Elfoil Data Logging, Transmission and Visualisation

GENG5012 Final Report

Jeremy Guo
School of Mechanical and Chemical Engineering
The University of Western Australia

Prof Thomas Bräunl
School of Electrical, Electronic and Computer Engineering
The University of Western Australia

Word Count: 7965

Abstract

Electrical hydrofoil jet-ski named Elfoil is an innovative design built by UWA, Australian start up Electro Aero and Tromes design funded by lithium producer Galaxy Resources originally. The ultimate goal for Elfoil is commercialise and provide a more fun experience for jet skis riders.

In any watercraft, the abilities the visualise and store the data is crucial. This project particularly will investigate the data logging, network and data visualisations using telemetry and ioT applications.

The areas of interest can be more specifically dived into data logging various sensors of data, data communications through different protocol and data visualisation in real time using telemetry and ioT applications. This project will aim to provide a serverless approach that client will not be required to have some sort of server in place.

The project will conduct researches on related literature reviews and background knowledge, investigate into potential methods, share the results as well as discuss the limitations and future work.

Acknowledgements

I would like to express my gratitude towards Prof. Thomas Braunl for his on-going support and supervision for the project. COVID has hindered the process of the project but has not hindered the cohesiveness of the team, I would like to express my gratitude towards the whole REV team for supporting and giving encouragements to each other. Last but not least, I would like to thank my family for all unconditional support throughout the years.

Table of Contents

List of Figures

List of Figures Section 4

List of Figures Section 5

List of Listings

List of Listings Section 4

List of Listings Section 5

List of Table

Nomenclature

3G	Third Generation
API	Application Programming Interface
BMS	Battery Management System
CAN	Controller Area Network
ESC	Electrical Speed Control
GPIO	General Purpose Input/Output
IMU	Inertial Measurement Unit
ioT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
LTE	Long Term Evolution
REV	Renewable Energy Vehicle
UART	Universal asynchronous receiver-transmitter
UI	User Interface

1.0 Project Background

The REV team advocates revolutionise transport by building zero emission vehicles. A petrol-based jet ski has been converted to electrical with adding in features such as hydrofoils and ailerons before the start of this project.
The watercraft is originally built by Electro Aero and water tested in 2019. It was then handed over to Tromes design working alongside with UWA for modifications and improvements. However, the modifications process has not been smooth, up until this point, there has been any water tests with REV members present yet. Fortunately, things are starting to move towards a positive direction that is a step closer towards water tests.

2.0 Problem identification

The abilities to monitor and trace back the key parameters are crucial. More specifically are abilities to log key parameters, establish communication between various source and able to live track and monitoring.
The advantages are, firstly, it allows the key parameters to be pushed to the display monitor, secondly it provides the abilities for users to trace back the state of the jet ski for troubleshooting and performance evaluation, thirdly live tracking and monitoring ability to provide safety features and allow users to monitor the jet ski off-shore.

3.0 Project objectives

The objectives of the jet ski are to eventually commercialise the jet ski.
The objectives can be separated into:
1) Develop a system to log all the key parameters such as water temperature pressure from depth sensor, battery remaining, current speed, GPS coordinates, pitch yaw and roll.
2) Develop a method for data transmitting network that allow various sensors to communicate to each other.
3) Develop a telemetry method that allow the jet ski to not only store the data locally but remotely onto a database. Also monitoring its position, trending its key parameters in real time and develop an ioT application that allow communication under LAN that the jet ski can display the data on to dashboards with any other device have the same access.
Also document all useful literature review and useful resources for future students or clients to quickly gain the equal understanding without spending the same amount of time.

3.1 The deliverables

The deliverables are a professional report document with all abovementioned resources and code

3.2 Project Scope

The scope of data logging is to develop a method and code to log key parameters and test them. The scope of the data network is to develop a communication method for various source but will not building a network bus for example this project will develop a method and using instrumentations for CAN network but will not cover building the data bus.
The scope of the live monitoring and telemetry is to develop a method, concept proof and conduct modular testings but will not actually be placed on to the jet ski as water testing are not ready at this stage.

4.0 Data logging

4.1 Literature Review

Firstly, the key parameters need to be identified, Woloszyn's thesis was on intrumentations and data logging on a different jet ski which have been defined as the speed of the vehicle, the batteries remaining, GPS coordinates and water conditions [1].
According to Pham "*The basic requirements that are present in almost every modern vehicle is: a speedometer, tachometer, odometer, fuel level gauge, water temperature gauge and warning indicators."* [2]
According to White [3] he recommended all the parameters that with relevance in safety should carefully be monitored and listed out some of the parameters.
Burden recommended that [4] water depth and wind speed are necessary for marine vehicles
Woloszyn provided a framework for choosing instrumentations which summarised as follow: [1]

- Are the specifications of the instrument provide the ability to carry the job with enough accuracy
- Simplicity, can the document provided be easily understood and is it user-friendly
- System integration, can it be easily integrated and compatible with current system
- Cost

4.2 Instrumentations

Using Woloszyn's framework as reference the following instrumentations have been selected and obtained:

Table 4. 1 Instrumentations

Instruments	Functions	Wiki & Datasheet
Raspberry Pi 3B	Data logging and transmitting	[5]
Seeedstudio analogue Pi hat	Allow analogue input reading on Pi	[6]
Seedstudio CAN data analyser	Debugging CAN	[7]
Seedstudio Arduino CAN v2 shield	Debugging CAN	[8]
Longan Serial CAN	Convert serial to CAN	[9]
Waveshare SIM7600X Cellular	Cellular for Pi	[10]
Columbus GPS V800	GPS for Pi	[11]
Huawei E8372 Wi-Fi dongle	Wi-Fi dongle for Pi	[12]

Of which Raspberry Pi 3B, Seeedstudio Arduino CAN shield are reused from previous projects. Raspberry Pi is also recommended by Leong for its multi-threading and processing power [13].

4.3 Logging structure and protocols

The design process for logging starts with a flow chart seen in Figure 4.1.

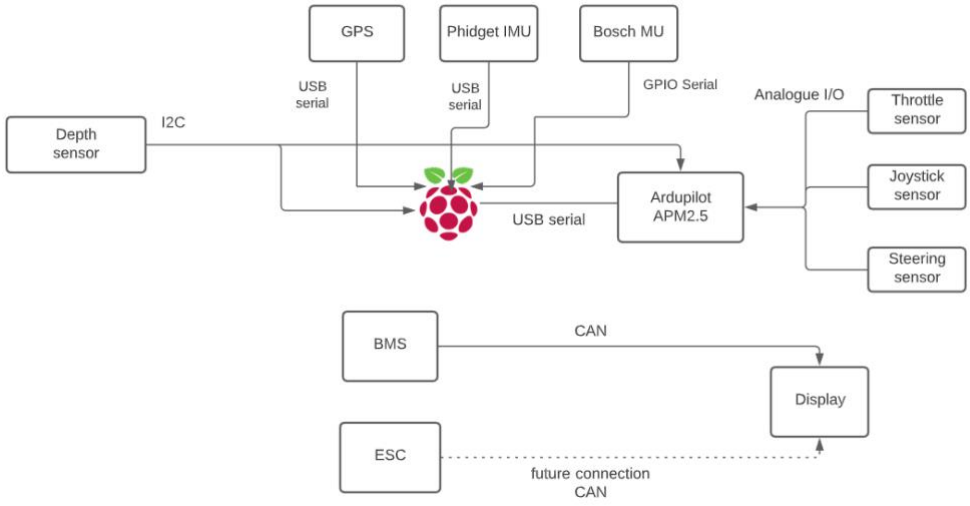


Figure 4. 1Connection diagram

The selections for the depth sensors, throttle sensor, joystick sensor, steering sensor, Bosch IMU, Phidget IMU, ESC and Ardupilot APM2.5 are not covered in this project and hence the instrumentations introduction will not be covered here.

Communications between embedded systems or devices are done by exchange data in the form of bits which has different rules in analogy of different languages has different grammars such rules are called communication protocols.
The depth sensor communicates to the Raspberry pi and Ardupilot via I2C protocol is a protocol that its output is synchronised by a clock signal controlled by the Master. The clock signal is carried on SCL and the data is transferred on SDA bit by bit [14].

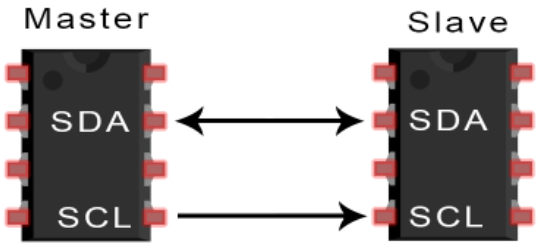


Figure 4. 2 I2C communication protocol, from [14]

GPS, Phidget IMU and Ardupilot are connect to pi via USB serial whereas Bosch's connection is done by GPIO serial. Serial protocol is a protocol sequentially transmitting data bit by bit.
The sensors at the front of the jet ski are steering, joystick and throttle which detect the states of them such as whether a joystick has been push up or down, steering bars current angles and how much throttle handle has been twisted. These are analogue sensors connected to Ardupilot on its analogue input pins that using A/D (analogue to digital converters) on-board to read the sensors reading return a value.

An optional set up uses an analogue grove purchased to allow direct connections to Raspberry Pi as Raspberry Pi does not have onboard A/D, split wires needs to be done to read the analogue signal. An implementation shown in Figure 4.3 has been done to test the viability and reverted for simplicity.

Figure 4. 3 Split wire implementation

BMS is connected to the display via CAN bus which is a broadcast serial bus [16]. Each connected electronic control unit under the bus is called a node. CAN bus runs wire run wires through every device and terminated on each end using a 120-ohm resistor. The two signal wires are differential signals called CAN High and CAN low measuring the resistance of the two wires results a total resistance of 60-ohm as they are in parallel. CAN bus messages are broadcast on the entire bus meaning messages are available at every node without needing

the direct connections. CAN messaged are sensed at every node but only particular node that has been coded to receive the message take actions on it, rest nodes ignore the message. Each node has a unique address on the CAN bus. If two or more modules start writing to the CAN network at the same time the one with the higher priority message take place first.
CAN message format can be seen in figure 4.4.

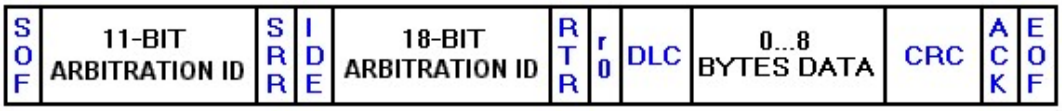


Figure 4. 4 CAN message format, from [16]

It consists start of frame and end of frame are named SOF and EOF respectively
A 11 bits or 29 bits arbitration ID depending on standard or extended frames it identifies the message ID and priority. The data field contains a 0 to 8 bytes of data and the remaining bits are for transmission error detection and handling.

4.5 Implementation

With student Leo Xu and Vladimir Pavkov help the very first implementation of the data logging system has been developed. The system developed the abilities to log either on a csv file or txt file.

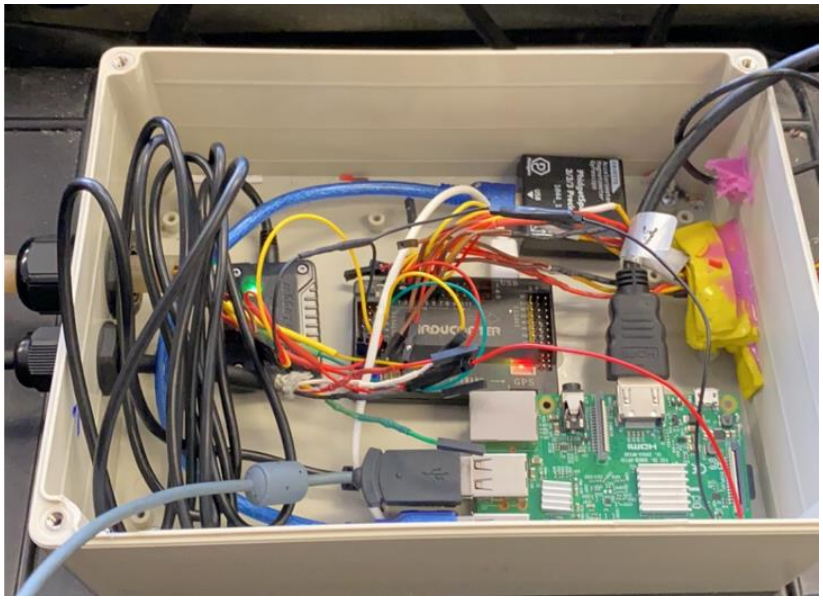


Figure 4. 5 Data logging implementation

To log data on Raspberry pi in a csv file follow a similar format as:

```
import csv
#define var1
#define var2
with open("<nameofcsvfile>.csv","a") as f:
        writer = csv.writer(f)
        writer.writerow([<var1>,<var2>])
        print(acceleration)
```

Listing 4. 1Write to csv file code structure

To log data on to a text file, follow a similar format as:

```
import logging
logging.basicConfig(filename='<nameoflogfile>.log', filemode='a',
format='%(created)f %(message)s', level=logging.INFO)
while True:
    #assign values to var1 and var2
    logging.info('var1={0:0.1f} <unit for var1> and
var2={1:0.1f}<unit for v2>'.format(var1, var2))
```

Listing 4. 2Write to txt file code structure

Performance evaluation is not covered in this project hence meaningless entries will not be shown.

4.6 Data network implementation

4.6.1 CAN Bus

To understand how to receive and store meaningful CAN message from ESC and BMS the data field packets format needs to be known i.e. as discussed, the data field contains a 0 to 8 byte message each byte is revealing different information this information structure needs known before logging.
Unfortunately, this information has not provided in the manufacturer's datasheet manual process is needed by using the Seeedstuio CAN analyser to figure out the data field structure. To do this, student Alishan Aziz took the lead together with REV members.

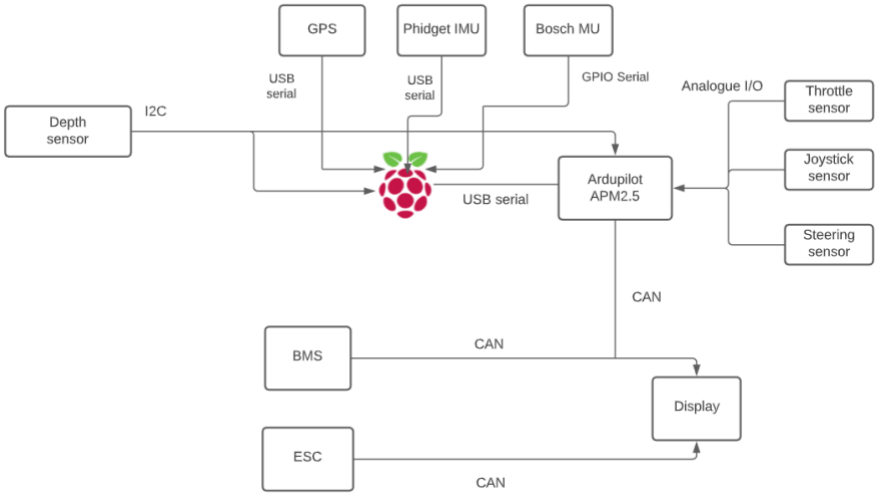


Figure 4. 6 Future CAN connection

The CAN bus has not been fully built yet, the plan is to implement a CAN bus network shown in Figure 4.6. Ardupilot will connect to the main CAN line and all messages will broadcast on the bus. Ardupilot will also serial transmit message to Raspberry Pi via USB serial to store the data both locally and remotely in real time.
As this has not been implemented yet the process of writing and writing has been hindered. However, the ability of writing the data to CAN network have been explored and modular tested to work. A Longan serial CAN bus module has been acquired to suit the application. It allows the serial UART protocol on Ardupilot to be converted to CAN protocol. Ardupilot follows a similar pin out to Arduino mega. Its pin 18 and pin 19 are configured as serial1 and pin 17 and pin 16 are configured as serial2 both supported for serial UART transmission.

A modular test designed to have the following setup in Figure 4.7

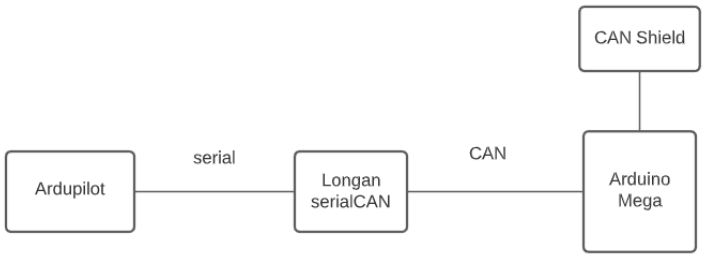


Figure 4. 7 Modular test connection

Ardupilot sending serial data to the Longan Serial CAN module and Arduino mega with CAN shield from Seeedstuido, reading the CAN message to test if it can be received. The implementation shown in Figure 4.8.

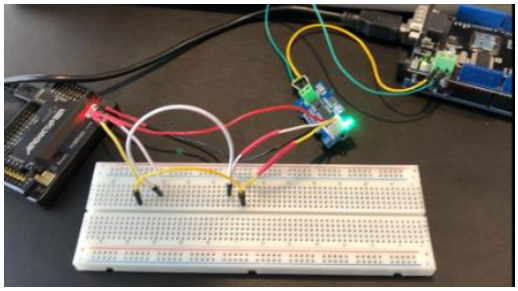


Figure 4. 8 CAN modular test implementation photo

It was successfully received on the mega for a testing data in listing 4.3, proven that it now has the ability to write to the CAN bus once the CAN network has been built.

```
unsigned char dta[8] = {1,2,3,4,5,6,7,8};
```

Listing 4. 3 testing data

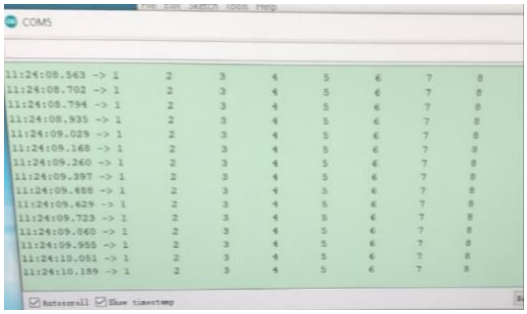


Figure 4. 9 CAN message received

4.6.2 Serial transmission

For serial transmissions there are two main types on a controller board namely the hardware serial and software serial. The native on-board serial supported pins are labelled Tx and Rx and called a UART. To replicate these functions by using software packages can enable other digital pins to do serial transmit.

5.0 Telemetry and ioT applications

To reiterate the goal is to develop a telemetry and ioT applications allow data transmission in real time and stored on the database, visualise and trend the data.

5.1 Literature Review

Aron Goldswordthy did a project on solar powered autonomous boat (SPAB) [17]. He used a way to transmit data from boat to a webserver via the Hypertext Transfer Protocol (HTTP) it requires a webserver, an open port 80 [13]. Goldswordthy covered a method of REST and had a local webserver on RPI with a 3G modem he can access the database host locally on Raspberry pi and access data on the website [17].

A flow chart shows in figure 5.1

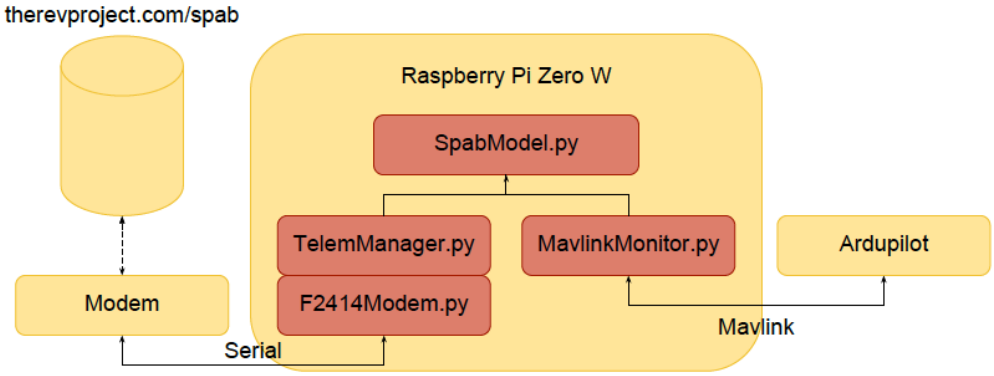


Figure 5. 1Goldsworthy's code architecture, from [17]

All messages were sent in JSON message format [17]. It performs a GET request periodically obtain the new commands from the server in JSON message format [17].
'TelemetryManager.py' defines the response of SPAB should take based on the commends from the modem. The 'modem.py class' decodes the JSON message and logging data to a database on UWA server. Leong has implemented a method using HTTP for data transfer on an Arduino mega with sending data to REV Server as well [13].

In order to understand how data transmission works background knowledge is needed. Data transmission over the internet follows OSI (Open Systems interconnection) model it is developed by ISO (Internation Orangnization of Standardization) in 1984, it streamlines data transmission into 7 layers [18]. This can be simplified into 4 layers seen in Figure 5.2.
From sender's perspective, it goes from top to bottom where at the application layer being websites and emails. Websites particularly are using Hyerpertext Transfer Protocol (HTTP) it transfers the data to the transport layer through port 80 using a protocol called Transmission Control Protocol (TCP).
TCP then splits the data into packets and send to the internet layer using a protocol known as Internet protocol (IP) it attaches headers which act like "instructions" on how to assemble the packets to make meaningful sense and error checking. IP protocols embedded with sender's IP and designation IP it passes to the network layer to determine the suitable route to send the data with minimal conjunction and latency [18]

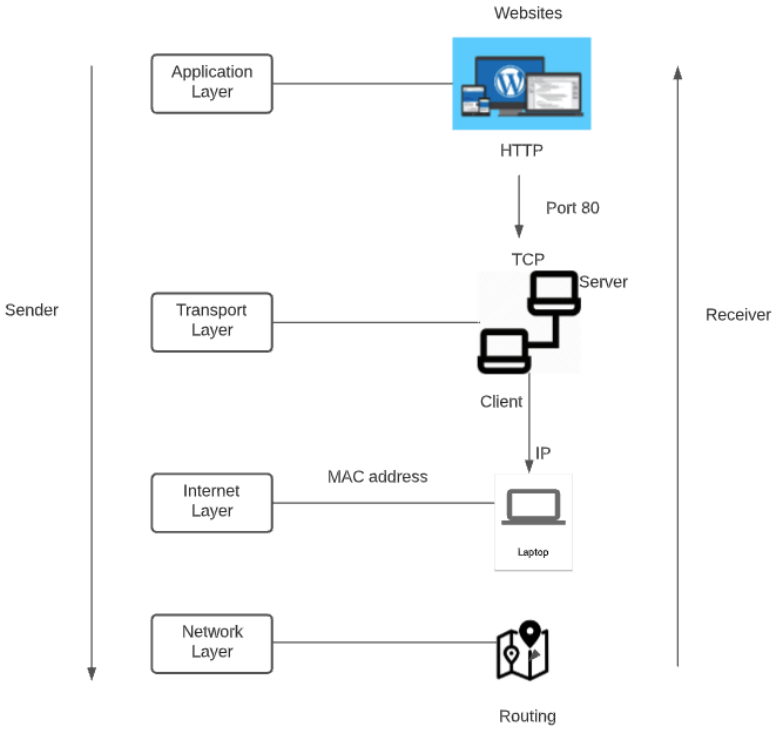


Figure 5. 2 Data transmission layers

The traditional method of transmitting the data involves develop a direct connection between the modem on Jetski and a TCP socket. On the server it will run a Daemon (multitasking computer system) to continuously parse and store the data [13]. However, this is not available at REV. So, a direct transmission on the Transport Layer has been crossed out. Leong and Goldsworths used a method of sending HTTP requests to a database hosting on a web server without the need for a daemon and direct web server access called Rest API. It sends HTTP requests to the webserver in order to operate data from the database hosting on the server. REST API is originally defined by Dr. Roy Fielding in his doctorate research paper in 2000 [19]. API (application program interface) acts like an intermediary request that data from a server and send back the response (the data have been requested) to the user [20]. To be considered as Rest or Restful API certain criteria has to be met which can be found here [21]. More specifically, on a server it will expose endpoints which are URL addresses for example 'revtracking.com/gps' a client can send HTTP request to the server to get data from a database. The '/gps' part is called resources. The main types of HTTP request are included as Get, Post, Put and Delete are called CRUD Operations [22] functionally means read, add, update and delete.

Another important area is internet protocol for telemetry and data transmission. The two types of IP are private and public, private is used to locally locate a device under LAN (local area network) whereas public IP is used for a device to be accessed publicly over the internet.

Private IP

In a local network a router, or more specifically DHCP (Dynamic Host Configuration Protocol) server assign free IP addresses to devices, each device is uniquely identifiable by an address called MAC address [23]. Hence, each time this address might be different. However, the private IP address can be configured to be static the advantages in this project are when Raspberry Pi needs to be SSH (secure shell) into (a remote way to connect to Raspberry Pi) also when implementing ioT applications which will be covered later.

To configure Raspberry Pi with a static private IPV4 address is to use a DHCP client daemon or DHCPCD which allows the Raspberry Pi communicates with the DHCP server (the one responsible to assign IP) and assign a static IP address to the Raspberry Pi. Such method has covered in Digital Guide IONOS article [24]

Public IP

Public IP on the other hand, it used for a device to reach or to be reached publicly over the internet. Most providers do not support for static public IP address and even do it's often come expansive [25]. The best alternative is to use a dynamic DNS (Domain Name System) service. This is done by linking a domain name to the public dynamic IP and when it changes it looks up the domain name and redirect to the new IP address [24]. This allows database host on Raspberry pi to be access remotely without keep changing the IP or it's useful to configure a server to allow access to the Raspberry Pi.

5.2 Instrumentation

5.3 Internet Modem

To achieve live data internet modem is needed. There are various ways to do it. Firstly, is to use a cellular hat designated to Raspberry Pi. Designated hat usually quite powerful yet but harder for beginner. Secondly use a USB internet modem, there are Wi-Fi network enabled and non-Wi-Fi network enabled. Wi-Fi network enabled allow any other devices connect to the Wi-Fi onboard. The cheapest way is configuring the Raspberry Pi to rider's cell phone hotspot.

5.4 Implemented instruments

Waveshare SIM7600X

It uses the SIM7600X chip which is produced by a global M2M (machine to machine) wireless telecommunication company named Simcom [26].

SIM7600X is a global version supports wireless communication modes of most 3G and 4G networks. It has been ensured that it supports most of the Australian providers bandwidth frequencies, such properties are usually measured in megahertz bands checking prior before purchasing any internet modem is important.

It has GPS, onboard audio, TF/SD card slot embedded onto the module. It supports most protocols for applications [10].

HUAWEI E8372

It been purchased to work along with the cellular hat. This is a cost-effective Wi-Fi modem that supports all LTE (4G network) and 3G network bands. It also provides up to 10 simultaneous users connect to it [28].

SIM cards

3G SIM card and 4G SIM card. REV is sponsored with a no on-going cost 3G SIM card. However, it was identified, and performance compared in Leong's thesis that 4G SIM card is recommended [13]. This project will mainly use a 4G SIM card.

5.4 Method and implementation

This project explores a methodology for a serverless implementation as the clients may not have servers and resources available as UWA REV team.

Google CloudSQL

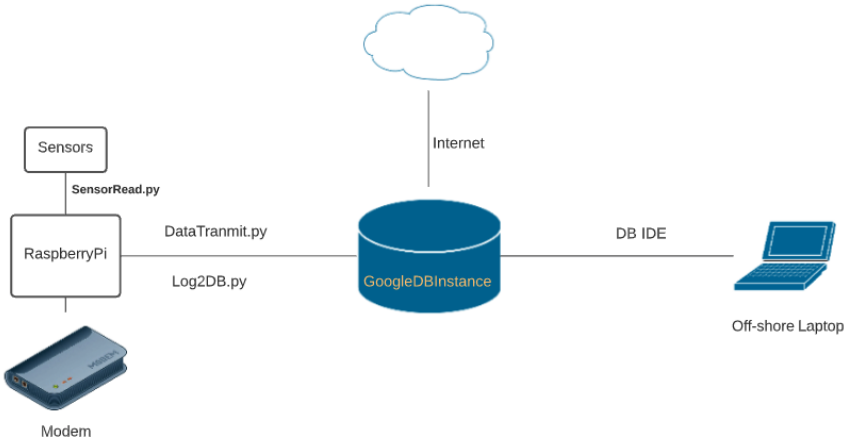


Figure 5. 3 CloudSQL structure

Google cloud-based Cloud SQL provides fully managed SQL databases in the cloud, it supports most of the databases such as PostgreSQL, SQL Server and MySQL. It has automatically backed up databases and it has automatic failover to make the availability of databases extremely high [28].
Google Cloud SQL provides a serverless solution for hosting the database instances and has a high ease of use as clients may not want to manage and maintain a server and the failover ensure the database is available most of the time. As of the time this project, Cloud SQL offers a free $300 credit for new users [28].
A MySQL instance has been created as REV team uses MySQL on the Blue Host server and MySQL can be easily migrated, duplicated and merged.

5.5 Connection methods

There are several ways to connect to the Cloud SQL. The user can set up the Cloud SQL instance with a private or public IP. The client machine needs to install MySQL client package.

If user choose the private IP, it assigns a private IP address to the database instance inside of Google's network which is the underlying network where the Cloud SQL instance resides, and client can connect to it using a virtual system called Virtual Private Cloud (VPC) [29]. This is method safer method compare with setting up your Cloud SQL with a public IP [30].

The Cloud SQL proxy is a safe way to connect to the database it can be set up on most operating systems such as Linux Windows and MacOS the structure is on Figure 5.4. It works by having a proxy client that connects to the proxy server via a secure tunnel on the Cloud SQL server which connects to the database [31]. It creates outgoing connection to Cloud SQL on port 3307 user should have the port open for it to work. Below on Figure 5.4 is an overview for how it works [30]. Client using the SQL proxy does not need to have a static public IP [31].

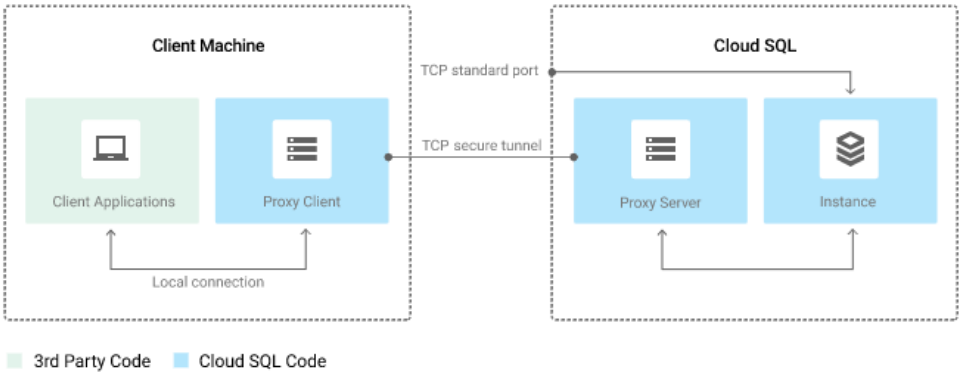


Figure 5. 4 Server proxy structure, from [30]

Another safe way is using SSL certificates which is a common way to keep sensitive or secretary information hidden by encrypting the data. The message is unreadable without the SSL keys [32]. Cloud SQL allows the client to create up to 10 client certificates to connect to the database instance. Client are required to download three key certificate and set them up on MySQL client [30].

Clients can also configure Cloud SQL with a public IP which allows authorised IP to connect to it. This is easier than connection for private IP Cloud SQL. Client's IP can be looked up by simply typing 'what's my public IP on Chrome. Authorise that IP address on CloudSQL, this impose to the problem that the public IP is keep changing as discussed in the literature review.

There are three solutions have been explored to resolve that issue.

1) Use the Cloud Proxy or SSL certificate
   Proxy client allows the secure connection to its proxy server which access the database instance. SSL on the other hand bypass any of that uses the authentication keys to verify the client identifies.
2) Allow for a CIDR (Classless inter-domain routing) range covers all client's provider IP address
   This would allow all IP address that with the same internet provider to access the database instance with the correct credentials.
3) Allow for all CIDR range that covers all internet providers
   This is done by authorizing IP address of: 0.0.0.0/0
   This is the least safe way and could be imposed to data breaching

Most connections on any database are followed similar concepts and knowledge applied here can help the future ski clients or students to connect to any other database.

The recommended way in this report after exploring most of the above methods is configure a public IP and using cloud proxy. For a guide on how to connect to the Cloud SQL refer to the Appendix A

5.6 Raspberry Pi writing data to database

To start, MySQL client package should be installed first please refer to Appendix B to see how to do it.
Once CloudSQL and MySQL package are set up, run MySQL on Raspberry pi user should see something like below:

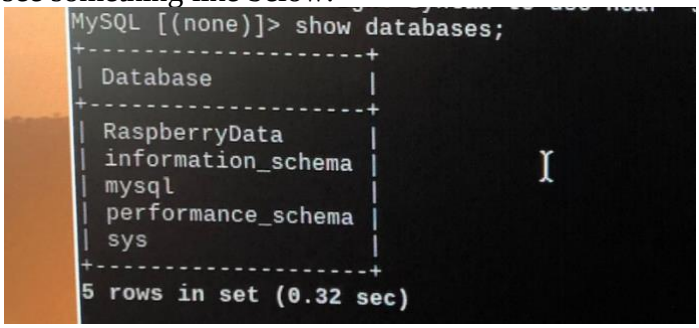


Figure 5. 5 MySQL Raspberry Pi connection

To write sensor data to the database include python library for MySQL by including

```
import mysql.connector
from mysql.connector import Error
```

Listing 5. 1MySQL python libraries

Use the library function to connect to the database and print out error messages

```
    try:
        connection = mysql.connector.connect(host='<DBpublicIP>',
                                             database='<databasename>',
                                             user='<username>',
                                             password='<yourpassword>')
    except Error as e:
        print('Connecction error')
        print(e)
```

Listing 5. 2 How to connection to MySQL database

To write or edit use call following method in sequence

```
cursor = cnx.cursor()
##Read Sensor data write to DB

#Execution
cursor.execute(add_longNlat,data_longNlat)

#Commit to changes
cnx.commit()

#Close connection
cursor.close()
cnx.close()
```

Listing 5. 3 How to write and edit on database

5.7 GPS live tracking

5.7.1 How GPS receiver work

The GPS receiver obtains signals from satellites when it receives at least 4 satellites it reaches a state called a lock or a fix [35]. Each satellite on board has an accurate atomic clock that is used to synchronise the time. This synchronised time along with the positional information are sent to the receiver via a radio frequency at 1.5 Ghz [36] it determines the position in three dimensions – east north and altitude [37].
There are three start states: cold, warm and hot. The time to receive the positional data various from the start state. In cold state, it takes the GPS receiver about 15 minutes to obtain the entire set of satellites data within its working group (around 12 other satellites) such group is called constellation and the data is called Almanac. With hot and warm start, it only requires getting a subset of the whole data which is called Ephemeris to work out the positional change [36] which only takes a comeback time at around 30 seconds.

5.7.2 NMEA sentence

Most receiver uses the serial transmit to transmit positional data to the microcontroller and in regular ASCII [35]. The GPS positional data conform with a format called NMEA 0813 which standards for National Marin Electronic Association uses a typical baud rate of 4800 ,9600 or 15600 protocol with a "$" sign at the start of the string [38] [35].
Two of the commonly used NMEA strings are $GPGGA and $GPRMC [35].

Figure 5. 6 GPGGA NMEA string format, from [35]

It gives information such as UTC time, latitude, longitude, how many satellites used to reach a fix, the HDOP which is an indication of accuracy, lower the better it stands for horizontal dilution of precision, altitude and the checksum for error handling [38].

5.7.3 NMEA sentence parsing on Raspberry Pi

5.7.3.1 Manual parsing

Knowing format manual parsing can be done by knowing the port of which the GPS receiver is connected to and its baud rate.

To figure out the baud rate, where GPS is connected and monitor its serial outputs use the following commands on Linux based system:

```
stty -F /dev/tty<portGPSConnected> #returns baud rate
ls /dev/tty*                       #returns a list with the prefix
cat /dev/tty<portGPSConnected>     #monitor the serial output
```

Listing 5. 4 Connect to GPS

First command returns the baud rate of GPS data transmission, second gives you the name list with the same prefix, plug and unplug the GPS to find out the port, and the last command monitor the serial outputs from the GPS. Manual method is mainly used when packages does not cover the attributes the user is looking for.
Follow a similar structure as Listing 5.5 to manually parse.

```
import serial
GPSport = "/dev/tty<PortGPSConnected>"
ser = serial.Serial(port, baudrate = 15200, timeout = 0.5)
data = ser.readline()

while True:
    if data[0:6] == "$GPRMC":
          GPSdata = data.split(",")
          lat = decode(GPSdata[3]) #Latitude
          dirLat = GPSdata[4]      #direction: N/S
          lon = decode(GPSdata[5]) #Longitute
          dirLon = GPSdata[6]      #direction: E/W
```

Listing 5. 5 Manual parsing NMEA

5.7.3.2 Packages

There are also dedicated packages that offers a robust way for NMEA sentence parsing and error handling one is called “pynmea2” it can parse the NMEA from a log file or parse as GPS reads. It has attributes such as timestamp, latitude, longitude and the HDOP [39].
Follow a similar structure in Listing 5.6 to parse using package

```
import serial
import pynmea2

port = "/dev/ttyUSB1"
ser = serial.Serial(port, baudrate = 15200, timeout = 0.5)
data = ser.readline().decode()

while True:
    if data[0:6] == "$GPRMC":
        msg = pynmea2.parse(data)
        lat = msg.latitude
        lng = msg.longitude
```

Listing 5.6 Pynmea parsing

5.8 GPS data transmit to Cloud Database

It's viable to write the GPS coordinates with time stamp and positional to the cloud database. Simply by combing code together covered above.

5.8.1 Result

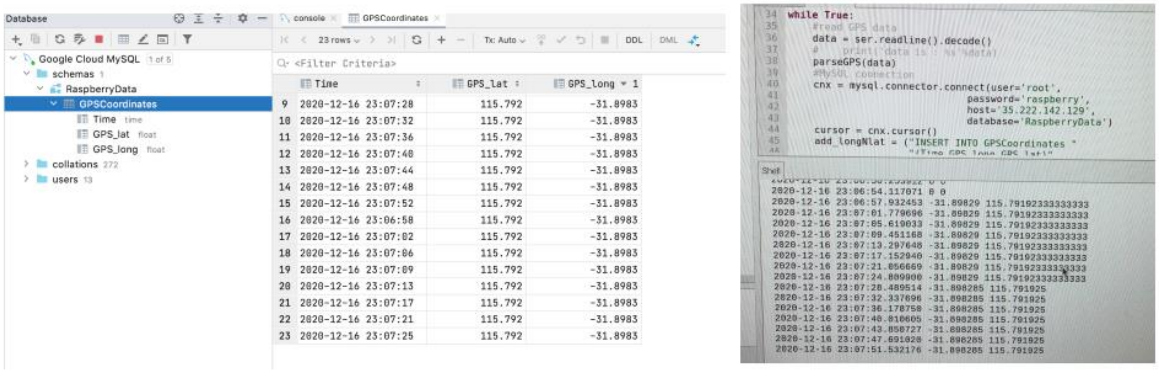


Figure 5. 7 Writing to MySQL on CloudSQL

The process can be implemented with transmitting any sensors data on the jet ski simply by swapping reading GPS reading sensor.
This provides a safe logging system for users to track any sensor data of the jet ski.
However, this is still a step away from monitoring position it in real time.

5.9 GPS live tracking real time

To achieve this PubNub has been used. PubNub is a real-time communication cloud network platform that provide real-time software framework for developers to build and scale their project [40]. Its APIs allow the developers to publish receive data, store, security control and real-time analytics. It has at least 15 data centres located globally to ensure reliable socket connections and support most common SDKs (software development kit) such as python and Java [40]. It also supports free accounts for low volume of data and 200 monthly active users which are suitable for the jet ski ioT (internet of things) applications, details about the free account can be found here [41].

5.9.1 PubNub ioT structure

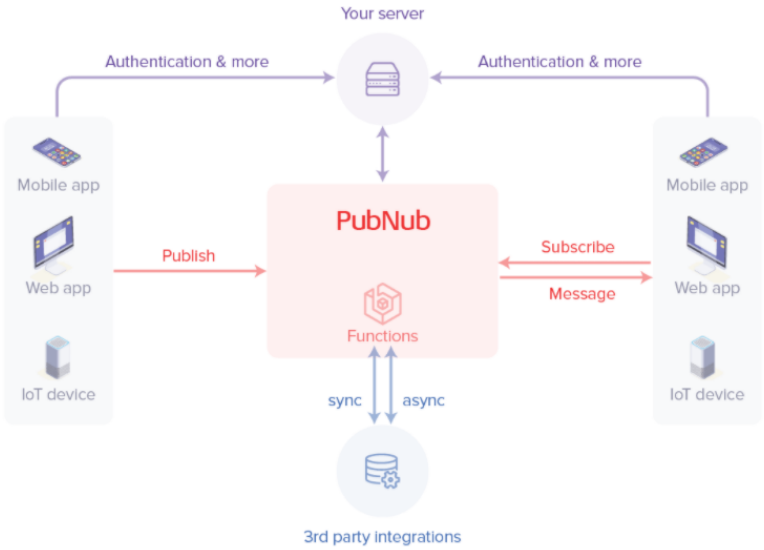


Figure 5. 8 PubNub cloud ioT communication, from [41]

The ioT structure can be seen on Figure 5.8. The objectives here is to create an ioT application that allow the raspberry pi to publish GPS coordinates to a webpage and displays it in real time.
PubNub has provided a helpful library that allows users to publish/subscribe with ease and it has included some open-source examples which formed the basis of this implementation [42].
Arijit Das has done a real time tracker project and created a useful button [54].
Start of the HTML code with including PubNub library, to connect to a PubNub channel an account needs to be registered first. Followed by creating a keyset for publishing and subscribing.

5.9.2 Implementation

The design process and architecture to implement a live tracking REV GPS is shown on Figure 5.9.

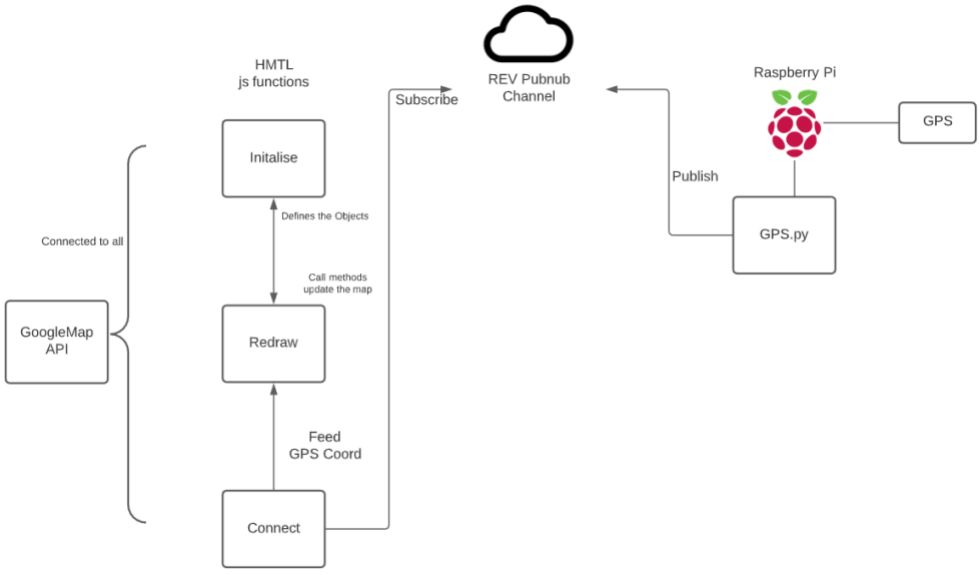


Figure 5. 9 GPS tracker architecture

The architecture is shown in Figure 5.9. First Google Map API need to be included at the very end of the code to ensure is accessible to all DOM (document object model). Conveniently, if user have used Cloud SQL account is already set up just need to enable the Google Map API.

The connection function subscribes to the PubNub channel and add the redraw function as a listener fetches the GPS coordinates published, the publisher is the Raspberry Pi.

Initialise function starts the map with an initialised latitude and longitude it places a marker on the map. It defines map and mark as global variables to hold the google.maps.Map and google.maps.Marker objects so they can be manipulated when the new position comes in. The method 'setCenter' and 'setPosition' under the object can be called to reposition the map.

The redraw function is the most important part of the code that fetches the latitude and longitude from the published source in this case the python code from Raspberry Pi.
It then passes the newly fetched 'lat' and 'lng' to Google Map API and reposition the map and reset the marker.

It finally draws a path by appending the new 'lat' and 'lng' to an array which was initialised as an empty array first.
Refer to Appendix B for a detailed guide and walk-through.

5.9.2.1 Raspberry pi GPS python

Include the library, established the connection and publish the data. Write a function like the following example listed in Listing 5.7.

```
def TransmitGPS(data):
    if data[0:6] == "$GPRMC":
        msg = pynmea2.parse(data)
        lng = msg.longitude
        try:
            envelope = pubnub.publish().channel(pnChannel).message({
            'lat':lat,
            'lng':lng
            }).sync()
            print("publish timetoken: %d" % envelope.result.timetoken)
        except PubNubException as e:
            handle_exception(e)
```

Listing 5. 7 Raspberry Pi transmit to PubNub channel

PubNub has also provided a guide in how to connect using python [53]. With the source code of the HTML file and access of the internet any device can run the application without a server. Simply by creating a HTML file and run it on browser where still possible to host on a server.

5.9.2.2 Results GPS live tracking

This provids an ability to track the jet ski's position in real time and allow user to trace back any historical position data when logging to database.
The HTML design is shown on Figure 5.10.

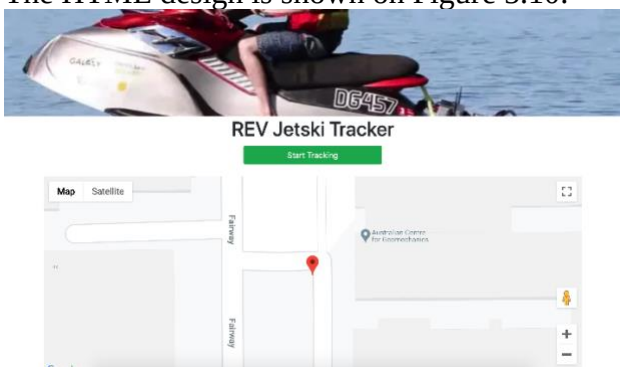


Figure 5.10 Jet ski tracker HTML page

As the jet ski is not ready for water test yet, a dry test has been done on Raspberry Pi and GPS. Result can be seen in Figure 5.11.

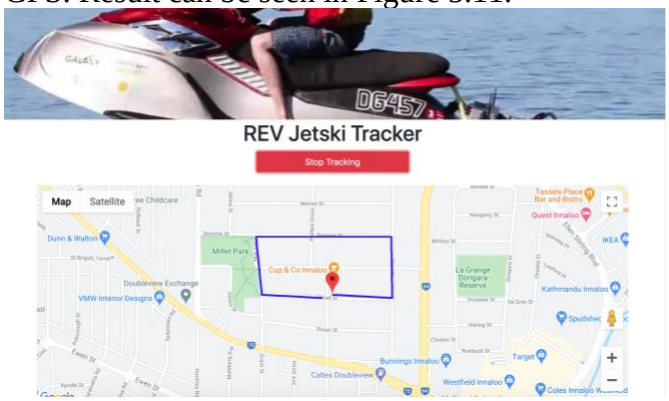


Figure 5.11Testing result

5.10 Data Visualisation

To view real time data and providing analytical trends are important for debugging and safety monitoring purposes. This project aims to concept proof and modularly test visualisations to minimise the time needed to implement when jet ski is ready for water test.
There will be two methods covered in this project one is using PubNub package called Eon another is using a local server on Raspberry pi by using telegraf and influxDB to develop dashboards on Grafana that are available to all local network device.

5.10.1 Dashboards PubNub Eon

Eon is an open-source software package PubNub has developed to allow real time visualisation of data and creating dashboards. It supports most of the popular chart types such as spline, bar, pie and gauge [43]. Both PubNub and Eon libraries need to be included in the HTML code. The design process and architecture will look like this in Figure 5.12.

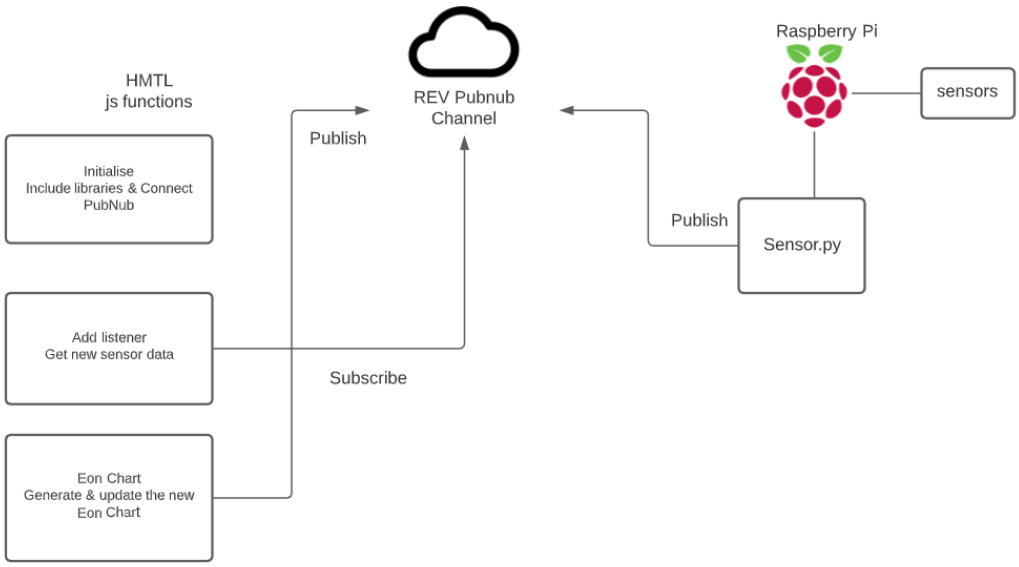


Figure 5. 12 Jet ski EON dashboard architure

Raspberry Pi takes the sensor readings from various source publish them to a dedicated channel for EON plots. On the HTML side, it listens to the channel gets the newly published data and generate the new EON chart and display it on the webpage.

5.10.2 Results EON chart

After setting all up, the results are as follow in Figure 5.13, here it has been tested with reading the speed only but replace the reading speed code with sensor reading code to visualise any other data

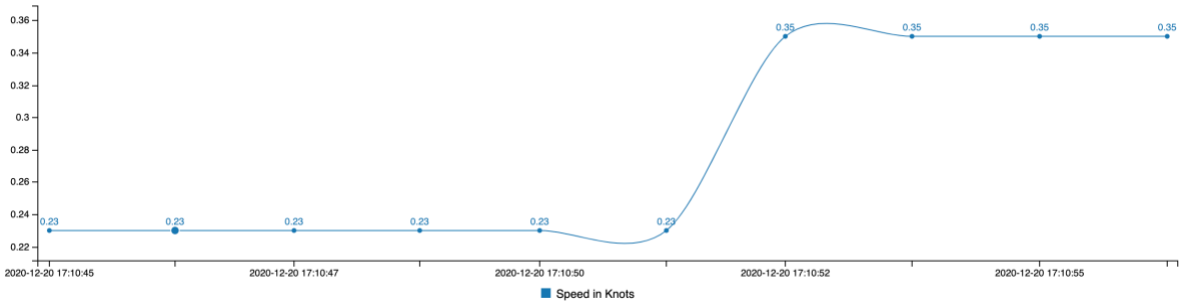


Figure 5. 13 Testing result EON chart

For a guide on how to set up please refer to Appendix C.

5.11 ioT applications local Raspberry Pi server

5.11.1 Grafana InfluxDB telegraf

This method allows the jetski to set up a local server on Raspberry Pi and create an influxDB locally. With the Wi-fi modem that have been purchased it creates LAN (local area networks) allow any devices to connect to it and display the dashboard. This method aimed to come up with alternatives for the display on the jet ski. Allow user to use any device such as tablets or iPad to monitor the crucial parameters just as what a display can do.
It also has the potential to be viewed remotely by using the MySQL on Cloud SQL.

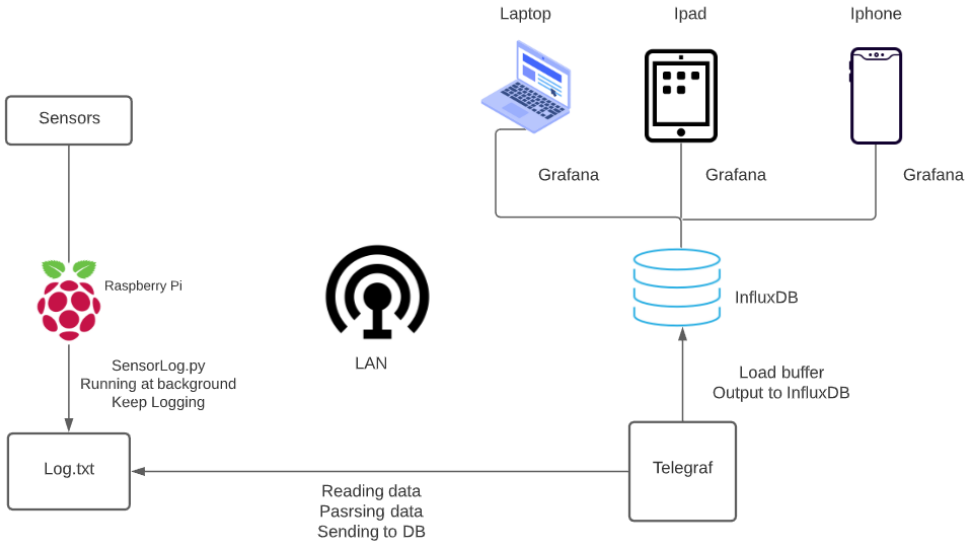


Figure 5. 14 Jet ski ioT Raspberry Pi local server architecture

5.11.2 Raspberry Pi side

The architecture and design process are shown in Figure 5.14. Raspberry Pi reads from various source of sensors and log them into a text file. The SensorLog.py keeps running in the background and consistently writing data to a text file.

5.11.3 Telegraf

Telegraf is the open-source server agent to collects metrics/data from sensors and distributing to various places in this case a database [44]. It supports databases like influxDB, MySQL and many others. In this case Telegraf will be configured to consistently read the log text file, parse the data and writing them to influxDB which host on Raspberry PI under LAN. To configure Telegraf is an important part there are many resources and prior of arts available. The github page on configuration provides many examples [45]. This method uses grok parser and it was covered in Kumar's project when he built a weather station to monitor temperature and humidity [46]

5.11.4 InfluxDB

InfluxDB is a time series database platform that empowers developers to build ioT applications to store and monitor time series data [47]. In this case it's used to store the data from sensors, and it is host on Raspberry Pi locally.
InfluxDB suits the jet ski application very well with all sensors data are collected with respect to time. It also commonly used in series with Telegraf and Grafana for ioT applications.

5.11.5 Grafana

Grafana is an open-source analytics and interactive virtualisation tool works very well with influxDB and compatible with most databases [48]. In this case Grafana will be configured to connect to the influxDB. Dashboards are designed to display the data on Grafana.
Every device under the LAN can access the Grafana through port 8086 by running the Raspberry Pi private IP follow by the port number.

As covered in literature review the DHCP server on LAN assign Raspberry pi with a different private IP this need be configured to the static private IP to work robustly.

5.11.6 Implementation

MySQL database on CloudSQL has potential to be used which would make everything accessible remotely and instead of connecting to the 'localhost' on Grafana it connects to the public IP address of the CloudSQL instance instead. This has not been implemented.

The method has been implemented reads the GPS speed by using the 'pymea2' that has been covered previously. It parses message under '$GPRMC' has an attribute to calculate speed in knots. This method implementation shall be used for all sensors when jet ski is ready to water test.

5.11.7 Results

Results for a speedometer dashboard is below on Figure 5.15. Waterproof tablet cases are readily available with typical one cost at $18.70 US dollars with 2 meters waterproof for 30 minutes immersed in water [49]. The device can be mounted on jet ski to replace the screen functions. Every other parameter such as battery remaining, water temperature and pressure, left and right motors rotational speed can all be design into panels and fit into this dashboard. Which functionally acts as an alternative for the display screen to saves the cost and in turn making the product more competitive in price. Alternatively, when the jet ski become commercialised it could have multiple plans with different price for clients to choose. Results can be seen below:

Figure 5. 15 Grafana speedometer result

For how to set up refer to Appendix E. For results showing the same dashboard on an iPad please refer to Appendix D

5.12 ioT sensors application MQTT

Another ioT application using similar set up is using MQTT (Message Queuing Telemetry Transport) which is a machine to machine (M2M) protocol used for communication and interaction between various devices over the internet [50].
Sensors are connected to a Wi-Fi enable microcontroller such as ESP32/ESP8266 which is a cost-effective Wi-Fi microchip with full TCP/IP stack and microcontroller capability it allows sensors to transmit data under the LAN [51]. The sensors connected with ESP32/ESP8266 act as MQTT publisher which will public the sensor data.
Raspberry pi will act as a MQTT broker through Mosquitto which is an open source message broker that implements the MQTT protocol it collects and message/data from the publisher and distributed to the data to the subscriber.
Raspberry Pi will also act as a subscriber subscribes to the various sensor topics this on jet ski could be sensors such as joystick sensors, steering sensors throttle sensors with different topics. It feed data to Telegraf which will distribute the message/data to the InfluxDB and connects to Grafana to display it.
This method will resolve the fact that currently all the sensors at the front and at the back need to run wires all the way front the front of the Jetski or the back of the jet ski.
The architecture of this application shown in Figure 5.16.

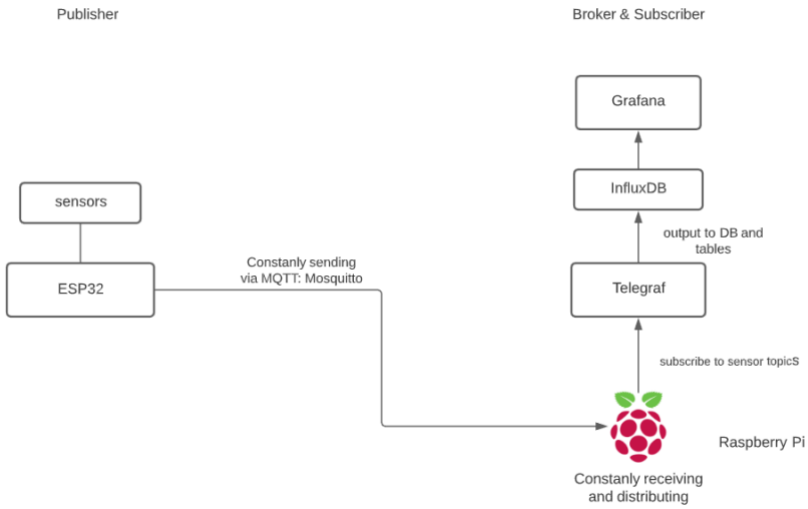


Figure 5. 16 MQTT publisher broker and subscriber architecture

Docker is a useful tool for such application it allows various applications to run seamlessly and simultaneously together [52]. It creates sufficient conditions for each application to run alone such environment is called a container. It streamlines the process and allows applications to be shuttled with ease.
However, with the time limit of this project this have not been fully explored nor implemented.

6.0 Overall results

The overall results are

1) Data logging system
   The data logging system can successfully log all sensors data which are depth sensors, Phidget IMU, Ardupilot, Bosch IMU, GPS this have been tested to log either onto csv or txt files
2) Data network
   The project has concept proof and modular tested the CAN data communication
3) Data transmission
   Data successfully transmitted remotely from Raspberry Pi. The CloudSQL method provide a method to store and manage database without managing a server. This allows jet ski to log any sensor data remotely.
4) GPS live tracking
   GPS live tracking have successfully been implemented on Raspberry Pi that allows the user to monitor the jet ski position in real time.
5) Dash visualization
   PubNub EON chart has tested with reading the speed this allow monitoring analytical data and trends in real time with any sensor data.
6) Raspberry Pi database ioT application
   This have been tested with a speedometer dashboard to visualize data on Grafana. It allows other device under LAN to access the same information.
   This project also covered a structure for MQTT applications that allows Wi-Fi controllers such as ESP32 to connect to sensors and transmit data over Wi-Fi.

7.0 Limitations

The limitations are as follow

1) Data logging system
   The data logging system have not been tested enough while there was only one water test at the very early stage
   Data Network
   The CAN bus has not been built yet it imposed limitations on exploring further with current method not been tested in real practice.
2) Data transmission, GPS live tracking, Visualization and Raspberry Pi database ioT application
   They faced the same limitation where currently it's only been partially tested which could impose to problems that cannot be unforeseen.

8.0 Future work

Future work can include into several areas

1) Water tests all data logging system, implement all sensors on telemetry and ioT applications
   More specifically, test the data logging systems on water tests, implement all sensor trends on PubNub Eon charts, implement all sensor on dashboards on Grafana
2) Improve the HTML design, UI design on Eon chart and dashboards on Grafana
   More specifically, improve on the css file, implement all sensors and
   improve the Eon charts UI and Grafana dashboards
3) Implement the ioT MQTT application
   Specifically, implementing the MQTT structure using dockers or alternative methods.

9.0 Conclusion

In conclusion, this project has covered and documented the methods, steps and results of the following areas.
Firstly, it has implemented data logging that allow all sensor to be logged on to csv files or txt file. It also covered how to write remotely onto CloudSQL MySQL database instance and tested to work.
Secondly, it has modular tested the CAN network in its writing and receiving ability by using a serial to CAN module prove that jet ski microcontroller now have the ability to write to the CAN bus.
Thirdly, the GPS live tracking implemented used a structure that use Raspbery Pi as a publisher and PubNub cloud server as a subscriber and implemented a tacking in real time application and it has been dry tested to work.
Moreover, the ioT application has developed ways to visualise data using PubNub Eon that subscribes to the channel of which the Raspberry Pi is posting the sensor data to and display and trending in real time.
Additionally, the ioT application on local server has developed a way to use Raspberry Pi consitantly reading the sensor data and populate to a text file, Telegraf constantly reading and parsing the log file and distributing it to a InfluxDB and Grafana connects to the InfluxDB and displays the data onto dashboards while other devices can visualise at the same time. The speedometer has been successfully tested to work as an example.
Lastly, this report has documented associated resources, background knowledge and literature review for the future students or clients to gain the understanding with spending less time.

The project generally has been imposed to limitations on the fact that the jet ski is not ready for water tests but under everyone's effort is start moving towards a positive direction. COVID has hindered the process by quite a lot as REV members were not able to go to labs for more than two months.
Above all, this project has outlined future work areas for future student to carry on.

10.0 Resources

Resource	URL
Source code	https://github.com/jeremyguo7/REVElfoil_jer

11.0 Reference list

[1] Woloszyn, M., 2018. *Instrumentation For The Revski; An Electric Personal Watercraft*. Master. The University of Western Australia.

[2] Pham, M., 2015. *Renewable Energy Vehicles' User Interface*. The University of Western Australia.

[3] White, R., 2013. *Electric Jet Ski*. Undergraduate. The University of Western Australia.

[4] Burden, T., 2016. *Selecting Navigational Instruments*. West Marine.

[5] n.d. *Raspberry PI 3 Model B*. [ebook] Available at: <https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf> [Accessed 16 December 2020].

[6] Seeedstudio. 2020. *8-Channel 12-Bit ADC For Raspberry Pi (STM32F030)*. [online] Available at: <https://wiki.seeedstudio.com/8-Channel_12-Bit_ADC_for_Raspberry_Pi-STM32F030/> [Accessed 16 December 2020].

[7] Seeedstudio. 2020. *USB To CAN Analyzer Adapter With USB Cable*. [online] Available at: <https://www.seeedstudio.com/USB-CAN-Analyzer-p-2888.html> [Accessed 16 December 2020].

[8] Seeedstudio. n.d. *CAN-BUS Shield V2.0*. [online] Available at: <https://wiki.seeedstudio.com/CAN-BUS_Shield_V2.0/> [Accessed 16 December 2020].

[9] Longan Docs. n.d. *Serial CAN Bus Module*. [online] Available at: <https://docs.longan-labs.cc/1030001/> [Accessed 16 December 2020].

[10] Waveshare. 2020. *SIM7600CE-T/E-H/A-H/SA-H/G-H 4G Modules*. [online] Available at: <https://www.waveshare.com/wiki/SIM7600E-H_4G_HAT> [Accessed 17 December 2020].

[11] Manualslib. 2012. *Columbus V-800+ Manuals*. [online] Available at: <https://www.manualslib.com/products/Columbus-V-800Plus-3695109.html> [Accessed 16 December 2020].

[12] Huawei. 2020. *E8372 Specifications*. [online] Available at: <https://consumer.huawei.com/au/routers/e8372/specs/> [Accessed 16 December 2020].

[13] Leong, D., 2019. *Revski Data Transmission, Storage, And Visualisation*. The University of Western Australia.

[14] Campbell, S., n.d. *Basics Of The I2C Communication Protocol*. [online] Circuit Basics. Available at: <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#:~:text=I2C%20is%20a%20serial%20communication,always%20controlled%20by%20the%20master.> [Accessed 15 December 2020].

[15] 2019. *Raspberry Pi Compute Module 3+ & 3+ Lite*. 1st ed. Raspberry Pi.

[16] Kvaser. n.d. *The CAN Protocol Tutorial*. [online] Available at: <https://www.kvaser.com/can-protocol-tutorial/> [Accessed 15 December 2020].

[17] Goldworthy, A., 2018. *Remote Control Of Autonomous Surface Vessels*. Master. The University of Western Australia.

[18] GeeksforGeeks. 2020. *Layers Of OSI Model - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/layers-of-osi-model/#:~:text=Transport%20layer%20provides%20services%20to,is%20referred%20to%20as%20Segments.&text=The%20transport%20layer%20also%20provides,if%20an%20error%20is%20found.> [Accessed 15 December 2020].

[19] Thomas, R., 2020. *Architectural Styles And The Design Of Network-Based Software Architectures*. Ph.D. University of California.

[20] MuleSoft. 2020. *What Is An API? (Application Programming Interface)*. [online] Available at: <https://www.mulesoft.com/resources/api/what-is-an-api> [Accessed 15 December 2020].

[21] Rouse, M., 2020. *Restful APR (REST API)*. [online] SearchAppArchitecture. Available at: <https://searchapparchitecture.techtarget.com/definition/RESTful-API> [Accessed 14 December 2020].

[22] Altvater, A., 2017. *What Are CRUD Operations? Examples, Tutorials & More*. [online] Stackify. Available at: <https://stackify.com/what-are-crud-operations/> [Accessed 14 December 2020].

[23] Teltonika. 2020. *Private And Public IP Addresses*. [online] Available at: <https://wiki.teltonika-networks.com/view/Private_and_Public_IP_Addresses> [Accessed 16 December 2020].

[24] IONOS. 2019. *How To Set A Raspberry Pi With A Static Ip Address?*. [online] Available at: <https://www.ionos.com/digitalguide/server/configuration/provide-raspberry-pi-with-a-static-ip-address/> [Accessed 16 December 2020].

[25] Microsoft Azure. 2020. *Public IP Address Pricing*. [online] Available at: <https://azure.microsoft.com/en-au/pricing/details/ip-addresses/#:~:text=All%20instance%2Dlevel%20public%20IP,are%20charged%20at%20%240.0055%2Fhour.> [Accessed 16 December 2020].

[26] Simcom.com. 2020. *Simcom Wireless Solutions - Wireless Modules And Solutions Supplier*. [online] Available at: <https://www.simcom.com/> [Accessed 15 December 2020].

[27] Simcom. n.d. *SIM7600X*. [online] Available at: <https://www.simcom.com/product/SIM7600X.html> [Accessed 16 December 2020].

[28] Google Cloud. 2020. *Cloud SQL*. [online] Available at: <https://cloud.google.com/sql> [Accessed 15 December 2020].

[29] Google Cloud. 2020. *VPC Overview*. [online] Available at: <https://cloud.google.com/vpc/docs/overview> [Accessed 15 December 2020].

[30] Google Cloud. 2020. *Connecting Overview*. [online] Available at: <https://cloud.google.com/sql/docs/mysql/connect-overview> [Accessed 15 December 2020].

[31] Google Cloud. 2020. *About The Cloud SQL Proxy*. [online] Available at: <https://cloud.google.com/sql/docs/mysql/sql-proxy#linux-64-bit> [Accessed 15 December 2020].

[32] Sslshopper. 2020. *Why SSL? The Purpose Of Using SSL Certificates*. [online] Available at: <https://www.sslshopper.com/why-ssl-the-purpose-of-using-ssl-certificates.html#:~:text=The%20primary%20reason%20why%20SSL,intended%20recipient%20can%20access%20it.&text=When%20an%20SSL%20certificate%20is,are%20sending%20the%20information%20to.> [Accessed 15 December 2020].

[33] Google Cloud. 2020. *Configuring SSL/TLS Certificates*. [online] Available at: <https://cloud.google.com/sql/docs/mysql/configure-ssl-instance#new-client> [Accessed 15 December 2020].

[34] WhatIsMyIPAddress. 2020. *What Is CIDR Notation*. [online] Available at: <https://whatismyipaddress.com/cidr#:~:text=Classless%20inter%2Ddomain%20routing%20(CIDR)%20is%20a%20set%20of,be%20sent%20to%20specific%20computers.> [Accessed 15 December 2020].

[35] Maptoaster. n.d. *How GPS Works*. [online] Available at: <https://www.maptoaster.com/maptoaster-topo-nz/articles/how-gps-works/how-gps-works.html#:~:text=The%20GPS%20receiver%20gets%20a,time%20the%20signals%20are%20sent.&text=So%20given%20the%20travel%20time,%2D%20east%2C%20north%20and%20altitude.> [Accessed 15 December 2020].

[36] Core Electronics. 2018. *How GPS Receivers Work*. [online] Available at: <https://core-electronics.com.au/tutorials/how-GPS-works.html> [Accessed 16 December 2020].

[37] Instructables circuits. n.d. *Intro To GPS With Microcontrollers*. [online] Available at: <https://www.instructables.com/Intro-to-GPS-with-Microcontrollers/> [Accessed 15 December 2020].

[38] Baddeley, G., 2001. *GPS - NMEA Sentence Information*. [online] Available at: <http://aprs.gids.nl/nmea/> [Accessed 15 December 2020].

[39] Lewis, B., 2020. GitHub repository, https://github.com/Knio/pynmea2.

[40] Pubnub. n.d. *Documentation*. [online] Available at: <https://www.pubnub.com/docs/platform/home> [Accessed 16 December 2020].

[41] PubNub. 2020. *Simple Pubnub Pricing To Help You Start Building*. [online] Available at: <https://www.pubnub.com/pricing/> [Accessed 16 December 2020].

[42] PubNub. 2019. *Flight Paths – Javascript Geolocation Tracking With Google Maps API (4/4)*. [online] Available at: <https://www.pubnub.com/blog/javascript-google-maps-api-flight-paths/> [Accessed 16 December 2020].

[43] PubNub. n.d. *An Open-Source Chart And Map Framework For Realtime Data.*. [online] Available at: <https://www.pubnub.com/developers/eon/> [Accessed 15 December 2020].

[44] InfluxData. 2020. *Telegraf*. [online] Available at: <https://www.influxdata.com/time-series-platform/telegraf/> [Accessed 15 December 2020].

[45] Soroka, S., 2020. *Telegraf / Docs / Configuration*. GitHub repository, https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md.

[46] Surprenant, C., 2014. *Logstash / Patterns / Grok-Patterns*. GitHub repository, https://github.com/elastic/logstash/blob/v1.4.2/patterns/grok-patterns.

[47] InfluxData. 2020. *Influxdb*. [online] Available at: <https://www.influxdata.com/> [Accessed 15 December 2020].

[48] Grafana Labs. 2020. [online] Available at: <https://grafana.com/> [Accessed 15 December 2020].

[49] alibaba. 2020. *Aicoo Waterproof Ip69k Full Cover Tablet Case For Ipad 10.2 Shockproof Cover With Kickstand And Hand Strap With Retail Box*. [online] Available at: <https://www.alibaba.com/product-detail/AICOO-Waterproof-IP69K-Full-Cover-Tablet_62425346989.html?spm=a2700.galleryofferlist.normal_offer.d_title.688b575ee5VbrP> [Accessed 13 December 2020].

[50] DIYI0T. 2020. *MQTT Tutorial For Arduino, ESP8266 And ESP32*. [online] Available at: <https://diyi0t.com/introduction-into-mqtt/> [Accessed 15 December 2020].

[51] Espressif. 2020. *ESP32*. [online] Available at: <https://www.espressif.com/en/products/socs/esp32> [Accessed 15 December 2020].

[52] Docker. 2020. *Docker*. [online] Available at: <https://www.docker.com/> [Accessed 15 December 2020].

[53] PubNub. n.d. *Python V4 Publish & Subscribe API Reference For Realtime Apps*. [online] Available at: <https://www.pubnub.com/docs/python/api-reference-publish-and-subscribe#subscribe_example_1> [Accessed 16 December 2020].

[54] Das, A., 2019. Make A Realtime GPS Tracker Device With Raspberry Pi. [online] SPARKLERS : We Are The Makers. Available at: <https://sparklers-the-makers.github.io/blog/robotics/realtime-gps-tracker-with-raspberry-pi/> [Accessed 16 December 2020].

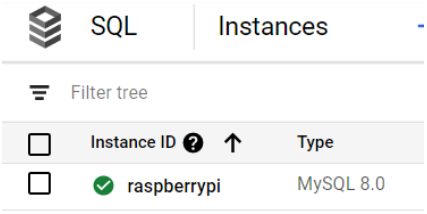
11.0 Appendix

11.1 Appendix A

This step by step guide will show how to connect using cloud proxy which is the recommended way for safety. And will also show how to add authorized network if using public IP address for the Cloud SQL instance.

Step 1 Login
Login into Google Cloud Platform first, register if needed;
Go to the top left corner click on the navigation menu, scroll down to SQL and click on it;

Step 2 Create instance
Create a data base instance with choice of database options include MySQL, PostgreSQL, SQL Server;
UWA Rev team uses MySQL;
Create instance ID, root password and version of Database;
Once complete navigate back user should see the instance just created would be something like this:



Step 3 Connection method
As discussed, configure the database instance with private or public IP Private IP is safer but harder to connect;
For raspberry pi connection via Cloud SQL please refer to a Stack Overflow post which can be found here [Reference?]; https://stackoverflow.com/questions/47267097/running-google-cloud-sql-proxy-on-raspberry
Once it's installed go the directory and run commands:
./cloud_sql_proxy -instances=<INSTANCE_CONNECTION_NAME>=tcp:3306

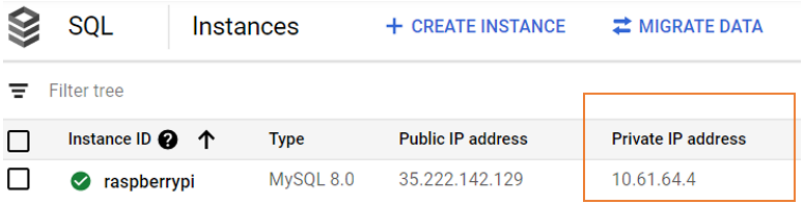
Instance connection name should conform the format of:
myproject:myregion:myinstance

Install MySQL client package or server package which would cover client package also by running command:
sudo apt install mariadb-server

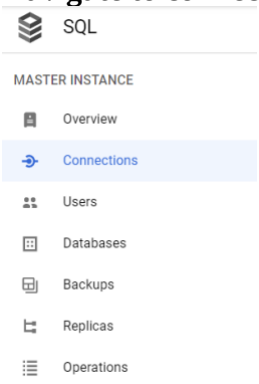
sudo mysql_secure_installation

Step 5 Private IP instance
By now the database connection should be successfully established. For private instance IP user need to enable the Virtual Private Cloud (VPC)

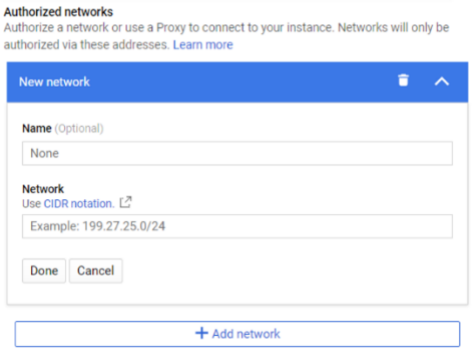
After it's been set up client should be able to see the private IP address as something like follows:



Step 6 Add authorised network, Public instance access
Navigate to connection once accessed the SQL from the main page



Click on connections



Navigate to Authorized network add IP address and assign name to the name as choice. This allows access publicly.

Once connection has been established on Raspberry pi run the commands:
mysql --host=[INSTANCE_IP] --user=root --password

Connection should be successfully established at this stage.

11.2 Appendix B Live tracking

Include the Gooogle API at the very end of the code

```
<script src="https://maps.goog-
leapis.com/maps/api/js?v=3.exp&key=YOUR_GOOGLE_MAPS_API_KEY&callback=ini-
tialize"></script>
  </body>
</html>
```

Connect to PubNub Channel, add the redraw function as a listener

```
var pnChannel = "<yourchannelName>";
var pubnub = new PubNub({
  publishKey:   'YOUR_PUB_KEY',
  subscribeKey: 'YOUR_SUB_KEY'
});
pubnub.subscribe({channels: [pnChannel]});
pubnub.addListener({message:redraw});
```

Initialise function

This initialise the start position and hold '.Map' and '.Marker' objects so they can be manipulated later to update the map

```
window.lat = <initialised lat>;
window.lng = <initlised lng>;
var initialize = function() {
  map  = new google.maps.Map(document.getElementById('map-canvas'),
{center:{lat:lat,lng:lng},zoom:12});
  mark = new google.maps.Marker({position:{lat:lat, lng:lng},
map:map});
};
```

Redraw function

Fetches the latitude and longitude by

```
  lat = payload.message.lat;
  lng = payload.message.lng;
```

Passes the 'lat' and 'lng' to Google Map API and update the map by:

```
  map.setCenter({lat:lat, lng:lng, alt:0});
  mark.setPosition({lat:lat, lng:lng, alt:0});
```

Draw a new path by initialising an empty array first

```
  var lineCoords = [];
  lineCoords.push(new google.maps.LatLng(lat, lng));
```

11.3 Appendix C EON set up and code walk through

Start of including both libraries of EON and PubNub

```
<script src="https://cdn.pubnub.com/sdk/javascript/pubnub.4.21.2.js"></script>
<script type="text/javascript" src="https://pubnub.github.io/eon/v/eon/1.0.0/eon.js"></script>
```

Include the keys to connect to PubNub

```
pubnub = new PubNub({
  publishKey : '<yourkey>',
  subscribeKey : '<yourkey>'
})
```

Generate a new EON curve by replacing the channel name with choice with desire, this does not need to create prior but must be consistent with Raspberry Pi python publish code. Make sure they are addressing the same channel

```
  eon.chart({
    debug: true,
    pubnub: pubnub,
    channels: ['<YourEonChartChannel>'],
    generate: {
      bindto: '#chart',
      data: {
        labels: true,
        type: '<ChartType>'
      },
      tooltip: {
          show: false
      }
    }
  });
```

On Raspberry Pi side

read the sensors first which will not be covered here

Import the necessary libraries by:

```
from pubnub.pnconfiguration import PNConfiguration
from pubnub.pubnub import PubNub
from pubnub.exceptions import PubNubException
```

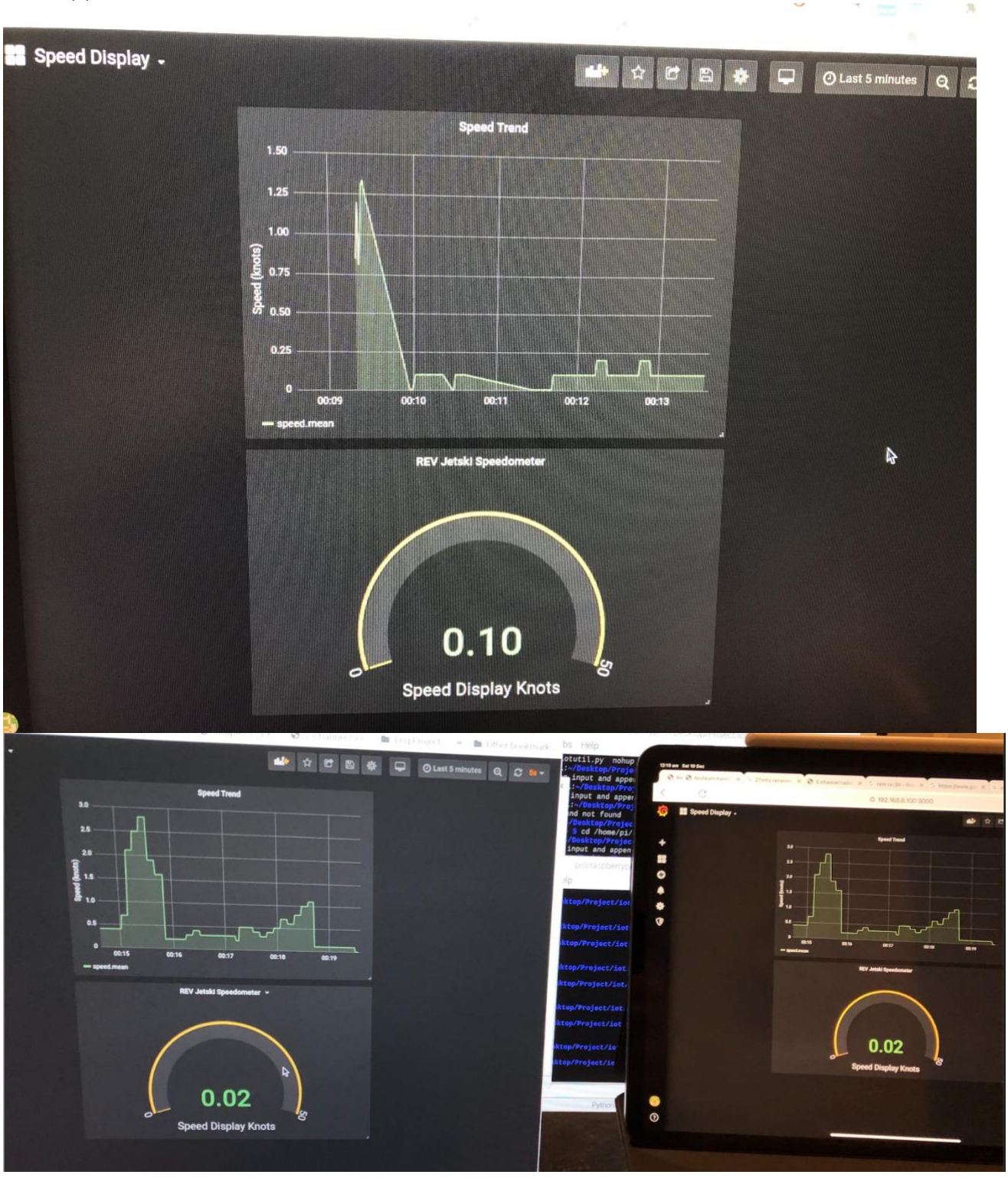
Connect to the same channel use user's own subscribe key and publish key

```
pnChannel = "<YourChannelSameNameInHTMLfile>";
pnconfig = PNConfiguration()
pnconfig.subscribe_key = "<YourKey>"
pnconfig.publish_key = "<YourKey>"
pnconfig.ssl = False
pubnub = PubNub(pnconfig)
```

Publish the sensor data to the channel

```
while True:
###################
#Call sensor reading function here
###################
    data = {"eon": {"<sensor1>": <variableForS1>, "<sensor2>": <variableForS2>}}
    pubnub.publish().channel(pnChannel).message(data).sync()
```

11.4 Appendix D results for dashboard tablet



11.5 Appendix E Telegraf Grafana InfluxDB

1. Install all packages, Grafana, InfluxDB and Telegraf
2. Download the code from github under ioT Grafana dashboard
3. Run the following code once everything is set up to start the application

```
nohup python <pythoncode>.py
telegraf --config <configfile>.conf &
sudo strart service grafana-server start
```

The commands does the following:

- 'nohup' runs the python code in the background,
- the'telegraf' command use the configuration file to configure Telegraf.
- Start the Grafana server on port 3000.