(frame)

        # blit to the screen and set the (x,y) coordinates
        self.screen.blit(frame, (0, 350))

    def stop_preview(self):
        # When everything done, release the capture
        self.cap.release()
        cv2.destroyAllWindows()
```

The above code utilises some opencv methods such as the `cap.read()` which returns a bool (True/False). If frame is read correctly, it will be True. The main purpose is achieved through the opencv library.

Errors and their solutions :

Error 1 : OSError: [Errno 25] Inappropriate ioctl for device
Solution :

- Type ls /dev/tty*.
- Choose /dev/ttyAMA0 or some other port.
- Make sure Arduino is plugged in properly.
- Choose the correct /dev/ttyXXXX.

Error 2 : Type error:cannot create a consistent method resolution order MRO for bases.
Solution : Do not inherit classes, which will be inherited by another class

For example

```
class Player:
    pass
class Enemy(Player):
    pass
```

```
class GameObject(Player, Enemy):
    pass
g = GameObject()
```

The `GameObject()` is inheriting from Player **and** Enemy. Because Enemy *already* inherits from Player Python now cannot determine what class to look methods up on first; either Player, or on Enemy, which would override things defined in Player.

You don't need to name all base classes of Enemy here; just inherit from that one class:

```
class GameObject(Enemy):
pass
```

Enemy already includes Player, you don't need to include it again.

3.7 Research

3.7.1 Tachometer using an AT Mega microcontroller

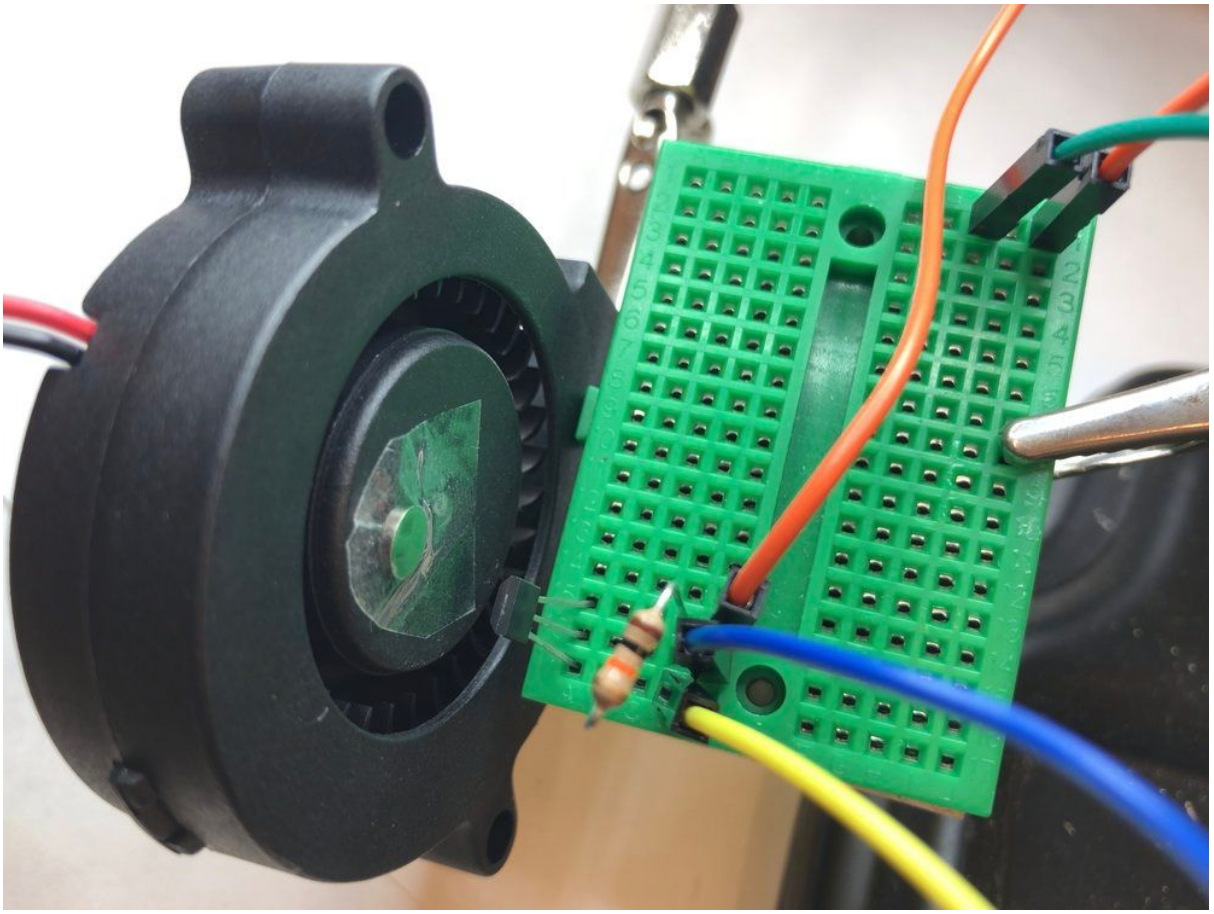


Figure 47 Tachometer setup

A tachometer is a useful tool for calculating the rotational motion of a part. Tachometers is the best choice when it comes to measuring speed of an rotating object. Tachometers read out revolutions per minute (RPM), which tells the user how often a rotating part completes one full rotation. RPM readings are used in the automotive, aerospace, and manufacturing fields.

We have researched on 2 types of sensors namely the IR sensor and the A3144 sensor.

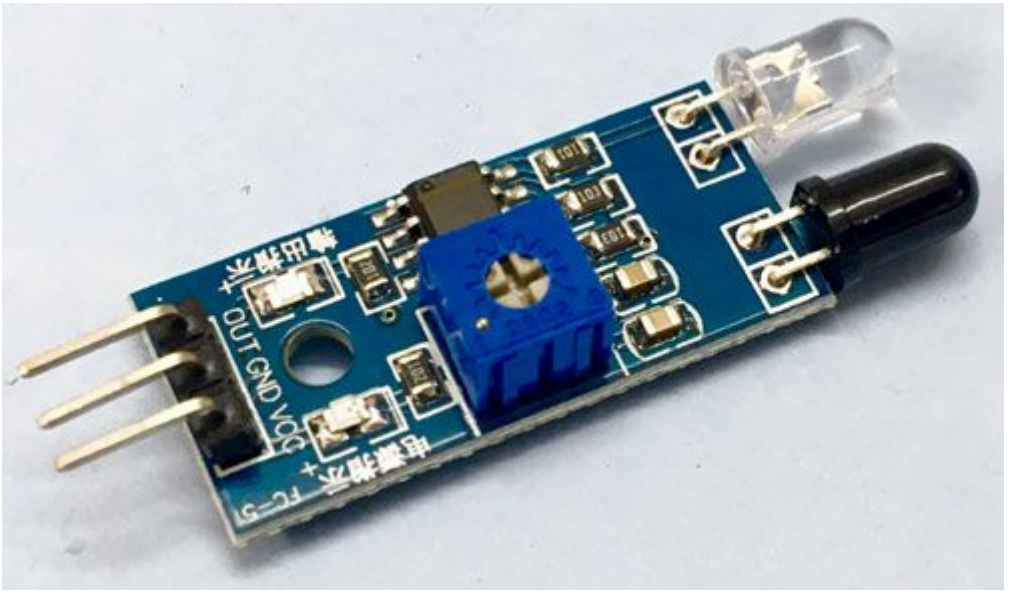


Figure 48 IR sensor

The Infrared Light Emitting Diode (LED) is a special purpose LED emitting infrared rays ranging from 700 nm to 1 mm wavelength. An IR sensor consists of two parts, the emitter circuit and the receiver circuit. It is also known as a photo-coupler or an optocoupler collectively.

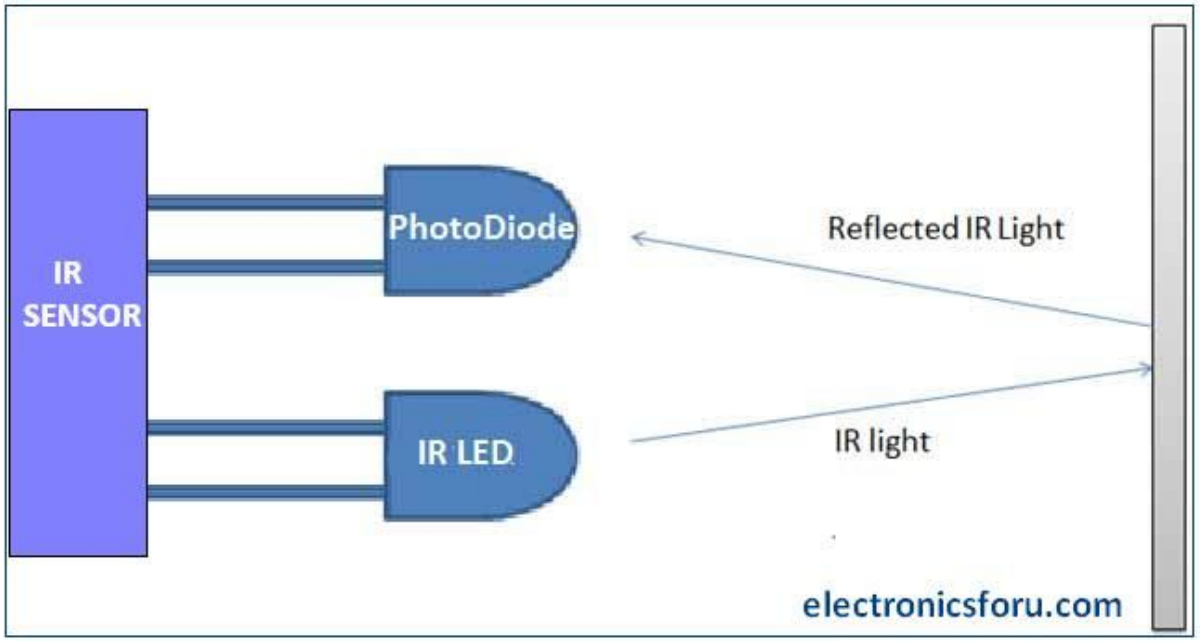


Figure 49 IR sensor working principles

IR LED is also known as an emitter and the IR photodiode is known as the detector. The IR photodiode is responsive to an infrared light emitted by the IR LED. Based on the infrared light received, the IR photodiode's resistance and output voltage will change proportionally. This is the fundamental working principle of an IR sensor.

The type of incidence can be direct incidence or indirect incidence. The IR LED is placed directly in front of a photodiode with no obstacle in between in a direct

incidence. Both diodes are placed beside with each other with an opaque object in front of the sensor in an indirect incidence. In the case of an indirect incidence, the light from the IR LED hits the opaque surface and reflects back to the photodiode. The time interval between each hits is taken into account, and is used to calculate the speed.

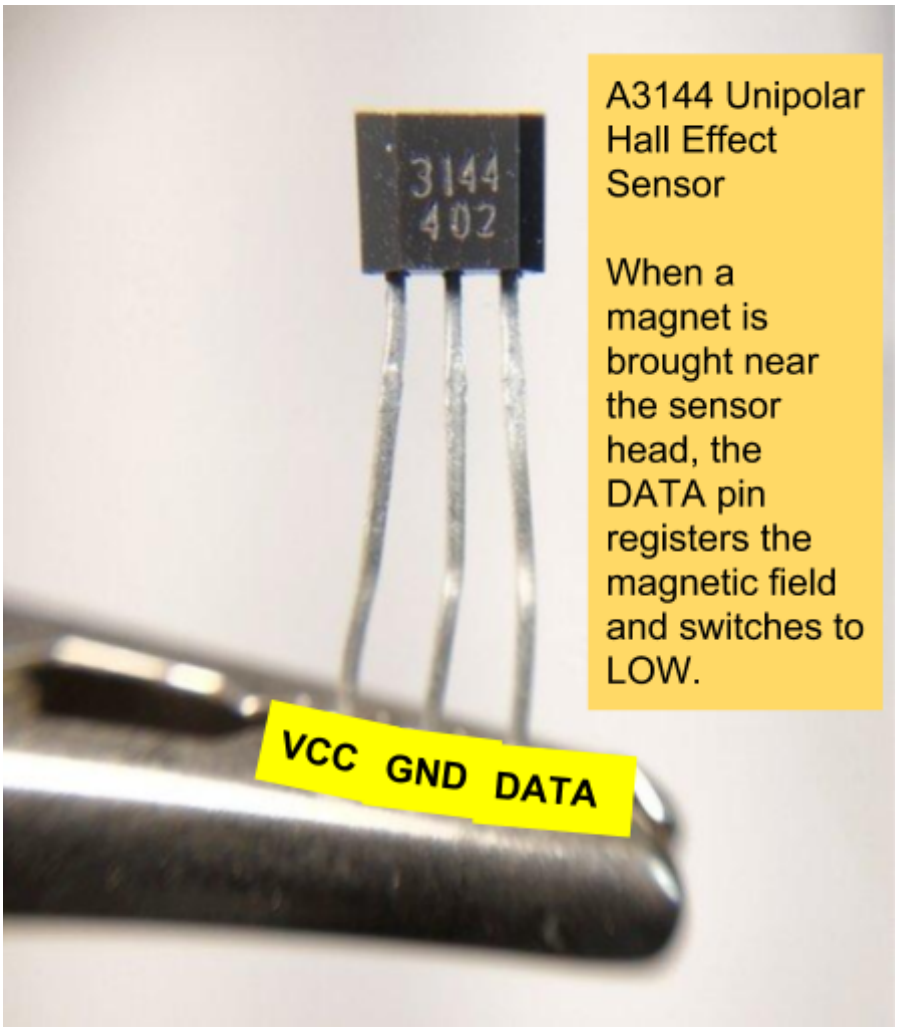


Figure 50 A3144 Hall Effect sensors

The hall effect allows electricity to flow through when a magnetic field is applied within the region of a given hall sensor. The A3144 hall effect sensor is an unipolar sensor. Only 1 pole of magnet is needed for unipolar sensors. This allows us to stick a magnet to a moving object and as it cycles through its rotation, each time it passes the hall sensor, the hall sensor registers its passing and we can say that one period has been completed.(Hrisko, 2019) The built-in hysteresis allows clean switching of the output even in the presence of external mechanical vibration and electrical noise.

The way we will be using the A3144 is by using a pullup resistor, which means that we add a resistor between VCC and DATA to keep the value of the data pin at HIGH

when a neodymium magnet is near. This is shown below:

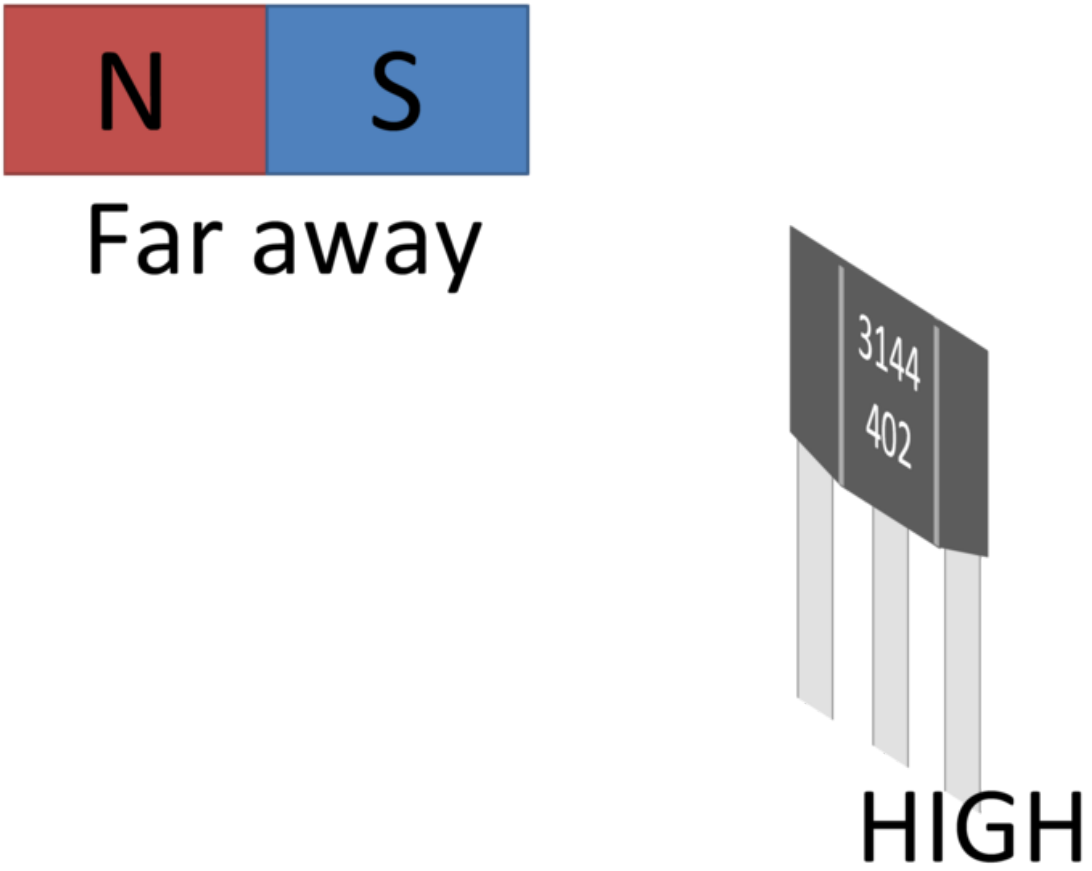


Figure 51 Data pin at HIGH when no magnet is near

Alternatively, when we place the neodymium magnet within the range of the hall sensor, we get a LOW value. This is also shown below:

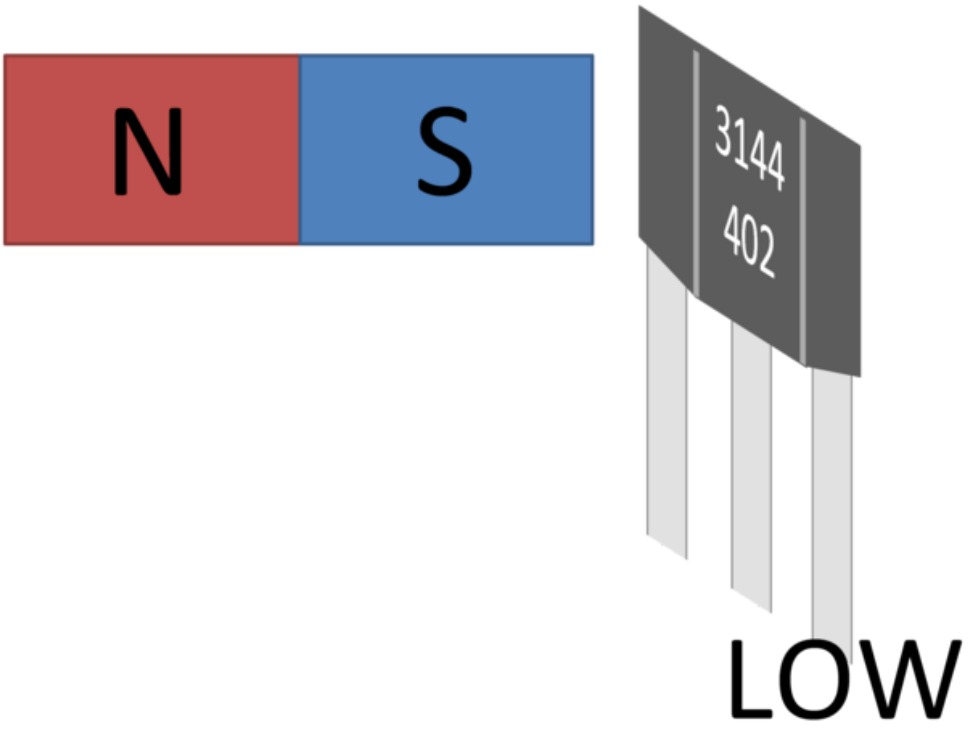


Figure 52 Data pin at LOW when a magnet is near

The cycle between LOW and HIGH shown above is what is expected when we attach a magnet to a rotating part that will bring the magnet within and out of reach of the hall sensor. The detection and the measurement of HIGH and LOW will be in a C program.

3.7.2 Computer Vision using raspberry pi

Our group also tried using computer vision to get the speed of the vehicle. The raspberry pi is only capable of getting up to ~0.9 frames per second when applying object detection with Python and OpenCV. (Rosebrock, 2019) We realised it is impossible to use computer vision to detect the speed.

3.7.3 FPGAs

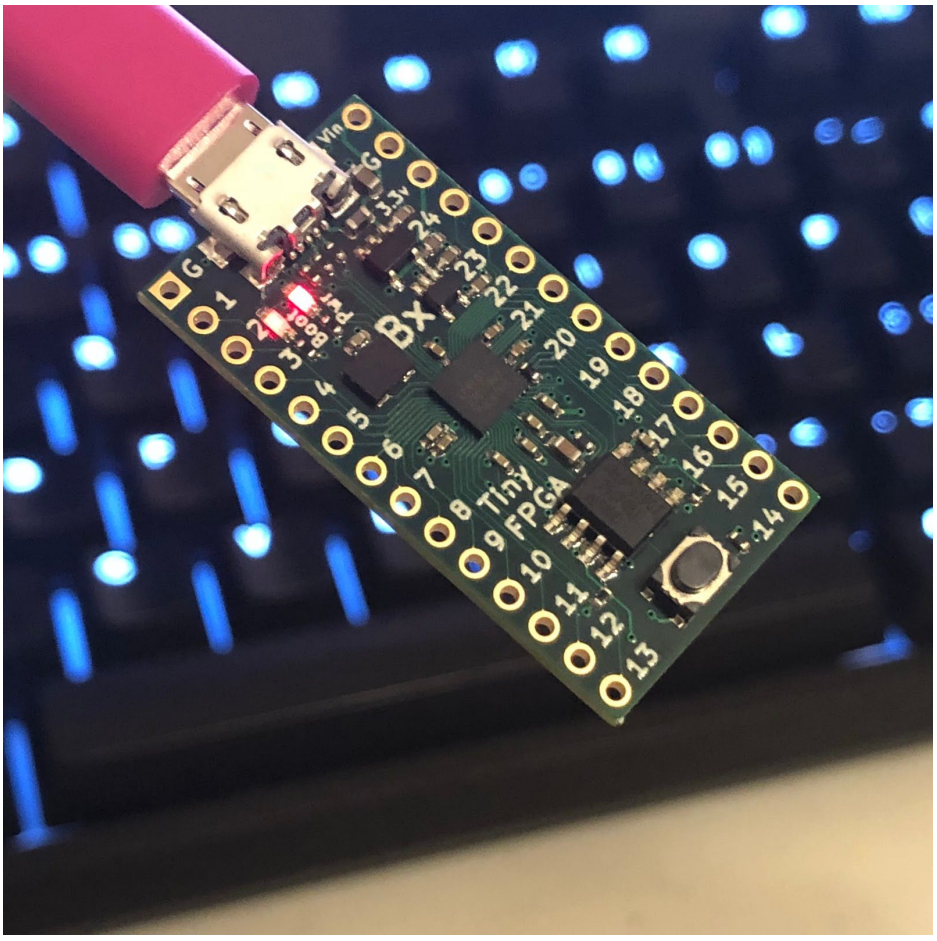


Figure 53 FPGA

FPGA (Field Programmable Gate Array) uses prebuilt logic blocks and programmable routing resources, one can configure implement any custom hardware functionality only limited by their imagination.

Advantages of FPGAs :

- ➔ Excellent performance due to parallel execution
- ➔ Customisable hardware architecture!
- ➔ Uses Verilog HDL / VHDL similar to software languages
- ➔ Able to receive and send data at higher speeds than any microcontrollers in the market

Disadvantages of FPGAs :

- ➔ One has to learn a HDL (Hardware Description Language) to program a FPGA

3.7.4 Future of SunSPEC Telemetry Systems

The future of Telemetry System lies in both NI CompactRIO and FPGAs.

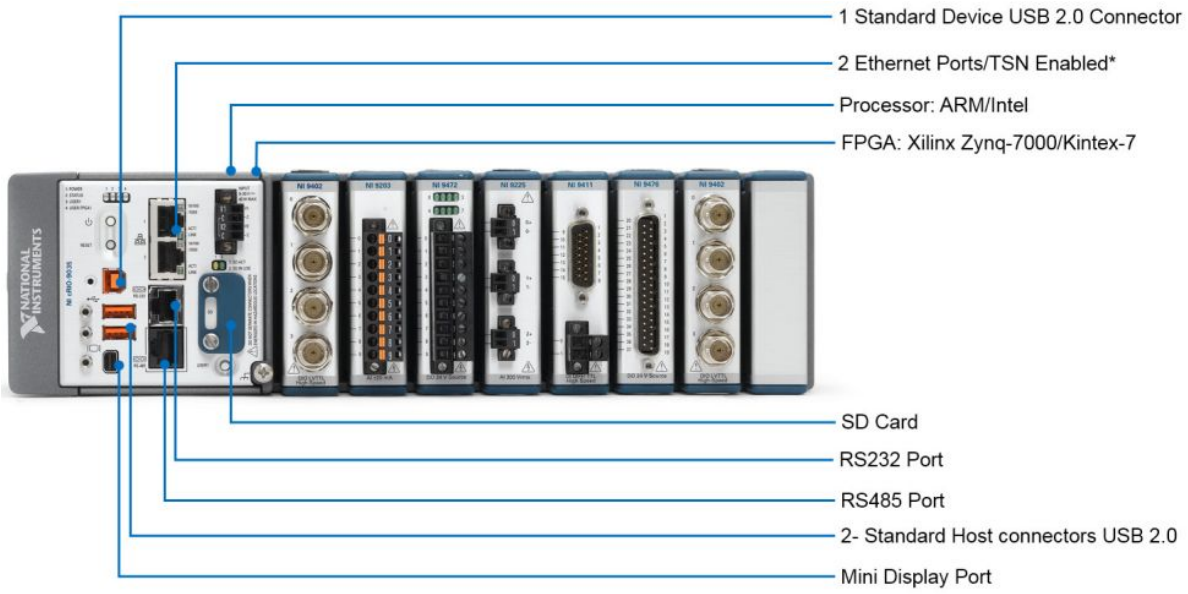


Figure 54 NI CompactRIO

The NI compactRIO consists of a controller running NI Linux Real-Time with a processor and a user-programmable FPGA with one or more I/O modules.

Features of a NI compactRIO are (Ni.com, 2019) :

- ➔ Modular, high-performance embedded controllers for industrial deployment
- ➔ NI Linux Real-Time OS for reliability, security, and determinism
- ➔ Industrial-grade processing component options from ARM, Intel, and Xilinx
- ➔ Time Sensitive Networking (TSN) enabled Ethernet ports
- ➔ Up to -40 °C to 70 °C temperature range, up to 50 g shock and 5 g vibration
- ➔ 200+ I/O modules to support a variety of sensors and signal types
- ➔ LabVIEW graphical development platform eliminates the need for hardware description language
- ➔ (HDL) expertise to use reconfigurable FPGA hardware

In conclusion, the NI is compact, rugged, uses real time operating system, has attained industry-standard certifications, modular I/O and very apt for use in the telemetry system.

Bibliography / References

ada, l. and Cooper, T. (2019). Overview | Adafruit Ultimate GPS Logger Shield | Adafruit Learning System. [online] Learn.adafruit.com. Available at: https://learn.adafruit.com/adafruit-ultimate-gps-logger-shield/overview [Accessed 5 Jan. 2019].

Crmagnetics.com. (2019). CR5310 DC Voltage Transducer - 0-5 Vdc output from CRMagnetics. [online] Available at: http://www.crmagnetics.com/dc-voltage-transducers/cr5310 [Accessed 5 Jan. 2019].

Selwood AM, C. and Pudney, P. (2019). 2019 Regulations | World Solar Challenge 2019. [online] Worldsolarchallenge.org. Available at: https://www.worldsolarchallenge.org/event-information/2019_regulations [Accessed 5 Jan. 2019].

Industries, A. (2019). SMA to uFL/u.FL/IPX/IPEX RF Adapter Cable. [online] Adafruit.com. Available at: https://www.adafruit.com/product/851 [Accessed 5 Jan. 2019].

En.wikipedia.org. (2019). Fleming's right-hand rule. [online] Available at: https://en.wikipedia.org/wiki/Fleming%27s_right-hand_rule [Accessed 13 Jan. 2019].

Store.arduino.cc. (2019). Arduino Mega 2560 Rev3. [online] Available at: https://store.arduino.cc/usa/arduino-mega-2560-rev3 [Accessed 25 Jan. 2019]. (Store.arduino.cc, 2019)

En.wikipedia.org. (2019). Microcontroller. [online] Available at: https://en.wikipedia.org/wiki/Microcontroller [Accessed 25 Jan. 2019]. (En.wikipedia.org, 2019)

Digikey.com. (2019). *CR5200 - CR Magnetics Inc. - Current | Online Catalog | DigiKey Electronics*. [online] Available at: https://www.digikey.com/catalog/en/partgroup/cr5200/22409 [Accessed 2 Feb. 2019].

Freertos.org. (2019). FreeRTOS - Open Source RTOS Kernel for small embedded systems - What is FreeRTOS FAQ?. [online] Available at: https://www.freertos.org/FAQWhat.html#WhyUseRTOS [Accessed 5 Feb. 2019]. (Freertos.org, 2019)

Burleson, D. (2019). *Advantages and Disadvantages of Object-Oriented Approach*. [online] Dba-oracle.com. Available at:

http://www.dba-oracle.com/t_object_oriented_approach.htm [Accessed 5 Feb. 2019].
(Burleson, 2019)

Docs.python.org. (2019). Built-in Types — Python 3.7.2 documentation. [online] Available at: https://docs.python.org/3/library/stdtypes.html#lists [Accessed 5 Feb. 2019].
(Docs.python.org, 2019)

Yveaux.blogspot.com. (2019). nRF24L01+ sniffer - part 1. [online] Available at: http://yveaux.blogspot.com/2014/07/nrf24l01-sniffer-part-1.html [Accessed 6 Feb. 2019].(Yveaux.blogspot.com, 2019)

GitHub. (2019). TMRh20/TMRpcm. [online] Available at: https://github.com/TMRh20/TMRpcm [Accessed 6 Feb. 2019]. (GitHub, 2019)

Scott, G. (2019). LoRa Module VS nRF24 VS Generic RF Module || Range & Power Test. [online] YouTube. Available at: https://www.youtube.com/watch?v=nP6YuwNVoPU [Accessed 6 Feb. 2019]. (Scott, 2019)

Hrisko, J. (2019). Arduino Tachometer - Using a Hall Effect Sensor (A3144) to Measure Rotations from a Fan. [online] Engineer's Portal. Available at: https://engineersportal.com/blog/2018/10/3/arduino-tachometer-using-a-hall-effect-sensor-to-measure-rotations-from-a-fan [Accessed 6 Feb. 2019]. (Hrisko, 2019)

Rosebrock, A. (2019). Raspberry Pi: Deep learning object detection with OpenCV - PyImageSearch. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/ [Accessed 6 Feb. 2019]. (Rosebrock, 2019)

Ni.com. (2019). CompactRIO Controllers. [online] Available at: http://www.ni.com/pdf/product-flyers/compactrio-controller.pdf [Accessed 6 Feb. 2019]. (Ni.com, 2019)

Appendix (Source Code and technical drawings)

Raspberry Pi C code (Arduino)

```
/*
 * Written and debugged by : Shankar 2018, 2019
  Aim : To gather data such as temperature, speed, battery voltage and current.
      : Send data from solar car to the chase vehicle

      Changes from draft2:
      + Used hardware serial parsing instead of software serial parsing (Recommended in GPS datasheet)
```

```
      + Supports only Arduino Mega and Leonardo because other
Arduino development boards cannot use interrupts.
*/

// ==============================DHT Sensor==================
#include "DHT.h"
#define DHTPIN 31     // what digital pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due
connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of
the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third
parameter to
// tweak the timings for faster processors.  This parameter is no
longer needed
// as the current DHT reading algorithm adjusts itself to work on
faster procs.
DHT dht(DHTPIN, DHTTYPE);
//
================================================================
==============



//
================================GPS-Speed=======================
===========
#include <Adafruit_GPS.h>

// what's the name of the hardware serial port?
#define GPSSerial Serial1

// Connect to the GPS on the hardware port
Adafruit_GPS GPS(&GPSSerial);
```

```
// Set GPSECHO to 'false' to turn off echoing the GPS data to the
Serial console
// Set to 'true' if you want to debug and listen to the raw GPS
sentences
#define GPSECHO false

uint32_t timer = millis();
//
============================================================================
===========



//
=================================Battery=========================
=========
# define battery_pin A8
//
=================================================================
=========

//
=================================Current=========================
=========
# define current_pin_1 A9
# define current_pin_2 A10
# define current_pin_3 A11
# define current_pin_4 A12
# define current_pin_5 A13

// constants won't change. Used here to set a pin number:
const int current_power_supply_pin_1 =  24; // the number of the
current_power_supply_pin_1 pin
const int current_power_supply_pin_2 =  26; // the number of the
current_power_supply_pin_2 pin
const int current_power_supply_pin_3 =  28; // the number of the
current_power_supply_pin_3 pin
const int current_power_supply_pin_4 =  30; // the number of the
current_power_supply_pin_4 pin
const int current_power_supply_pin_5 =  32; // the number of the
current_power_supply_pin_5 pin
```

```
//
=================================================================
=========

// lambda declarations
float collect_temperature();
float collect_speed();
float collect_battery();

float collect_current_sensor_1();
float collect_current_sensor_2();
float collect_current_sensor_3();
float collect_current_sensor_4();
float collect_current_sensor_5();

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

  // ============================DHT
22=============================================
  dht.begin();
  // ============================End of DHT
22=======================================


  // ============================Ultimate GPS
shield=================================

  //while (!Serial);  // uncomment to have the sketch wait until
Serial is ready

  // connect at 115200 so we can read the GPS fast enough and echo
without dropping chars
  // also spit it out
  // Serial.begin(9600);
  // Serial.println("Adafruit GPS library basic test!");

  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's-
some use 4800
  GPS.begin(9600);
  // uncomment this line to turn on RMC (recommended minimum) and
```

```
GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended"
data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either
RMC only or RMC+GGA since
  // the parser doesn't care about other sentences at this time
  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort
thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  delay(1000);

  // Ask for firmware version
  // GPSSerial.println(PMTK_Q_RELEASE);

  // ========================End of Ultimate GPS
shield=================================

  //
===================Battery=================================================
================

  // ===================End of
Battery=============================================

  //
===================Current=================================================
================
  // set the digital pin as output:
  pinMode(current_power_supply_pin_1, OUTPUT);
  pinMode(current_power_supply_pin_2, OUTPUT);
  pinMode(current_power_supply_pin_3, OUTPUT);
  pinMode(current_power_supply_pin_4, OUTPUT);
  pinMode(current_power_supply_pin_5, OUTPUT);
```

```
  // ===================End of
Current============================================================

}

void loop() {

  digitalWrite(current_power_supply_pin_1, HIGH);
  digitalWrite(current_power_supply_pin_2, HIGH);
  digitalWrite(current_power_supply_pin_3, HIGH);
  digitalWrite(current_power_supply_pin_4, HIGH);
  digitalWrite(current_power_supply_pin_5, HIGH);

  //  testing purposes
  //  float temperature_value = 25;
  //  float speedometer_value = 100;
  //  float battery_value = 75;
  //  float current_value_1 = 0.10;
  //  float current_value_2 = 0.20;
  //  float current_value_3 = 0.30;
  //  float current_value_4 = 0.40;
  //  float current_value_5 = 0.50;

  float temperature_value = collect_temperature();
  float speedometer_value = collect_speed();
  float battery_value = collect_battery();

  float current_value_1 = collect_current_sensor_1();
  float current_value_2 = collect_current_sensor_2();
  float current_value_3 = collect_current_sensor_3();
  float current_value_4 = collect_current_sensor_4();
  float current_value_5 = collect_current_sensor_5();

  Serial.print("DATA");
  Serial.print(",");
  Serial.print("TIME");
  Serial.print(",");
  Serial.print(temperature_value);
  Serial.print(",");
  Serial.print(speedometer_value);
  Serial.print(",");
```

```
  Serial.print(battery_value);
  Serial.print(",");
  Serial.print(current_value_1);
  Serial.print(",");
  Serial.print(current_value_2);
  Serial.print(",");
  Serial.print(current_value_3);
  Serial.print(",");
  Serial.print(current_value_4);
  Serial.print(",");
  Serial.println(current_value_5);

}

float collect_temperature() {
  // Wait a few seconds between measurements.
  delay(250);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very
slow sensor)
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  // float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  return (t);
}

float collect_speed() {

  // read data from the GPS in the 'main loop'
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
  if (GPSECHO)
    if (c) Serial.print(c);
```

```
  // if a sentence is received, we can check the checksum, parse
it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or
data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print
out data
    // Serial.println(GPS.lastNMEA()); // this also sets the
newNMEAreceived() flag to false
    if (!GPS.parse(GPS.lastNMEA())) // this also sets the
newNMEAreceived() flag to false
      return; // we can fail to parse a sentence in which case we
should just wait for another
  }
  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis()) timer = millis();

  // approximately every 2 seconds or so, print out the current
stats
  if (millis() - timer > 2000) {
    timer = millis(); // reset the timer

    // Serial.print("\nTime: ");
    // Serial.print(GPS.hour, DEC); Serial.print(':');
    // Serial.print(GPS.minute, DEC); Serial.print(':');
    // Serial.print(GPS.seconds, DEC); Serial.print('.');
    // Serial.println(GPS.milliseconds);
    // Serial.print("Date: ");
    // Serial.print(GPS.day, DEC); Serial.print('/');
    // Serial.print(GPS.month, DEC); Serial.print("/20");
    // Serial.println(GPS.year, DEC);
    // Serial.print("Fix: "); Serial.print((int)GPS.fix);
    // Serial.print(" quality: ");
Serial.println((int)GPS.fixquality);
    if (GPS.fix) {
      // Serial.print("Location: ");
      // Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
      // Serial.print(", ");
      // Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);
      // Serial.print("Speed (knots): ");
```

```
      return (GPS.speed * 1.852);
      // Serial.print("Angle: "); Serial.println(GPS.angle);
      // Serial.print("Altitude: "); Serial.println(GPS.altitude);
      // Serial.print("Satellites: ");
Serial.println((int)GPS.satellites);
    }
  }
}

float collect_battery() {
  // read the input on analog pin 0:
  //  float batteryValue = analogRead(batteryPin); // Read OUTPUT
= 0 V to 5 V.
  //  float value = (batteryValue / 1023) * 5; //
  //  float realValue = value * 30;

  return (((analogRead(battery_pin) / 1023) * 5) * 30);
}

float collect_current_sensor_1() {
  delay(5);
  float output_voltage_value_1 = analogRead(current_pin_1) * (5.0
/ 1023.0);
  float current_value_1 = (output_voltage_value_1 - (5.0 / 2.0)) /
0.037;
  return current_value_1;
}

float collect_current_sensor_2() {

}

float collect_current_sensor_3() {

}

float collect_current_sensor_4() {

}

float collect_current_sensor_5() {
```

```
}
```

Raspberry Pi Python code

```
#
# # Written and debugged by Shankar
# Date: 12 September 2018
# Purpose: This code is a dashboard for a solar vehicle. Refer to README
# All rights reserved.

try:
   import pygame
   import math
   import random
   import picamera
   from pygame.locals import *
   import serial
   import time
except ImportError as ImpErr:
   print("Check your imports !")
   print(str(ImpErr))


class Initialise(object):
   # Colour definitions (Red ,Green, Blue)
   # All the below code are class variables
   BLACK = (0, 0, 0)
   WHITE = (255, 255, 255)
   RED = (255, 0, 0)
   GREEN = (0, 255, 0)
   BLUE = (0, 0, 255)
   YELLOW = (255, 255, 0)
   CYAN = (0, 255, 255)
   BROWN = (83, 91, 36)
   SILVER = (192, 192, 192)

   pygame.init()
   pygame.display.set_caption("Dashboard")
   clock = pygame.time.Clock()
   resolution = (1024, 600)
   screen = pygame.display.set_mode(resolution, pygame.RESIZABLE)
```

```
    ser = serial.Serial('/dev/ttyACM0', 9600, 8, 'N', 1, timeout=5)


class Text(Initialise):
    def message_display(self, text, x_position, y_position):
        largeText = pygame.font.Font("freesansbold.ttf", 20)
        # The text is inside a rectangle and can be referenced by a rectangle.
        textSurface = largeText.render(text, True, Initialise.CYAN)

        # TextSurface, TextRect = text_objects(text, largeText)
        TextRect = textSurface.get_rect()
        TextRect.center = (x_position, y_position)
        Initialise.screen.blit(textSurface, TextRect)


class Battery(Text):

    def draw_rect(self, battery_value):
        pygame.draw.rect(Initialise.screen, Battery.SILVER, (956, 65, 15, 30))
        pygame.draw.rect(Initialise.screen, Battery.WHITE,
                         (800, 25, 158, 120), 3)

        # pygame.draw.rect(screen, color, (x,y,width,height), thickness)
        # pygame.draw.rect(self.screen, Battery.GREEN, (802, 27, 145, 97))
        if (battery_value > 70 and battery_value <= 140):
            pygame.draw.rect(Initialise.screen, Initialise.GREEN,
                             (802, 27, battery_value, 117))
        elif ((battery_value > 42) and (battery_value <= 70)):
            pygame.draw.rect(Initialise.screen, Battery.YELLOW,
                             (802, 27, battery_value, 117))
        elif (battery_value >= 0 and battery_value <= 42):
            pygame.draw.rect(Initialise.screen, Initialise.RED,
                             (802, 27, battery_value, 117))
            self.bolt_image = pygame.image.load("bolted.png")
            self.bolt_image = pygame.transform.scale(self.bolt_image,
                                                     (100, 110))
            Initialise.screen.blit(self.bolt_image, (830, 33))
            Text.message_display(self, text="LOW Battery !!!", x_position=865,
                                 y_position=200)
```

```
class Speedometer(Initialise):

    def load_image(self):
        self.speedometer_image = pygame.image.load("speed.png")
        self.speedometer_image = pygame.transform.scale(self.speedometer_image,
                                                        (700, 450))
        Initialise.screen.blit(self.speedometer_image, (155, -40))

    def draw_arc(self, speed_value):
        # pygame.draw.arc()
        # Good example arc below ->
        # pygame.draw.arc(self.screen, Speedometer.YELLOW,
        # (235, 75, 525, 525), math.radians(-42), math.radians(223), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
        """ Draws an elliptical arc on the Surface.
        The rect argument is the area that the ellipse will fill.
        The two angle arguments are the initial and final
        angle in radians, with the zero on the right. The width argument is the
        thickness to draw the outer edge.

        TAKE NOTE: <Worth mentioning>
        the initial angle must be less than the final angle;
        otherwise it will draw the full ellipse. """
        pygame.draw.arc(Initialise.screen, Speedometer.YELLOW,
                        (277, 5, 450, 400), math.radians(speed_value),
                        math.radians(224), 5)


class Temperature(Text):

    def draw_arc(self, temperature_value):

        # Good example arc below ->
        # pygame.draw.arc(Initialise.screen, Initialise.YELLOW, (50, 75,
        # 200, 200), math.radians(0), math.radians(180), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
        """ Draws an elliptical arc on the Surface.
```

```
        The rect argument is the area that the ellipse will fill.
        The two angle arguments are the initial and final angle in radians,
        with the zero on the right.
        The width argument is the thickness to draw the outer edge.

        TAKE NOTE: <Worth mentioning> the initial angle must be less
        than the final angle; otherwise it will draw the full ellipse. """
        if (temperature_value >= 0 and temperature_value < 45):
            pygame.draw.arc(Initialise.screen, Initialise.RED,
                            (10, 75, 250, 250),
                            math.radians(temperature_value), math.radians(180),
                            25)
            Text.message_display(self, text="HIGH Temperature !!!",
                                 x_position=150, y_position=235)

            self.high_temperature_image = pygame.image.load("temperature.jpg")
            self.high_temperature_image = pygame.transform.scale(
                self.high_temperature_image, (50, 50))
            Initialise.screen.blit(self.high_temperature_image, (115, 125))

        elif (temperature_value >= 45 and temperature_value < 135):
            pygame.draw.arc(Initialise.screen, Initialise.YELLOW,
                            (10, 75, 250, 250),
                            math.radians(temperature_value), math.radians(180),
                            25)
        elif (temperature_value >= 135):
            pygame.draw.arc(Initialise.screen, Initialise.GREEN,
                            (10, 75, 250, 250),
                            math.radians(temperature_value), math.radians(180),
                            25)
            Text.message_display(self, text="LOW Temperature !!!",
                                 x_position=150, y_position=250)


class Camera(Initialise):
    # The below are declared as class variables
    camera_position = (0, 300, 1024, 300)
    camera_block = pygame.Surface((1024, 300))

    def __init__(self):
        self.camera = picamera.PiCamera()
```

```
        self.camera.preview_fullscreen = False
        self.camera.preview_window = (0, 350, 1024, 300)
        self.camera.resolution = (1024, 300)

    def use_camera(self):
        # If your preview is upside-down, you can rotate it with the following
        # code
        # self.camera.rotation = 180
        self.camera.start_preview()
        Initialise.screen.blit(
            Camera.camera_block,
            (Camera.camera_position[0], Camera.camera_position[1]))

    def stop_camera(self):
        self.camera.stop_preview()


class UpdateValues(Initialise):
    def __init__(self):
        self.temperature_value = 4.5
        self.temperature_value_original = 39

        self.speed_value = 30.11
        self.speed_value_original = 25

        self.battery_value = 135
        self.battery_value_original = 75

    def transferValues(self):
        if (Initialise.ser.in_waiting):
            try:
                values = Initialise.ser.readline()
                print('values', values)
                decoded = values.decode("utf-8")
                print('Decoded', decoded)
                decoded = str(decoded)
                split_decoded = decoded.split(",")

                self.data = split_decoded[0].strip()
                self.time = split_decoded[1].strip()
                self.temperature_value_original = float(
```

```
                split_decoded[2].strip())
            self.speed_value_original = float(split_decoded[3].strip())
            self.battery_value_original = float(split_decoded[4].strip())
            self.current_1 = float(split_decoded[5].strip())
            self.current_2 = float(split_decoded[6].strip())
            self.current_3 = float(split_decoded[7].strip())
            self.current_4 = float(split_decoded[8].strip())
            self.current_5 = float(split_decoded[9].strip())

            #   data = split_decoded[0].strip()
            #   time = split_decoded[1].strip()
            #   temperature = float(split_decoded[2].strip())
            #   speed = float(split_decoded[3].strip())
            #   battery = float(split_decoded[4].strip())
            #   current = float(split_decoded[5].strip())

            self.temperature_value = (-9 / 2 *
                                      self.temperature_value_original) + 180
            self.speed_value = (-133 / 110 *
                                self.speed_value_original) + 224
            self.battery_value = self.battery_value_original

            print("Temperature = {}, speed = {}, battery = {}".format(
                self.temperature_value_original, self.speed_value_original,
                self.battery_value_original))
        except:
            print(
                "I cannot convert string (Failed to read from DHT sensor) "
                "to float")


if __name__ == "__main__":

    initialise = Initialise()

    battery = Battery()

    speedometer = Speedometer()
    speedometer.load_image()

    temperature = Temperature()
```

```
text = Text()

camera = Camera()

updatevalues = UpdateValues()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_f:
                initialise.resolution = (1024, 600)
                initialise.screen = pygame.display.set_mode(
                    initialise.resolution, pygame.FULLSCREEN)
            elif event.key == pygame.K_g:
                initialise.resolution = (1024, 600)
                initialise.screen = pygame.display.set_mode(
                    initialise.resolution, pygame.RESIZABLE)
            elif event.key == pygame.K_c:
                camera.use_camera()
            elif event.key == pygame.K_v:
                camera.stop_camera()

    initialise.screen.fill(initialise.BLACK)

    speedometer.load_image()
    speedometer.draw_arc(updatevalues.speed_value)

    battery.draw_rect(updatevalues.battery_value_original)

    temperature.draw_arc(updatevalues.temperature_value)

    text.message_display(text="{} Volts".format(
        updatevalues.battery_value_original), x_position=865,
        y_position=175)

    text.message_display(text="{}".format(
        updatevalues.speed_value_original), x_position=500, y_position=200)
```

```
        text.message_display(text="{} degrees".format(
            updatevalues.temperature_value_original), x_position=150,
            y_position=200)

        updatevalues.transferValues()

        initialise.clock.tick(60)
        pygame.display.update()

pygame.quit()
quit()
```

Lattepanda C code (Arduino)

```
/*  Written and debugged by Shankar

   Aim : This code reads in data from various sensors and passes them to a
graphical user interface
   GUI and passess them to LoRa.
   All rights reserved. Copyright 2018
*/

// ====================================DHT
Sensor=====================================
#include "DHT.h"
#define DHTPIN 2     // what digital pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321

// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor

// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors.  This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);
```

```
//
=============================================================
=============================

// ========================Aadfruit
GPS=============================================
// Test code for Adafruit GPS modules using MTK3329/MTK3339 driver
//
// This code shows how to listen to the GPS module in an interrupt
// which allows the program to have more 'freedom' - just parse
// when a new NMEA sentence is available! Then access data when
// desired.
//
// Tested and works great with the Adafruit Ultimate GPS module
// using MTK33x9 chipset
//    ------> http://www.adafruit.com/products/746
// Pick one up today at the Adafruit electronics shop
// and help support open source hardware & software! -ada

#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

// If you're using a GPS module:
// Connect the GPS Power pin to 5V
// Connect the GPS Ground pin to ground
// If using software serial (sketch example default):
//   Connect the GPS TX (transmit) pin to Digital 3
//   Connect the GPS RX (receive) pin to Digital 2
// If using hardware serial (e.g. Arduino Mega):
//   Connect the GPS TX (transmit) pin to Arduino RX1, RX2 or RX3
//   Connect the GPS RX (receive) pin to matching TX1, TX2 or TX3

// If you're using the Adafruit GPS shield, change
// SoftwareSerial mySerial(3, 2); -> SoftwareSerial mySerial(8, 7);
// and make sure the switch is set to SoftSerial

// If using software serial, keep this line enabled
// (you can change the pin numbers to match your wiring):
SoftwareSerial mySerial(8, 7);

// If using hardware serial (e.g. Arduino Mega), comment out the
```

```
// above SoftwareSerial line, and enable this line instead
// (you can change the Serial number to match your wiring):

//HardwareSerial mySerial = Serial1;

Adafruit_GPS GPS(&mySerial);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences.
#define GPSECHO  false

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy

// ===================================End Of Adafruit
GPS=======================================


//
=====================================Battery============
=======================================
# define battery_pin A0
//
================================================================
=========================================

//
=====================================Current============
=======================================
# define current_pin_1 A1
# define current_pin_2 A2
# define current_pin_3 A3
# define current_pin_4 A4
# define current_pin_5 A5
//
================================================================
=========================================
```

```
// Global Variable declarations
float temperature_value;
float speedometer_value;
float battery_value;
float current_value_1;
float current_value_2;
float current_value_3;
float current_value_4;
float current_value_5;


// Function prototypes
float collect_temperature();
float collect_speed();
float collect_battery();

float collect_current_sensor_1();
float collect_current_sensor_2();
float collect_current_sensor_3();
float collect_current_sensor_4();
float collect_current_sensor_5();


void setup() {

  Serial.begin(9600);
  Serial1.begin(9600);

  //============================DHT temperature
sensor=====================================
  dht.begin();
  //============================End of DHT temperature
sensor================================

  // ============================Adafruit GPS setup()
=======================================
  // connect at 115200 so we can read the GPS fast enough and echo without
dropping chars
  // also spit it out
  //  Serial.begin(115200);
  //  Serial.println("Adafruit GPS library basic test!");
```

```
  // 9600 NMEA is the default baud rate for Adafruit MTK GPS's- some use 4800
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For parsing data, we don't suggest using anything but either RMC only or
RMC+GGA since
  // the parser doesn't care about other sentences at this time

  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
  // For the parsing code to work nicely and have time to sort thru the data, and
  // print it out we don't suggest using anything higher than 1 Hz

  // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  // the nice thing about this code is you can have a timer0 interrupt go off
  // every 1 millisecond, and read data from the GPS for you. that makes the
  // loop code a heck of a lot easier!
  useInterrupt(true);

  delay(1000);
  // Ask for firmware version
  //  mySerial.println(PMTK_Q_RELEASE);

  //============================End of Adafruit GPS setup()
=====================================

  //============================Battery setup()
================================================
  //============================End of Battery setup()
=========================================

  //============================Current setup()
================================================
  //============================End of Current setup()
```

```
//================================================
}

// IMPORTANT CODE for Adafruit GPS !

// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect) {
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
#ifdef UDR0
  if (GPSECHO)
    if (c) UDR0 = c;
  // writing direct to UDR0 is much much faster than Serial.print
  // but only one character can be written at a time.
#endif
}

void useInterrupt(boolean v) {
  if (v) {
    // Timer0 is already used for millis() - we'll just interrupt somewhere
    // in the middle and call the "Compare A" function above
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  } else {
    // do not call the interrupt function COMPA anymore
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}

uint32_t timer = millis();


void loop() {

  delay(500);

  temperature_value = collect_temperature();
  speedometer_value = collect_speed();
  battery_value = collect_battery();
```

```
//  current_value_1 = collect_current_sensor_1();
//  current_value_2 = collect_current_sensor_2();
//  current_value_3 = collect_current_sensor_3();
//  current_value_4 = collect_current_sensor_4();
//  current_value_5 = collect_current_sensor_5();

// The below code is only for Testing purposes
// temperature_value = 25;
// speedometer_value = 100;
// battery_value = 75;
current_value_1 = 0.10;
current_value_2 = 0.20;
current_value_3 = 0.30;
current_value_4 = 0.40;
current_value_5 = 0.50;


Serial.print("DATA");
Serial.print(",");
Serial.print("TIME");
Serial.print(",");
Serial.print(temperature_value);
Serial.print(",");
Serial.print(speedometer_value);
Serial.print(",");
Serial.print(battery_value);
Serial.print(",");
Serial.print(current_value_1);
Serial.print(",");
Serial.print(current_value_2);
Serial.print(",");
Serial.print(current_value_3);
Serial.print(",");
Serial.print(current_value_4);
Serial.print(",");
Serial.println(current_value_5);

// Send to LoRa
Serial1.print("DATA");
Serial1.print(",");
```

```
  Serial1.print("TIME");
  Serial1.print(",");
  Serial1.print(temperature_value);
  Serial1.print(",");
  Serial1.print(speedometer_value);
  Serial1.print(",");
  Serial1.print(battery_value);
  Serial1.print(",");
  Serial1.print(current_value_1);
  Serial1.print(",");
  Serial1.print(current_value_2);
  Serial1.print(",");
  Serial1.print(current_value_3);
  Serial1.print(",");
  Serial1.print(current_value_4);
  Serial1.print(",");
  Serial1.println(current_value_5);
}


float collect_temperature() {
  // Wait a few seconds between measurements.
  delay(250);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  // float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  return (t);
}

float collect_speed() {
  // in case you are not using the interrupt above, you'll
```

```
  // need to 'hand query' the GPS, not suggested :(
  if (! usingInterrupt) {
    // read data from the GPS in the 'main loop'
    char c = GPS.read();
    // if you want to debug, this is a good time to do it!
    if (GPSECHO)
      if (c) Serial.print(c);
  }

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
    // a tricky thing here is if we print the NMEA sentence, or data
    // we end up not listening and catching other sentences!
    // so be very wary if using OUTPUT_ALLDATA and trytng to print out data
    //Serial.println(GPS.lastNMEA());   // this also sets the newNMEAreceived() flag
to false

    if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived() flag
to false
      return;  // we can fail to parse a sentence in which case we should just wait
for another
  }

  // if millis() or timer wraps around, we'll just reset it
  if (timer > millis())  timer = millis();

  // approximately every 2 seconds or so, print out the current stats
  if (millis() - timer > 2000) {
    timer = millis(); // reset the timer

    //    Serial.print("\nTime: ");
    //    Serial.print(GPS.hour, DEC); Serial.print(':');
    //    Serial.print(GPS.minute, DEC); Serial.print(':');
    //    Serial.print(GPS.seconds, DEC); Serial.print('.');
    //    Serial.println(GPS.milliseconds);
    //    Serial.print("Date: ");
    //    Serial.print(GPS.day, DEC); Serial.print('/');
    //    Serial.print(GPS.month, DEC); Serial.print("/20");
    //    Serial.println(GPS.year, DEC);
    //    Serial.print("Fix: "); Serial.print((int)GPS.fix);
    //    Serial.print(" quality: "); Serial.println((int)GPS.fixquality);
```

```
    if (GPS.fix) {
      //      Serial.print("Location: ");
      //      Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
      //      Serial.print(", ");
      //      Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);
      //      Serial.print("Location (in degrees, works with Google Maps): ");
      //      Serial.print(GPS.latitudeDegrees, 4);
      //      Serial.print(", ");
      //      Serial.println(GPS.longitudeDegrees, 4);

      //      Serial.print("Speed (knots): ");
      return (GPS.speed * 1.852);
      //      Serial.print("Angle: "); Serial.println(GPS.angle);
      //      Serial.print("Altitude: "); Serial.println(GPS.altitude);
      //      Serial.print("Satellites: "); Serial.println((int)GPS.satellites);
    }
  }
}

float collect_battery() {
  // read the input on analog pin 0:
  //  float batteryValue = analogRead(batteryPin); // Read OUTPUT = 0 V to 5 V.
  //  float value = (batteryValue / 1023) * 5; //
  //  float realValue = value * 30;

  return (((analogRead(battery_pin) / 1023) * 5) * 30);
}

float collect_current_sensor_1() {
  delay(5);
  float output_voltage_value_1 = analogRead(current_pin_1) * (5.0 / 1023.0);
  float current_value_1 = (output_voltage_value_1 - (5.0 / 2.0)) / 0.037;
  return current_value_1;
}

float collect_current_sensor_2() {

}

float collect_current_sensor_3() {
```

```
}

float collect_current_sensor_4() {

}

float collect_current_sensor_5() {

}
```

Lattepanda Python code

```
#
# Written and debugged by Shankar
# Date: 12 September 2018
# Purpose: This code is a dashboard for a solar vehicle. Refer to README
# All rights reserved.

try:
   import pygame
   import math
   import random
   from pygame.locals import *
   import serial
   import time
   import numpy as np
   import sys
   import cv2
except ImportError as ImpErr:
   print("Check your imports !")
   print(str(ImpErr))


class Initialise(object):
   # Colour definitions (Red ,Green, Blue)
   # All the below code are class variables
   BLACK = (0, 0, 0)
   WHITE = (255, 255, 255)
   RED = (255, 0, 0)
   GREEN = (0, 255, 0)
   BLUE = (0, 0, 255)
   YELLOW = (255, 255, 0)
```

```
    CYAN = (0, 255, 255)
    BROWN = (83, 91, 36)
    SILVER = (192, 192, 192)

    pygame.init()
    pygame.display.set_caption("Dashboard")
    clock = pygame.time.Clock()
    resolution = (1024, 600)
    screen = pygame.display.set_mode(resolution, pygame.RESIZABLE)

    # raspberry pi code
    # ser = serial.Serial('COM5', 9600, 8, 'N', 1, timeout=5)

    # lattepanda code
    ser = serial.Serial('COM5')


class Text(Initialise):
    def message_display(self, text, x_position, y_position):
        largeText = pygame.font.Font("freesansbold.ttf", 20)
        # The text is inside a rectangle and can be referenced by a rectangle.
        textSurface = largeText.render(text, True, Initialise.CYAN)

        # TextSurface, TextRect = text_objects(text, largeText)
        TextRect = textSurface.get_rect()
        TextRect.center = (x_position, y_position)
        Initialise.screen.blit(textSurface, TextRect)


class Battery(Text):

    def draw_rect(self, battery_value):
        pygame.draw.rect(Initialise.screen, Battery.SILVER, (956, 65, 15, 30))
        pygame.draw.rect(Initialise.screen, Battery.WHITE,
                         (800, 25, 158, 120), 3)

        #       pygame.draw.rect(screen, color, (x,y,width,height), thickness)
        # pygame.draw.rect(self.screen, Battery.GREEN, (802, 27, 145, 97))
        if (battery_value > 70 and battery_value <= 140):
            pygame.draw.rect(Initialise.screen, Initialise.GREEN,
                             (802, 27, battery_value, 117))
```

```
        elif ((battery_value > 42) and (battery_value <= 70)):
            pygame.draw.rect(Initialise.screen, Battery.YELLOW,
                             (802, 27, battery_value, 117))
        elif (battery_value >= 0 and battery_value <= 42):
            pygame.draw.rect(Initialise.screen, Initialise.RED,
                             (802, 27, battery_value, 117))
            self.bolt_image = pygame.image.load("bolted.png")
            self.bolt_image = pygame.transform.scale(self.bolt_image,
                                                     (100, 110))
            Initialise.screen.blit(self.bolt_image, (830, 33))
            Text.message_display(self, text="LOW Battery !!!", x_position=865,
                                 y_position=200)


class Speedometer(Initialise):

    def load_image(self):
        self.speedometer_image = pygame.image.load("speed.png")
        self.speedometer_image = pygame.transform.scale(self.speedometer_image,
                                                        (700, 450))
        Initialise.screen.blit(self.speedometer_image, (155, -40))

    def draw_arc(self, speed_value):
        # pygame.draw.arc()
        # Good example arc below ->
        # pygame.draw.arc(self.screen, Speedometer.YELLOW,
        #  (235, 75, 525, 525), math.radians(-42), math.radians(223), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
        """ Draws an elliptical arc on the Surface.
        The rect argument is the area that the ellipse will fill.
        The two angle arguments are the initial and final
        angle in radians, with the zero on the right. The width argument is the
        thickness to draw the outer edge.

        TAKE NOTE: <Worth mentioning> the initial angle must be less than the
        final angle; otherwise it will draw the full ellipse. """
        pygame.draw.arc(Initialise.screen, Speedometer.YELLOW,
                        (277, 5, 450, 400), math.radians(speed_value),
                        math.radians(224), 5)
```

```
class Temperature(Text):

    def draw_arc(self, temperature_value):

        # Good example arc below ->
        # pygame.draw.arc(Initialise.screen, Initialise.YELLOW,
        # (50, 75, 200, 200), math.radians(0), math.radians(180), 4)

        # draw a partial section of an ellipse
        # arc(Surface, color, Rect, start_angle, stop_angle, width=1) -> Rect
        """ Draws an elliptical arc on the Surface.
        The rect argument is the area that the ellipse will fill.
        The two angle arguments are the initial and final angle in radians,
        with the zero on the right.
        The width argument is the thickness to draw the outer edge.

        TAKE NOTE: <Worth mentioning> the initial angle must be less
        than the final angle; otherwise it will draw the full ellipse. """
        if (temperature_value >= 0 and temperature_value < 45):
            pygame.draw.arc(Initialise.screen, Initialise.RED,
                            (10, 75, 250, 250),
                            math.radians(temperature_value), math.radians(180),
                            25)
            Text.message_display(self, text="HIGH Temperature !!!",
                                 x_position=150, y_position=235)

            self.high_temperature_image = pygame.image.load("temperature.jpg")
            self.high_temperature_image = pygame.transform.scale(
                self.high_temperature_image, (50, 50))
            Initialise.screen.blit(self.high_temperature_image, (115, 125))

        elif (temperature_value >= 45 and temperature_value < 135):
            pygame.draw.arc(Initialise.screen, Initialise.YELLOW,
                            (10, 75, 250, 250),
                            math.radians(temperature_value), math.radians(180),
                            25)
        elif (temperature_value >= 135):
            pygame.draw.arc(Initialise.screen, Initialise.GREEN,
                            (10, 75, 250, 250),
```

```
                        math.radians(temperature_value), math.radians(180),
                        25)
            Text.message_display(self, text="LOW Temperature !!!",
                                 x_position=150, y_position=250)


class Camera(Initialise):
    def __init__(self):
        # NOTE -> tutorial link :
        # https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui
        # /py_video_display/py_video_display.html

        self.cap = cv2.VideoCapture(0)

        # set the width and height
        self.cap.set(3, 1000)
        self.cap.set(4, 250)

        # Check if camera is opened successfully
        if (self.cap.isOpened() == False):
            print("Error opening video stream or file")

    def use_camera(self):
        # Capture frame-by-frame
        ret, frame = self.cap.read()
        frame = np.rot90(frame)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = pygame.surfarray.make_surface(frame)

        # blit to the screen and set the (x,y) coordinates
        self.screen.blit(frame, (0, 350))

    def stop_preview(self):
        # When everything done, release the capture
        self.cap.release()
        cv2.destroyAllWindows()

class UpdateValues(Initialise):
    def __init__(self):
        self.temperature_value = 4.5
        self.temperature_value_original = 39
```

```
        self.speed_value = 30.11
        self.speed_value_original = 25

        self.battery_value = 135
        self.battery_value_original = 75

    def transferValues(self):
        if (Initialise.ser.in_waiting):
            try:
                values = Initialise.ser.readline()
                print('values', values)
                decoded = values.decode("utf-8")
                print('Decoded', decoded)
                decoded = str(decoded)
                split_decoded = decoded.split(",")

                self.data = split_decoded[0].strip()
                self.time = split_decoded[1].strip()
                self.temperature_value_original = float(
                    split_decoded[2].strip())
                self.speed_value_original = float(split_decoded[3].strip())
                self.battery_value_original = float(split_decoded[4].strip())
                self.current_1 = float(split_decoded[5].strip())
                self.current_2 = float(split_decoded[6].strip())
                self.current_3 = float(split_decoded[7].strip())
                self.current_4 = float(split_decoded[8].strip())
                self.current_5 = float(split_decoded[9].strip())

                #   data = split_decoded[0].strip()
                #   time = split_decoded[1].strip()
                #   temperature = float(split_decoded[2].strip())
                #   speed = float(split_decoded[3].strip())
                #   battery = float(split_decoded[4].strip())
                #   current = float(split_decoded[5].strip())

                self.temperature_value = (-9 / 2 *
                                          self.temperature_value_original) + 180
                self.speed_value = (-133 / 110 *
                                    self.speed_value_original) + 224
                self.battery_value = self.battery_value_original
```

```
            print("Temperature = {}, speed = {}, battery = {}".format(
                self.temperature_value_original, self.speed_value_original,
                self.battery_value_original))
        except:
            print(
                "I cannot convert string (Failed to read from DHT sensor) "
                "to float")


if __name__ == "__main__":

    initialise = Initialise()

    battery = Battery()

    speedometer = Speedometer()
    speedometer.load_image()

    temperature = Temperature()

    text = Text()

    camera = Camera()

    updatevalues = UpdateValues()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_f:
                    initialise.resolution = (1024, 600)
                    initialise.screen = pygame.display.set_mode(
                        initialise.resolution, pygame.FULLSCREEN)
                elif event.key == pygame.K_g:
                    initialise.resolution = (1024, 600)
                    initialise.screen = pygame.display.set_mode(
                        initialise.resolution, pygame.RESIZABLE)
```

```
            elif event.key == pygame.K_c:
                pass
            elif event.key == pygame.K_v:
                camera.stop_preview()
                # The below line requires to be in a While loop
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    camera.stop_preview()

        initialise.screen.fill(initialise.BLACK)

        speedometer.load_image()
        speedometer.draw_arc(updatevalues.speed_value)

        battery.draw_rect(updatevalues.battery_value_original)

        temperature.draw_arc(updatevalues.temperature_value)

        text.message_display(text="{} Volts".format(
            updatevalues.battery_value_original), x_position=865,
            y_position=175)

        text.message_display(text="{}".format(
            updatevalues.speed_value_original), x_position=500, y_position=200)

        text.message_display(text="{} degrees".format(
            updatevalues.temperature_value_original), x_position=150,
            y_position=200)

        camera.use_camera()

        updatevalues.transferValues()

        initialise.clock.tick(60)
        pygame.display.update()

pygame.quit()
quit()
```

Motor

List of Illustrations for motor

1.0 Introduction to Motor System

Figure 1 Mitsuba Motor

The motor system consists of 2 main components, motor and the motor controller.

1.1 Technical Specifications

M2096 specification

motor	
model number	M2096D-Ⅲ
dimension	φ 262mm × L73mm
weight	11kg
type	DC brushless motor in wheel type (direct drive type)
nominal power	2.0kW
maximum power	about 5000W (see note)
efficiency	more than 95% (including motor controller efficiency)
nominal load rotation speed	810rpm
rotating direction	forward : left turn (when see the wheel) / right turn optionally
controller	
model number	M2096C
dimension	W203mm × D213mm × H93.5mm
weight	3.5kg
cooling operation	natural air cooling
nominal voltage	96V
input voltage	45～160V
operation	120 degrees Square-wave control
control mode	
current control	checking input current and automatic adjust PWM DUTY
manual PWM control	direct control PWM Duty
reverse switch	available (with speed limit)
generation brake system	power adjust and voltage limiter (program by use)

note: maximum power which depend on voltage and battery

Figure 2 Motor and Motor Controller technical specifications

1.2 Brushed Motor

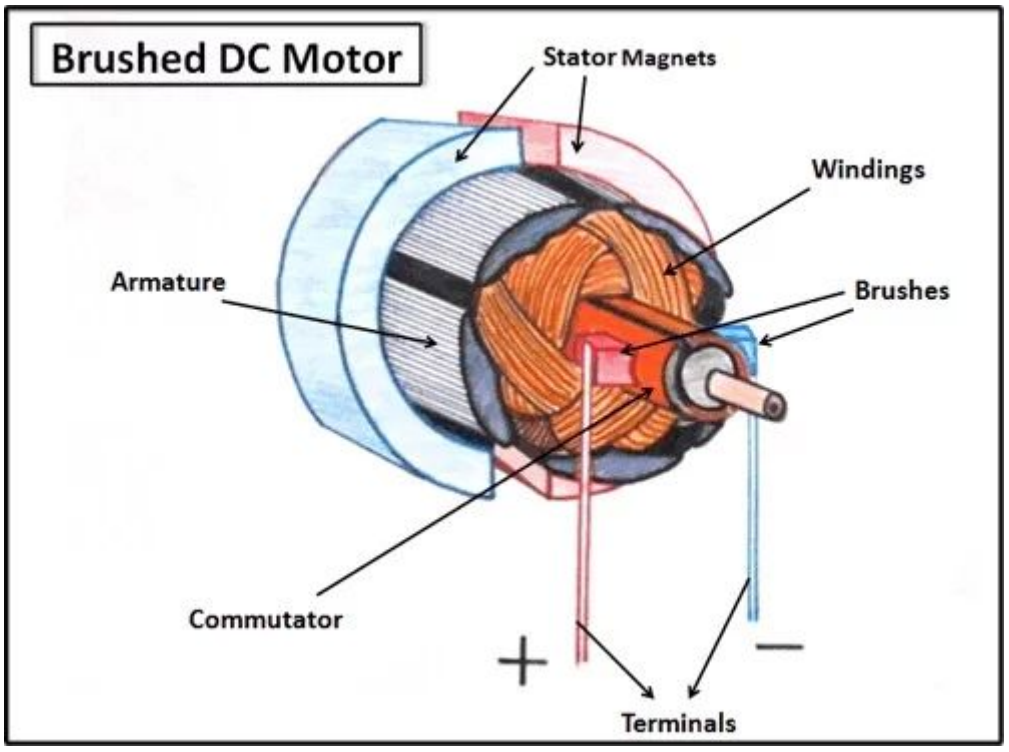


Figure 3 Brushed Motor (Microchip.com, 2019)

The stator generates a stationary magnetic field that surrounds the rotor and this magnetic field is generated by either permanent magnets or electromagnetic windings. Rotor, also known as the armature is made up of one or more windings. When these windings are energized they produce a magnetic field. The magnetic poles of this rotor field will be attracted to the opposite poles created by the stator, causing the rotor to rotate. As the motor rotates, the windings are constantly being energized in a different sequence so that the magnetic poles generated by the rotor do not overrun the poles generated in the stator. This switching of the field in the rotor windings is called Commutation. (Microchip.com, 2019)

The brushes charge the commutator inversely in polarity to the permanent magnet, in turn causing the armature to rotate. The rotation direction, clockwise and/or counterclockwise, can be reversed easily by reversing the polarity of the brushes, i.e., reversing the leads on the battery.

Advantages :

- Simplified wiring
    - Brush motors can be wired directly to DC power and control can be a simple as a switch.
- Low cost

Disadvantages :

- Less efficient
- Electrically Noisy

    - The switching action of the commutators constantly creating and breaking inductive circuits creates a great deal of electrical and electromagnetic noise.
- Lifespan
    - Brushes and commutators wear out in time because they are in perpetual physical contact with the shaft.

1.3 Brushless motor

Brushless DC motors (BLDC) feature high efficiency, excellent controllability and have power-saving advantages relative to other motor types. In a BLDC motor, the commutation occurs electrically and not mechanically.

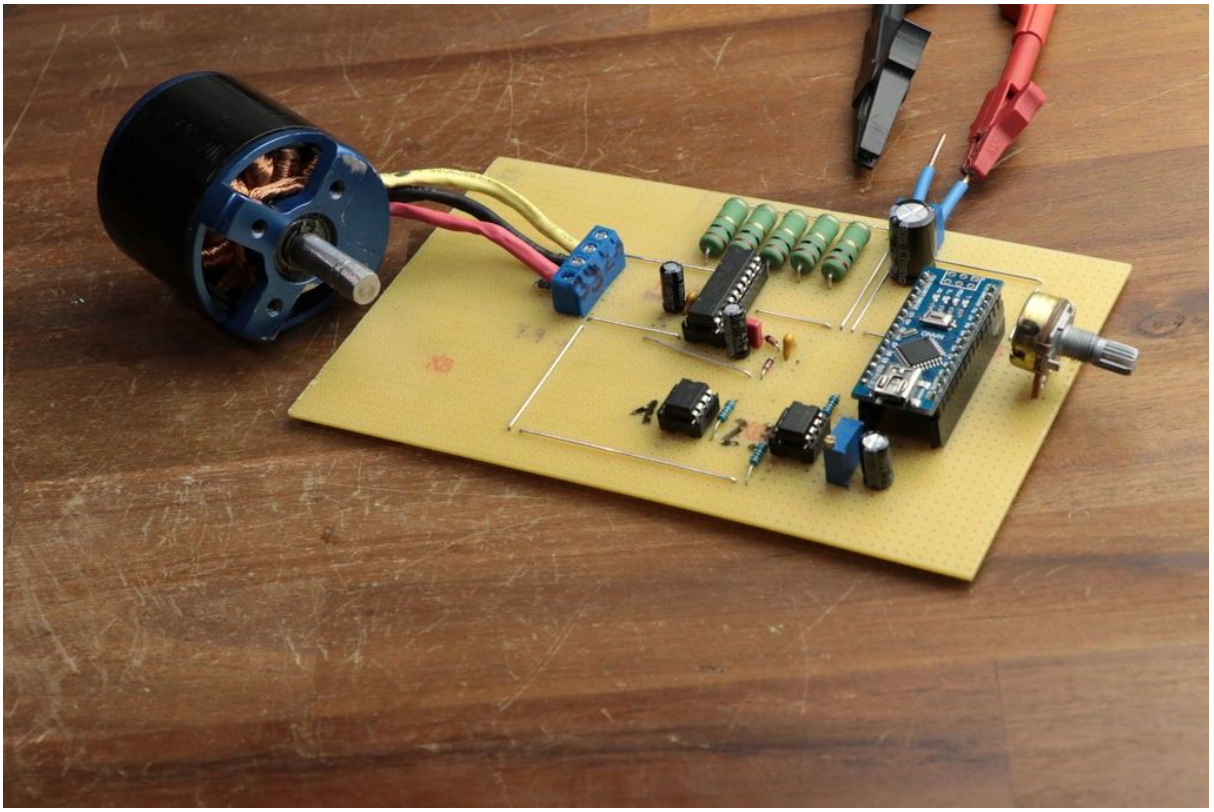


Figure 4 BLDC motors and a microcontroller for the motor control

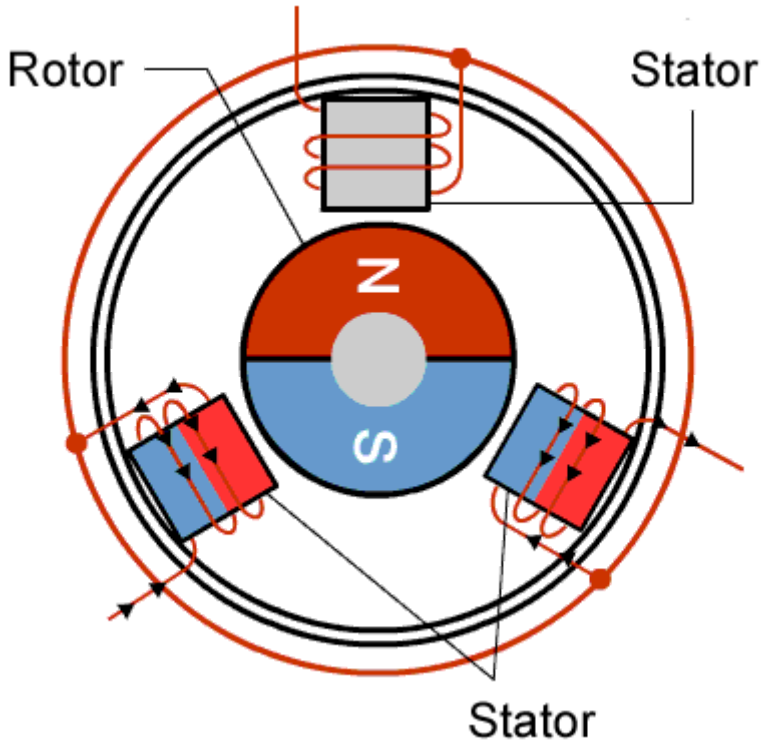


Figure 5 Brushless DC Motors (Renesas Electronics, 2019)

The coils of a BLDC are not located on the rotor like a brushed motor. Instead, the rotor is a permanent magnet. The coils are stationary and are fixed in place on the stator. There is no need for brushes and a commutator because the coils are stationary. In a BLDC motor, the permanent magnet rotates. changing the direction of the magnetic fields generated by the surrounding stationary coils. We change the direction of the magnetic fields generated by the surrounding stationary coils, thus rotating the motor. A motor controller or a ESC is used to control the rotation by adjusting the magnitude and direction of the current into these coils.

Advantages :

- ➢ Efficiency
  - BLDC motors can control continuously at maximum rotational force (torque).
- ➢ High durability and low electric noise generation
- ➢ High Controllability
  - Feedback mechanisms are used to control BLDC motors to deliver the precise desired torque and rotation speed. Energy consumption and heat generation are reduced, the battery could provide power for a extended time.
- ➢ Higher safety compared to Brushed motors (No internal sparks)

Disadvantages :

- ➢ High initial cost (motor controller)

1.4 Terminals of a BLDC controller

controller – terminal

description	parts terminal of number and position	wire color	terminal bar terminal number	note
main switch	center	white	01	
	side	black	20	
acceleration volume	2	white	02	
	1	black	21	
	3	red	03	
generating brake volume	2	white	22	
	1	black	04	
	3	red	23	
power / eco mode switch	center	white	05	
		—	24	prohibit to connect
	side	black	06	
forward / reverse switch	center	white	25	
	side	black	07	
			26	prohibit to connect
			08	prohibit to connect
			27	motor rotation pulse out put signal (0-5V)
			09	prohibit to connect
			28	GND
			10	prohibit to connect
			29	prohibit to connect
			11	Map_GND
			30	prohibit to connect
digital switch		red	12	Map_Bit0
		White	31	Map_Bit1
		green	13	Map_Bit2
		yellow	32	Map_Bit3
LED	K (-) :3	black / (brown)	14	LED_GND-0V
			33	prohibit to connect
LED	A (+) :1	white	15	LED_+
			34	prohibit to connect
			16	prohibit to connect
			35	prohibit to connect
			17	prohibit to connect
			36	prohibit to connect
			18	prohibit to connect
			37	prohibit to connect
			19	prohibit to connect

motor sensor signal cable – controller

sensor circuit board	wire color	panel connector positions of R05-PB6N	
CON01	yellow	A	power input (+)
CON02	black	B	GND (0V)
CON03	red	C	A line
CON04	white	D	B line
CON05	green	E	C line
—	sealed	F	sealed

motor – controller

motor	controller
red	A
white	B
black	C

battery – controller

battery	controller
positive +	+
negative —	—

if make mistake connection which will be make damage or broken this kit.

【signals for speed】
①pulse out put signal
・you will have 0/5V (off/on) at terminal 27 – 28(GND(0V))
・16pulse/1rotation
【acceleration volume】
Mitsuba recommend volume switch 5kΩ ～10kΩ and at full acceleration you will have 4.7V～4.8V at 02 – 21GND(0V)which are correct voltage.

Figure 6 description of electric terminals of a BLDC

1.5 Motor Connection for the SunSPEC vehicle

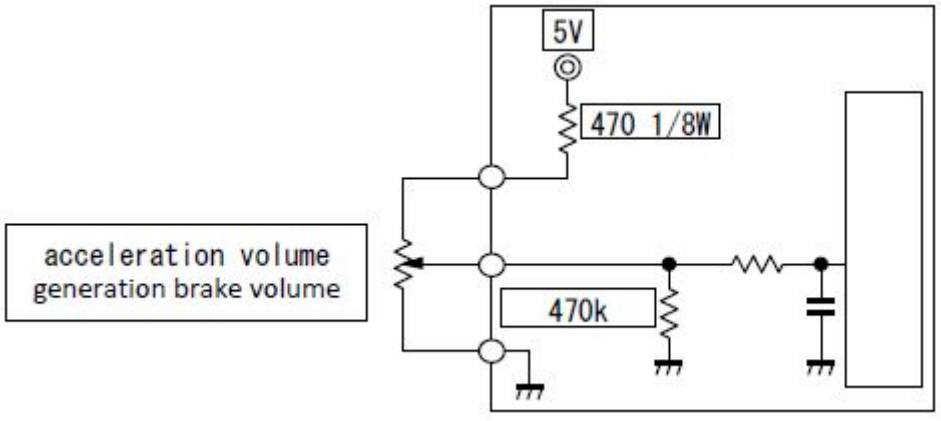


Figure 7 input / output composition figure

Terminals 1 and 20 have a switch in between to turn ON and OFF the vehicle. Terminals 2,21 and 3 are connected together in a potentiometer to control the acceleration of the vehicle.

1.6 Motor Software

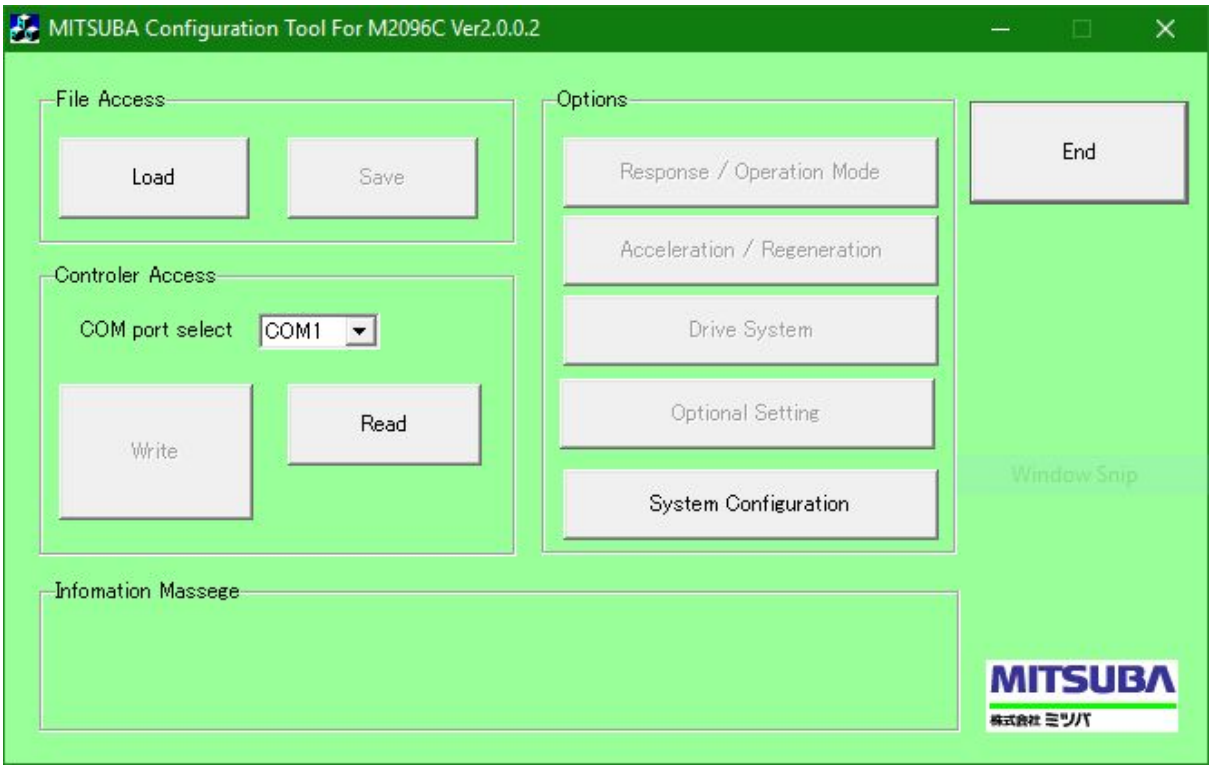


Figure 8 Mitsuba Configuration Tool for M2096C

The Mitsuba configuration tool allows us to configure the settings of the motor to our needs. We also have the option to save and load a sph file which consists of parameters setting for the motor.

<u>2 steps to connect to the motor controller software</u> :

1. Use a USB cable from the motor controller to connect to a computer which has the mitsuba configuration tool.
2. Select the correct COM port under the Device Manager

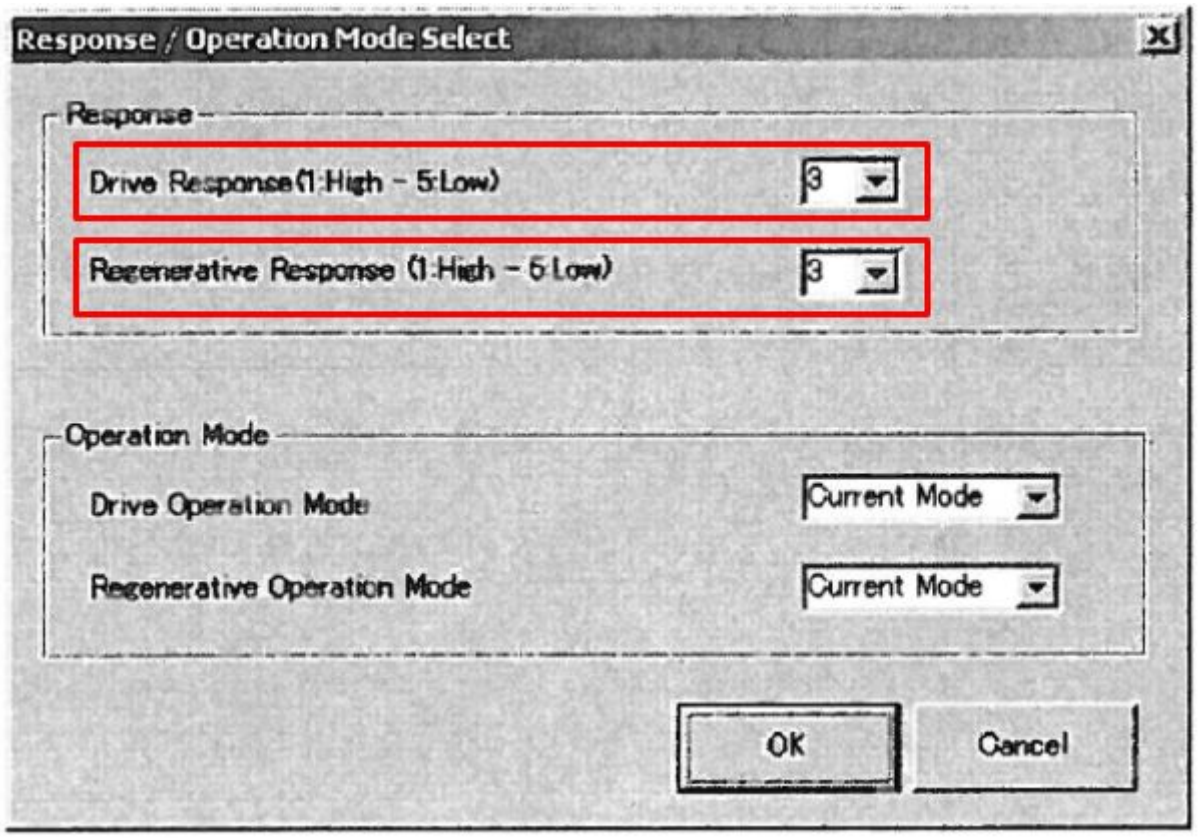


Figure 9 Response / Operation Mode Select

The default settings are recommended by Mitsuba, change them if your specifications require.

In the above figure, the drive response and the regenerative response will have some delay when the vehicle is moving forward and when the vehicle is coming to a stop.

Drive Operation Mode : Current Mode

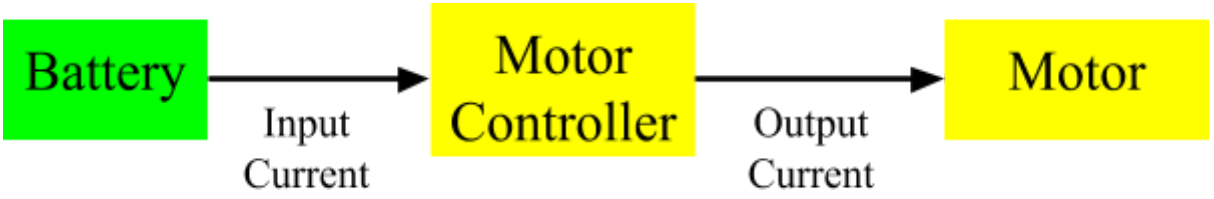


Figure 10 Drive Operation Mode : Current Mode

In the drive operation mode, the battery supplies current to the motor. There are 2 modes to select. They are Current and Pulse Width Modulation (PWM) mode. We are able to control the input current from the battery but not the output current from the motor controller. The output current is determined by the duty cycle from the motor controller.

We will need to limit the input current from the battery. If we fail to do so, we may drain the battery too quickly. As a result, we choose to operate in current mode.

Regenerative Response Mode : PWM Mode

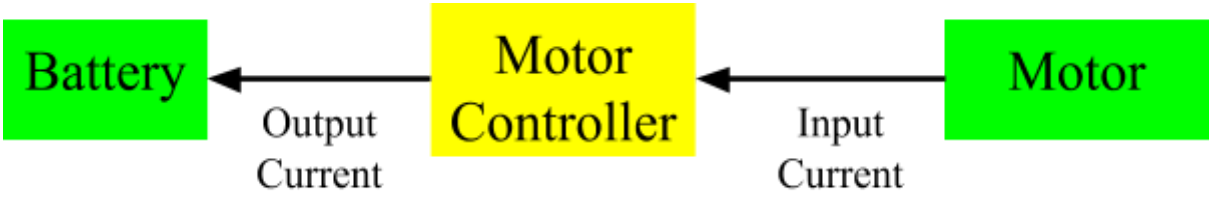


Figure 11 Regenerative Response Mode : PWM Mode

The regeneration mode occurs when the solar vehicle is rolling on a sharp downhill slope and forces the motor to turn fast enough resulting in an electromotive force to be generated. Regenerative braking also occurs when an energy recovery mechanism slows the solar vehicle by converting its kinetic energy into electricity. The motor acts as a generator pumping current to the battery. There are Current and Pulse Width Modulation (PWM) mode. As mentioned earlier, we are able to control the input current from the battery but not the output current from the motor controller. Here, we do not wish to limit the current, so we use PWM mode to get as much energy from the motor. This will result in the solar car moving more distance in the race.

Acceleration / Regeneration Setup

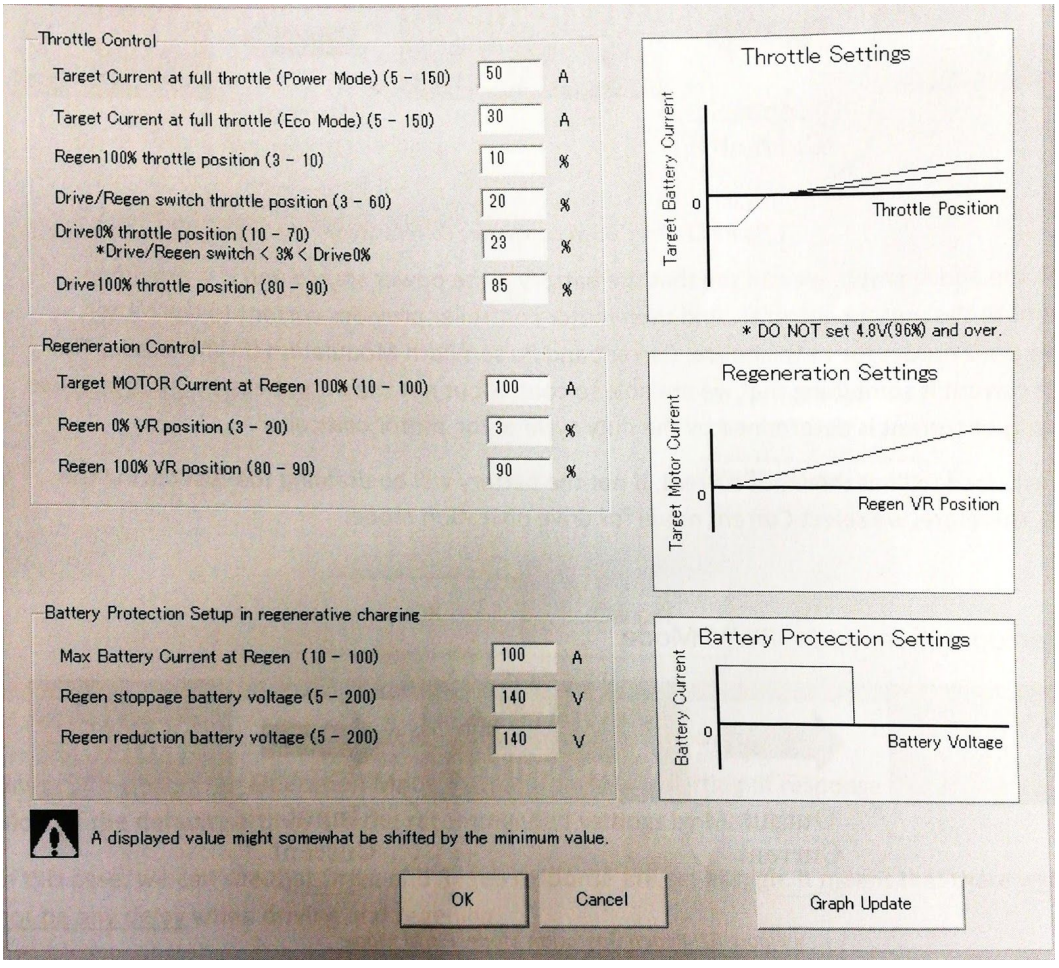


Figure 11 Acceleration / Regeneration Setup

In the drive operation mode, we set the limit of the current consumption under the throttle control section. A potentiometer is also added to allow the car to be in cruise mode. Since the speed corresponds to the throttle position, we need to adjust the throttle position accordingly.

1.7 Research and improvements

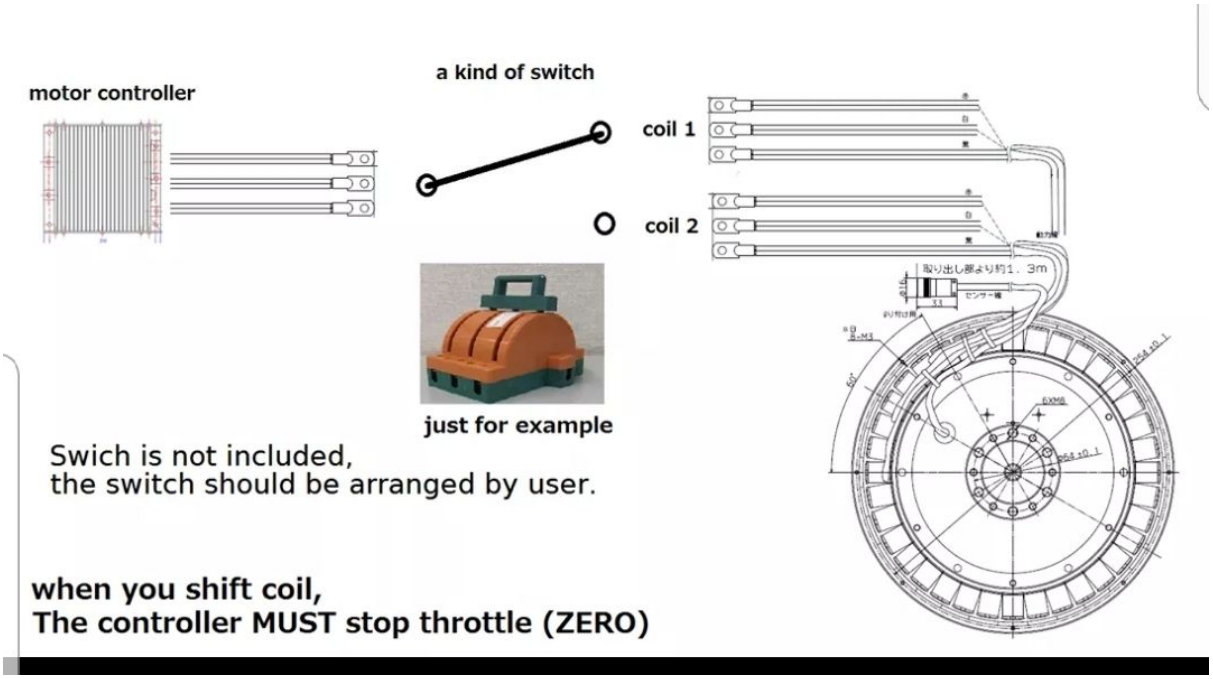


Figure 12 Circuit Diagram of the switch to change coil

Figure 13 Switch to change coil

We have introduced 2 coils to extend the speed-torque limit of the vehicle. The two coils represent 2 different speed-torque curves. 1 speed-torque coil would limit the vehicle to a certain speed for example 75 km/h. 2 speed-torque coils would increase the range of the speed, for example, 1 speed-torque coil for 75-100 km/h and another speed-torque coil for 100-130 km/h. Thus, switching these 2 coils from low speed to high speed will enable us to have a wider range of speed.

Bibliography

Renesas Electronics. (2019). What are Brushless DC Motors. [online] Available at: https://www.renesas.com/sg/en/support/technical-resources/engineer-school/brushless-dc-motor-01-overview.html [Accessed 6 Feb. 2019]. (Renesas Electronics, 2019)

Microchip.com. (2019). Brushed DC Motor Basics. [online] Available at: https://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en543041.pdf [Accessed 6 Feb. 2019]. (Microchip.com, 2019)