zet1n22@soton.ac.uk (33398151)

Summer Internship Report

Prepared by: Zhi En Tee

Week 1 (1/7-5/7)

- Familiarise with **PX4**, **QGroundControl (QGC)**, **MAVLink**, **MAVLink Router**, **MAVROS and Wireshark**
- Setup **ZeroTier connection.**

Details:

Familiarise with PX4, QGroundControl (QGC), MAVLink, MAVLink Router, MAVROS and Wireshark

PX4 (Autopilot Software)
PX4 is an open-source flight control software widely used for drones and other unmanned vehicles. It provides functionalities like flight control, sensor integration, and mission planning. It acts as the core flight controller, managing the drone's flight operations.

QGroundControl (QGC)
QGC is a ground control station (GCS) software that provides a graphical interface for managing drones running PX4. It provides a user-friendly interface for mission planning and real-time monitoring.

MAVLink (Micro Air Vehicle Link)
MAVLink is a communication protocol between drones and ground stations or other components. The key MAVLink message types such as HEARTBEAT could be focused on connection status.

MAVLink Router
MAVLink Router is a tool that routes MAVLink messages between different endpoints, such as QGC and onboard computers or other networked systems. It ensures that multiple systems receive the data they need to operate in parallel. The data could be verified if routed correctly using tools like **Wireshark** to inspect the MAVLink message flow.

MAVROS (MAVLink to ROS Interface)
MAVROS is a ROS (Robot Operating System) package that allows the integration of drones running PX4 with the ROS ecosystem. It enables the integration of the drone into the **ROS** ecosystem for autonomous behaviours and advanced control algorithms.

Wireshark
Wireshark is a widely used, open-source network protocol analyser that can capture, inspect, and analyse network traffic in real time. It provides detailed visibility into the data packets traveling across a network, helping users understand how network communication works, troubleshoot network issues, and inspect security vulnerabilities.

ZeroTier Connection Setup

ZeroTier is a software-defined networking (SDN) solution that could create secure, encrypted virtual networks over the internet, effectively simulating a local network. It connects devices (computers, smartphones, servers, etc.) across the globe as if they were all on the **same local network (LAN)**, making it easy to share resources, communicate, or remotely access systems.

ZeroTier simplifies networking by **creating a virtual network** overlay that connects devices across different physical networks securely and easily. It is widely used for **remote access,** IoT projects, gaming, and creating secure, private networks without the need for traditional VPNs or advanced networking knowledge.

Steps:
On Windows

- Sign in to ZeroTier. (https://www.zerotier.com/)
- Download **ZeroTier 1.6.6** for Windows 11. (https://download.zerotier.com/RELEASES/1.6.6/dist/ZeroTier%20One.msi)
- Sign in to ZeroTier Central. (https://my.zerotier.com/)
- Create a network and copy the network ID.
- Find the ZeroTier One icon by clicking the arrow on the status bar on the window's bottom right.
- Select "**Join a Network**".
- Paste the network ID in and click "**Join**"

On Jetson Terminal (Ubuntu)

- Run the **ubuntuzerotiersetup.sh** by changing the **9e1948db6317a4d2 to network ID** in the ZeroTier website network created.

Connection

- Disconnected all wifi and open a command prompt and type "**ipconfig**" for Windows or "**ifconfig**" for Ubuntu/Linux.
- Run the **start-mav-route.sh** by changing the IP address to the **QGC device IP address** without using Wifi.

ubuntuzerotiersetup.sh

```
# On Jetson Terminal
sudo apt-get install gnupg
curl -s 'https://raw.githubusercontent.com/zerotier/ZeroTierOne/main/doc/contact%40zerotier.com.gpg' | gpg --import && \
if z=$(curl -s 'https://install.zerotier.com/' | gpg); then echo "$z" | sudo bash; fi
sudo zerotier-cli join 9e1948db6317a4d2
sudo zerotier-cli status
sudo zerotier-cli listnetworks
```

start-mav-route.sh

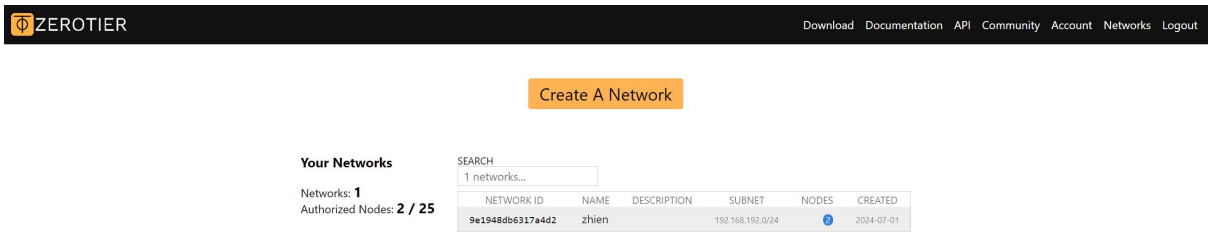
```
#!/bin/bash

#Run heartbeat_sender.py first before using mavlink routing
python3 heartbeat-sender.py /dev/ttyACM0 &
mavlink-routerd /dev/ttyACM0 -e 127.0.0.1:14550 -e 192.168.192.227:14550 -r &
wait
```

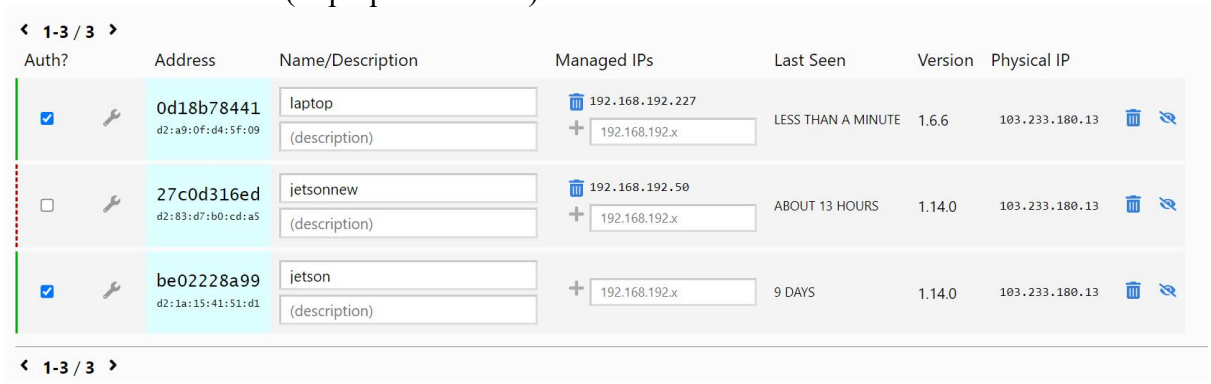
zet1n22@soton.ac.uk (33398151)

ZeroTier Central

Network ID



Authorized Members (Laptop and Jetson)



ipconfig on Laptop (QGC)

```
Ethernet adapter ZeroTier One [9e1948db6317a4d2]:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::20c8:cb54:7a21:6566%48
   IPv4 Address. . . . . . . . . . . : 192.168.192.227
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 25.255.255.254
```

This IPv4 address **(192.168.192.227)** is used in the **start-mav-route.sh** script as the destination for routing MAVLink messages to QGroundControl, bypassing WiFi.

zet1n22@soton.ac.uk (33398151)

Week 2 (8/7-12/7)

- Familiarise with **LoRa, LoRaWAN, Wisgate OS, WisDM, TTN.**
- Understand **LoRaWAN Network Architecture**
- **Extract data from TTN** by creating a NodeJS server via MQTT integration API and display it in Git Bash Terminal without successfully forward to QGC.

Details:

Familiarise with LoRa, LoRaWAN, Wisgate OS, WisDM, TTN

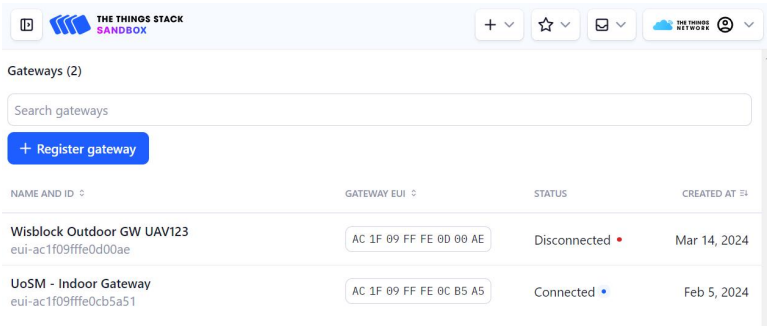
LoRa (Long Range)
LoRa is a wireless modulation technique that uses chirp spread spectrum (CSS) technology to provide long-range communication. It is designed to transmit small amounts of data over long distances with very low power consumption, making it ideal for battery powered IoT devices.

LoRaWAN (Long Range Wide Area Network)
LoRaWAN is the protocol layer that defines how devices communicate over LoRa networks. It adds networking features like device addressing, secure communication, and adaptive data rates to make LoRa scalable for large IoT deployments. Devices send data to a LoRaWAN gateway, which forwards the data to a network server. The network server manages the connections, security, and data routing.

WisGate OS
WisGate OS is the operating system used by RAKwireless gateways for managing LoRaWAN network infrastructure. It provides the tools and configuration options necessary to operate the gateway, integrate it with network servers, and handle communication between devices.

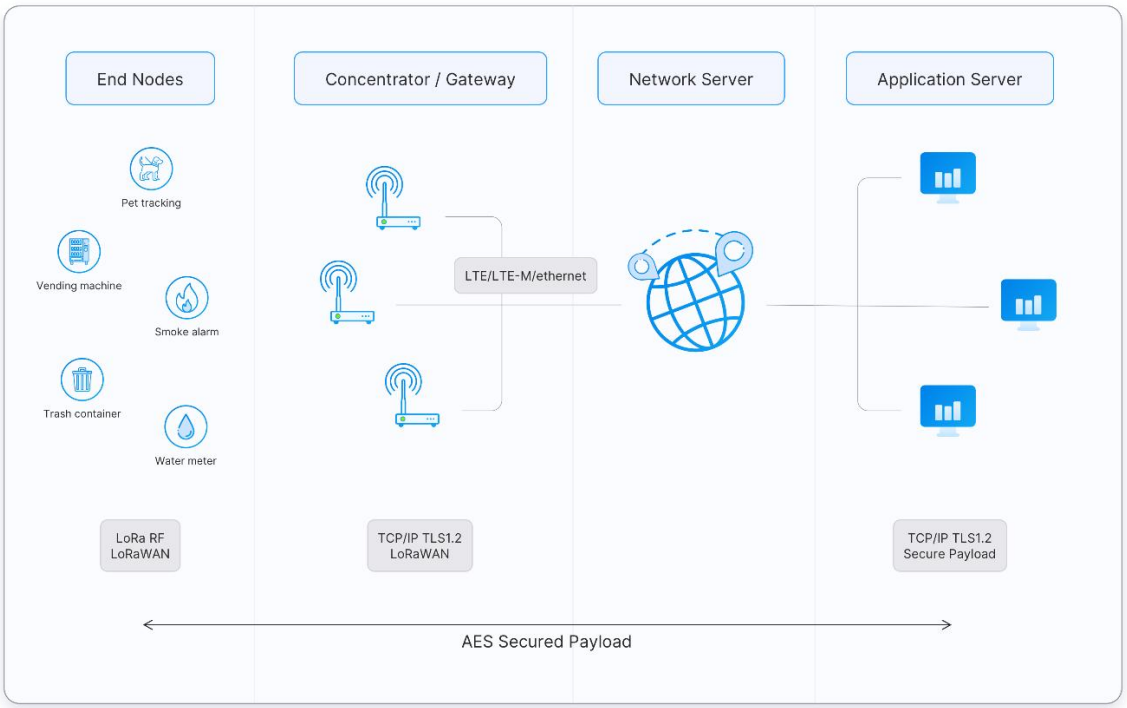
WisDM (WisGate Device Management)
WisDM is a cloud-based platform by RAKwireless for managing multiple gateways remotely. It allows users to monitor, configure, and update their LoRaWAN gateways from a central location.

The Things Network (TTN)
TTN is a global open-source infrastructure for building and managing LoRaWAN networks. It provides a network server and various tools to connect LoRaWAN-enabled devices and manage the flow of data to applications.
Website:
https://au1.cloud.thethings.network/console/gateways



LoRaWAN Network Architecture

LoRaWAN follows a **star-of-stars topology**, where end devices communicate with gateways, which in turn connect to a network server over a backhaul (typically IP-based). The architecture can be broken down into the following components:



End Devices (Nodes)

- These are the **LoRa-enabled devices** (sensors, actuators) that gather and transmit data using the **LoRa** modulation technique.
- End devices typically use very low power and can remain operational for long periods on battery power.
- These devices communicate with multiple gateways using **LoRaWAN**, which handles the communication protocol and ensures secure, encrypted transmission.
- End devices communicate wirelessly with the gateways running **WisGate OS**, which forwards the data to **TTN** for further processing.
- **WisDM** can be used to manage the gateways that receive data from these end devices.

Gateways

- Gateways serve as a bridge between the end devices and the network server.
- They receive LoRa signals from the end devices and forward them to a **LoRaWAN Network Server** using a standard IP connection (e.g., Ethernet, cellular, or Wi-Fi).
- A single gateway can handle thousands of devices and typically covers a large geographic area.
- The gateways run **WisGate OS**, which manages the data flow from end devices to network servers.
- These gateways can be managed remotely via **WisDM**, which can monitor their performance, update firmware, and resolve issues without being physically present.
- Gateways forward the data to a **network server**, such as **The Things Network (TTN)**, for processing.

LoRaWAN Network Server (LNS)

- The network server is a critical component of the architecture, as it manages all the communication between the gateways and the application servers.
- It handles:
  a. **Device Management**: Keeps track of which devices are connected to the network.
  b. **Routing**: Ensures the data packets received from gateways are routed to the correct application servers.
  c. **Security**: Implements encryption and ensures that only authorized devices are communicating on the network.
  d. **De-duplication**: If a device's data is received by multiple gateways, the network server eliminates duplicate packets.
- **TTN** acts as a **LoRaWAN Network Server** in many LoRaWAN deployments. It receives data from gateways and handles all the backend tasks such as device registration, routing, and security.
- **WisGate OS** helps to connect the gateways to **TTN** or any other network server, ensuring smooth operation of the LoRaWAN network.

Application Servers

- The application server is where the data from the end devices is delivered, processed, and used by applications or services.
- After the data is routed by the network server, it is forwarded to an application server that performs functions such as:
  a. Storing the data in a database
  b. Visualizing the data in dashboards
  c. Sending alerts or triggering actions based on the received data (e.g., turning on an irrigation system based on soil moisture data)
- **TTN** facilitates this data forwarding, allowing users to integrate their LoRaWAN data with services like **MQTT, Webhooks, or HTTP APIs**.
- Data collected from devices (like temperature, location, etc.) is processed and visualized in dashboards such as **Node-RED** or can be integrated into larger applications.

zet1n22@soton.ac.uk (33398151)

Extract Data From TTN

Create a **NodeJS server** to interact with the **TTN MQTT Integration API** to extract device data.

Reference:

https://youtu.be/C32umTkMkb0?si=Y-E4ae7xnkoDsDez
https://youtu.be/hxIKyFAgZCE?si=GyVj3Lt0k6Q4Digq

Server.js (partial code)

The server.js script was used to **connect to the TTN MQTT broker, subscribe to topics and receive data from TTN devices.** The data was displayed in Git Bash Terminal as the payloads were successfully retrieved from TTN.

server.js | start-mav-route.sh | udp_heartbeat.py | Print.h C:\...\arduino | Print.h C:\...\api | eui-ac1f09fffe0cb5a51_live_data_1720081126682.json

TTN_OWN_SERVER > server.js > client.on('message') callback > getDataFromTTN

```
var express = require("express");
var app = express();
app.use(express.static("public")); //Procedure to set public folder contain process files (css,image,.js,..)
app.set('view engine', 'ejs'); // Use ejs instead of html
app.set("views","./views"); // view folder contain .ejs files

// Parse URL-encoded bodies (as sent by HTML forms)
const bodyParser = require('body-parser')
app.use(bodyParser.urlencoded({extended: true}))

// Create server
var server = require("http").Server(app);
var io = require("socket.io")(server);
server.listen(process.env.PORT || 3000, () => {
    console.log('listening on *:3000');
});

// MQTT setup
var mqtt = require('mqtt');
var options = {
    port: 1883,
    clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
    username: 'wisblock-rak@ttn',
    password: 'NNSXS.O6PB56L5SOSCMGENHXACDWTUUGKR3TFC6YVFJEI.WKGTAYKEYLG4475JXDV2KSP6MIYURCFB2OS44WS53EBWDX74DYPA',
    keepalive: 60,
    reconnectPeriod: 1000,
    protocolId: 'MQIsdp',
    protocolVersion: 3,
    clean: true,
    encoding: 'utf8'
};
var client = mqtt.connect('https://au1.cloud.thethings.network',options);
```

Git Bash Terminal Output

The output in the terminal included **real-time data received from TTN**. However, there were **issues forwarding this data to QGroundControl (QGC)**, which remained unresolved.

```
User@LAPTOP-LCFQCL7H MINGW64 ~/OneDrive - University of Southampton/Documents/dr
one/TTN_OWN_SERVER
$ node server.js
listening on *:3000
Client connected to TTN
Data from TTN (hexadecimal):  48656c6c6f2148484821
SNR:  14.3
RSSI:  -55
Custom MAVLink message sent to QGroundControl
Data from TTN (hexadecimal):  48656c6c6f2148484821
SNR:  14.3
RSSI:  -55
Custom MAVLink message sent to QGroundControl
Data from TTN (hexadecimal):  48656c6c6f2148484821
SNR:  14
RSSI:  -58
Custom MAVLink message sent to QGroundControl
Data from TTN (hexadecimal):  48656c6c6f2148484821
SNR:  14.3
RSSI:  -55
Custom MAVLink message sent to QGroundControl
Data from TTN (hexadecimal):  48656c6c6f2148484821
SNR:  14
RSSI:  -62
Custom MAVLink message sent to QGroundControl
```

Week 3 (15/7-19/7)

- Node's **maximum payload length testing** and **spreading factor modification.**
- Modify the NodeJS server to **forward the payload message to another laptop** by using ZeroTier and Wireshark.

Details:

Node's Maximum Payload Length Testing

Objective:
Test how increasing the data rate affects the maximum payload length that the node can send.

Findings:

- When the **data rate** increases from 0 to 5, the **maximum payload length** that could be sent increases.
- At **data rate 5**, the maximum payload length was determined to be **192 bytes.**

Spreading Factor Modification

The spreading factor directly impacts the data rate and range of LoRa communication. Modifying it can optimize communication based on application needs.

Reference:
https://forum.rakwireless.com/t/rak4630-set-spreading-factor/8195/6
https://docs.rakwireless.com/Knowledge-Hub/Learn/Installation-of-Board-Support-Package-in-Arduino-IDE/

Arduino IDE (partial code)
Code was written and tested in **Arduino IDE** to modify the spreading factor of the node.

```
//Set Spreading Factor
void set_spreading_factor(uint8_t data_rate) {
    uint8_t spreading_factor;
    //Map the spreading factor to the corresponding data rate
    switch (data_rate) {
        case DR_0:
            spreading_factor=12;
            break;
        case DR_1:
            spreading_factor=11;
            break;
        case DR_2:
            spreading_factor=10;
            break;
        case DR_3:
            spreading_factor=9;
            break;
        case DR_4:
            spreading_factor=8;
            break;
        case DR_5:
            spreading_factor=7;
            break;
        default:
            Serial.println("Invalid data rate");
            return;
    }
    //Set the data rate with ADR disabled
    lmh_datarate_set(data_rate, false);
}
```

zet1n22@soton.ac.uk (33398151)

TTN Live Data

Data rate 0 (Spreading factor 10)

Data rate 1 (Spreading factor 10)

Data rate 2 (Spreading factor 10)

Data rate 3 (Spreading factor 9)

Data rate 4 (Spreading factor 8)

Data rate 5 (Spreading factor 7)

Spreading Factors Observed:

- Data Rate 0: **Spreading Factor 10**
- Data Rate 1: **Spreading Factor 10**
- Data Rate 2: **Spreading Factor 10**
- Data Rate 3: **Spreading Factor 9**
- Data Rate 4: **Spreading Factor 8**
- Data Rate 5: **Spreading Factor 7**

zet1n22@soton.ac.uk (33398151)

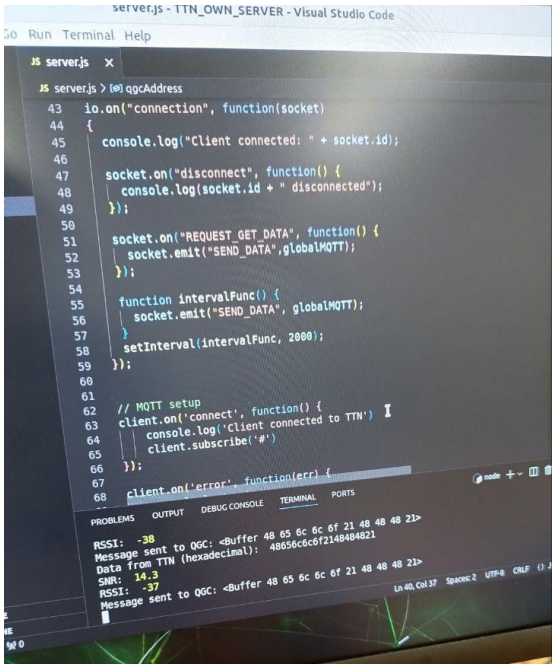
Forward the Payload Message to Another Laptop

Modified NodeJS Server: The NodeJS server was updated to forward payload messages, previously extracted from TTN, to another laptop on the network.

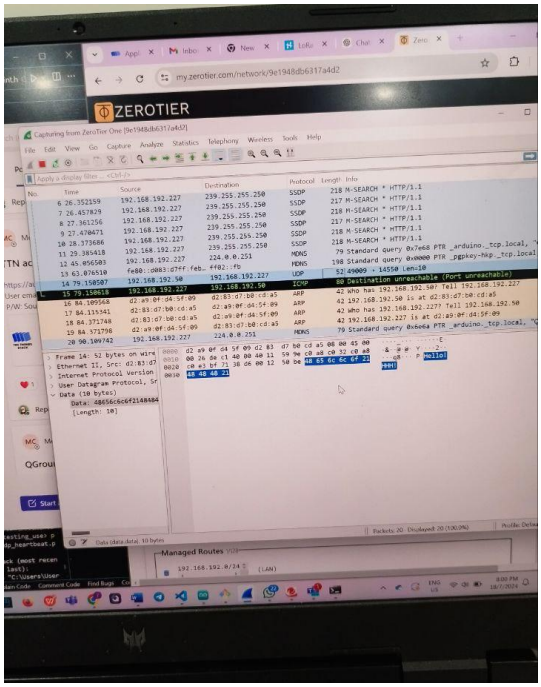
Networking Tools:

e. **ZeroTier:** Used to establish a private network connection between the Jetson device and the target laptop.
f. **Wireshark:** Utilized to monitor and verify that the payload messages were successfully forwarded and received on the target device.

Jetson



Wireshark



zet1n22@soton.ac.uk (33398151)

Week 4 (22/7-26/7)

- Set up **RealVNC Viewer** for Raspberry Pi.
- Create a **Node-RED dashboard** to extract data from TTN and display it in the dashboard.

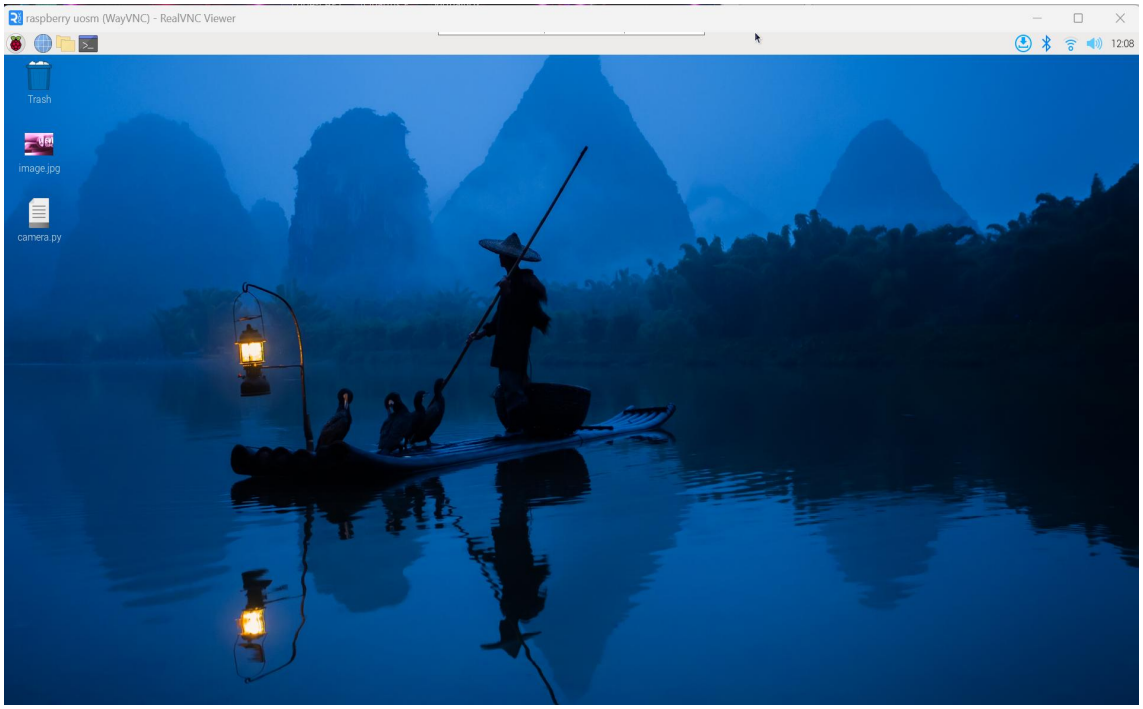
Details:

Set up RealVNC Viewer for Raspberry Pi

RealVNC Viewer for Raspberry Pi is a remote access tool that allows users to view and control the desktop environment of a Raspberry Pi from another device over a network.

Installation:
https://www.realvnc.com/en/connect/download/viewer/

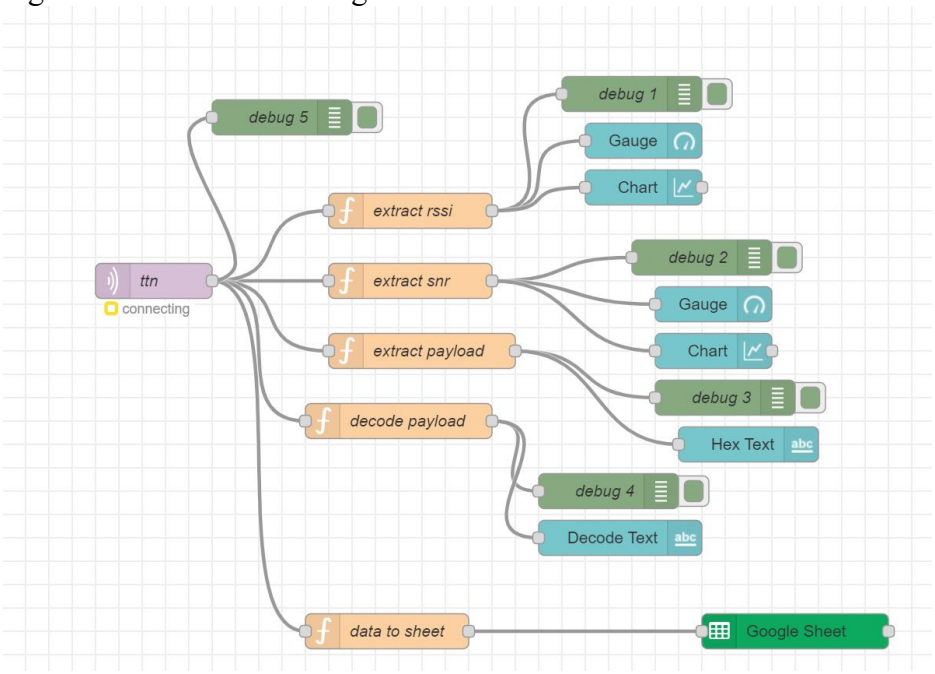
Steps:
Use **RealVNC Viewer** to enable remote access to the Raspberry Pi with IP address 10.100.19.237.



Reference:
https://youtu.be/NWBmYnNvN3A?si=_MhgHeTVuIsoBOuJ

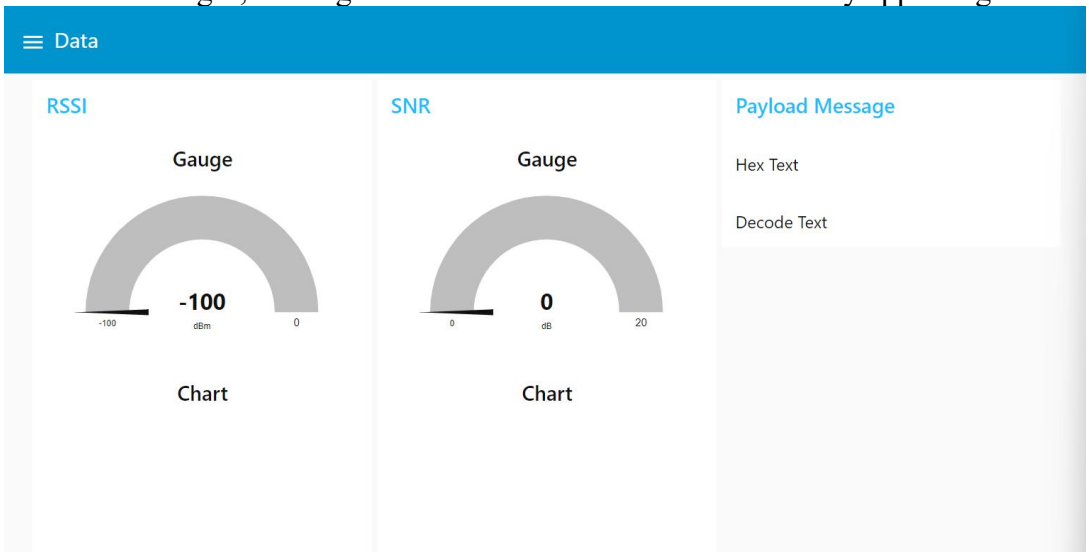
Node-RED Dashboard to Extract Data from TTN:

- Created a flow in Node-RED to stream **RSSI**, **SNR**, and **Payload Message** data from TTN.
- **Decoding Payloads**: Integrated a decoding function to convert payload messages into readable text, then displayed on the dashboard.
- **Data Recording**: Incoming messages were recorded in **Google Sheets** to maintain a log of received TTN messages.



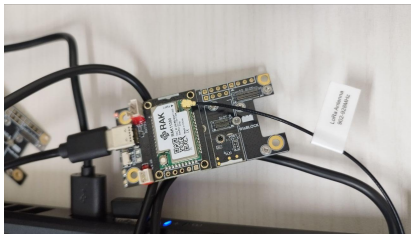
Node-Red Dashboard (Data)

Displayed real-time data from TTN, including signal strength indicators (RSSI, SNR), and decoded messages, making the information accessible and visually appealing.



By connecting the RAK11300 node to the local machine, it **transmits the uplink message** to TTN and the **Node-RED dashboard** takes the data from TTN and then displays it. The data is recorded in the **Google Sheet.**

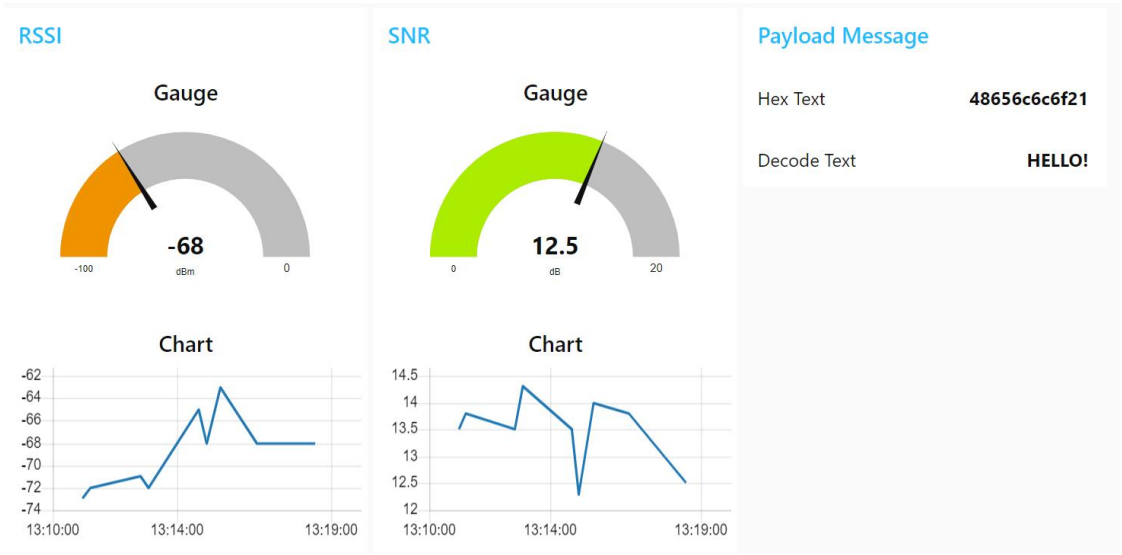
RAK11300 Transmitter Node



TTN Live Data

↑ 12:55:44	eui-ac1f09fffe06b7e1	Forward join-accept message	DevAddr:	26 0D 33 D1	JoinEUI:	00 00 00 00 00 00 00 00	DevEUI:	AC 1F 09 FF FE 06 B7 E1	
↑ 12:55:42	eui-ac1f09fffe06b7e1	Successfully processed join-re…	DevAddr:	26 0D 1D 4F	JoinEUI:	00 00 00 00 00 00 00 00	DevEUI:	AC 1F 09 FF FE 06 B7 E1	
12:55:42	eui-ac1f09fffe06b7e1	Accept join-request	DevAddr:	26 0D 33 D1	JoinEUI:	00 00 00 00 00 00 00 00	DevEUI:	AC 1F 09 FF FE 06 B7 E1	
↑ 12:51:35	eui-ac1f09fffe06b7e1	Forward uplink data message	DevAddr:	26 0D 1D 4F	48 65 6C 6C 6F 21	FPort: 2	Data rate: SF7BW125	SNR: 12.8	RSSI: -75
↑ 12:51:00	eui-ac1f09fffe06b7e1	Forward uplink data message	DevAddr:	26 0D 1D 4F	48 65 6C 6C 6F 21	FPort: 2	Data rate: SF7BW125	SNR: 13.3	RSSI: -75
↑ 12:49:57	eui-ac1f09fffe06b7e1	Forward uplink data message	DevAddr:	26 0D 1D 4F	48 65 6C 6C 6F 21	FPort: 2	Data rate: SF7BW125	SNR: 11.5	RSSI: -77
↑ 12:49:35	eui-ac1f09fffe06b7e1	Forward uplink data message	DevAddr:	26 0D 1D 4F	48 65 6C 6C 6F 21	FPort: 2	Data rate: SF7BW125	SNR: 12.8	RSSI: -78

Node-RED Dashboard



Google Sheet

	Timestamp	SNR (dB)	RSSI (dBm)	Payload Message	Decoded Payload Message
2	2024-09-26T05:10:57.623354152Z	13.5	-73	48656c6c6f21	HELLO!
3	2024-09-26T05:11:11.718862506Z	13.8	-72	48656c6c6f21	HELLO!
4	2024-09-26T05:12:50.426635812Z	13.5	-71	48656c6c6f21	HELLO!
5	2024-09-26T05:13:04.520944428Z	14.3	-72	48656c6c6f21	HELLO!
6	2024-09-26T05:14:43.225227382Z	13.5	-65	48656c6c6f21	HELLO!
7	2024-09-26T05:14:57.320238445Z	12.3	-68	48656c6c6f21	HELLO!
8	2024-09-26T05:15:25.538707857Z	14	-63	48656c6c6f21	HELLO!
9	2024-09-26T05:16:36.025171686Z	13.8	-68	48656c6c6f21	HELLO!
10	2024-09-26T05:18:28.823588679Z	12.5	-68	48656c6c6f21	HELLO!

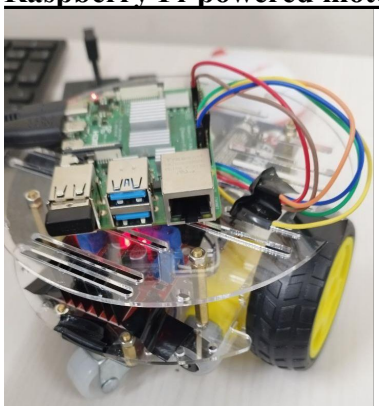
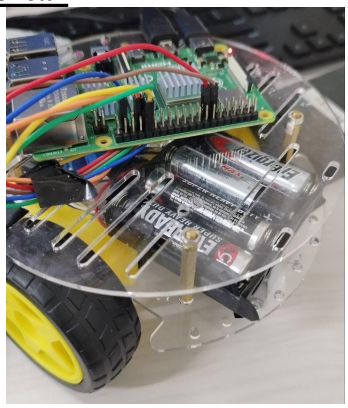
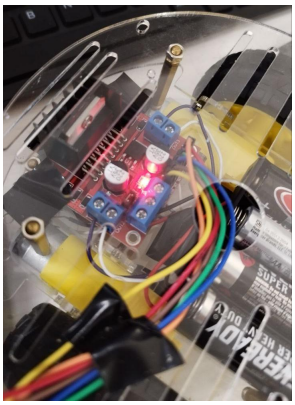
zet1n22@soton.ac.uk (33398151)

Week 5 (29/7-2/8)

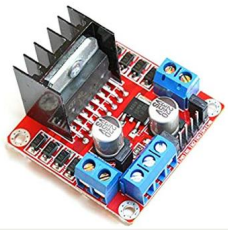
- Build a **Raspberry Pi-powered motorcar**.
- Create a **Node-RED dashboard** to input node location and show it on a map.

Details:

Raspberry Pi-powered motorcar

L298N Motor Driver Pinout and Datasheet



L298N Motor Driver Module | L298N Motor Driver Module Pinout

L298N Module Pinout Configuration

Pin Name	Description
IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B
ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A
OUT3 & OUT4	Output pins of Motor B
12V	12V input from DC power Source
5V	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin

Node-RED Dashboard to Input and Display Node Locations (Map):

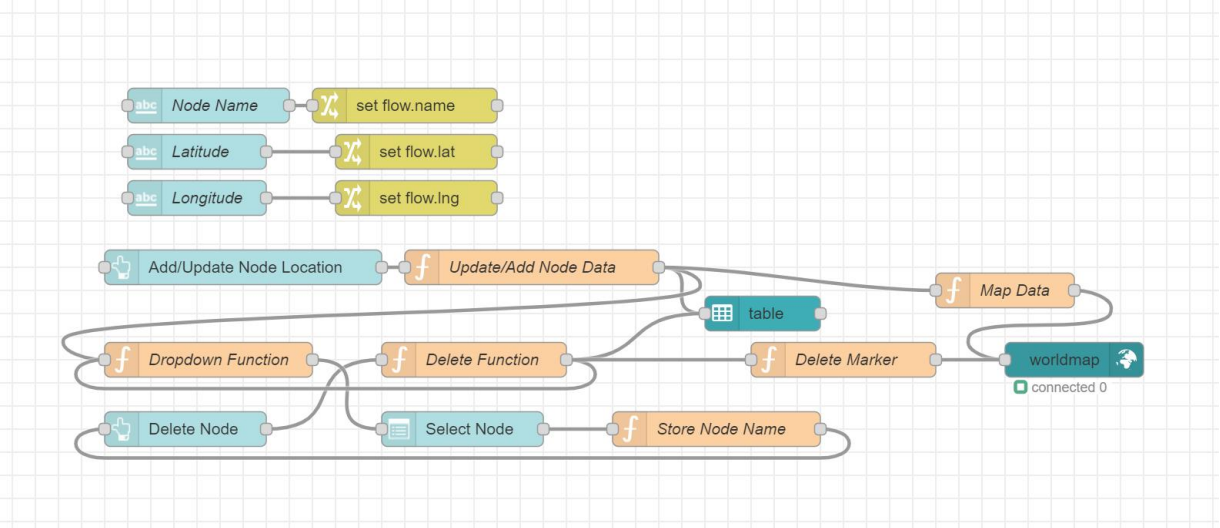
User Inputs:
The dashboard allowed users to input node location data, including the **node name**, **latitude**, and **longitude**.

Add/Update Node Location:

g. When the user clicks the **"Add/Update Node Location"** button, the node information is added to a table, and a **map marker** appears on the map.
h. If a node with the same name was input with updated coordinates, the existing node information and map marker would be automatically updated.

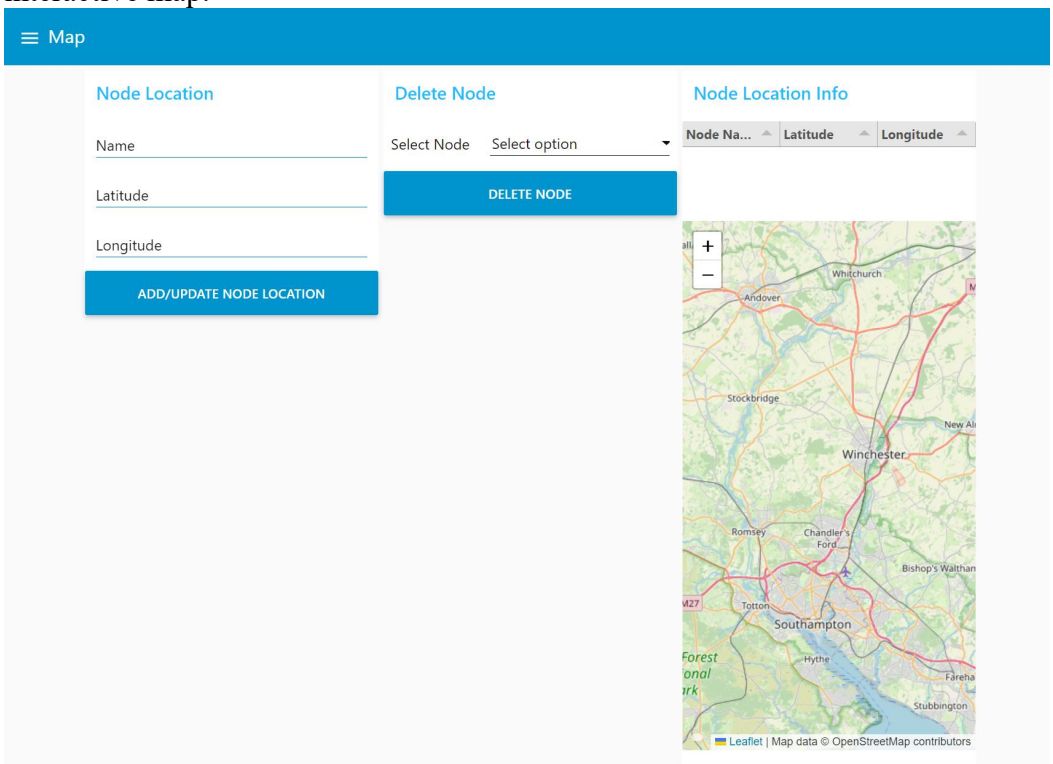
Delete Node Location:

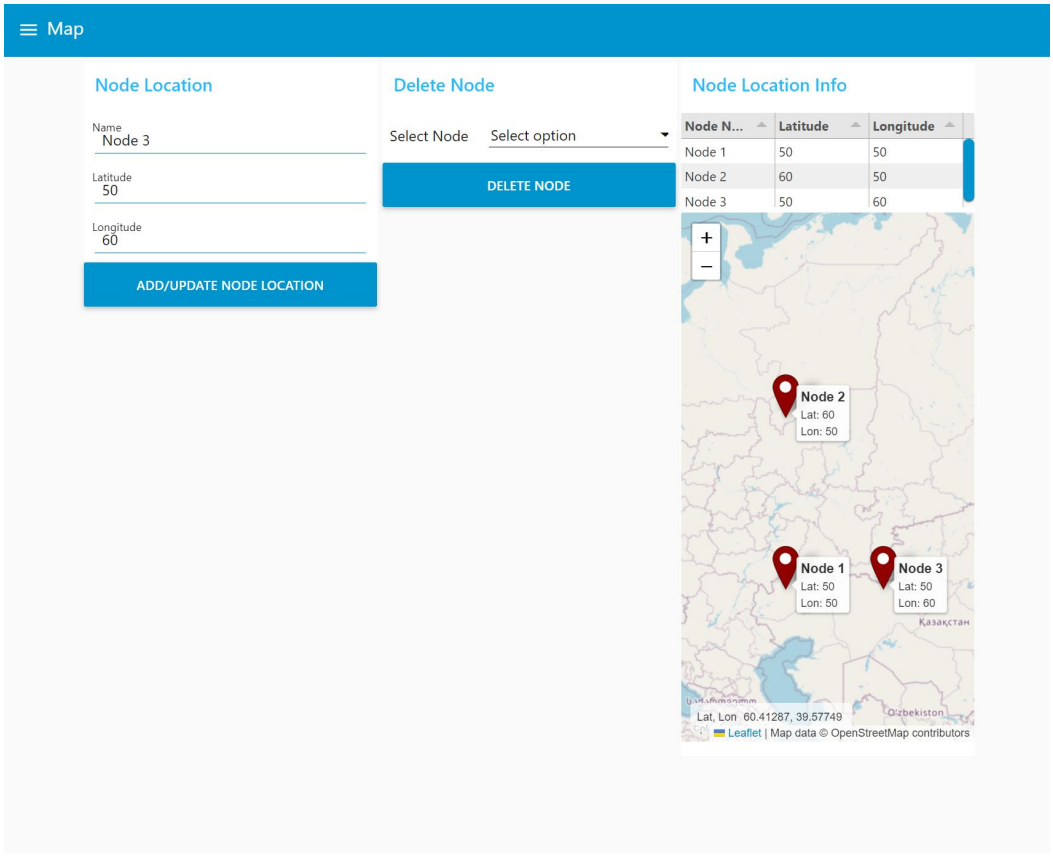
i. Users can delete specific nodes by selecting a node and clicking the **"Delete Node"** button. This action removed the node's info and marker without affecting other nodes.



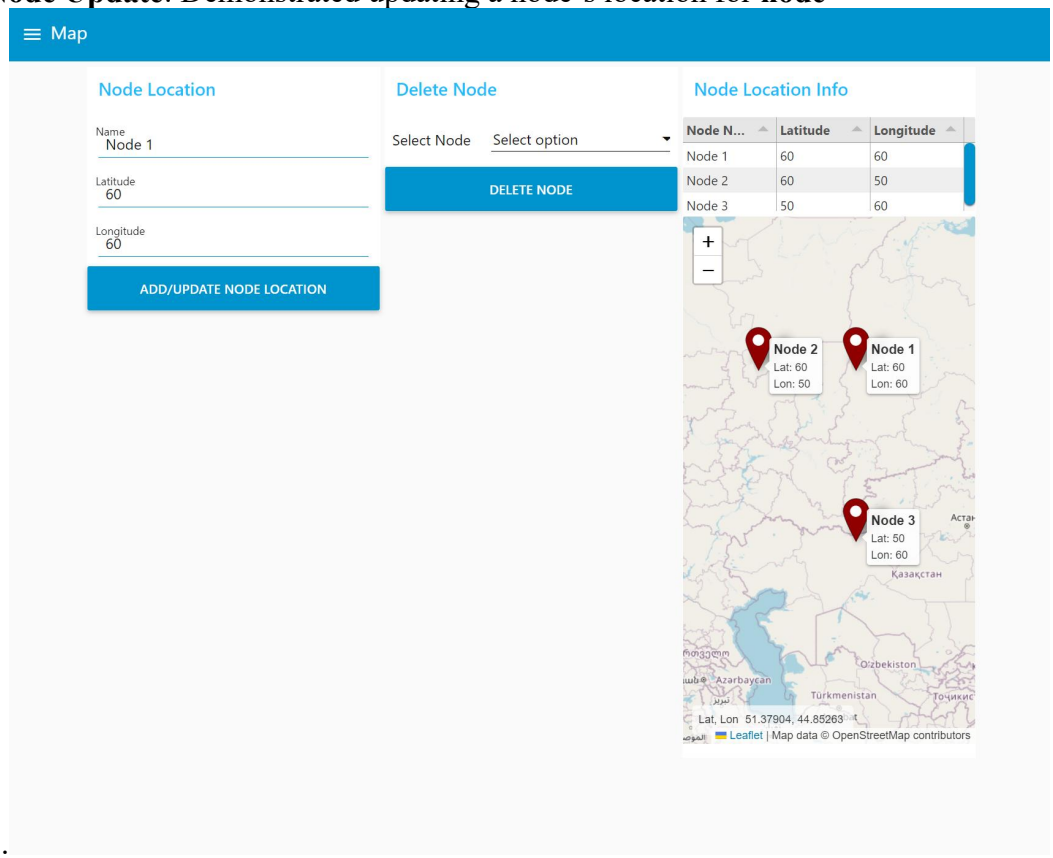
Node-RED Dashboard (Map)

The dashboard displayed the node location information and visualized the nodes on an interactive map.



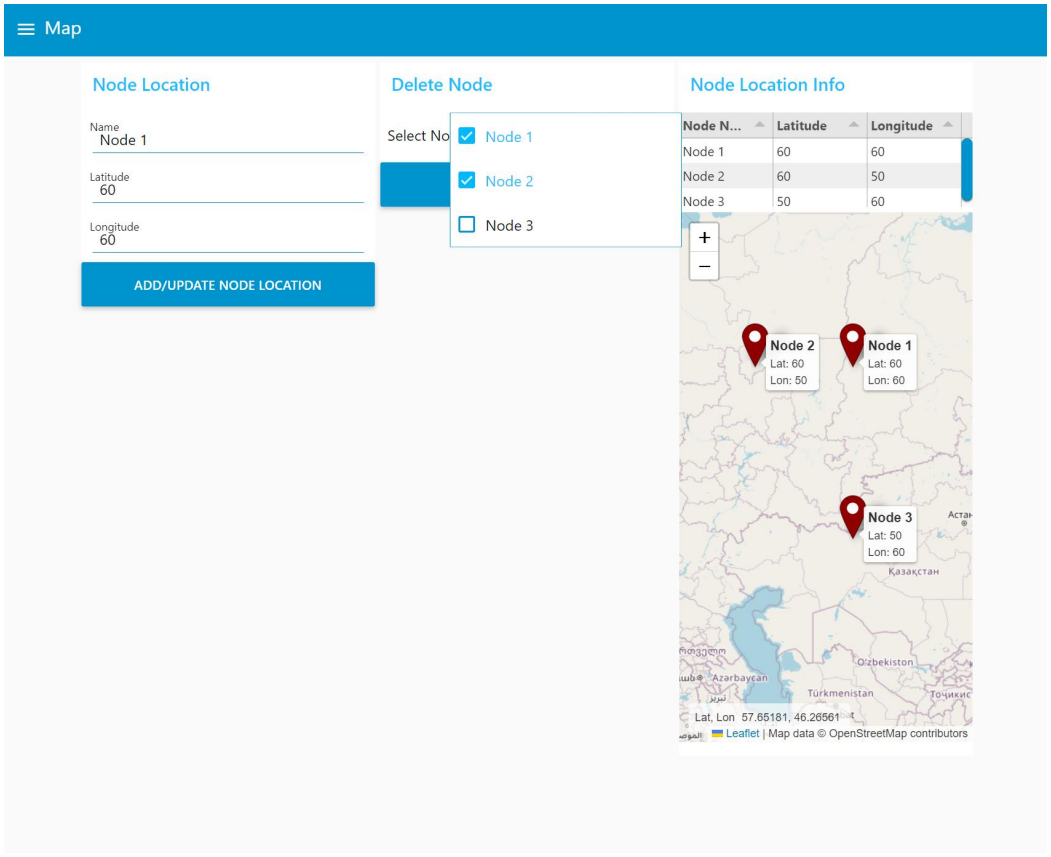


Node Update: Demonstrated updating a node's location for **node**

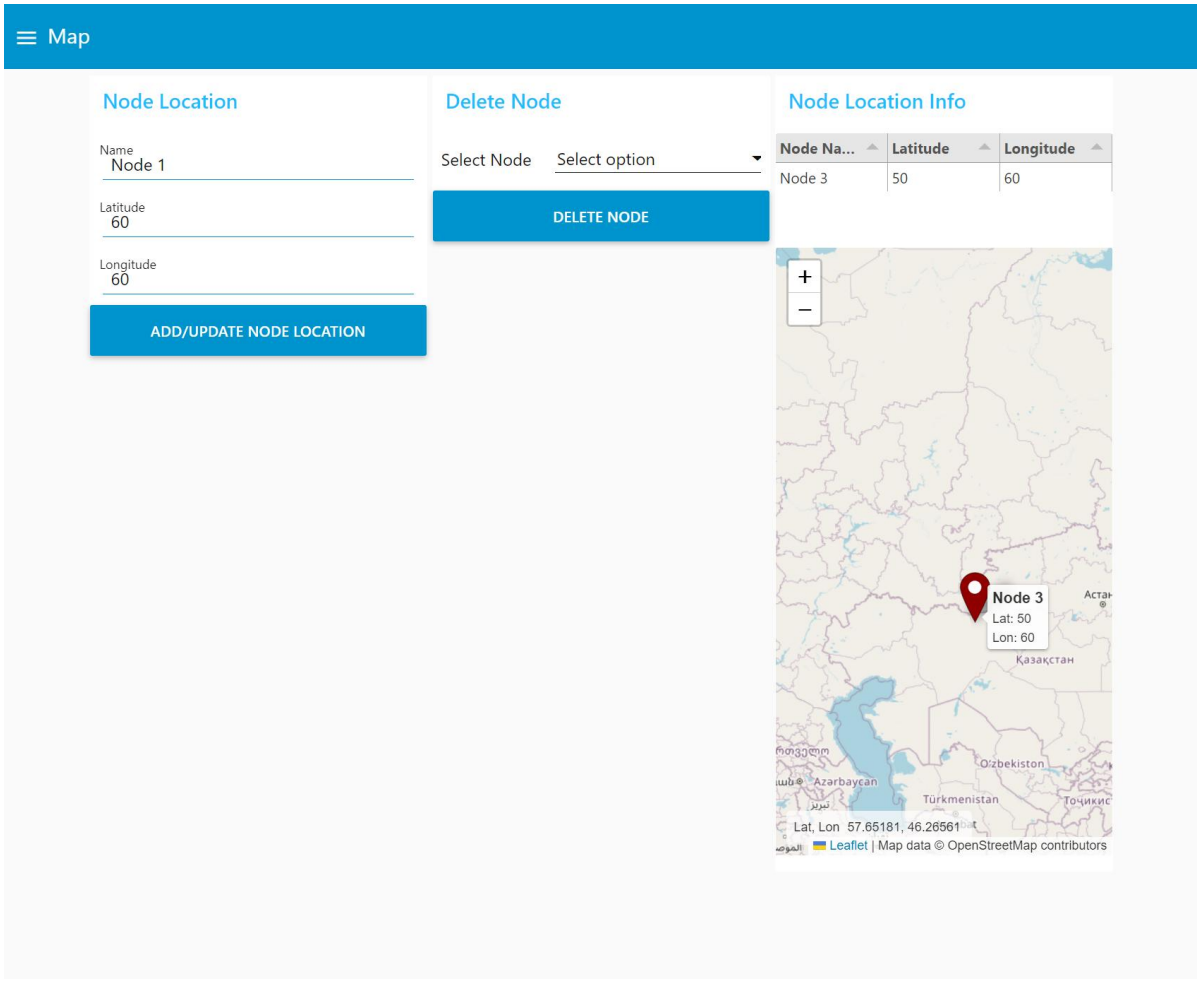


1.

Multiple Node Deletion: Allowed simultaneous deletion of multiple nodes while preserving other entries.



The dashboard interface after the nodes is deleted.



zet1n22@soton.ac.uk (33398151)

Week 6 (5/8-9/8)

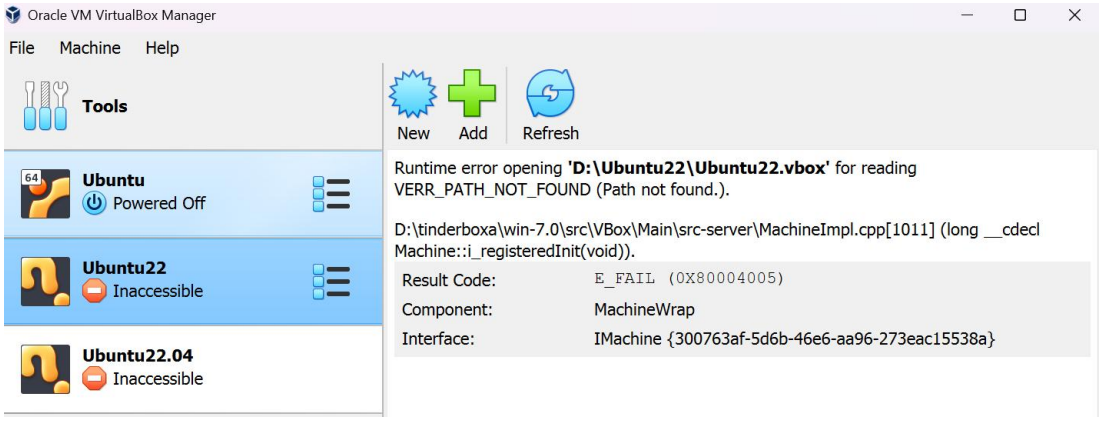
- Set up **Oracle VM VirtualBox Manager** for Ubuntu server on Windows.
- Set up **PX4 SITL Gazebo Simulation with QGroundControl** on Ubuntu 22.

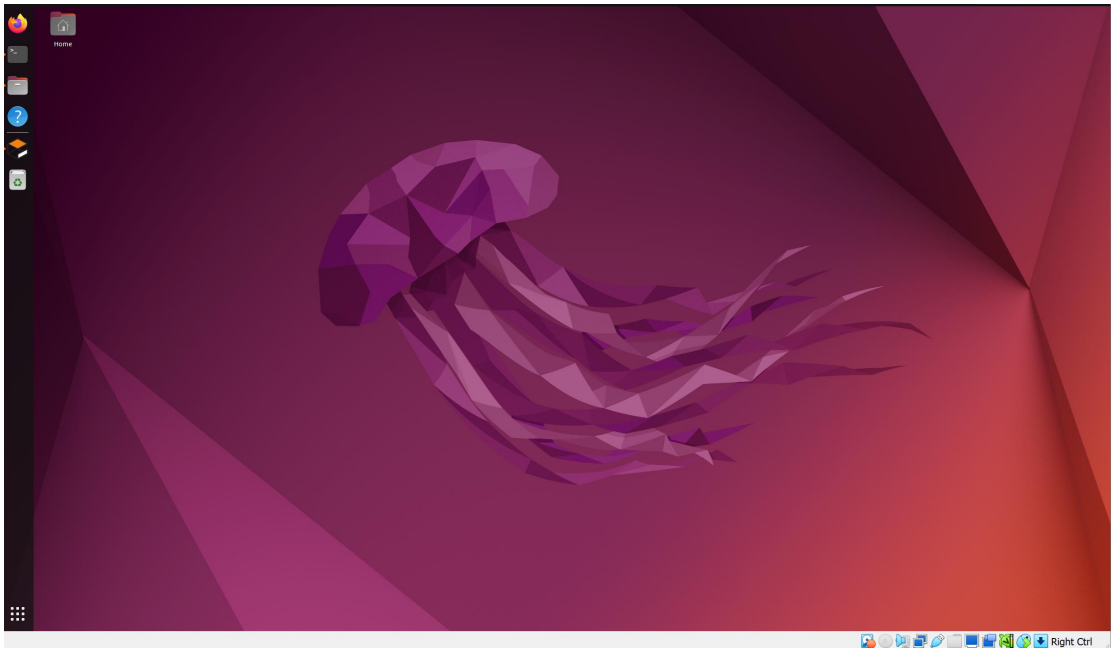
Details:

Oracle VM VirtualBox Manager Setup

Installation:
https://www.oracle.com/virtualization/technologies/vm/downloads/virtualbox-downloads.html

Objective:
Install and configure **Oracle VM VirtualBox** to run an **Ubuntu server (Ubuntu 22)** on Windows for development and testing purposes.





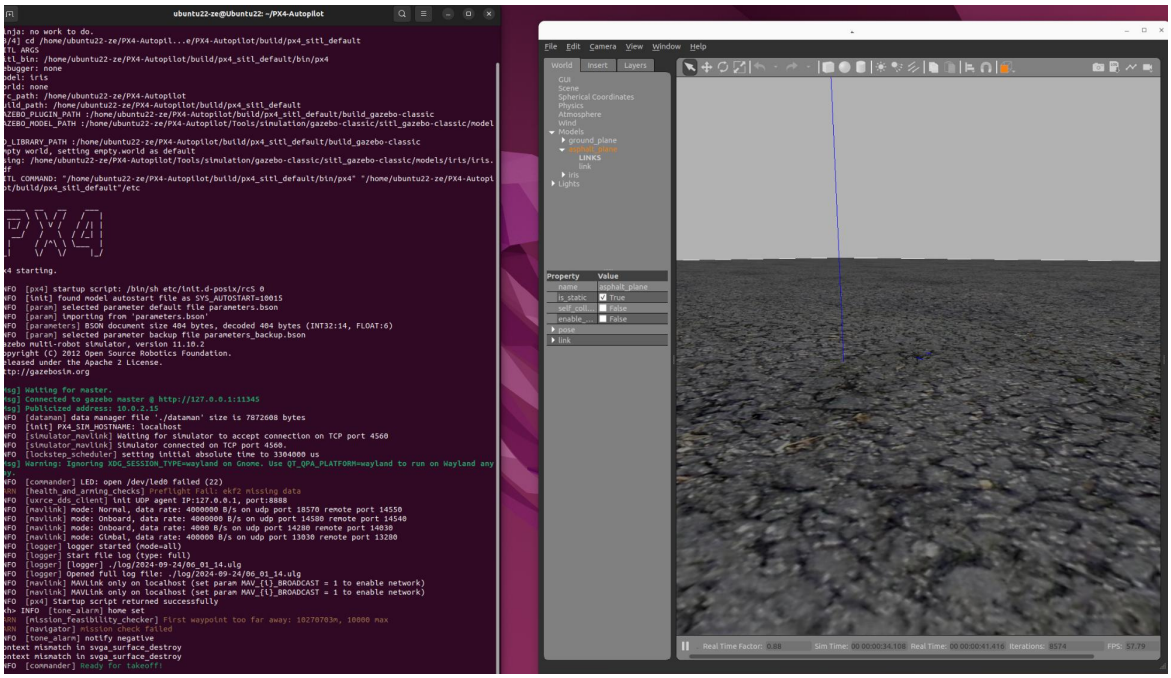
zet1n22@soton.ac.uk (33398151)

PX4 SITL Gazebo Simulation with QGroundControl on Ubuntu 22

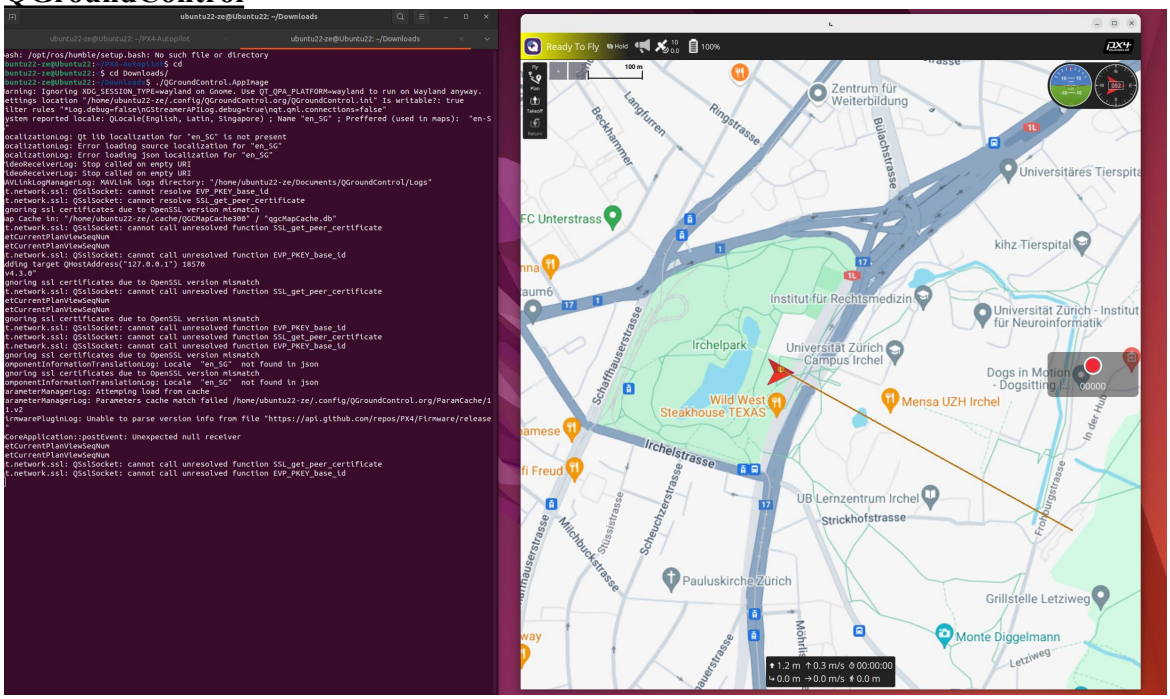
Objective:
Set up the **PX4 Software-in-the-Loop (SITL)** simulation in **Gazebo** and interface it with **QGroundControl** for testing and development.

Reference:
https://github.com/MrStealYoCurls/Gazebo-PX4-Setup-Guide

Outcome:
Gazebo Simulation



QGroundControl



zet1n22@soton.ac.uk (33398151)

Week 7 (12/8-16/8)

- Simulate **QGroundControl** and **Gazebo.**
-

Details:

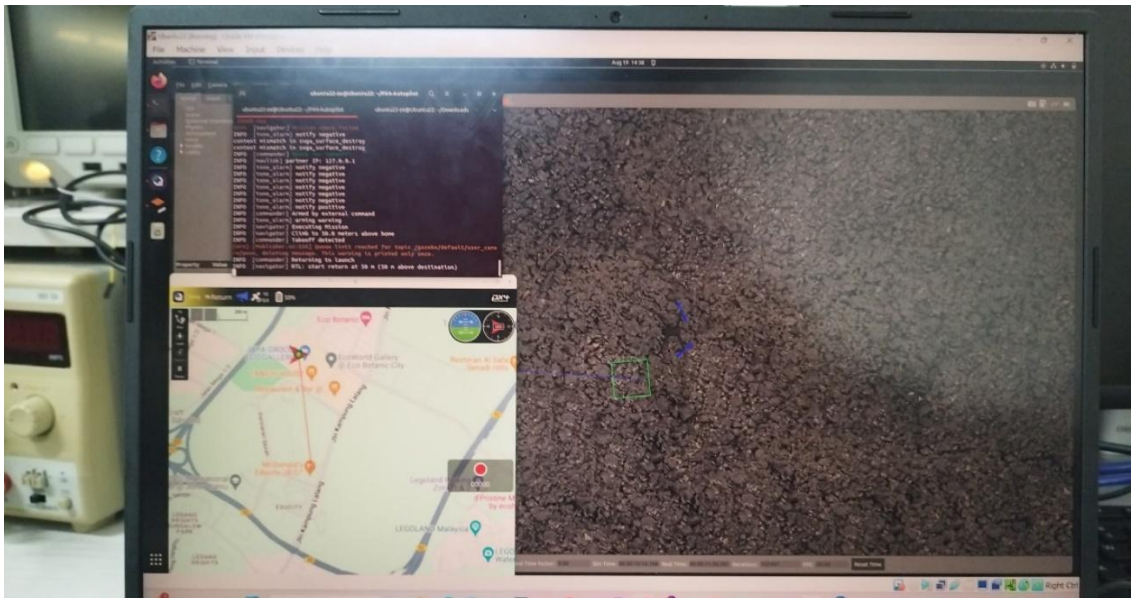
Simulate QGroundControl and Gazebo

Objective:
Simulate a drone flight mission in Gazebo using **QGroundControl** for mission planning.

Mission Details:

- Planned a flight mission from **Eco Galleria** to **McDonald's EduCity** within QGroundControl.
- Used a **multirotor** drone model with a flight speed of **5.0 m/s**.
- The entire simulated mission took approximately **12 minutes** to complete.

Outcome:



Reference:

https://docs.px4.io/main/en/simulation/
https://docs.px4.io/v1.12/en/simulation/gazebo.html
https://docs.px4.io/main/en/flight_modes_mc/mission.html
https://docs.px4.io/main/en/flying/missions.html

Week 8 (19/8-23/8)

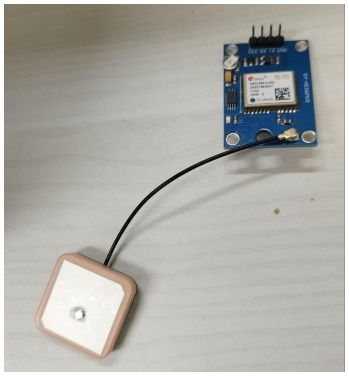
- Work out with **NEO-6M GPS Module Arduino.**
- Create a **Node-RED dashboard** to connect to the GPS Module.
- Check it on the **latitude and longitude finder website** to confirm if the GPS Module works perfectly or not.

Details:

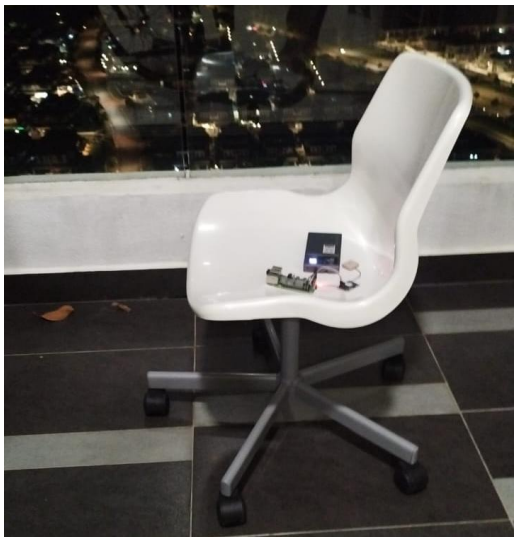
Working with NEO-6M GPS Module and Arduino:

Objective:
Set up the **NEO-6M GPS module** with Arduino, powered by a Raspberry Pi, to capture GPS coordinates.

Setup:
Connected the GPS module outdoors at **EcoNest** for testing to ensure accurate reception of satellite signals.
Once the GPS module LED starts to **blink**, this means that it is working.



GPS VCC to **Pin 2**
GPS TX to **Pin 10**
GPS GND to **Pin 6**



Reference:
https://randomnerdtutorials.com/guide-to-neo-6m-gps-module-with-arduino/
https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/

NMEA Sentences on Raspberry Pi Terminal

```
admin@raspberrypi:~ $ sudo cat /dev/serial0
$GPVTG,,T,,M,1.389,N,2.573,K,A*23
$GPGGA,151707.00,0126.31847,N,10337.13280,E,1,05,7.98,128.4,M,5.0,M,,*62
$GPGSA,A,3,21,14,07,08,30,,,,,,,,8.47,7.98,2.84*0B
$GPGSV,3,1,12,02,60,059,,03,21,151,,04,32,177,,06,00,210,*74
$GPGSV,3,2,12,07,56,348,25,08,11,028,07,09,43,219,,14,08,315,30*7F
$GPGSV,3,3,12,17,32,258,,19,14,238,,21,42,048,20,30,26,332,25*7F
$GPGLL,0126.31847,N,10337.13280,E,151707.00,A,A*6E
$GPRMC,151708.00,A,0126.31892,N,10337.13284,E,1.560,,260924,,,A*7D
```

Creating a Node-RED Dashboard for GPS Data (GPS):

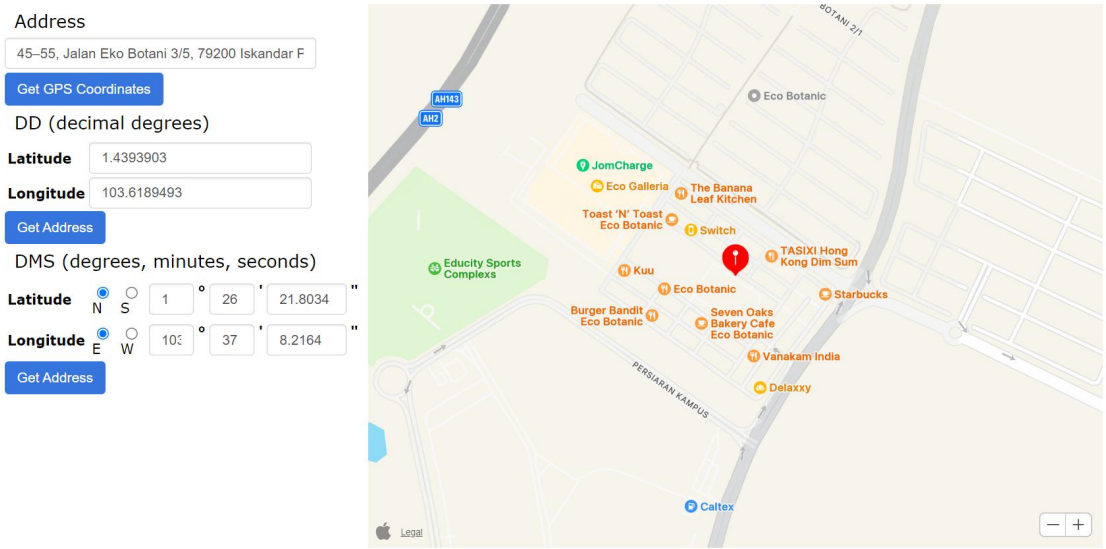
Objective:
Develop a **Node-RED dashboard** to connect with the GPS module and display the real-time latitude and longitude data.

Confirming GPS Accuracy with Coordinate Converter Website:

Verified the GPS data by checking the coordinates on the coordinate converter website.

Website:
https://gps-coordinates.org/coordinate-converter.php

Location Page URL:
https://gps-coordinates.org/my-location.php?lat=1.4393903&lng=103.6189493



Week 9 (26/8-30/8)

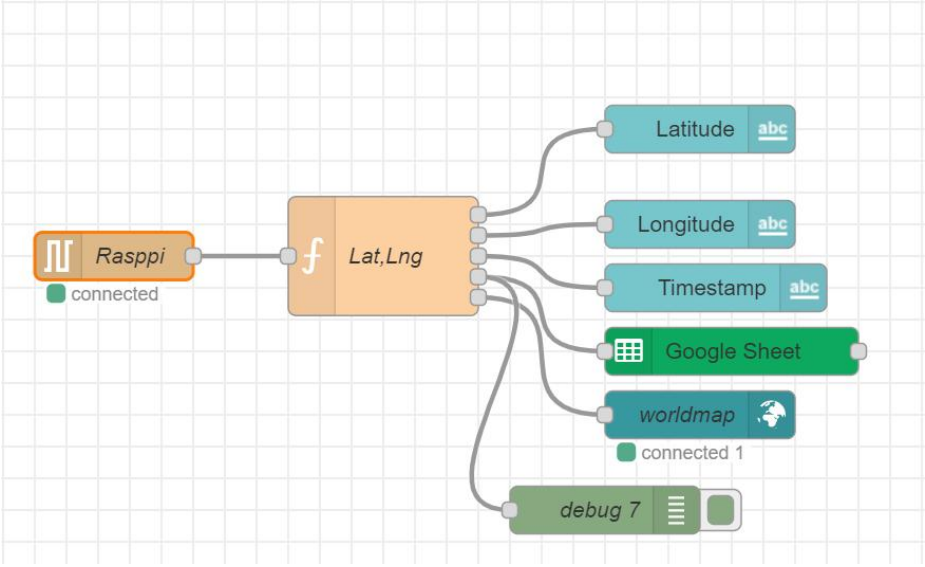
- Enhance the **Node-Red dashboard** that shows the live time GPS location and displays it on the map.
- **Record the data** in the Google Sheet

Details:

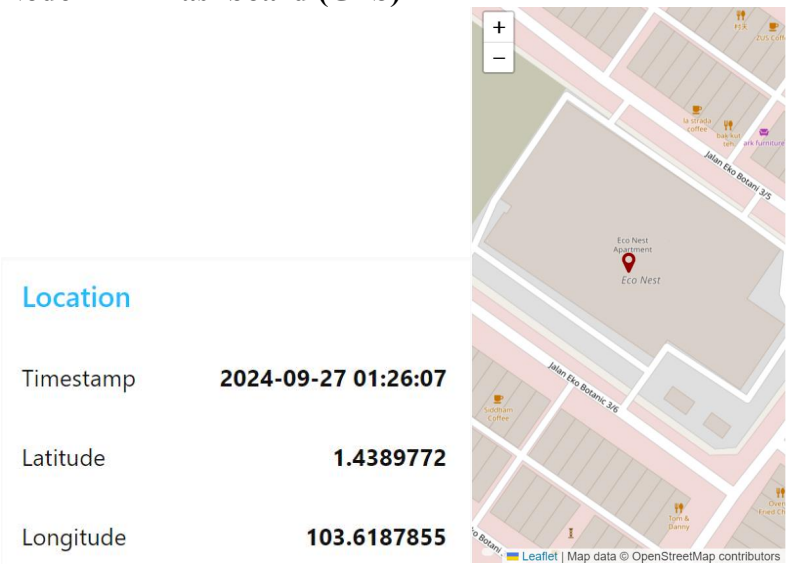
Node-RED Dashboard Update for Live GPS Tracking (GPS):

Objective:
Enhance the **Node-RED dashboard** to display real-time GPS location data, including **timestamps** and visualizing it on a **map**.

The dashboard now dynamically updates with the **live GPS location** of the connected device (from the NEO-6M module), showing both latitude and longitude along with a timestamp.



- **Node-RED Dashboard (GPS)**



zet1n22@soton.ac.uk (33398151)

Recording GPS Data in Google Sheets:

Objective:
Log the GPS data, including the coordinates and timestamp, in a **Google Sheet** for historical tracking and analysis.

A	B	C
Timestamp	Latitude	Longitude
8/27/2024 21:23:	1.4387445	103.6189383
8/27/2024 21:23:	1.4387533	103.6189405
8/27/2024 21:23:	1.4387557	103.6189405
8/27/2024 21:23:	1.4387497	103.6189395
8/27/2024 21:23:	1.4387498	103.6189402
8/27/2024 21:23:	1.4387630	103.6189420
8/27/2024 21:23:	1.4387612	103.6189418
8/27/2024 21:23:	1.4387732	103.6189442
8/27/2024 21:23:	1.4387955	103.6189485
8/27/2024 21:23:	1.4388068	103.6189502
8/27/2024 21:23:	1.4388117	103.6189507
8/27/2024 21:23:	1.4388180	103.6189523
8/27/2024 21:23:	1.4388203	103.6189527
8/27/2024 21:24:	1.4388262	103.6189533
8/27/2024 21:24:	1.4388510	103.6189572
8/27/2024 21:24:	1.4388680	103.6189590
8/27/2024 21:24:	1.4388838	103.6189578
8/27/2024 21:24:	1.4389078	103.6189613
8/27/2024 21:24:	1.4389380	103.6189663
8/27/2024 21:24:	1.4389588	103.6189693
8/27/2024 21:24:	1.4389803	103.6189723
8/27/2024 21:24:	1.4389890	103.6189738
8/27/2024 21:24:	1.4390198	103.6189787
8/27/2024 21:24:	1.4390438	103.6189822
8/27/2024 21:24:	1.4390480	103.6189850
8/27/2024 21:24:	1.4390625	103.6189882
8/27/2024 21:24:	1.4390682	103.6189892
8/27/2024 21:24:	1.4390500	103.6189938

Week 10 (2/9-6/9)

- Create a **Node-RED dashboard** to **control the Raspberry Pi-powered motorcar** to move forward, left, right and stop.

Details:

Node-RED Dashboard for Robot Car Control (Movement):

Objective:
Develop a **Node-RED dashboard** to remotely control a Raspberry Pi-powered motorcar's movements—**forward, left, right and stop**.

Components Overview:

1. **Motor Control:**

- The Raspberry Pi **GPIO pins** are being used to control the **motors of the car**, with specific pins handling forward, reverse, and **speed control** through **PWM (Pulse Width Modulation).**
- There are four output GPIO pins for left and right motor control in forward and reverse directions **(6, 27, 26, 22)**, and two pins **(12, 13)** controlling motor speed through PWM.
- A **control function** determines the car's movement **(forward, left, right, stop)** based on input signals.

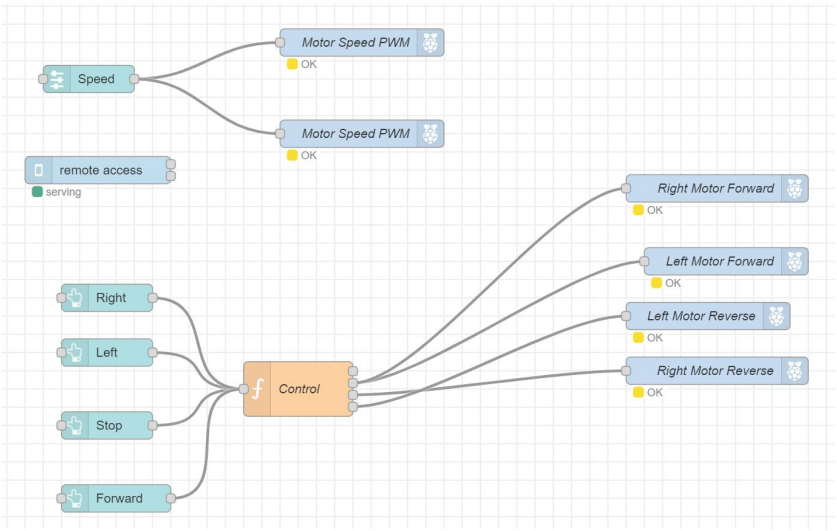
2. **User Interface (UI) for Control:**

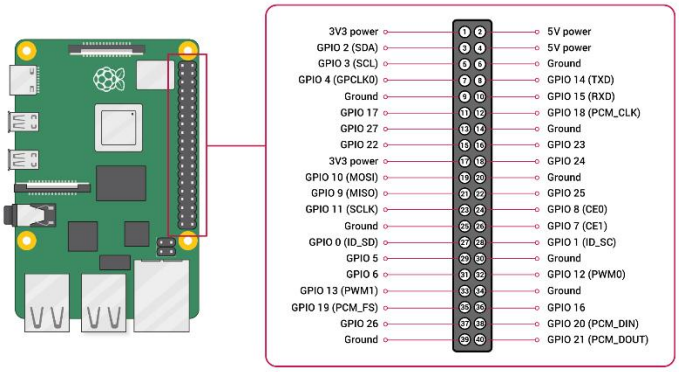
- A set of **buttons** (left, right, forward, stop) allows manual control of the car's movement from the Node-RED Dashboard.
- A **slider** is provided to adjust the speed of the car, affecting the PWM signal sent to the motor pins.

3. **Remote Access:**

- A **remote-access node** is present, which connects the Node-RED flow to the **Remote-RED app.** This enables remote control and monitoring of the car from anywhere with a phone.

Node-RED Flow (Movement):
Implemented a control flow where user inputs from the dashboard are translated into commands sent to the robot car.

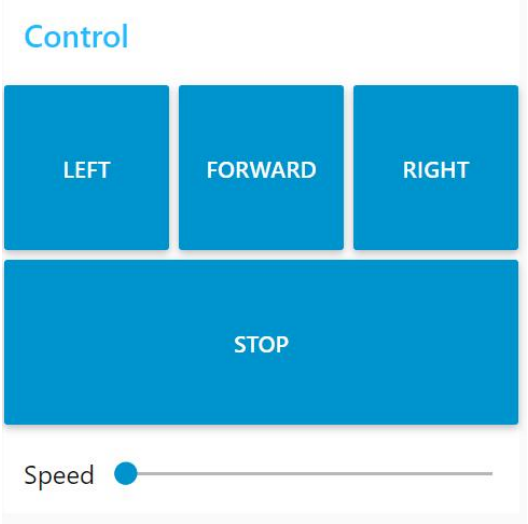




Left Motor Speed PWM-**GPIO 12**
Right Motor Speed PWM-**GPIO 13**
Left Motor Forward-**GPIO 6**
Right Motor Forward-**GPIO 27**
Left Motor Reverse-**GPIO 22**
Right Motor Reverse-**GPIO 26**

Node-RED Dashboard (Movement):

Features:
The dashboard contains buttons or a control interface that allows users to move the robot car in the desired direction by clicking buttons to send movement commands.



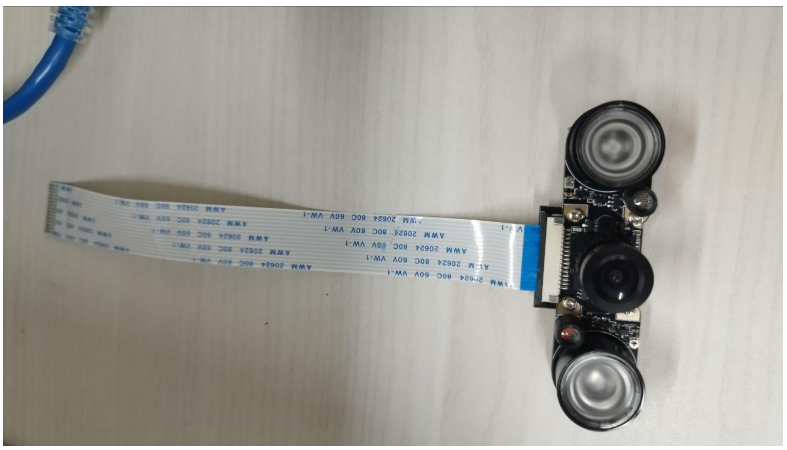
Week 11 (9/9-13/9)

- Work out with Raspberry Pi Night Vision Camera.
- Livestream the camera video and display it on the Node-RED dashboard.

Details:

Working with Raspberry Pi Night Vision Camera

Objective:
Set up the **Raspberry Pi Night Vision Camera** to live stream video.

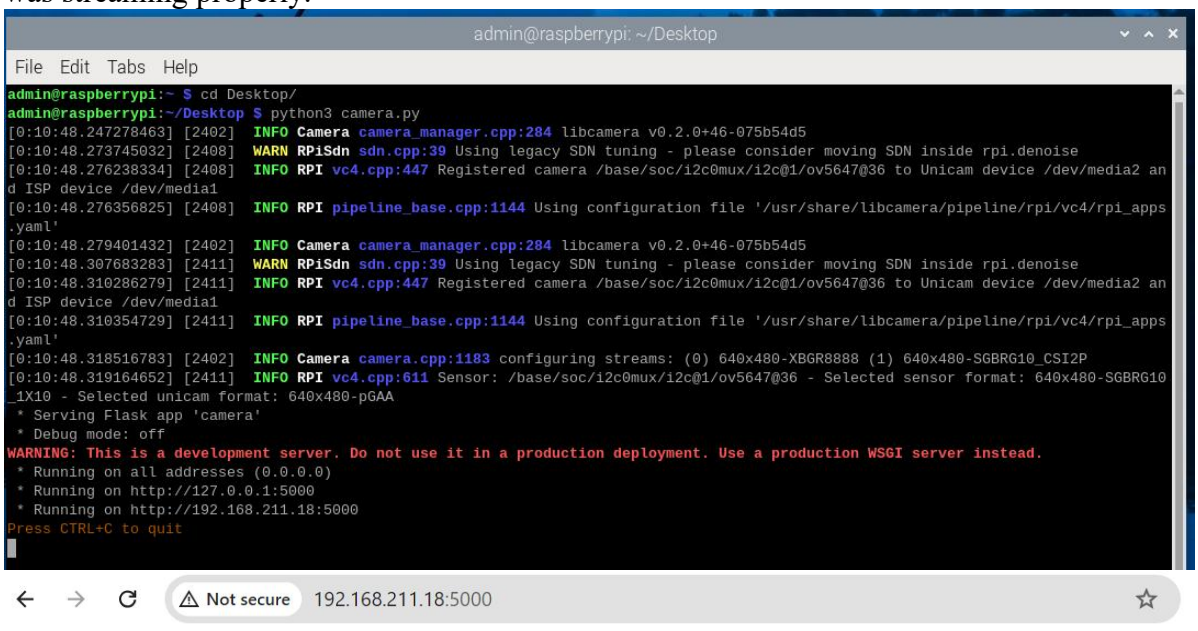


Implementation:
Created a Python script (**camera.py**) to stream the camera video over **port 5000**.

- The code imports the necessary libraries (**Flask, Response** from **flask, Picamera2** from **picamera2**, and **PIL** from **Pillow**).
- A Flask app is created to handle HTTP routes.
- The **gen_frames()** function captures a video frame from the Pi camera using **camera.capture_array(),** converts it to a PIL image, rotates it 180 degrees, ensures the image is in RGB format if necessary, and then converts the image to JPEG for streaming.
- The **@app.route('/')** function provides a basic HTML page with a header and an image element that points to the **/video_feed** route.
- The **@app.route('/video_feed')** function generates a response that provides the video feed in multipart JPEG format for continuous streaming.
- Finally, the **if __name__ == "__main__"** section initializes the camera, configures the preview, starts the camera, and runs the Flask app on **0.0.0.0** with **port 5000.**

Reference:
https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/2
https://www.raspberrypi.com/documentation/accessories/camera.html

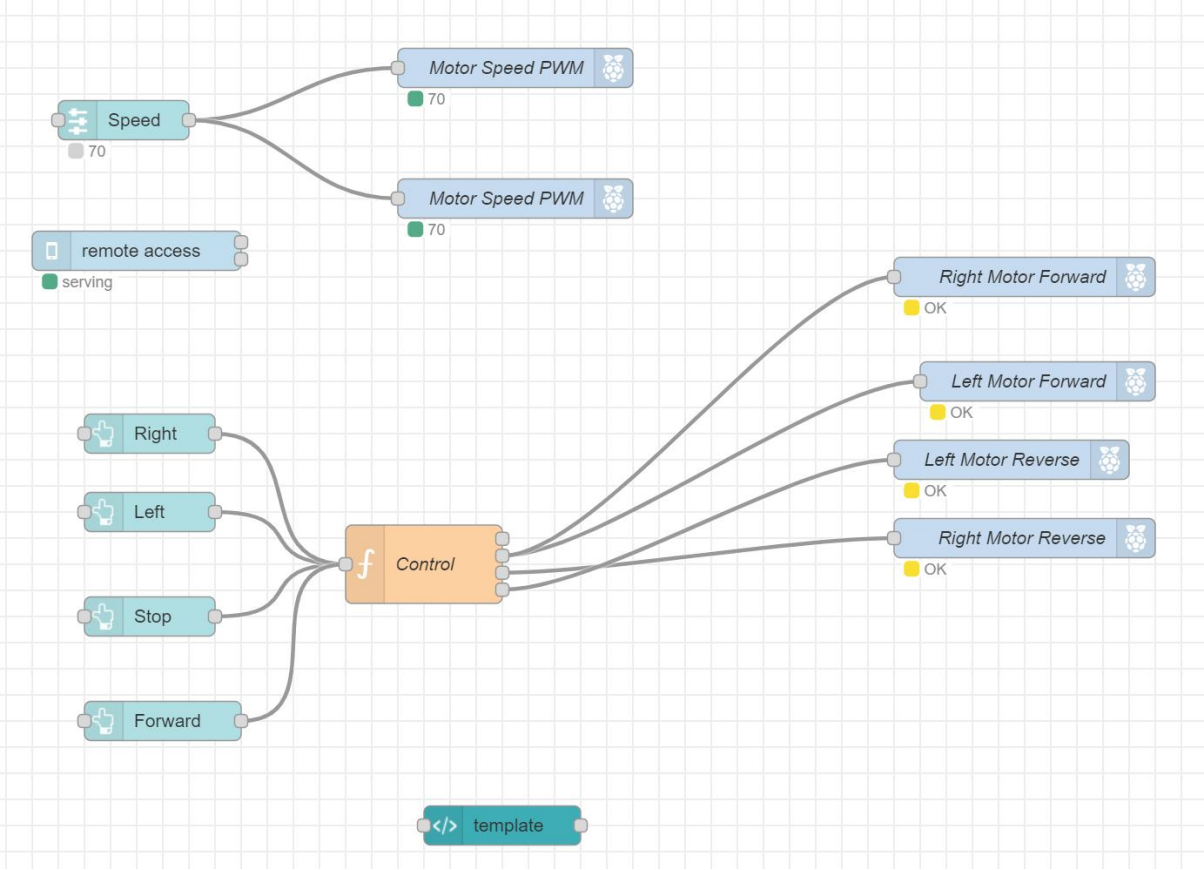
Calling the Python Script:
Executed the **camera.py** script in the terminal, successfully initiating the live stream. The video stream was accessible via **http://<raspberry_pi_ip>:5000**, confirming that the camera was streaming properly.



Raspberry Pi Camera Stream

Integrating Video Stream into Node-RED Dashboard:

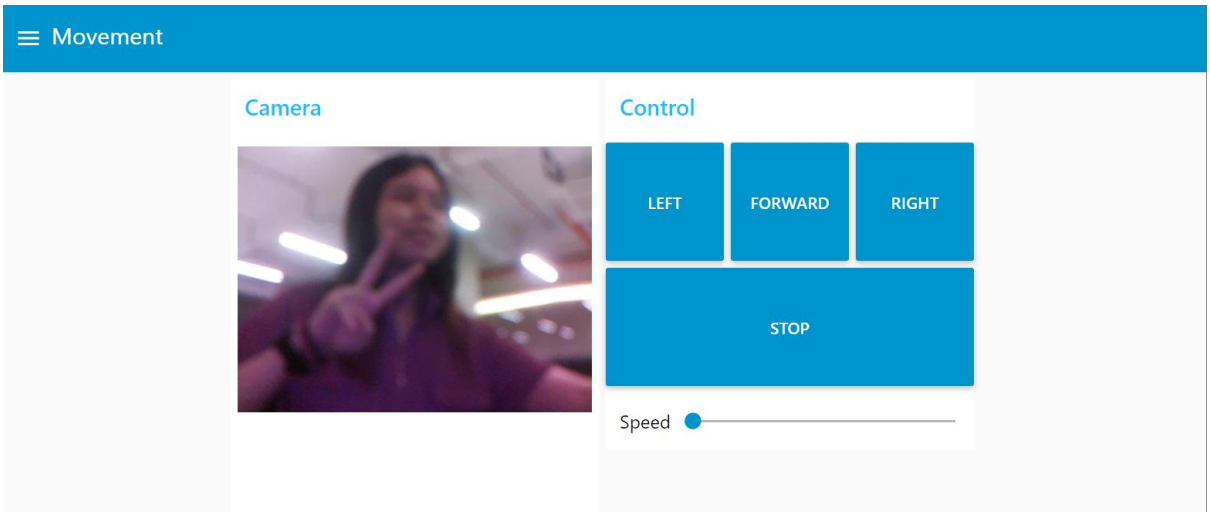
- Updated the previous **Node-RED Dashboard (Movement)** by adding a **template node** to embed the live stream video from port 5000.
- The video stream was successfully displayed on the Node-RED dashboard alongside the existing control interface for the robot car.



In the **template node,**

```
<div style="text-align:center;">
    <img src="http://192.168.211.18:5000/video_feed" style="width:100%; max-width:800px;">
</div>
```

The **IP address** should be the same as the **Raspberry Pi IP address.**



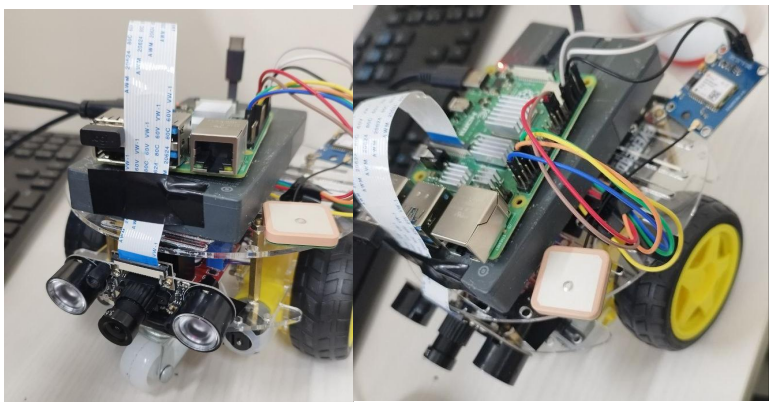
Week 12 (16/9-20/9)

- Assembly of **the camera and GPS module** on the Raspberry Pi-powered motorcar Test the car by **controlling wirelessly** using a phone.

Details:

Assembly of Camera and GPS Module:

Objective:
Integrate the **Raspberry Pi Night Vision Camera** and **NEO-6M GPS module** onto the **Raspberry Pi-powered motorcar.**



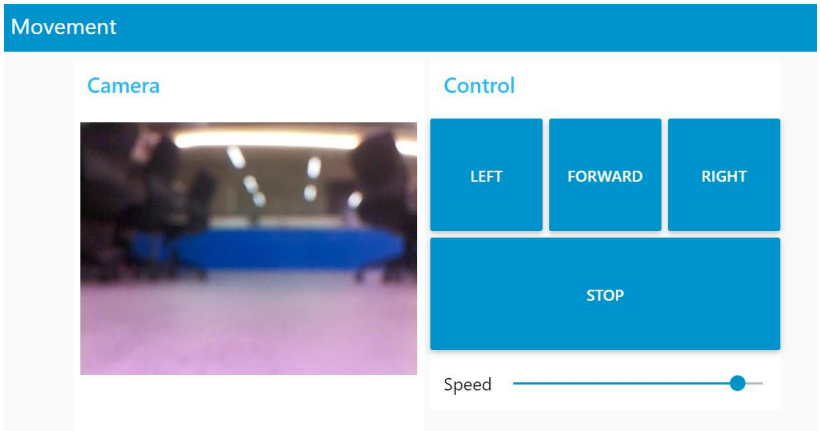
The camera was securely mounted to provide a clear forward view, while the GPS module was positioned to ensure accurate location tracking during movement.

Wireless Control Testing:

Objective:
Test the functionality of the robot car by controlling it wirelessly using a mobile phone.

- Connected to the Node-RED dashboard from a smartphone, enabling remote control of the car's movement.
- Verified that both the camera livestream and GPS data were accessible and functional through the dashboard interface.

Outcome:



Week 13 (23/9-27/9)

- **Machine learning** with RAK11300 and Edge Impulse.
- Access a Remote Terminal with SSH on Raspberry Pi
- Use two RAK11300, transmitter and receiver respectively to **gesture control** the Arduino robot car to move forward, left and right.
-

Details:

Machine learning with RAK11300 and Edge Impulse.

Figure: RAK11300 with 3-axis accelerometer

RAK11300 and 3-Axis Accelerometer Setup

The RAK11300 is a powerful microcontroller that can be paired with sensors such as 3-axis accelerometer to capture motion data.

Objective:

To recognise gestures such as forward, left and right using machine learning models deployed on the RAK11300.

Installation and setup:

- Follow the RAKwireless documentation [https://docs.rakwireless.com/Knowledge-Hub/Learn/Getting-Started-with-WisBlock-and-Edge-Impulse/] to install the necessary environment, including the Arduino IDE, RAK11300 board definitions and libraries.

3-Axis Accelerometer Arduino Code:

- Using the 3-Axis Accelerometer Arduino Example Code with several modification as below, the code could be written to read data from 3-axis accelerometer.
- The sensor captures X, Y and Z axis values, which represent accelerations along different directions.

Edge Impulse Platform

Edge Impulse is a machine learning platform that simplifies the process of training models.

Account Setup and Data Collection:

- Create an account on Edge Impulse Studio [https://studio.edgeimpulse.com/studio/profile/projects?autoredirect=1].
- Create a new project to capture the data from the RAK11300's 3-axis accelerometer, which will be used to train a gesture recognition model.

Installation:

- Follow the Edge Impulse CLI installation guide [https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-installation] to install the CLI on computer.
- The CLI is used to interact with the Edge Impulse platform, including data collection, training and deployment.

Installation - Windows

1. Create an Edge Impulse account.
2. Install Python 3 on your host computer.
3. Install Node.js v20 or above on your host computer.
   - For Windows users, install the **Additional Node.js tools** (called **Tools for Native Modules** on newer versions) when prompted.
4. Install the CLI tools via:

```
npm install -g edge-impulse-cli --force
```

You should now have the tools available in your PATH.

- After installation, connect the RAK11300 to the computer and run the **edge-impulse-data-forwarder** command on the terminal to send the accelerometer data from the device to the Edge Impulse platform.

```
PS C:\Users\User> edge-impulse-data-forwarder
Edge Impulse data forwarder v1.27.1
? What is your user name or e-mail address (edgeimpulse.com)? teezhien
? What is your password? [hidden]
Endpoints:
    Websocket: wss://remote-mgmt.edgeimpulse.com
    API:       https://studio.edgeimpulse.com
    Ingestion: https://ingestion.edgeimpulse.com

? Which device do you want to connect to? COM6 (Microsoft)
[SER] Connecting to COM6
[SER] Serial is connected (51:84:60:E6:22:65:6A:53)
[WS ] Connecting to wss://remote-mgmt.edgeimpulse.com
[WS ] Connected to wss://remote-mgmt.edgeimpulse.com

? To which project do you want to connect this device? Zhi En Tee / control2
[SER] Detecting data frequency...
[SER] Detected data frequency: 47Hz
? 3 sensor axes detected (example values: [0.47,-0.63,9.74]). What do you want to call them? Separate the names with ','
: AccX,AccY,AccZ
[WS ] Device "rak11300" is now connected to project "control2". To connect to another project, run `edge-impulse-data-fo
rwarder --clean`.
[WS ] Go to https://studio.edgeimpulse.com/studio/522144/acquisition/training to build your machine learning model!
```

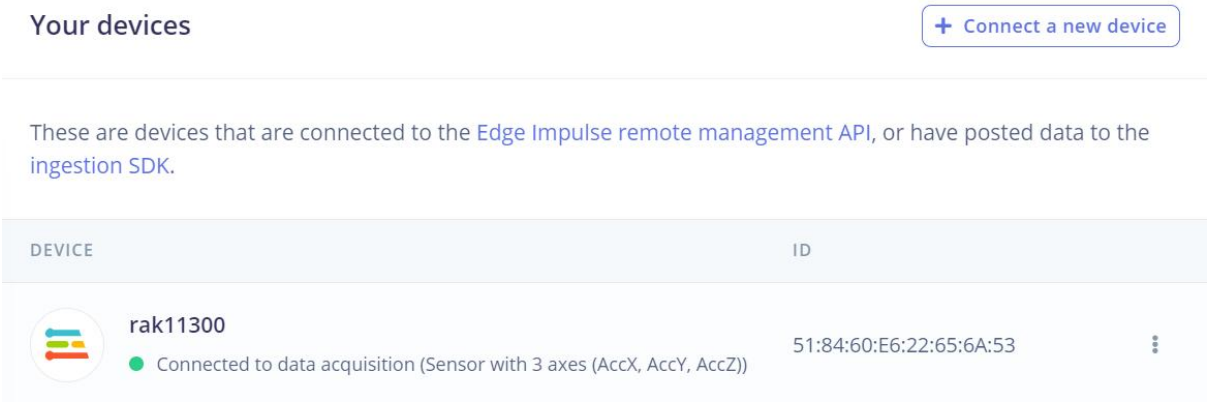
Switching Projects:

- If projects need to be changed (work on different models), the **edge-impulse-data-forwarder --clean** command could be run in the terminal.
- The command clears the current project configuration, allowing you to link to a new project on Edge Impulse.

Model Training:

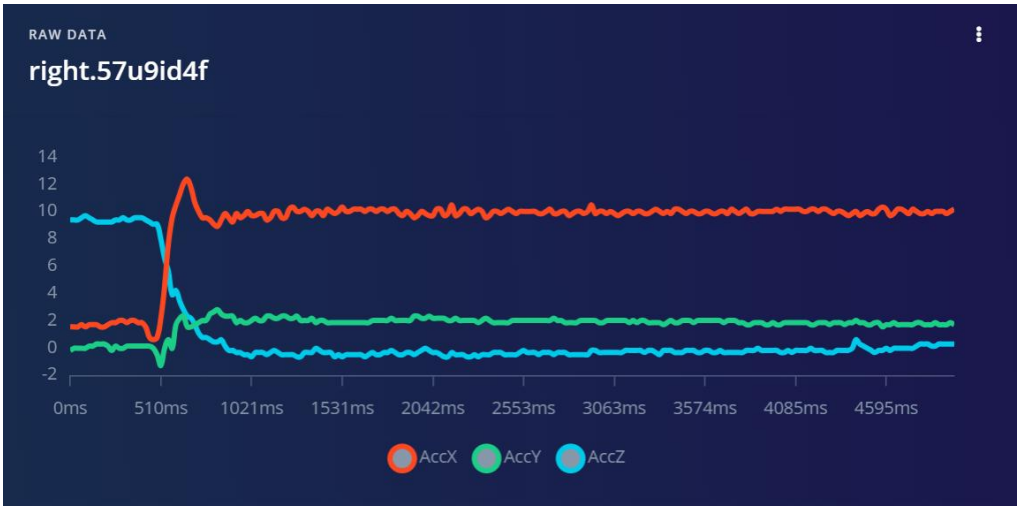
1. Device Connection

After setting up the RAK11300 with Edge Impulse, ensure that the device is connected. Verify this by checking the **Devices** tab in Edge Impulse Studio. The RAK11300 should appear as a connected device ready to forward data.



2. Data Acquisition

- Navigate to the **Data Acquisition** tab to begin collecting data for the gestures you want to recognize.
- Perform each gesture with the RAK11300 and collect accelerometer data corresponding to different actions. Here are examples of gestures:
  a. **Forward:** Move the RAK11300 from the center position to down.
  b. **Left:** Move the RAK11300 from the center to the left.
  c. **Right:** Move the RAK11300 from the center to the right.
  d. **Idle:** Hold the RAK11300 in the center position without any movement.
- Collect **5-10 minutes** of data for each gesture to ensure you have a balanced and robust dataset. Make sure to correctly **split the data** between the training and test datasets.



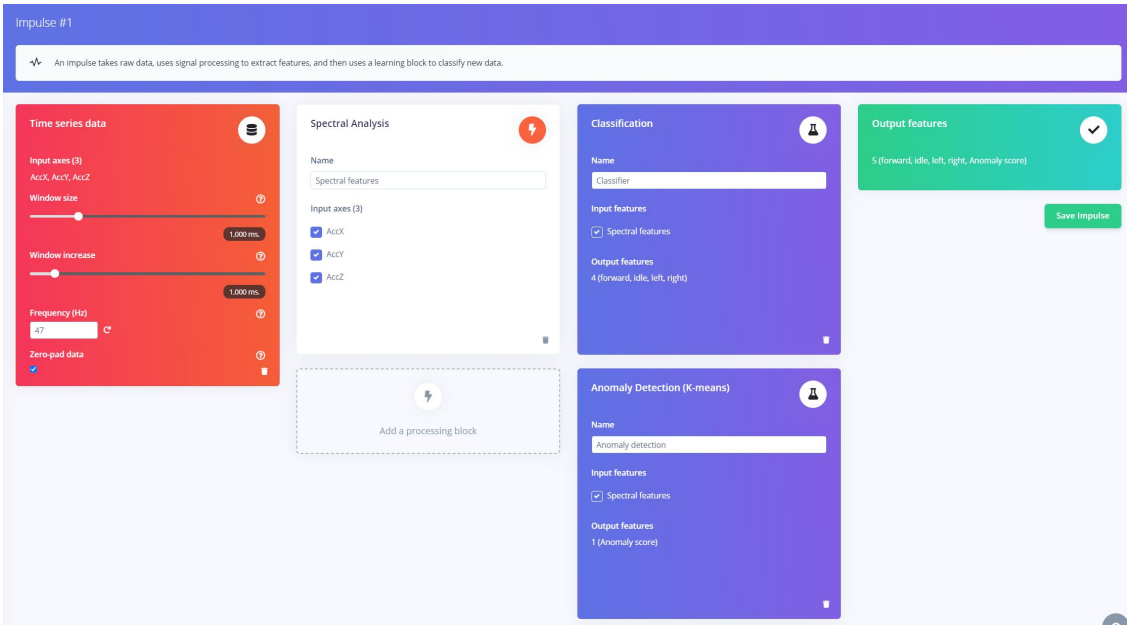
- Collect approximately 5-10 minutes of data and make sure the training and test dataset split is perfect.

3. Create Impulse

In the **Impulse Design** section, navigate to the **Create Impulse** tab to define the structure of the machine learning model.

- Add a **processing block** such as **Spectral Analysis** to process the raw accelerometer data.
- Then, add a **learning block** such as **Classification** to train the model for recognising gestures.
- Optionally, add **Anomaly Detection** for the model to detect unknown or unexpected patterns.

Once the impulse is designed, click **Save Impulse.**

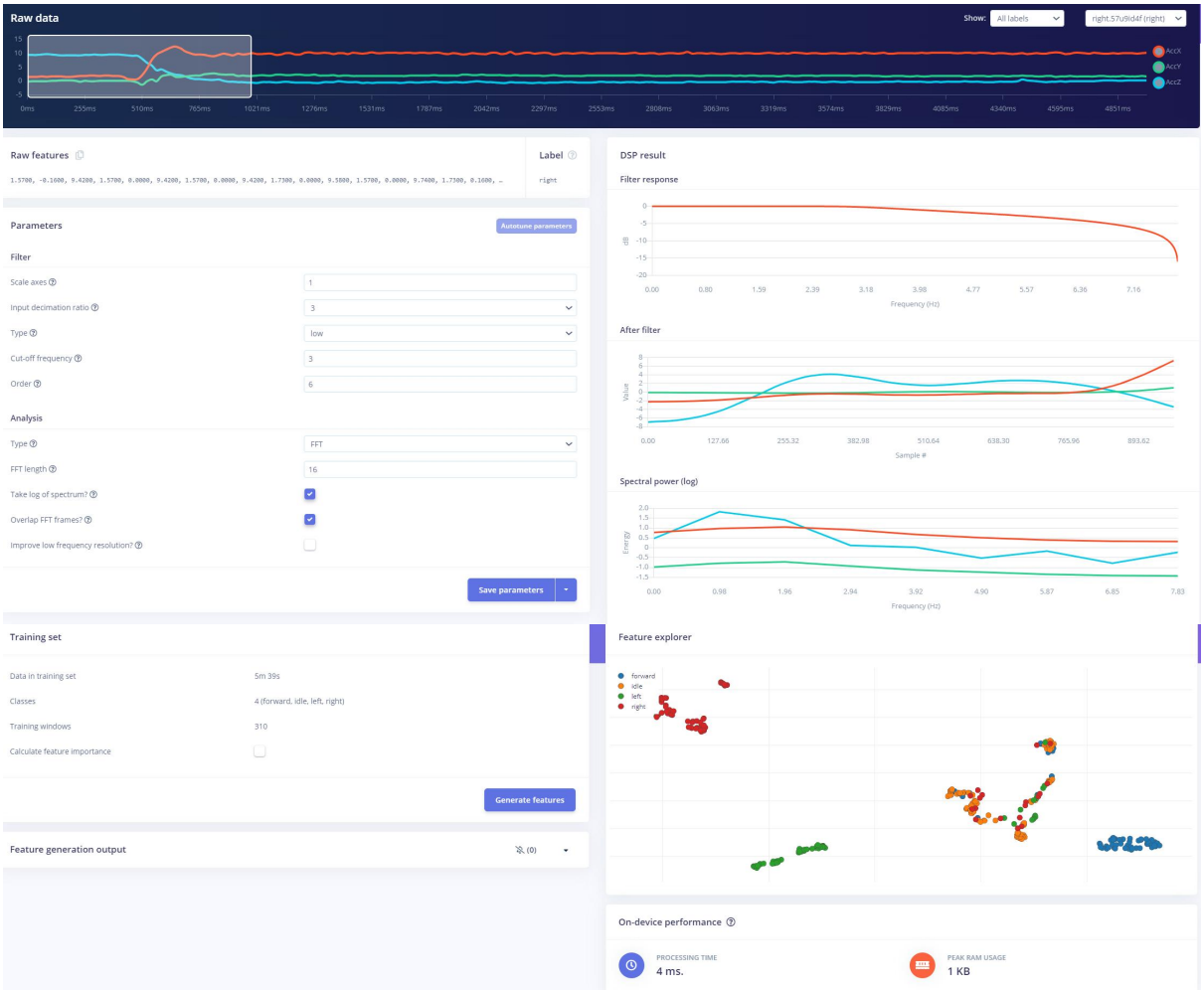


4. Spectral Features

Move on to the **Spectral Features** tab. Adjust the parameters of the DSP (Digital Signal Processing) block.

- Tweak the settings to achieve better results based on the gesture data (e.g., frame size, windowing, etc.). The goal is to ensure that the data for each gesture is properly transformed into spectral features that the model can learn from.

Once satisfied with the DSP results, click **Save Parameters and Generate Features** to apply the settings to the collected data.



5. Model Training

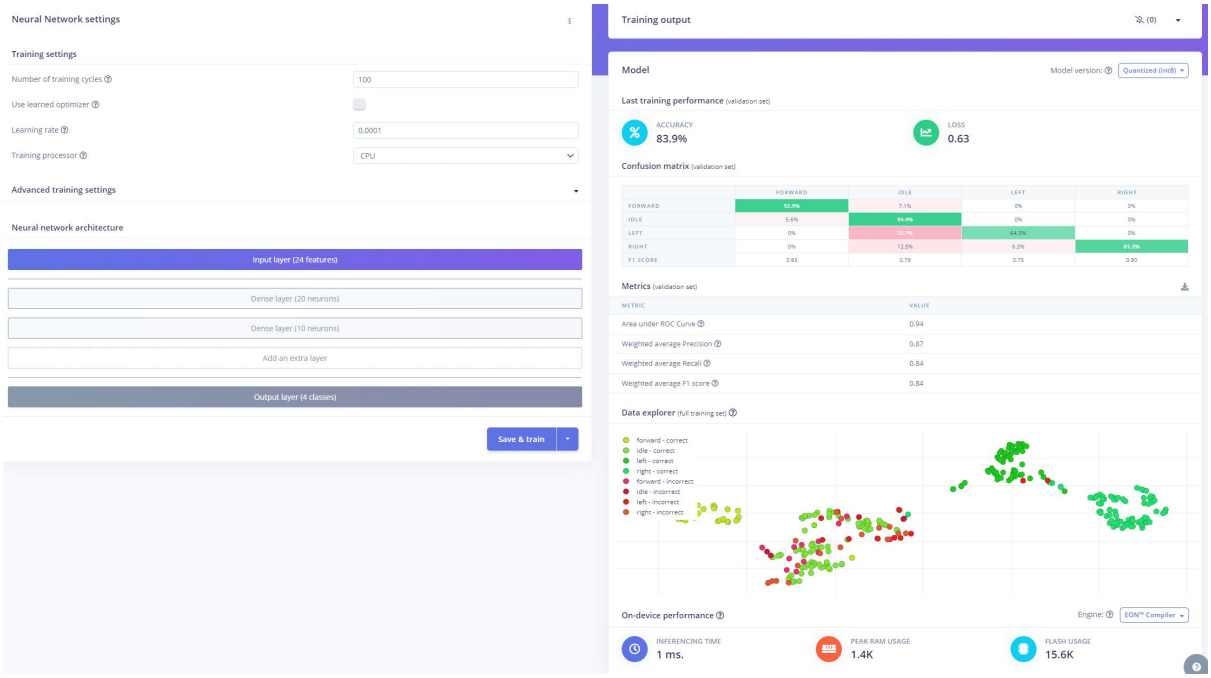
In the **Classifier** tab, adjust the **training settings** such as:

- **Number of training cycles:** Defines how many times the model will iterate over the training data. (Example: 100)
- **Learning rate:** Controls how fast the model learns from the data. (Example: 0.0001)

After adjusting the settings, click **Save & Train** to start the training process.
For this specific gesture recognition model, the training results showed:

- **Accuracy: 83.9%**
- **Loss: 0.63**

If the model's performance is **not satisfactory** (low accuracy or high loss), **tweak the settings or collect more data** to improve it.

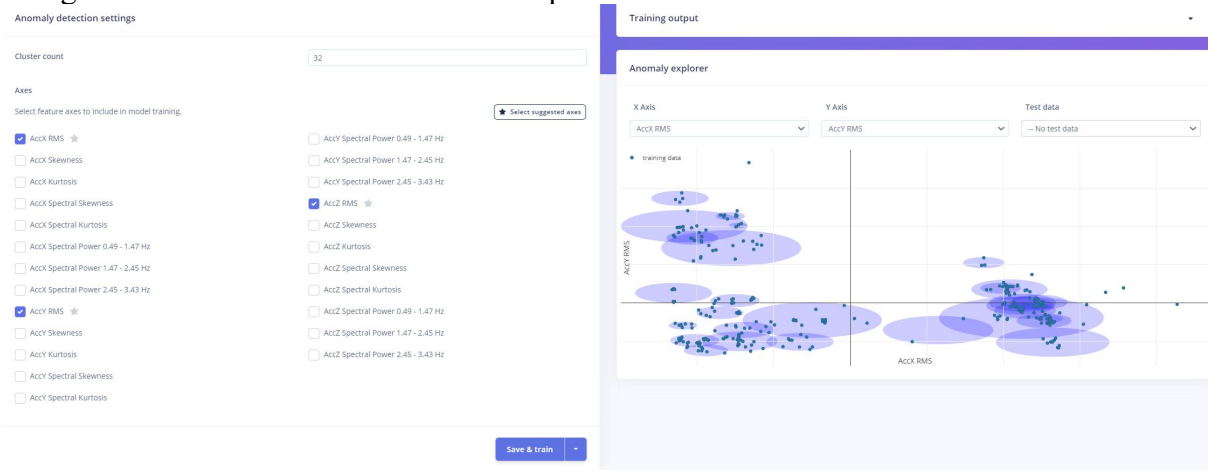


6. Anomaly Detection

Go to the **Anomaly Detection** tab and select the accelerometer data for analysis:

- Choose **AccX RMS, AccY RMS, and AccZ RMS.**

Click **Save & Train** to build the anomaly detection model. This model will help detect any unexpected gestures that do not match the trained patterns.

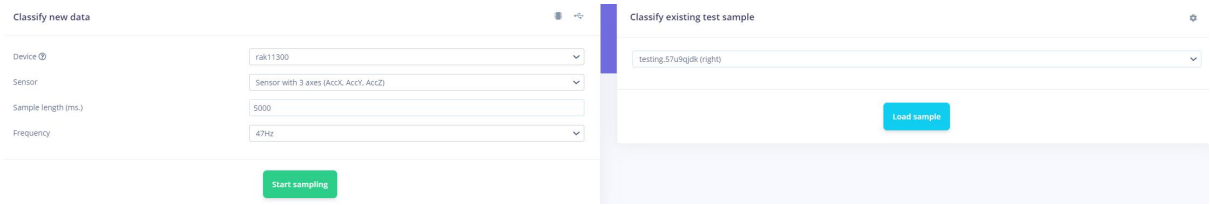


7. Live Classification

The **Live Classification** tab allows the trained model to be tested in real time.

- **Perform the gestures** (forward, left, right, idle) with the RAK11300, and see if the model correctly recognises them.

If the model misclassifies gestures, collect more data or retrain the model to improve its accuracy.

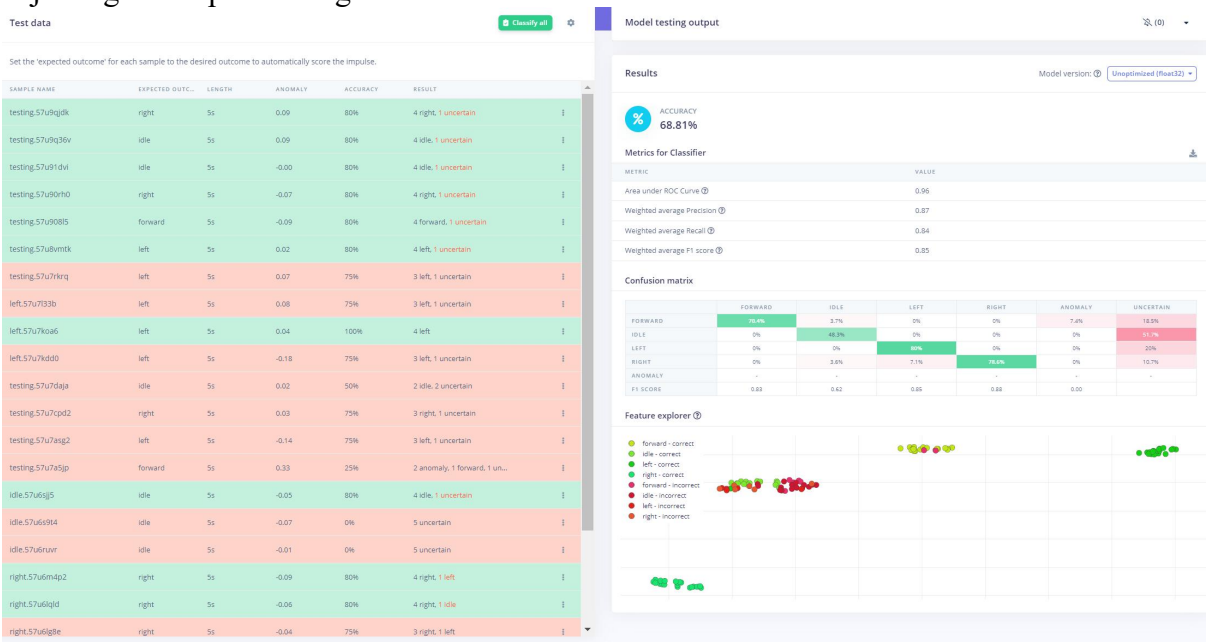


8. Model Testing

The **Model Testing** tab is where the performance of the model could be evaluated on the test dataset.

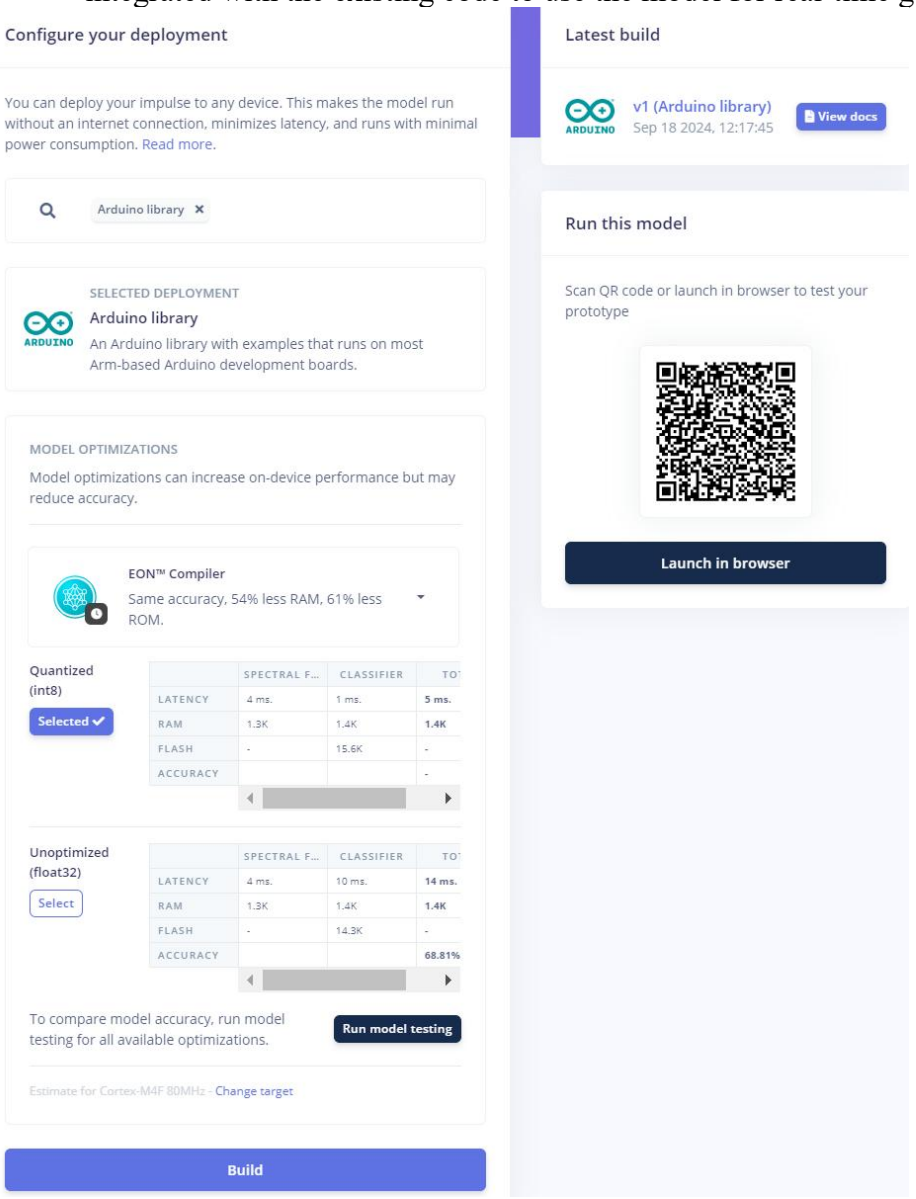
- **Analyse the results** for each test data point and ensure that the model can accurately classify the gestures.

If the test results are not satisfactory, consider retraining the model with more data or adjusting the impulse design.



9. **Deployment**

- In the **Deployment** tab of Edge Impulse Studio, select the deployment option for **Arduino Library.**
- Click the **Build** button, and Edge Impulse will automatically generate a **C++ library** that can be used in the Arduino environment.
- Once the build is complete, a **Zip file** containing the library will be automatically downloaded to the local machine. This file includes the necessary code to run the trained model on the RAK11300 device.
- **Unzip the file** and **import the library** into the Arduino IDE, where it can be integrated with the existing code to use the model for real-time gesture classification.



Reference:

https://docs.rakwireless.com/Knowledge-Hub/Learn/Getting-Started-with-WisBlock-and-Edge-Impulse/RAK11310-Edge-Impulse-Guide/
https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/continuous-motion-recognition

Accessing a Remote Terminal with SSH on Raspberry Pi

SSH (Secure SHell) is a widely used protocol that enables secure access to the command-line interface of a remote device over a network.

1. **Enabling SSH on Raspberry Pi:**

By default, SSH is disabled on Raspberry Pi OS for security reasons. To connect to a Raspberry Pi remotely via SSH, the SSH server must first be enabled.

- **Launch the Raspberry Pi Configuration tool**:
- On the Raspberry Pi desktop, click on the Raspberry icon (top-left corner).
- Navigate to **Preferences** and select **Raspberry Pi Configuration**.
- **Enable SSH:**
- In the Raspberry Pi Configuration window, click on the **Interfaces** tab.
- Find the **SSH** option and select **Enabled.**
- **Apply Settings:**
- Click **OK** to save the changes and exit.

2. **Connecting to Raspberry Pi via SSH:**

Once SSH is enabled, the Raspberry Pi can be connected from another device using a terminal or command-line interface. The Raspberry Pi's IP address and login credentials are needed.

Finding the IP Address

To find the Raspberry Pi's IP address, the **hostname -I** command could be run.

Connecting to the Raspberry Pi

Once the IP address is obtained, open a terminal on Windows with SSH enabled and run the **ssh <username>@<ip_address>** command.

3. **Command Line Access to Raspberry Pi:**

After entering the correct password, the Raspberry Pi command-line prompt should be seen. It will look something like this:

```
PS C:\Users\User> ssh admin@192.168.211.18
The authenticity of host '192.168.211.18 (192.168.211.18)' can't be established.
ED25519 key fingerprint is SHA256:zgHx2V0yJ/C7BW10c5bfurPuw0GvCSckaROgyaXYNuU.
This host key is known by the following other names/addresses:
    C:\Users\User/.ssh/known_hosts:4: 10.100.19.237
    C:\Users\User/.ssh/known_hosts:7: 10.100.27.167
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.211.18' (ED25519) to the list of known hosts.
admin@192.168.211.18's password:
Linux raspberrypi 6.6.20+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.20-1+rpt1 (2024-03-07) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 26 14:33:29 2024
admin@raspberrypi:~ $
```

4. **Closing the SSH Section:**

The SSH connection could be safely closed by typing the **exit** command to terminate the SSH session and return to the local machine's terminal.

Reference:

https://www.onlogic.com/blog/how-to-ssh-into-raspberry-pi/ssh

Remote Access to a Raspberry Pi Folder Using Samba

This section outlines the process for sharing a folder from a Raspberry Pi and accessing it remotely from other devices using the Samba protocol. Samba allows devices on the same network to share files, making it accessible from systems like Windows and macOS.

Steps:

Sharing a Folder from your Raspberry Pi

Firstly, create a folder to share. This example creates a folder called `shared` in the `home` folder of the current user:

```
$ cd ~
$ mkdir shared
$ chmod 0740 shared
```

Now we need to tell Samba about your default user account when accessing that folder. When prompted, enter your password, replacing the `<username>` placeholder with the username of your primary user account:

```
$ sudo smbpasswd -a <username>
```

Now we need to tell Samba to share this folder, using the Samba configuration file.

```
sudo nano /etc/samba/smb.conf
```

At the end of the file, add the following to share the folder, giving the remote user read/write permissions. Replace the `<username>` placeholder with the username of the primary user account on your Raspberry Pi:

```
[share]
    path = /home/<username>/shared
    read only = no
    public = yes
    writable = yes
```

In the same file, find the `workgroup` line, and if necessary, change it to the name of the workgroup of your local Windows network.

```
workgroup = <your workgroup name here>
```

The shared folder should now appear to Windows or macOS devices on the network. Enter your Raspberry Pi username and password to mount the folder.

Reference:

https://www.raspberrypi.com/documentation/computers/remote-access.html

- The trained model could also be deployed in the **C++ library.**
- Once the folder is shared, paste the file folder under **control2_inferencing \src** in the zip file to the shared folder.
- Then, follow the **steps** on the website below:

https://docs.edgeimpulse.com/docs/run-inference/cpp-library/deploy-your-model-as-a-c-library

Outcome:

The inference could be shown in the Raspberry Pi terminal.

```
admin@raspberrypi:~ $ ls
Bookshelf                 build-Test-Desktop-Release  Downloads             Music     shared     Test2
build-Test-Desktop-Debug  Desktop                     mjpg-streamer-master  Pictures  Templates  Videos
build-Test-Desktop-Profile  Documents                 mjpg-streamer.zip     Public    Test       wiringpi
admin@raspberrypi:~ $ cd shared
admin@raspberrypi:~/shared $ ls
edge-impulse
admin@raspberrypi:~/shared $ cd edge-impulse
admin@raspberrypi:~/shared/edge-impulse $ ls
demo
admin@raspberrypi:~/shared/edge-impulse $ cd demo
admin@raspberrypi:~/shared/edge-impulse/demo $ ls
build  edge-impulse-sdk  main.cpp  main.o  Makefile  model-parameters  tflite-model
admin@raspberrypi:~/shared/edge-impulse/demo $ cd build
admin@raspberrypi:~/shared/edge-impulse/demo/build $ ls
app
admin@raspberrypi:~/shared/edge-impulse/demo/build $ ./app
run_classifier returned: 0
Timing: DSP 0 ms, inference 0 ms, anomaly 0 ms
Predictions:
  idle: 0.00781
  snake: 0.95312
  updown: 0.03906
  wave: 0.00000
Anomaly prediction: -0.193
```

Use two RAK11300, transmitter and receiver respectively to gesture control the Arduino robot car to move forward, left and right.

Transmitter Node

Overview:
The transmitter node detects gestures using a **LIS3DH accelerometer** and classifies the motion using a pre-trained **Edge Impulse model**. Based on the classified gesture, it transmits a command (forward, left, right or idle) to the receiver via **LoRa communication**.

Arduino Code Breakdown:

1. **Libraries:**
- **SparkFunLIS3DH.h** for accelerometer handling.
- **LoRaWan-Arduino.h** for LoRa communication.
- **edge-impulse-sdk/classifier/ei_run_classifier.h** for Edge Impulse model inference.

2. **Gesture Detection:**
- **LIS3DH Sensor**: The accelerometer captures **x, y, and z-axis** data. The values are processed through the Edge Impulse model.
- **Edge Impulse Model:** Classifies the data into four possible gestures: **forward, left, right, and idle.** Based on the confidence scores, the gesture is determined.

3. **LoRa Setup:**
- Configured with appropriate settings such as frequency (868.3 MHz), power (22 dBm), bandwidth, and spreading factor.
- On detecting a valid gesture, the system sends the gesture name via **LoRa** using **Radio.Send().**

4. **Data Flow:**
- Accelerometer readings **(x, y, z)** are stored in a buffer.
- Once the buffer fills, it runs the inference to classify the gesture.
- The detected gesture is then sent to the receiver node via LoRa.

Receiver Node

Overview:
The receiver node listens for commands transmitted by the gesture detection node. Upon receiving a valid command (forward, left, right), it controls the robot's movements accordingly. It does this by interpreting the received LoRa message and sending the command to the Arduino-based robot car system.

Arduino Code Breakdown:

1. **Libraries:**
- **LoRaWan-Arduino.h** for LoRa communication.
- **SPI.h** for interfacing with LoRa.

2. **LoRa Setup:**
- Similar LoRa parameters are used as in the transmitter: frequency, bandwidth, spreading factor, etc.
- Configured to listen indefinitely for new messages using **Radio.Rx().**

3. **Message Reception:**
- The **OnRxDone()** callback function is triggered when a message is received.
- The payload (gesture command) is copied into a buffer and the command is displayed on the serial monitor.
- The received command could be passed to motor control functions (not included in the code) that drive the robot's motors based on the command (e.g., move forward, turn left, or right).

4. **Error Handling:**
- **OnRxTimeout()** is called if the node does not receive any messages within a certain period.
- **OnRxError()** handles any errors during reception and restarts the radio to listen again.

Test Outcome

- The transmitter node was programmed with the Edge Impulse gesture recognition model to detect basic gestures: forward, left, right, and idle.
- These gestures were then transmitted to the receiver node using LoRa.

Transmitter: Detects gestures and prints them on the serial monitor.

```
15:02:48.711 -> Sending gesture via LoRa: forward
15:02:49.660 -> Gesture: forward
15:02:49.660 -> Sending gesture via LoRa: forward
15:02:50.595 -> Gesture: forward
15:02:50.595 -> Sending gesture via LoRa: forward
15:02:51.527 -> Gesture: idle
15:02:51.527 -> Sending gesture via LoRa: idle
15:02:52.491 -> Gesture: idle
15:02:52.491 -> Sending gesture via LoRa: idle
15:02:53.438 -> Gesture: idle
15:02:53.438 -> Sending gesture via LoRa: idle
15:02:54.358 -> Gesture: right
15:02:54.358 -> Sending gesture via LoRa: right
15:02:55.306 -> Gesture: right
15:02:55.306 -> Sending gesture via LoRa: right
15:02:56.259 -> Gesture: right
15:02:56.259 -> Sending gesture via LoRa: right
```

Receiver: Receives the gesture via LoRa and prints it on its serial monitor.

```
15:02:48.712 -> Sending gesture via LoRa: forward
15:02:49.662 -> Gesture: forward
15:02:49.662 -> Sending gesture via LoRa: forward
15:02:50.595 -> Gesture: forward
15:02:50.595 -> Sending gesture via LoRa: forward
15:02:51.528 -> Gesture: idle
15:02:51.528 -> Sending gesture via LoRa: idle
15:02:52.492 -> Gesture: idle
15:02:52.492 -> Sending gesture via LoRa: idle
15:02:53.439 -> Gesture: idle
15:02:53.439 -> Sending gesture via LoRa: idle
15:02:54.358 -> Gesture: right
15:02:54.358 -> Sending gesture via LoRa: right
15:02:55.307 -> Gesture: right
15:02:55.307 -> Sending gesture via LoRa: right
15:02:56.260 -> Gesture: right
15:02:56.260 -> Sending gesture via LoRa: right
```

Node-RED dashboard for Robot Car Gesture Control (Gesture Control)

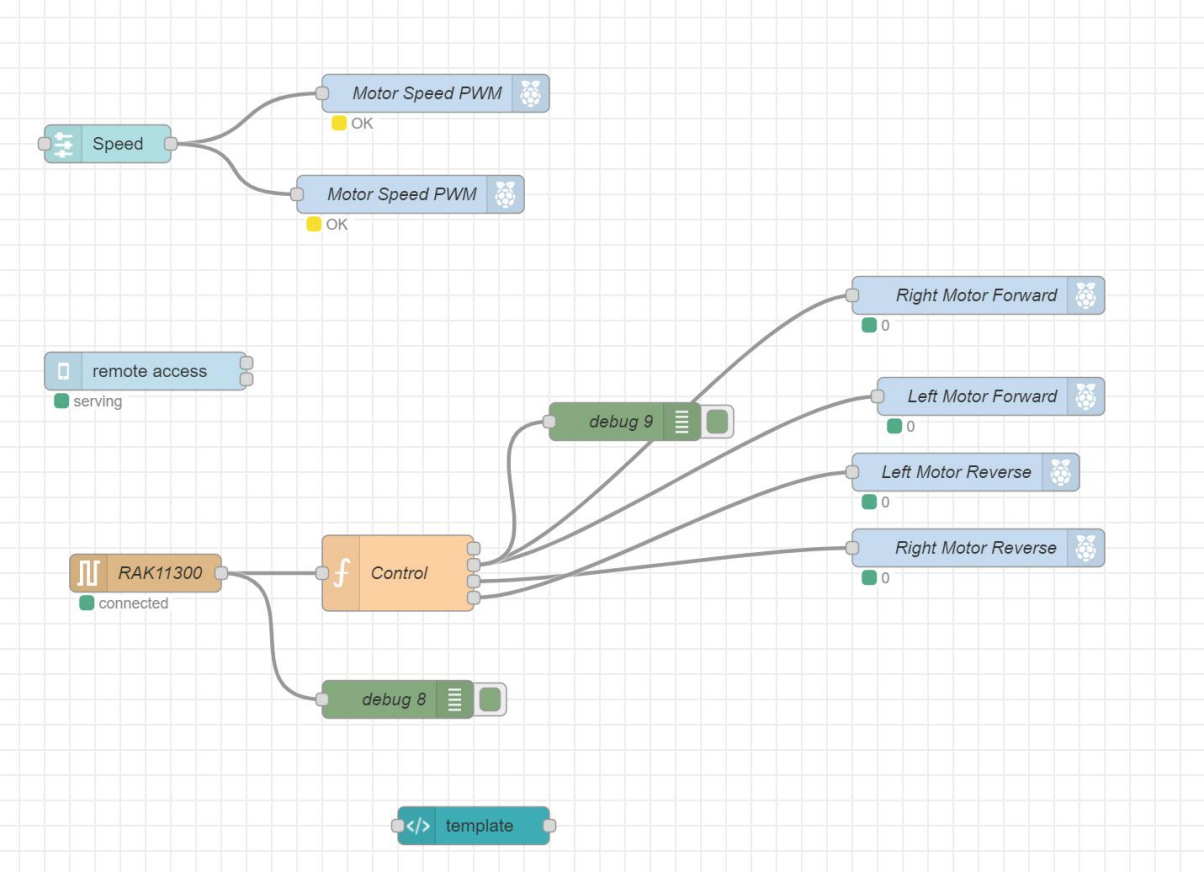
Overview:
It is designed to control a Raspberry Pi-powered motor car using gesture commands transmitted from a **RAK11300** LoRa receiver node. The gestures such as **forward, left, right, and idle** are recognized and used to control the car's movement. Additionally, the flow includes a camera feed integrated into the dashboard for live monitoring of the car's environment.

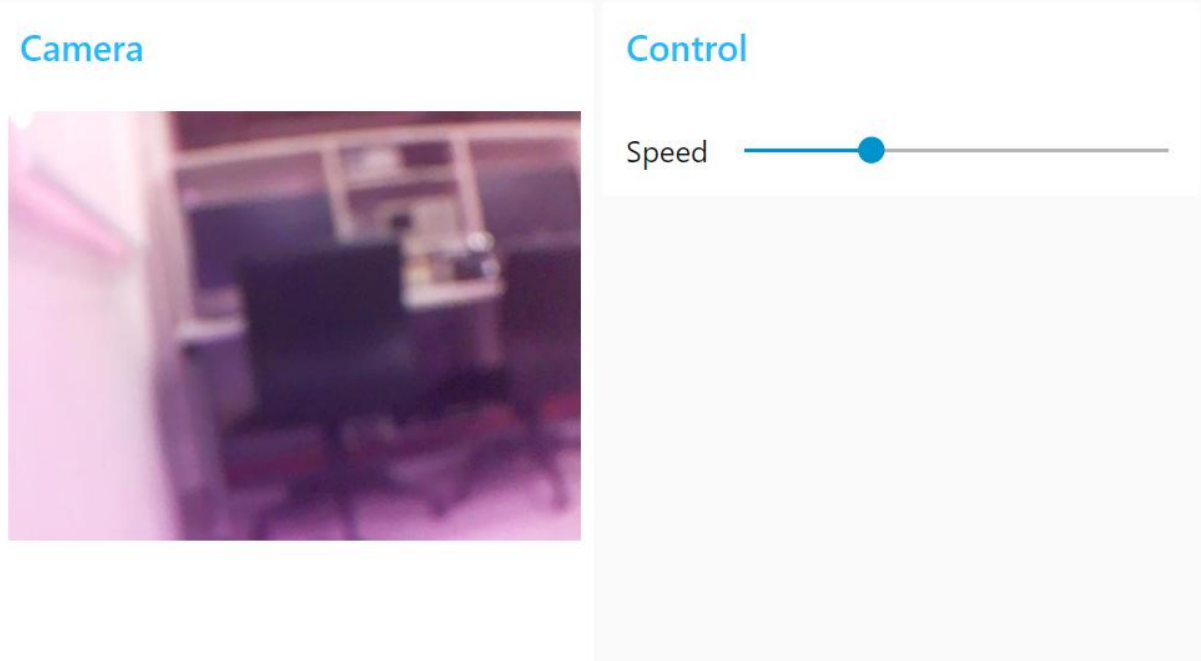
System Components:
- **Gesture Receiver Node**: The RAK11300, connected via serial to the Raspberry Pi, receives gesture commands from the 3-axis accelerometer.
- **Motor Control**: GPIO pins on the Raspberry Pi are used to control motor speed and direction via Pulse Width Modulation (PWM).
- **Node-RED Dashboard**: The control interface is presented through a web-based dashboard, displaying a live camera feed and manual controls for speed adjustment.
- **Serial Communication**: The Raspberry Pi communicates with the RAK11300 via a serial interface to receive gesture commands and execute them.

Gesture Command Interpretation:
- A function node named **Control** reads the received gesture from the RAK11300 node. The possible gestures are **forward, left, right, and idle.**
- Depending on the gesture, it adjusts the GPIO outputs to move the motors accordingly:
    a. **Forward**: Both left and right motors move forward.
    b. **Left**: The left motor moves backward, enabling a left turn.
    c. **Right**: The right motor moves backward, enabling a right turn.
    d. **Idle**: All motors stop.
- The outputs from this function are sent to the relevant motor control pins (GPIO pins 6, 27, 26, and 22).

Serial Input (RAK11300):
- A **serial in** node captures gesture commands from the RAK11300 via **/dev/ttyACM0** at a baud rate of **115200.**
- The data received is passed through the **Control** function node, where it is interpreted and converted into motor control commands.

Node-RED Dashboard (Gesture Control)


Speed
Motor Speed PWM
OK
Motor Speed PWM
OK
remote access
serving
RAK11300
connected
Control
debug 9
debug 8
template
Right Motor Forward
0
Left Motor Forward
0
Left Motor Reverse
0
Right Motor Reverse
0


Camera
Control
Speed

Week 14 (30/9)

- Conclude and finalise all.

Details:

Problems that I Encountered

Problem 1: Motor Battery Consumption

- **Description:** The motors consume significant power, causing the 9V battery to drain quickly, reducing the runtime of the robot car.
- **Solution:** Use **four 1.5V AA batteries** in series instead of a single 9V battery. AA batteries provide more capacity (mAh) than 9V batteries, allowing for longer operation without needing to replace or recharge the batteries frequently.
- **Elaboration:** The higher capacity of AA batteries ensures a more stable power supply for the motors, reducing the risk of voltage drops that could affect motor performance. Additionally, using multiple batteries in parallel could further increase capacity and runtime.

Problem 2: Raspberry Pi Overheating During Camera Streaming

- **Description:** The Raspberry Pi tends to overheat when streaming video, especially when running multiple tasks or for extended periods.
- **Solution:** Offload some tasks to another microcontroller like **ESP32**, which can handle simpler operations like sensor data processing or communication. Consider adding a **heat sink or fan** to the Raspberry Pi for improved cooling.
- **Elaboration:** Distributing tasks between the Raspberry Pi and ESP32 not only reduces the heat load but also enhances system efficiency. By assigning non-critical tasks to the ESP32, the Raspberry Pi can focus on more intensive processes like video streaming, reducing thermal stress. An active cooling system (fan) or passive (heat sink) can help maintain optimal operating temperatures, preventing throttling or shutdowns due to overheating.

Problem 3: GPS Module Not Receiving Signal Indoors

- **Description:** The Neo-6M GPS module does not blink indoors because it is unable to acquire a GPS signal due to interference from walls and ceilings.
- **Solution:** Use the GPS module outdoors where it has a clear line of sight to satellites. For indoor testing, consider using a **GPS simulator or a GPS repeater.**
- **Elaboration:** GPS modules need unobstructed access to satellites to calculate their position accurately. In environments like urban canyons or buildings, signals are weak or non-existent. A GPS repeater can extend outdoor signals indoors, while a GPS simulator can allow for testing in confined spaces by mimicking satellite signals.

Problem 4: Gesture Control Delay

- **Description:** The current gesture control system has noticeable delays when processing input, causing slow response times.
- **Solution: Retrain the machine learning model** with more data, improve its accuracy, and optimize the gesture recognition algorithm to reduce latency.
- **Elaboration:** Delays in gesture control can result from a combination of low model accuracy, poor data quality, or inefficient algorithms. By collecting a larger, more diverse dataset and retraining the model, the detection speed and accuracy of gestures could be improved. Additionally, optimizing the code, such as reducing model complexity or running the algorithm on faster hardware, can further reduce delays.

Problem 5: Node-RED Connectivity and Location Limitation

- **Description:** Using Node-RED connected to a private hotspot or local ethernet limits the geographical range, reducing flexibility in applications that require mobility.
- **Solution:** Connect the system to the **university's WiFi network**, which offers a broader range and consistent internet access.
- **Elaboration:** Connecting to a larger WiFi network provides greater coverage and mobility for the system. This solution is especially beneficial in projects where data needs to be accessed remotely or from different physical locations. Alternatively, using a mobile hotspot or cellular connection could also increase flexibility when off-campus.

Problem 6: Limited Access to Raspberry Pi Display

- **Description:** Only having one local machine makes it difficult to directly access and control the Raspberry Pi's display for debugging or monitoring purposes.
- **Solution:** Use **RealVNC Viewer or SSH** to remotely connect to the Raspberry Pi. This allows full control and access to the Raspberry Pi's interface from another device.
- **Elaboration:** RealVNC Viewer enables graphical access to the Raspberry Pi's desktop, while SSH allows for command-line access, both of which eliminate the need for direct physical interaction with the Raspberry Pi. These methods are especially useful when the Raspberry Pi is placed in hard-to-reach locations or when working on headless setups.

Problem 7: Limited Power Supply for Portable Projects

- **Description:** Portable projects using the Raspberry Pi and peripherals like motors, sensors, and cameras require a reliable power source.
- **Solution:** Use a **portable power bank** with sufficient capacity to power the Raspberry Pi and other connected components.
- **Elaboration:** A high-capacity power bank can offer a stable power supply, allowing the project to be mobile without being tethered to an outlet. Ensure the power bank has enough output current (at least 2.5A) to power all the devices connected to the Raspberry Pi, including peripherals.

Problem 8: LoRaWAN Data Transmission Delay

- **Description:** Data transmitted via LoRaWAN experiences some delay due to the low data rate and long-range communication protocol.
- **Solution:** Use appropriate data rates and optimize message payload sizes. Additionally, consider setting up an **alternative communication method** for time-critical data.
- **Elaboration:** LoRaWAN is ideal for low-power, long-range communication, but it’s not suited for real-time applications due to its latency. Minimizing message size and optimizing the data rate can reduce delays, and using dual communication systems, like WiFi or cellular, for time-sensitive information could be an alternative approach.

Problem 9: Limited Processing Power for Machine Learning Models

- **Description:** The Raspberry Pi may struggle to process complex machine learning models, leading to slow performance.
- **Solution: Offload model inference to a more powerful device**, such as a Jetson Nano, or optimize the model using TensorFlow Lite for edge computing.
- **Elaboration:** For resource-constrained devices, using lighter versions of machine learning models (e.g., TensorFlow Lite) can maintain good accuracy while significantly reducing computational overhead. Alternatively, using dedicated hardware like the Jetson Nano allows more complex models to run efficiently.

Problem 10: Inconsistent Data Logging

- **Description:** Data logged from sensors and the camera stream can be inconsistent or incomplete due to network issues or Raspberry Pi performance limitations.
- **Solution:** Implement buffering and error-checking mechanisms in the data logging process.
- **Elaboration:** Adding data buffers and retry mechanisms ensures that if the connection drops or the Raspberry Pi temporarily underperforms, data is not lost. This can be particularly important in real-time monitoring applications.

Conclusion

During this summer internship, I had the opportunity to work on diverse and cutting-edge technologies, including **LoRa**, **drones**, **IoT**, **robotics**, **Node-RED**, and **machine learning**. Through hands-on experience, I developed a deeper understanding of these fields and their applications in real-world scenarios.

LoRa

LoRa (Long Range) communication emerged as a key solution for enabling wireless communication in rural or hard-to-reach areas. Throughout the internship, I explored how LoRa networks, such as The Things Network (TTN), could be used to transmit data from remote nodes to a central system, with a focus on low power consumption and long-range capabilities. I also gained insights into payload optimization, managing communication delays, and using tools like Wireshark to verify packet forwarding.

Drones

Working with drones, particularly on the integration with PX4 and QGroundControl (QGC), provided valuable experience in autonomous vehicle operation and telemetry systems. I explored how drones could be controlled and monitored using MAVLink and how LoRa could be employed to communicate data from remote sensors to a base station. This laid the foundation for future work in drone communication using LoRa for long-range applications, especially in rural or underdeveloped areas.

IoT (Internet of Things)

The internship involved building IoT systems that integrated sensors, wireless communication (LoRa), and data processing platforms such as Node-RED. By working on the design and implementation of IoT nodes, I enhanced my understanding of the sensor-to-cloud pipeline, which is essential for smart systems like remote monitoring, industrial automation, and environmental sensing.

Robotics

In the domain of robotics, I worked on a variety of projects, including gesture-controlled robots and Raspberry Pi-based car systems. I explored how sensor data, including data from accelerometers, could be processed and used to control robotic movements via gesture recognition. The integration of IoT and robotics showed me the vast potential for creating interconnected and autonomous systems that can operate remotely.

Node-RED
Node-RED played a central role in visualising and managing data from various sensors and systems. I used it to create dashboards for monitoring real-time data, controlling devices, and building IoT workflows. Through this, I learned how to build flexible, low-code IoT applications and integrate them with external APIs such as Google Maps and Google Sheet, as well as leverage the power of MQTT for real-time communication.

Machine Learning
In the field of machine learning, I focused on training models for gesture recognition using the Edge Impulse platform. This experience gave me hands-on practice in data collection, model training, and optimization, particularly in embedded systems. By deploying models on low-power hardware like the RAK11300 with a 3-axis accelerometer, I gained a better understanding of edge AI and how to build intelligent systems that process data locally.

Acknowledge and Gratitude

I would like to take this opportunity to express my heartfelt gratitude to my mentor, Melissa and Dr Jing Huey for their invaluable guidance and support throughout my internship. Your expertise and encouragement played a crucial role in my learning and development during this journey.

Thank you for sharing your knowledge and insights, helping me navigate challenges, and inspiring me to push my boundaries. Your mentorship has greatly enhanced my understanding of technologies such as LoRa, drone systems, IoT, robotics, Node-RED, and machine learning. I appreciate the time you took to explain complex concepts and the constructive feedback you provided on the projects.

This internship has been a transformative experience, and I am truly grateful for the chance to learn from you. I look forward to applying the skills and knowledge I have gained under your mentorship in my future endeavours.

Thank you once again for everything!