Hellenic Mediterranean University

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Internet of Things: Technologies and Applications

Final Project Report

Koutsourelakis Charilaos, MSc,

MTP323

Table of Contents

Introduction

This report is the continuation of the previous interim report in the subject ***Internet of Things: Technologies and Applications***, which covered the fields of Sleep and Low Power modes, BLE, Bluetooth, FreeRTOS, SPIFFS, OTA and the overall self-hosted server, along with the motor control and functions of the car, in the ESP32 development board.

Docker and Containers

Docker is a tool that allows us to run applications inside "containers." A container includes everything the app needs to work, like code and settings, but it runs isolated from the rest of the system. This makes it easy to manage and move applications across different machines. Containers are lightweight because they share the host system's operating system, unlike virtual machines that need a full OS for each app.

Node-Red

InfluxDB is a time-series database, meaning it's designed to store data that changes over time, like sensor readings. In my setup, InfluxDB is another container that stores the data coming from Node-RED. When Node-RED receives sensor information from the ESP32, it forwards that data to InfluxDB, which keeps it for later use. Since both containers are part of the same Docker network, data flows smoothly between them.

InfluxDB

InfluxDB is a time-series database, meaning it's designed to store data that changes over time, like sensor readings. In my setup, InfluxDB is another container that stores the data coming from Node-RED. When Node-RED receives sensor information from the ESP32, it forwards that data to InfluxDB, which keeps it for later use. Since both containers are part of the same Docker network, data flows smoothly between them.

Grafana

Grafana is a tool used to display data in graphs and charts. It runs in a Docker container and connects to InfluxDB to pull stored data. Using Grafana, I can create dashboards to visualize the sensor data coming from the ESP32. Grafana accesses the data stored in InfluxDB, and everything happens within the same Docker network, making it easy to display and analyze the data in real time.

ESP32 & Docker Communication

The ESP32 is connected to the same network as my Docker containers. It sends data over MQTT to the Node-RED container, which then passes it to InfluxDB for storage. Grafana later pulls this data from InfluxDB to create visual dashboards. Since all the containers and the ESP32 are on the same network, they communicate efficiently without any issues.

Docker & Containers: How to

First, we need to enable Docker in our host machine, this is done with the following commands:

Note: In my case the host OS is GNU/Linux, but the commands are almost identical for Windows, except for the `sudo` flag.

- Installation of docker and stating the docker daemon

```
sudo pacman -S docker
sudo systemctl enable docker.service
sudo systemctl start docker.service
```

- Adding user to the docker group

```
sudo usermod -aG docker $USER
```

- Use this command to re-source the changes on the machine (avoids logging out for effects to take place)

```
newgrp docker
```

Docker should be now running in the background.

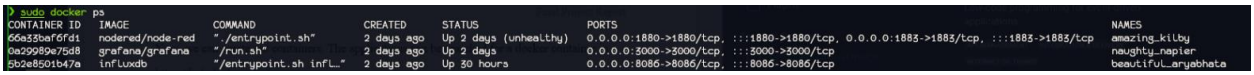
Now, we can create out containers. The applications can be searched for a docker container in

https://hub.docker.com/

If it exists, there is a docker container for it.

```
sudo docker run -p 8086:8086 influxdb
sudo docker run -d -p 1880:1880 -p 1883:1883 nodered/nodered
sudo docker run -d -p 3000:3000 grana/grafana
```

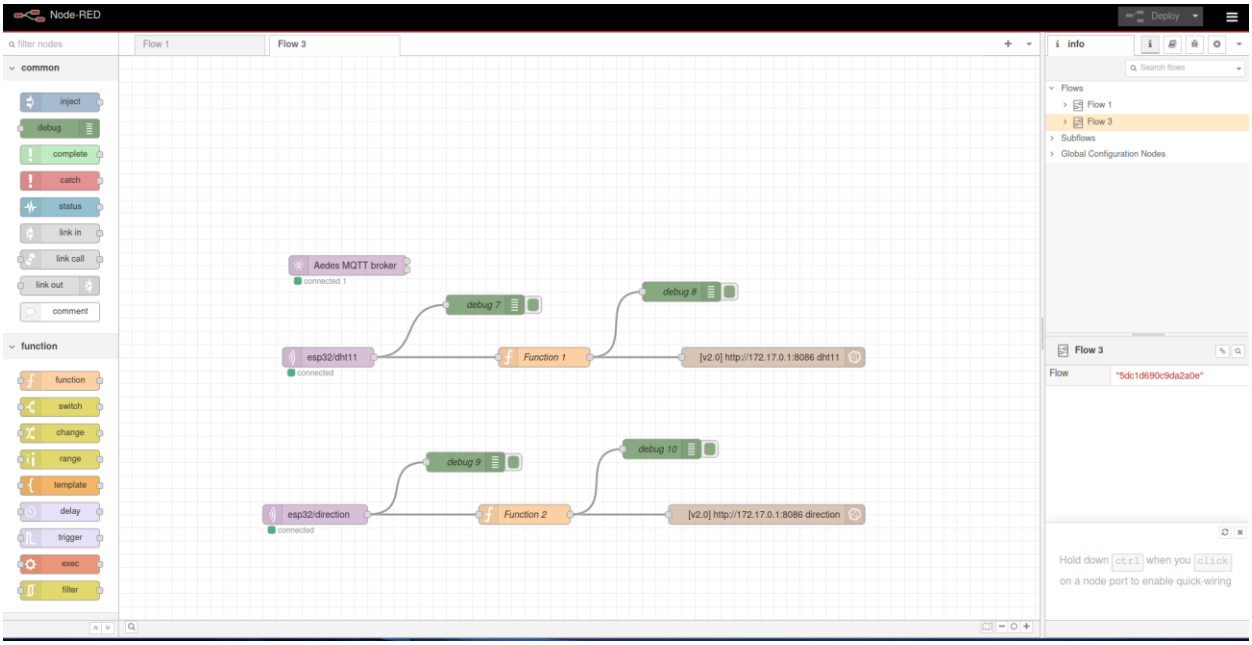
Once every container has been downloaded and run, we should see something similar:

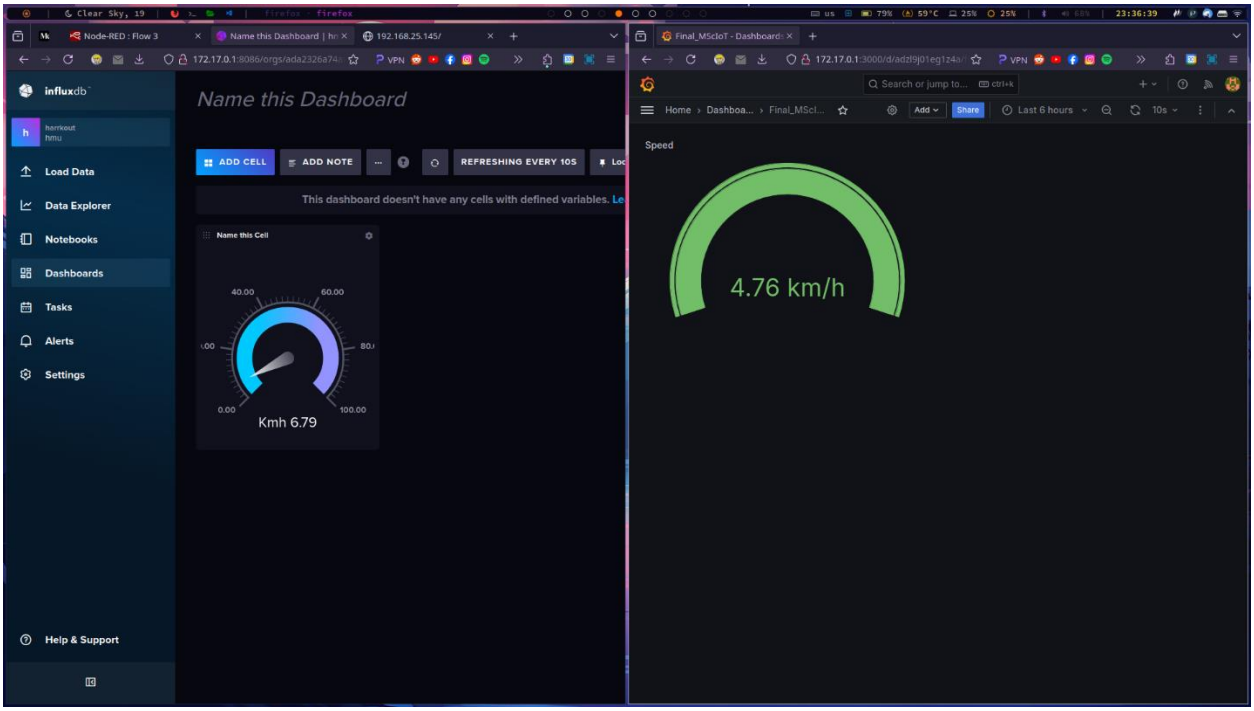


Or we can use a TUI (Terminal User Interface) for a more detailed and easy-of-use UI, like ***oxker***.

Next the Node-Red, InfluxDB and Grafana self-hosted services are shown, along with the flows and their WebSockets.

Note that when Docker is initialized, it creates a local IP inside the host network. So our services (InfluxDB and Grafana) need to be added to the Node-Red with the like: ***<docker-ip>:port*** (as seen in the image). The Docker networking part needs further research though.

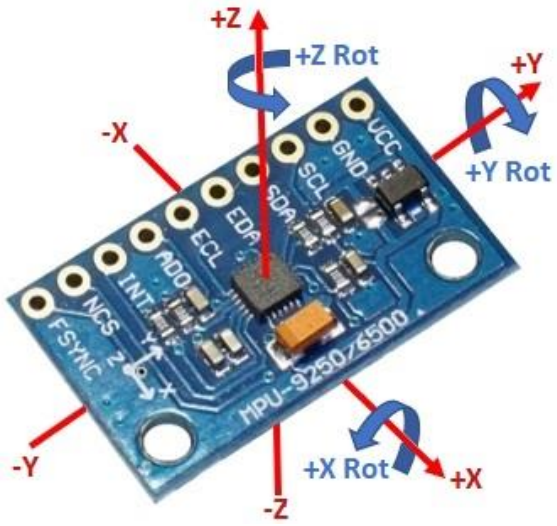




MPU9250

The device used for the motion mobilities' calculations of the car is the Motion Processing Unit (MPU) 9250. The MPU-9250 is a versatile 9-axis Motion Processing Unit (MPU) that integrates a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer into a single chip. This allows it to accurately track motion and orientation, making it ideal for applications requiring precise motion sensing, such as robotics, drones, and wearable devices. The MPU-9250 also features a Digital Motion Processor (DMP) that can handle complex sensor fusion algorithms, reducing the computational load on the main processor. It communicates via I²C or SPI, offering flexibility for integration into various systems.

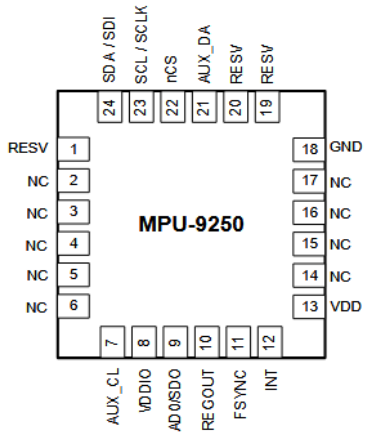
In the case of this project, only the 3-axis gyroscope of the MPU-9250 was used to measure the angular velocity of the vehicle. This data helps calculate the car's rotational movement, enabling precise control overturning and orientation adjustments. The gyroscope outputs real-time data on the rate of rotation around the X, Y, and Z axes, which is critical for maintaining stability and executing smooth maneuvers. By focusing on the gyroscope, the project optimizes computational efficiency while still ensuring accurate motion tracking for navigation and control.



4.1 Pin Out and Signal Description

Pin Number	Pin Name	Pin Description
1	RESV	Reserved. Connect to VDDIO.
7	AUX_CL	I²C Master serial clock, for connecting to external sensors
8	VDDIO	Digital I/O supply voltage
9	AD0 / SDO	I²C Slave Address LSB (AD0); SPI serial data output (SDO)
10	REGOUT	Regulator filter capacitor connection
11	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	INT	Interrupt digital output (totem pole or open-drain)
13	VDD	Power supply voltage and Digital I/O supply voltage
18	GND	Power supply ground
19	RESV	Reserved. Do not connect.
20	RESV	Reserved. Connect to GND.
21	AUX_DA	I²C master serial data, for connecting to external sensors
22	nCS	Chip select (SPI mode only)
23	SCL / SCLK	I²C serial clock (SCL); SPI serial clock (SCLK)
24	SDA / SDI	I²C serial data (SDA); SPI serial data input (SDI)
2 – 6, 14 - 17	NC	Not internally connected. May be used for PCB trace routing.

Table 9 Signal Descriptions



4.2 Typical Operating Circuit

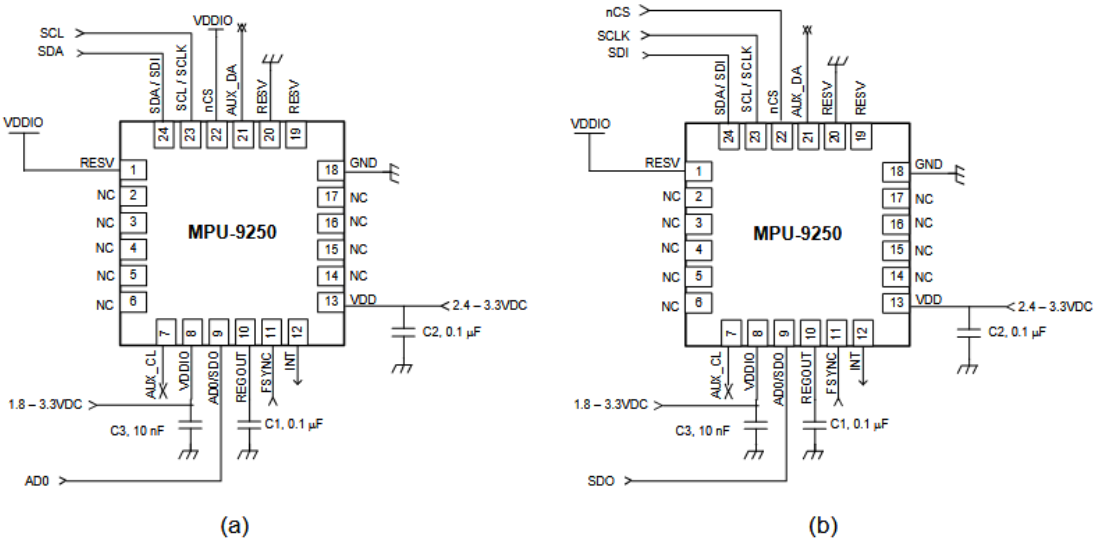


Figure 2 MPU-9250 QFN Application Schematic: (a) I2C operation, (b) SPI operation

The MPU9250 was firmly placed on the upper body of the car for stable motion tracking. However, the placement of the device is irrelevant as long as it is either face-up or face-down, since it measures angular velocity in both positive and negative directions along each axis, ensuring accurate readings regardless of its orientation, as shown in the figure below.

9.1 Orientation of Axes

The diagram below shows the orientation of the axes of sensitivity and the polarity of rotation. Note the pin 1 identifier (•) in the figure.

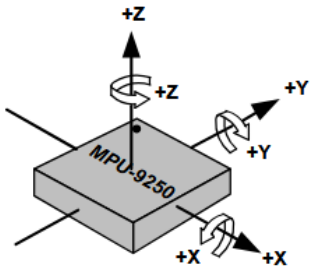


Figure 4. Orientation of Axes of Sensitivity and Polarity of Rotation for Accelerometer and Gyroscope

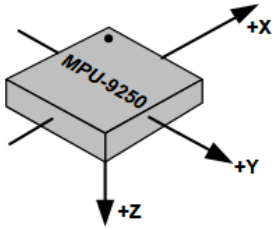
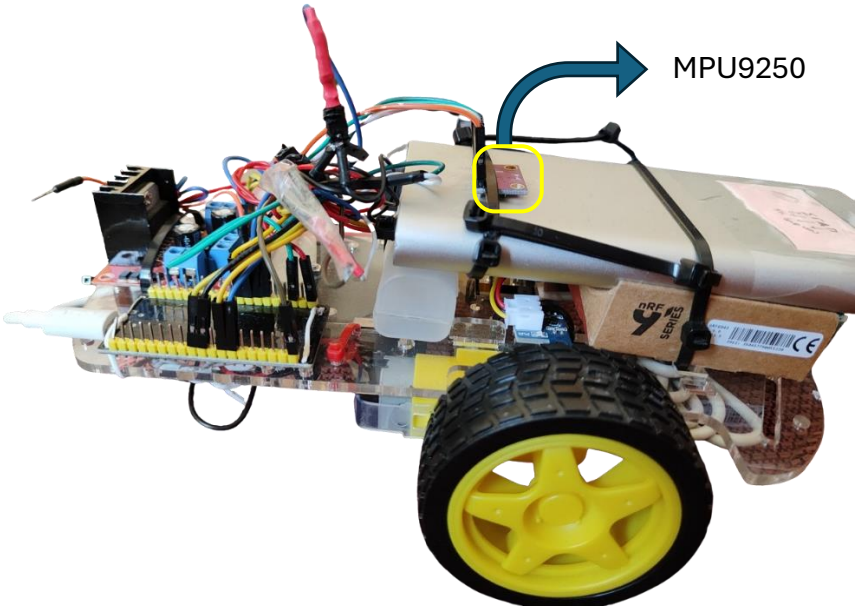


Figure 5. Orientation of Axes of Sensitivity for Compass



Edge AI

In order to implement an Edge AI functionality within the ESP32 with the MPU9250, we need to follow a few steps. First, we need to log the data from the MPU9250.

Below we see a python script (*located at /data/log.py*) that logs the live data from the «***Rover_MPU_DataLogger»*** application in a .csv file named «***imu_gyro_data.csv***».

In the case of the project, 6 datasets were created for the movements:

- Idle
- Forward
- Backward
- Left
- Right
- Wheelie

Note: in every generated dataset, we need to add a label column with the name of the movement, like so:

```
1 | gyro_x  gyro_y  gyro_z  label
2 | -0.3    -0.07   0.22    forward
3 | -0.25   -0.02   0.31    forward
4 | -0.2    0       0.04    forward
5 | -0.18   0.06    0.01    forward
6 | -0.25   0.03    0.04    forward
7 | -0.2    0.05    -0.22   forward
8 | -0.22   0.03    -0.25   forward
9 | -0.25   0.05    0.13    forward
10| -0.19   0.03    0.12    forward
11| -0.14   0.07    0.02    forward
12| -0.15   0       -0.13   forward
13| -0.27   0.07    -0.07   forward
14| -0.25   0.04    -0.08   forward
15| -0.24   0       0.26    forward
16| -0.74   0.07    0.42    forward
```

/data/log.py

```
# author: koutsourelakis

import serial
import csv

# Open serial connection to ESP32
# Linux: ser = serial.Serial('/dev/ttyUSB*', 115200, timeout=2)
# Windows: ser = serial.Serial('COM*', 115200, timeout=2)
# Note: change the asterisk (*) with your port number
try:
    # Add timeout to avoid infinite blocking
    ser = serial.Serial('/dev/ttyUSB2', 115200, timeout=2)
except serial.SerialException as e:
    print(f"Error opening serial port: {e}")
    exit()

# Open CSV file for logging gyro data
with open('imu_gyro_data.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    # CSV header for gyro data
    writer.writerow(['gyro_x', 'gyro_y', 'gyro_z'])

    print("Logging started. Press Ctrl+C to stop.")

    try:
        while True:
            # Read line from serial and decode
            line = ser.readline().decode('utf-8').strip()

            if line:
                print(f"Raw Line: {line}")  # Print raw data for debugging

                # Split the line by spaces
                data = line.split()  # Use default whitespace split

                # Ensure there are 12 values in the output while running the app
                '''
                 ------------------------------
                | No |  Data type  | Function |
                |:--:|:-----------:|:--------:|
                |  0 | new_imu_data |          |
                |  1 | new_mag_data |          |
                |  2 | accel_x_mps2 |          |
                |  3 | accel_y_mps2 |          |
                |  4 | accel_z_mps2 |          |
                |  5 | gyro_x_radps |   USED   |
                |  6 | gyro_y_radps |   USED   |
                |  7 | gyro_z_radps |   USED   |
                |  8 |   mag_x_ut   |          |
                |  9 |   mag_y_ut   |          |
                | 10 |   mag_z_ut   |          |
                | 11 |  die_temp_c  |          |
                 ------------------------------
                '''
                if len(data) == 12:
                    # Extract gyro values by selecting the columns we want to log
                    # Get gyro_x, gyro_y, gyro_z
                    gyro_data = data[5:8]
                    # Write only gyro data to the CSV
                    writer.writerow(gyro_data)
                    # print parsed data to monitor
                    print(f"Logged Gyro Data: {gyro_data}")
                else:
```

```
64|                     print("Unexpected data format")
65|
66|     except KeyboardInterrupt:
67|         print("\nLogging stopped by user.")
68|
69|     except Exception as e:
70|         print(f"Error during logging: {e}")
71|
72|     ser.close()
73|     print("Serial connection closed.")
```

Now that we have the logs ready, we can move onto the generation of our model. This can be done in various ways with various tools, my choice was with the following Python application that as input the 6 datasets created earlier, and with the use of the DecisionTree classifier, a header file will be generated that will contain our hardcoded Machine Learning model.

```
1 | import pandas as pd
2 | from sklearn.model_selection import train_test_split
3 | from sklearn.tree import DecisionTreeClassifier
4 | import joblib
5 |
6 | # Load the four datasets, assuming you have separate CSVs for each direction
7 | forwards_data = pd.read_csv('forward.csv')
8 | left_data = pd.read_csv('left.csv')
9 | right_data = pd.read_csv('right.csv')
10| idle_data = pd.read_csv('idle.csv')
11| wheelie_data = pd.read_csv('wheelie.csv')
12| backwards_data = pd.read_csv('backwards.csv')
13|
14|
15| # Assign labels to each dataset
16| forwards_data['label'] = 'forwards'
17| left_data['label'] = 'left'
18| right_data['label'] = 'right'
19| # Combine the datasets into a single DataFrame
20| imu_data = pd.concat([forwards_data, left_data, right_data, idle_data,
wheelie_data, backwards_data])
21|
22| # Remove leading and trailing whitespace from column names
23| imu_data.columns = imu_data.columns.str.strip()
24|
25| # Check the shape of the combined DataFrame
26| print("Combined DataFrame shape:", imu_data.shape)
27|
28| # Check for missing values
29| print("Missing values in each column:")
30| print(imu_data.isnull().sum())
31|
32| # Convert the label column to numeric for model training
33| imu_data['label'] = imu_data['label'].astype('category').cat.codes
34|
35| # Split the data into features (X) and labels (y)
36| X = imu_data.drop(columns=['label'])
37| y = imu_data['label']
38|
39| # Split the data into training and testing sets
```

```
40| X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
41|
42| # Train a decision tree classifier
43| model = DecisionTreeClassifier()
44| model.fit(X_train, y_train)
45|
46| # Save the model to a file
47| joblib.dump(model, 'decision_tree_model.joblib')
48| print("Model saved as 'decision_tree_model.joblib'.")
49|
50| # Load the model from the file for demonstration
51| loaded_model = joblib.load('decision_tree_model.joblib')
52| print("Model loaded from 'decision_tree_model.joblib'.")
53|
54| # Evaluate the model's accuracy
55| accuracy = loaded_model.score(X_test, y_test)
56| print(f"Accuracy on the test set: {accuracy}")
57|
58| # Example usage: making predictions
59| predictions = loaded_model.predict(X_test)
60| print("First 10 predictions:", predictions[:10])
61| print("First 10 actual labels:", y_test[:10].values)

>>Combined DataFrame shape: (145, 4)
>>Missing values in each column:
>>gyro_x    0
>>gyro_y    0
>>gyro_z    0
>>label     0
>>dtype: int64
>>Model saved as 'decision_tree_model.joblib'.
>>Model loaded from 'decision_tree_model.joblib'.
>>Accuracy on the test set: 0.9310344827586207
>>First 10 predictions: [4 1 3 3 3 5 5 2 2 3]
>>First 10 actual labels: [4 0 3 3 3 5 5 2 2 3]
```

Next we generate the header (.h) file:

```
from micromlgen import port
print(port(model))
```

```
#pragma once
#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        if (x[2] <= 0.9799999743700027) {
                            if (x[2] <= -1.1449999809265137) {
                                return 4;
                            }

                            else {
```

```
                                if (x[0] <= -0.19500000029802322) {
                                    if (x[0] <= -0.20499999821186066) {
                                        if (x[2] <= 0.19500000029802322) {
                                            if (x[1] <= 0.0249999999441206455) {
                                                return 0;
                                            }

                                            else {
                                                if (x[0] <= -0.24000000208616257) {
                                                    return 1;
                                                }

                                                else {
                                                    if (x[0] <= -0.22500000149011612)
{

                                                        return 0;
                                                    }

                                                    else {
                                                        return 1;
                                                    }
                                                }
                                            }
                                        }

                                        else {
                                            return 1;
                                        }
                                    }

                                    else {
                                        if (x[1] <= 0.0049999999888241291) {
                                            if (x[1] <= -0.014999999664723873) {
                                                return 0;
                                            }

                                            else {
                                                if (x[2] <= 0.034999999921768904) {
                                                    return 2;
                                                }

                                                else {
                                                    return 1;
                                                }
                                            }
                                        }

                                        else {
                                            if (x[1] <= 0.044999999925494194) {
                                                return 0;
                                            }

                                            else {
                                                return 1;
                                            }
                                        }
                                    }
                                }

                                else {
                                    if (x[1] <= -0.0049999999888241291) {
                                        if (x[1] <= -0.0249999999441206455) {
                                            return 0;
```

```
                                    }

                                    else {
                                        return 5;
                                    }
                                }

                                else {
                                    if (x[1] <= 0.019999999552965164) {
                                        if (x[1] <= 0.004999999888241291) {
                                            return 1;
                                        }

                                        else {
                                            return 5;
                                        }
                                    }

                                    else {
                                        return 1;
                                    }
                                }
                            }
                        }

                        else {
                            return 3;
                        }
                    }

                protected:
                };
            }
        }
    }
```

Important Note: The decision tree logic generated above (.h file) is directly written as a C++ class in the project and does not depend on any external machine learning library. Details follow below.

Rover_MPU9250_EdgeAI Application

The final application can be found in the folder « Rover_MPU9250_EdgeAI». What is important to note is how the header file that was generated is used.

The following snippet contains every line of code in the application, that uses and utilizes the machine learning model. No external libraries are included in order to use the model and have some output based on the real-time values of the MPU9250.

How it works

Absence of Libraries

In this code, there is no external library utilized for the decision tree model. This means that the implementation is entirely self-contained within the C++ code (.h header file), eliminating the need for any additional machine learning libraries. The decision tree logic is directly embedded in the project, simplifying the integration process.

Creating the Decision Tree Model

The line of code:

```
/* Decision Tree model */
Eloquent::ML::Port::DecisionTree model;
```

creates an instance of the DecisionTree class, which we call ‘model’. This lets you use the ‘predict()’ function to make predictions based on sensor data. In simple terms, this means the decision tree is helping to decide what actions the car should take based on the information it gets.

```
#include "rover.h"

/* Mpu9250 object */
bfs::Mpu9250 imu;
/* Decision Tree model */
Eloquent::ML::Port::DecisionTree model;

.
.
.

/*============ IMU ================
  /* Check if new IMU data is available */
  if (imu.Read() && imu.new_imu_data()) {
      /* Read gyroscope data */
```

```
      float gyroX = imu.gyro_x_radps();
      float gyroY = imu.gyro_y_radps();
      float gyroZ = imu.gyro_z_radps();

      /* Prepare input for the decision tree (only gyro data) */
      float input[3] = { gyroX, gyroY, gyroZ };

      /* Predict direction using the decision tree model */
      int prediction = model.predict(input);

      /* Print gyroscope data */
      Serial.print("Gyro Data: ");
      Serial.print("GyroX: "); Serial.print(gyroX);
      Serial.print("\tGyroY: "); Serial.print(gyroY);
      Serial.print("\tGyroZ: "); Serial.print(gyroZ);

      /* Convert prediction to string */
      switch (prediction) {
          case 0: direction = "Wheelie"; break;
          case 1: direction = "Forward"; break;
          case 2: direction = "Idle"; break;
          case 3: direction = "Turning Left"; break;
          case 4: direction = "Turning Right"; break;
          case 5: direction = "Reverse"; break;
          default: direction = "Unknown"; break;
      }

      Serial.print("\tPredicted Direction: "); Serial.println(direction);
      Delay(1000);

.
.
.
```

Demonstration and Code

In the following links a demonstration video can be found, along with the code repository:

Github: https://github.com/harrkout/Rover_MPU9250_EdgeAI_/

Video: https://www.youtube.com/watch?v=ysg2b4TgAaQ