Manish Subedi

Leveraging Robotics for 5G Test Optimization

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

30 October 2024

Abstract

Author:	Manish Subedi
Title:	Leveraging Robotics for 5G Test Optimization
Number of Pages:	49 pages + 7 appendices
Date:	30 October 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Smart IoT Systems / Embedded Systems
Supervisors:	Saana Vallius, Senior Lecturer Tuomo Rouhiainen, Team Manager

---

The objective of this final year project was to develop a robot to optimize 5G network testing and integrate the robot control mechanism into the internal automation framework of the Nokia Field Verification team. The project was aimed at implementing a simple mobile test system that was interfaced with the automation framework. The API interfaces enable the remote operation of the robot allowing the user to schedule periodic operations.

This thesis elaborates on the application of a simple design in building a mobile test system. The goals were achieved with the development of a FastAPI server deployed on a Raspberry Pi single-board computer. The endpoints of the server were secured with the OAuth2 protocol. The scheduling tool of the automation framework, FiCAT, interacts with the server using REST APIs. Moreover, the STM32 motor controller board was used to control the motors of the robot. A line following algorithm was developed for the motor controller. A USART interface was configured between the Raspberry Pi and the STM32 motor controller to relay the robot control signal sent by FiCAT to the Raspberry Pi server. FiCAT invokes FastAPI client methods with the help of Robot Framework cases deployed in the code base of the automation framework.

The API interfaces were tested before and after the integration of the OAuth2 protocol with Postman ensuring stability and reliability. End-to-end tests were performed to ascertain the successful delivery of the requirements. The system was able to navigate along a line paved through unique network conditions. Furthermore, the system successfully allowed operation and control over the network.

Keywords: Robotics, 5G, STM32 MCU, Raspberry Pi, 5G KPIs

Contents

List of Abbreviations

1 Introduction 1

2 Background 3

2.1 Robots in Industrial Environments 3
2.2 Robot Control Systems 4
2.3 5G Technology 6
2.4 USART 7
2.5 APIs 7
2.6 Sensors 9
2.7 DC Motor 10

3 System Overview 11

3.1 Development Board 13
3.2 Sensor Modules 15
3.3 Motor 19

4 Development Tools 20

4.1 Development Environment 20
4.2 Motor Controller 21
4.3 Robot Framework 21
4.4 FastAPI 22
4.5 3D Design 24

5 Implementation 25

5.1 Mechanical Assembly 25
5.2 Network Configuration 28
5.3 Motor Driver Configurations 29
5.4 Encoders 31
5.5 Software Interrupt and Closed-loop Controller 33
5.6 USART 34
5.7 ADC and Reflectance 34
5.8 Line Following and Control Algorithm 35
5.9 Integration to Automation Framework 37

5.10 Raspberry Pi Server 40

6 System Test and Verification 41

6.1 Robot Control Test 41

6.2 API Calls Test 42

7 Conclusion 44

References 46

Appendices

Appendix 1: Motor Controller Layout

Appendix 2: Overview of STM32CubeIDE Interface

Appendix 3: PWM Configuration for Motors

Appendix 4: External Interrupt Handler and Closed-Loop Controller

Appendix 5: Reading Reflectance

Appendix 6: Calculate and Return Mean

Appendix 7: Test Environment for the Robot

List of Abbreviations

4G:	Fourth Generation
5G:	Fifth Generation
ADC:	Analog-to-Digital Converter
API:	Application Programming Interface
ARM:	Advanced RISC Machine
ASGI:	Asynchronous Server Gateway Interface
BFAT:	Basic Flexible Automatic Test
BLDC:	Brushless Direct Current
BTS:	5G Base Transceiver Station
CCW:	Counterclockwise
CMS:	Centre-aligned Mode Selection Bits
CMSIS:	Common Microcontroller Software Interface Standard
CW:	Clockwise
DC:	Direct Current
DHCP:	Dynamic Host Configuration Protocol
DMP:	Digital Motion Processor
DNS:	Domain Name System

EOC: End of Conversion

EVS: Enhanced Voice Service

FDM: Fused Deposition Modeling

GPIO: General Purpose Input Output

HTTP: Hyper Text Transmission Protocol

IC: Integrated Circuit

IIC: Inter-Integrated Circuit

IDE: Integrated Development Environment

IOC: Input Output Controller

IR: Infra-Red sensors

ISR: Interrupt Service Routine

JSON: JavaScript Object Notation

KPI: Key Performance Indicators

LDPC: Low Density Parity Check

LTE: Long-Term Evolution

MCU: Microcontroller Unit

NR: New Radio

NSA: Non-Standalone

NVIC: Nested Virtual Interrupt Controller

OAuth: Open Authorization

OAuth2: Open Authorization 2.0

PI: Proportional-Integral

PLA: Polylactic Acid

PWM: Pulse-width Modulation

RAN: Radio Access Network

REST: Representational State Transfer

reST: reStructuredText

RISC: Reduced Instruction Set Computer

RPC: Remote Procedural Calls

RTD: Resistance Temperature Detector

RXNE: Read Data Register Not Empty

RXNEIE: RXNE Interrupt Enable

SA: Standalone

SIM: Subscriber Identity Module

SSH: Secure Shell

SOAP: Simple Object Access Protocol

SUT: System Under Test

TSV: Tab-Separated Values

UE: User Equipment

URL: Uniform Resource Locator

USART: Universal Synchronous Asynchronous Receiver Transmit

USB: Universal Serial Bus

Wi-Fi: Wireless Fidelity

WWAN: Wireless Wide Area Network

1 Introduction

Each generation brings numerous unprecedented innovations into existence with the evolution of wireless technologies. The innovations must only be released and deployed after a series of tests and validations under a dynamic environment. The test environment substantially determines the accuracy and reliability of the test output for any system under test. Network performance can be tested, analysed and verified better out in the field than in a simulated laboratory scenario.

In this project, a mobile test system was developed, enabling the performance testing of 5G base stations with mobile user entity (UE) for Nokia. Established in 1865, the company currently stands as one of the leaders in the telecommunications industry. It specialises in mobile networks, cloud and network services, network infrastructure, and Nokia technologies [1]. Nokia has innovated over 6,000 families of Standard Essential Patents for 5G such as 5G radio stack design, Low Density Parity Check (LDPC) channel coding, Enhanced Voice Service (EVS), and power saving features in the 5G network components [2].

The objectives of this final year project can be listed as follows:

- Build a line following robot that simulates mobile UE under different network conditions.
- Integrate the control logic of the robot into the internal automation framework.

The goal was to implement a simple and low maintenance design using easily available components to build an in-house robot. The robot comprises of a chassis platform with four direct current (DC) motors controlled by an STM32 ARM Cortex microcontroller unit (MCU). An array of infrared (IR) sensors and an ultrasonic sensor were interfaced with STM32 MCU and Raspberry Pi respectively. The Raspberry Pi acts as the brain of the robot, handling application programming interface (API) requests. APIs enable the integration of control

mechanisms with the internal automation framework of the Nokia Field Verification team. The motors are equipped with hall encoder sensors that enable the possibility to calculate the speed and direction of the motors. The Raspberry Pi is connected to a long-term evolution (LTE) network for remote connectivity and control features. The UE logs base station (BTS) performance data while other network analysis applications make use of the data for further analysis.

2 Background

This section introduces the concept of terms and technologies used to achieve the goals of this project. It starts with historical introduction of robotics followed by control systems and 5G technology. The section extends further into explaining the working principle of the components used in this project such as ultrasonic and IR sensors.

2.1 Robots in Industrial Environments

The term ‘robot’ was coined in 1920 by a Czech writer Karel Capek who wrote the play ‘Rossum’s Universal Robots’, a science fiction tale [3]. With time, science fiction introduced robots as mechanical artifacts. In the 1940s, Russian science fiction writer Isaac Asimov introduced the term ‘robotics’ as the science of the study of robots with the following fundamental laws:

> 1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
>
> 2. A robot must obey the orders given by human beings, except when such orders would conflict with the first law.
>
> 3. A robot must protect its own existence, as long as such protection does not conflict with the first or second law.
>
> [3.]

For over a century, humans have tried to innovate intelligent systems to mimic human-like characteristics. Some systems have persevered while others have perished. Humanity has come a long way as robots are significantly changing people’s lives in many aspects. They operate everywhere from ordinary work environments to risky and toxic ones. These dangerous environments such as radioactive spaces of nuclear powerplants and ablaze buildings full of toxic gases are impossible for humans to sustain. Although Megatrons are still far-fetched, robots such as Spot by Boston Dynamics have already been deployed in rescue missions [4].

The first industrial robot was developed by George Devol and Joseph Engelberger in 1959 [5]. The robot was known as the Unimate robot which was installed by General Motors in its manufacturing line in 1961 [6]. The robot performed tasks such as welding, stacking hot pieces of diecast metal, and material handling to a precision of one of ten-thousandth of an inch [5]. The market leader of industrial robots, ABB, released YuMi as a collaborative robot in the year of 2015. YuMi was designed to collaborate with humans in challenging industrial environments that required humanly impossible precision [7]. This progress in just over half a century suggests that the field of robotics has advanced at a swift pace and contributed largely to some of the most significant factors shaping today's world. The manufacturing industries worldwide rely on about 3.9 million robots. These robots are installed in a variety of environments ranging from the assembly line of a car manufacturer to fully automated production of optical devices leveraging smart robotics. Among those robots, just below one third are in automotive industries [8].

2.2 Robot Control Systems

All robotic systems are developed to perform a specific set of tasks and have a controller that ensures precision of the implementation. Robots are no exception to this and have a control system that consists of several tools and functionalities. These functions enable capabilities such as sensing, intelligence, and data processing [3]. Figure 1 below shows a fundamental control system of a robot.

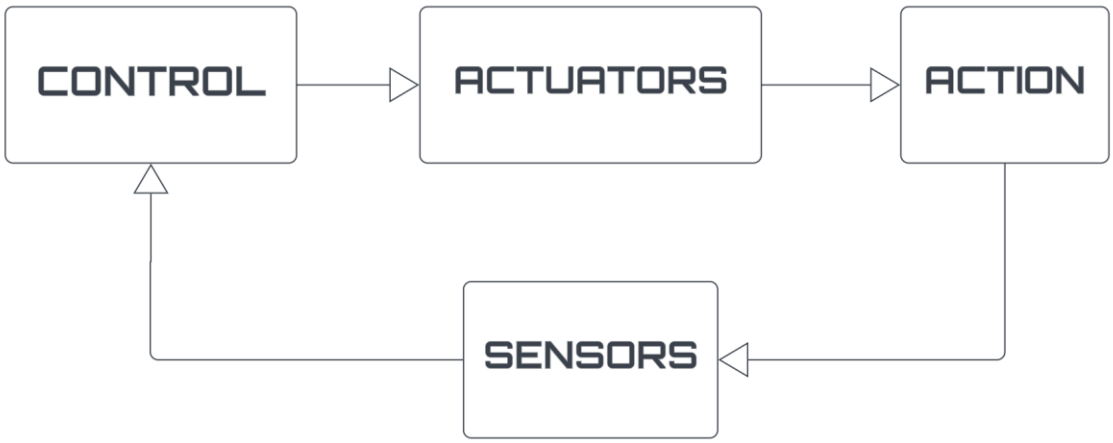


Figure 1. Control system.

As seen in Figure 1, the robot control system has a controller that reads data from sensors, processes the data, and mobilises actuators based on the processed data. Actuators are an integral part of robotic systems enabling movement by converting energy into controlled output in the form of mechanical force. A robot typically consists of multiple actuators depending on the functionality. As an actuator outputs mechanical force, the required input energy can be anything from electric current to hydraulic and pneumatic forces [9]. When the system performs a certain action, sensors constantly monitor the process and feed data to the controller. The cycle continues enabling the system to perform controlled actions.

Line following robots typically follow a black line leveraging an array of IR sensors. These sensors read the reflectance of the surface underneath and feed the data to the controller. The process of reading the reflectance relies on the concept that dark surfaces have less reflectance value than lighter ones. In other words, dark surfaces absorb more light. The controller then makes a decision to adjust line diversion whenever applicable. [10.]

2.3 5G Technology

Fifth generation (5G) technology ensures reliability, ultra-low latency, increased speed and capacity among many other enhancements. The infrastructure can be deployed in primarily two ways, standalone (SA) and non-standalone (NSA). Figure 2 shows the difference between the architectures of the SA and NSA deployments.

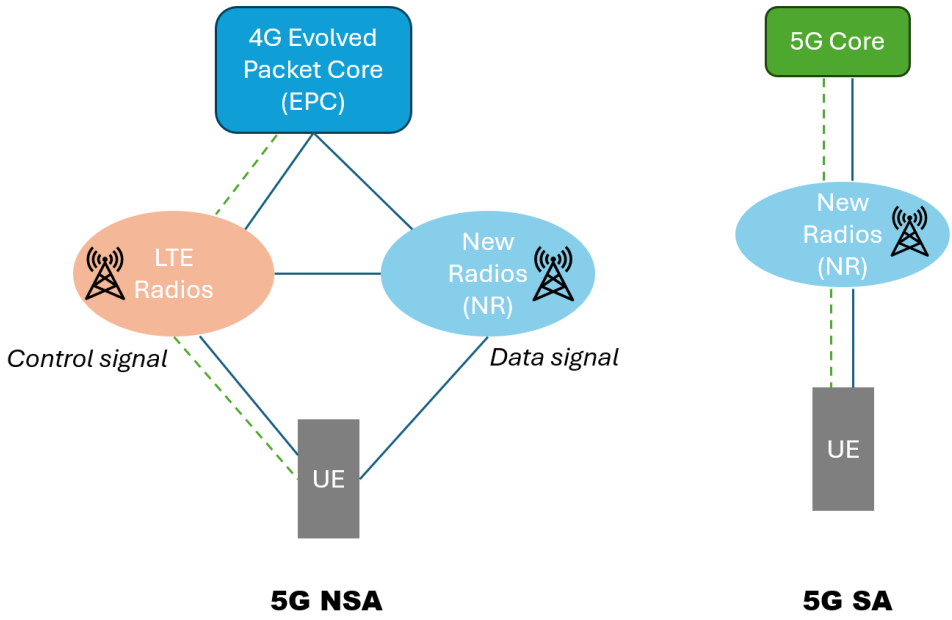


Figure 2. 5G NSA vs 5G SA [11].

As seen in Figure 2, the SA architecture consists solely of 5G components and does not rely on any other technologies. It has its own core network called the 5G core. This deployment offers the full features of a 5G network such as ultra-low latency, higher throughput rates than LTE, network slicing, and enhanced broadband. Moreover, it has a network controller to control network management functions. The NSA architecture consists of the Long-Term Evolution (LTE) core network while the Radio Access Network (RAN) units are 5G components. It provides quicker 5G deployment than SA and higher throughput than LTE. The UE connects to both fourth generation (4G) LTE and 5G networks where the control signals are handled by LTE radios and data signal is handled by 5G new radio (NR). [11.]

2.4 USART

USART enables full-duplex, asynchronous communications with separate enable bits for transmission and receiving. The word length is configurable with either 8 or 9 bits with a programmable baud rate of up to 4.5 megabits per second. The asynchronous mode needs at least two pins, the Rx Pin and the Tx Pin [12, p. 790]. Figure 3, extracted from the reference manual of STM32F103RCT6, shows the word length programming for USART.

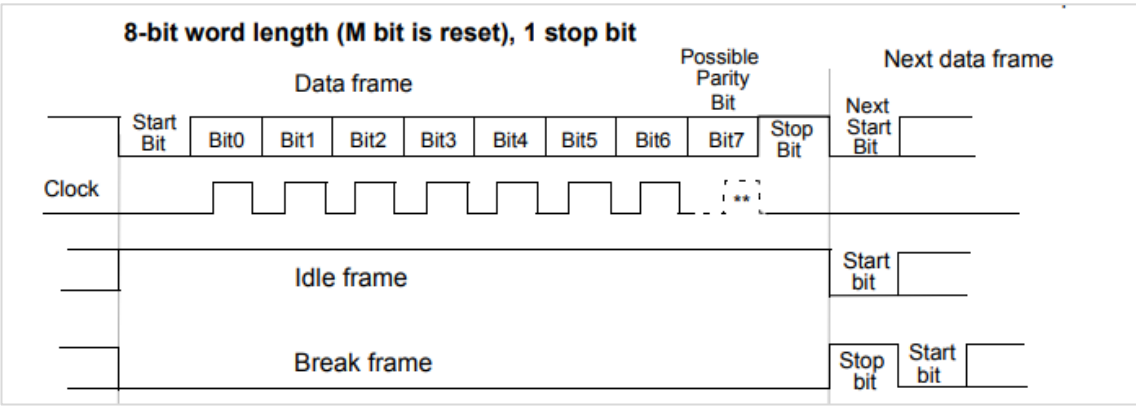


Figure 3. Word length programming for USART. Modified from [12, p. 790].

As illustrated in Figure 3, topmost is the data frame, followed by the clock signal, the idle frame, and a break frame. A frame is considered an as an idle frame if it consists of ones from start to stop bit. On the other hand, a break frame consists of zeros from start to stop bits. A data frame starts with the transmission (Tx) pin set low. An extra parity bit succeeds the data bits if the USART configuration uses a parity bit. Finally, the Tx pin is set high after the last bit is transmitted [36, p. 790].

2.5 APIs

The application Programming Interface (API) enables communication between two or more distributed applications programmatically. The applications are unaware of each other's functionality and follow standard protocols for safety and security. APIs work in a server-client model. The server application handles API

requests and serves the relevant resource. On the other hand, client applications request for resources from the server. The server and client exchange XML messages in Simple Object Access Protocol (SOAP) API architecture. There is also another implementation of an API called Remote Procedural Calls (RPC) APIs where the client makes a function call with certain parameters. The server executes the called function and returns the response back to the client.

WebSocket API implements modern architecture supporting duplex communication between a server and a client. They exchange JavaScript Object Notation (JSON) objects enhancing the efficiency in contrast with Representational State Transfer (REST) API. The latter relies on clients to initialise requests such as GET, POST, PUT, and DELETE [13]. Table 1 describes these methods in brief.

Table 1. REST API method description

Method	Description
GET	Request resource from a server
POST	Add data or create a new entry on the server
PUT	Update a data or resource on the server
DELETE	Delete an entry from the server

REST API was used in the final year project due to its uniform interface, statelessness, flexibility, scalability, and platform independent deployment. In this project, the FiCAT client made GET and POST requests to communicate with the Raspberry Pi server.

2.6 Sensors

Sensors are fundamental components of a system, collecting data on the different physical properties of the environment. They transform and make sense of the energy, for instance, kinetic energy and heat, by converting them to electrical energy. A machine can interpret the electrical signals for further processing. As an example, a Resistance Temperature Detector (RTD) has an increase in resistance under higher temperatures. It has a linear relationship between temperature and resistance, making it easy to read the temperature [14].

Sensors can measure properties such as level, temperature, flow, pressure, speed, position, humidity, and reflectance. They can be primarily classified as active and passive sensors, based on the whether they require an external power supply to operate or not. Active sensors, such as thermocouples, require external power to operate. Passive sensors, such as an RTD, do not require external power to operate [15]. This project leveraged two types of sensors working on completely different forms of energy, light and sound.

The TCRT5000 IR Sensor module uses light to gather data on the surroundings with the help of phototransistors. The phototransistor is an active semiconductor that reacts to light exposure with a response time of one to 10 microseconds [16]. It has a large base collector exposed to light. As light falls on the collector, photocurrent is generated resulting in increased collector current. This current is amplified by hundreds to several thousands. The increased current drops the output voltage [17]. This output voltage data can be measured for reading the luminescence.

The HCSR05 ultrasonic sensor, on the other hand, works on the principle of echolocation [18]. Echolocation is a technique where ultrasonic waves are transmitted, and the reflected waves are analysed for navigation and obstacle detection. It was first discovered being used by bats in the eighteenth century. As waves hit an obstacle in its trajectory, the time taken for the reflected wave to be perceived by the ears of the bat determines the distance to the obstacle [19].

Similarly, ultrasonic sensors can calculate the distance to an object implementing echolocation with a transmitter and a receiver. The distance is calculated as shown in Formula 1:

$$\boldsymbol{s} = \frac{(\boldsymbol{v} * \boldsymbol{t})}{\mathbf{2}} \tag{1}$$

In formula 1,
‘s’ is the distance between the sensor and the object
‘v’ is the velocity of sound wave in air
‘t’ is the measured time

Piezoelectric crystals in the transmitter generate sound waves which reflect to the receiver upon hitting an object [20]. The time taken by the emitted wave to hit back the receiver gives the distance to the object as shown in Formula 1.

2.7 DC Motor

Motors are the key component in mechanical systems transforming electrical energy into mechanical energy. A stepper motor consists of a rotor, a stator, a commutator and brushes. A Brushless DC (BLDC) motor is used in this project that does not include a commutator and brushes. They work on the principle of electromagnetic induction. When the conductor inside the motor’s magnetic field is supplied with current, the conductor starts rotating. [21.]

Motor shaft is an extension for a rotor to transmit the torque. The rotor is made up of steel and mounted with permanent magnets. It converts electrical energy into mechanical energy as it spins inside a stator. The stator, stationed around the rotor, is made up of laminated steel and copper wire windings to generate a magnetic field as current is supplied [22]. The case encloses the vital components to avoid external damage and interference.

3 System Overview

This section introduces the devices and hardware components that were used in the project. As seen in Figure 4, the system comprises of a Raspberry Pi, a STM32 motor controller, an ultrasonic sensor, IR sensors, DC motors, and a battery. There is also a logic converter that converts an output of five volts from the echo pin of the ultrasonic sensor to 3.3 volts for the GPIO pin of the Raspberry Pi.

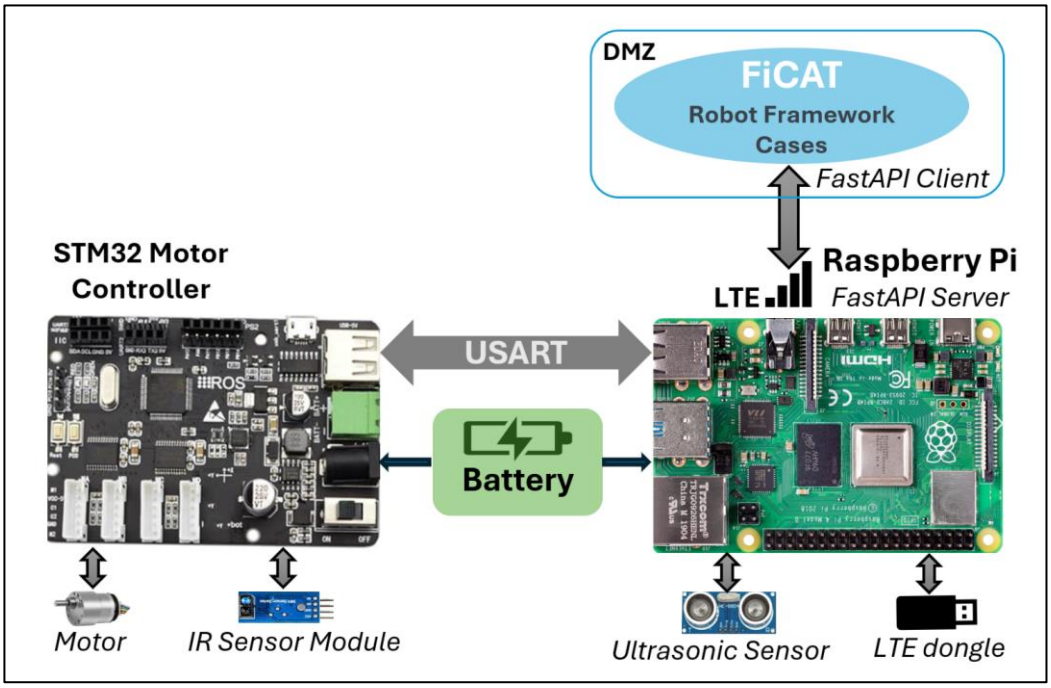


Figure 4. System architecture

An overview of the system hardware is illustrated in Figure 4. The Raspberry Pi handles networking and user commands while the STM32 board controls the motors based on the commands sent by the Raspberry Pi. These two components communicate over the USART protocol at a baud rate of 9600 bits per second. Raspberry Pi sends 8-bit word frames to the STM32 motor controller. The parity bit is not used; thus, the last bit of the payload is immediately followed by a stop bit. Once the API request is processed by the Raspberry Pi, it sends a control signal to the STM32 motor controller and remains idle until the next

request. The communication interface is terminated with a break frame if the server running on the Raspberry Pi is shut down intentionally or exits with errors.

An overview of the supply voltages required for the components of the system is shown in Table 2.

Table 2. Supply voltages for different components

Component	Supply **Voltage**
Raspberry Pi 4	5 volts
STM32 Motor Controller Board	6 to 12 Volts (3.3 Volts micro-USB)
Ultrasonic Sensor	5 Volts
IR Sensor	3.3 to 5 Volts
DC Motors	12 Volts

As seen from Table 2, the ultrasonic sensor operates at five volts meaning that the output of the sensor is five volts. The logic converter plays a significant role converting the voltage to 3.3 volts as the GPIO pins of Raspberry Pi can only tolerate up to 3.3 volts. [23.] The STM32 board has a barrel jack connector with a supply voltage range of 6 to 12 volts.

3.1 Development Board

The STM32 motor controller board used in this project is armed with a STM32F103RCT6 chip. Some of the important features of the chip are listed in Table 3.

Table 3. Features of an STM32F103RCT6 chip [24, p. 1].

Features	Value
Operating voltage	2 to 3.3 Volts
MCU	ARM Cortex-M3 32-bits
Timers	11
Flash memory	512 Kilobytes
Clock frequency	72 Megahertz
Debug Protocols	SWD and JTAG
NVIC priority levels	16
Interrupt channels	Up to 60
GPIO pins	64 (5V tolerant)
Communication protocols	IIC and USART

The NVIC of STM32F103RCT6 MCU supports tail-chaining. Tail-chaining is the immediate execution of an ISR without a context restore. Typically, the processor finishes executing a higher priority task and restores the context to resume the execution of a low priority task [24].

Appendix 1 illustrates the layout of the STM32 motor control board. It can be seen from Appendix 1 that the board has two TB6612 motor drivers. This motor driver can handle the voltage rating of the motors used in this project. Moreover, it can drive the amount of current required by the motors [25]. Figure 5 shows a block diagram of the TB6612 motor driver.

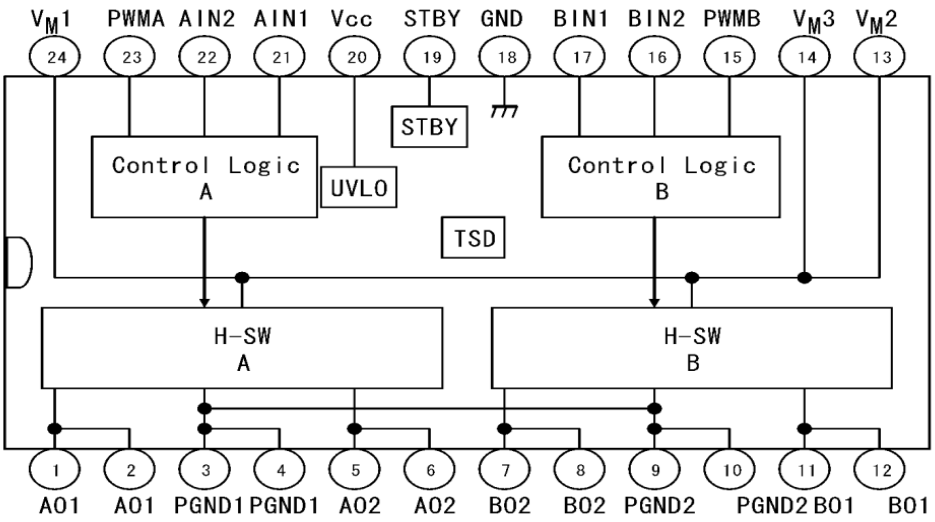


Figure 5. Block diagram of a TB6612FNG motor driver. Extracted from [26, p. 2]

As seen in Figure 5, there are two control logic blocks with dedicated H-Bridge Switch blocks. Thus, each motor driver can control two motors offering four different modes as listed below:

- Clockwise (CW)
- Counterclockwise (CCW)
- Short brake
- Stop mode

A maximum of 15 volts can be supplied to the motor driver. It comes with a standby state to save power with built-in thermal shutdown and a low voltage detection circuit. [26.]

Furthermore, the board is equipped with MPU6050. It is an integrated 6-axis motion tracking device that combines a 3-axis gyroscope, a 3-axis accelerometer, and a Digital Motion Processor (DMP). The gyroscope measures angular momentum, the accelerometer measures acceleration across linear plane, and the DMP utilises the motion data from the gyroscope and the accelerometer to determine the position and orientation of an object. MPU6050 stores the processed data in its data registers. The data registers can be read over the IIC protocol. The IC consists of three 16-bit ADC converters dedicated for gyroscope readings and another three for the accelerometer [27].

3.2 Sensor Modules

In addition to the board discussed above, the system consists of two more sensor modules as listed below:

- HW-270 IR module
- HC-SR04 ultrasonic sensor

Figure 6 shows the IR sensor used in this project. It transmits both digital and analogue data. An ADC channel must be configured to read the analogue data [28]. The ADC output data ranges between zero and 4500 and is read from the A0 pin.

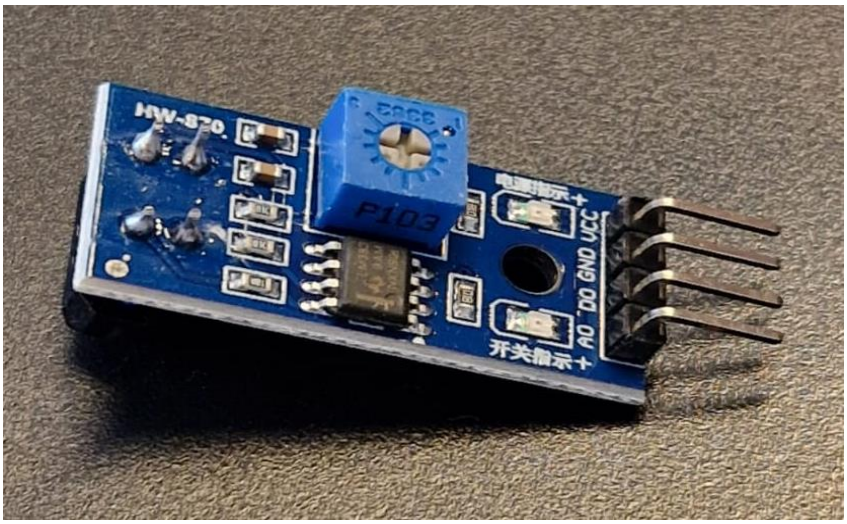


Figure 6. HW-270 IR Module

The HW-270 IR module consists of a potentiometer, the blue cube with a knob, as seen in Figure 6. It adjusts the voltage across the on-board LED that turns off upon hitting the threshold set with the potentiometer. It transmits a digital output signal of ‘1’ in this case. When the output value is below the threshold, the digital output value is ‘0’. The supply voltage for the sensor lies within the range of 3.3 to 5 volts and most accurately detects reflecting distance between 1 to 25 millimetres. Infrared light is within the wavelength of 700 to 1000 nanometres’ range.

An IR sensor comprises of an emitter and a receiver, combinedly known as a photocoupler [20] as illustrated by Figure 7.

Figure 7. Emitter and receiver of an IR Sensor

Figure 7 is an image of HW-270 IR Module showing an emitter and a receiver. The transparent and slightly tinted LED of the photocoupler package is the emitter. It transmits an IR light with a wavelength of 950 nanometres [29]. The dark tinted sphere is the receiver made up of a sensitive phototransistor.

The HC-SR04 ultrasonic sensor measures distance to objects at two to 400 centimetres with 3 mm tolerance. The module consists of a control circuit, ultrasonic transmitters, and receivers [30]. Figure 8 shows an image of the front view of the HC-SR04 module.

Figure 8. The HC-SR04 module ultrasonic sensor used in the project.

Both the Vcc pin and the trig pin seen in Figure 8 are supplied with five. From the perspective of Raspberry Pi, the trig pin must be configured as a GPIO output pin, and the echo pin must be configured as a GPIO input pin. Figure 9 below illustrates timing diagram of the module.

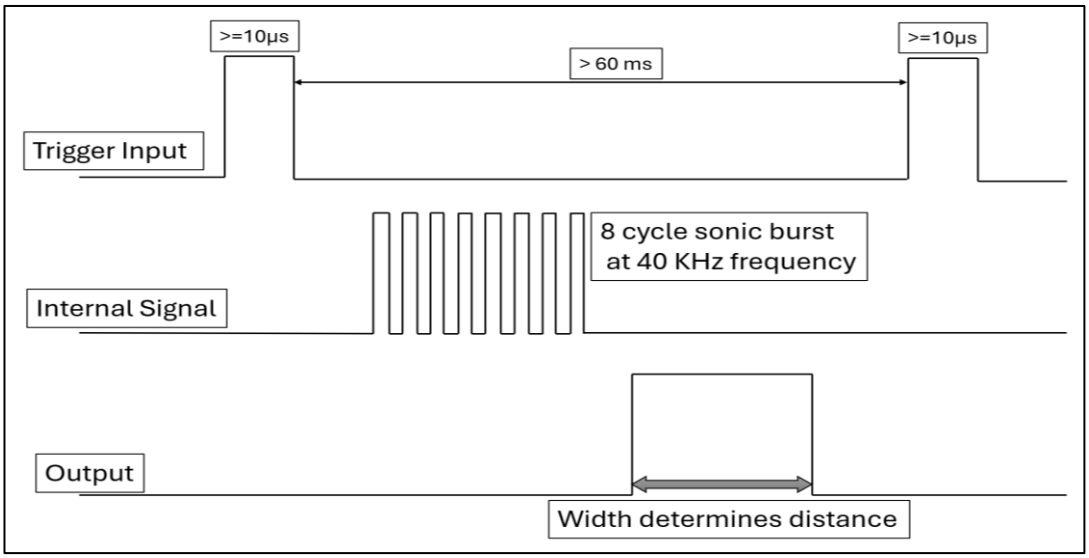


Figure 9. Timing diagram of HC-SR04

The trigger input pin of the module is set high for a minimum of 10 microseconds. This is a signal that initializes the transmitter to send eight cycles of sonic bursts at 40 kHz as seen in Figure 9. The receiver sets the echo pin high and waits for the reflected ultrasonic burst. The echo pin is set low if the receiver detects ultrasonic waves. Figure 10 shows the signal analysis performed during the integration of the HC-SR04 ultrasonic module into the system.

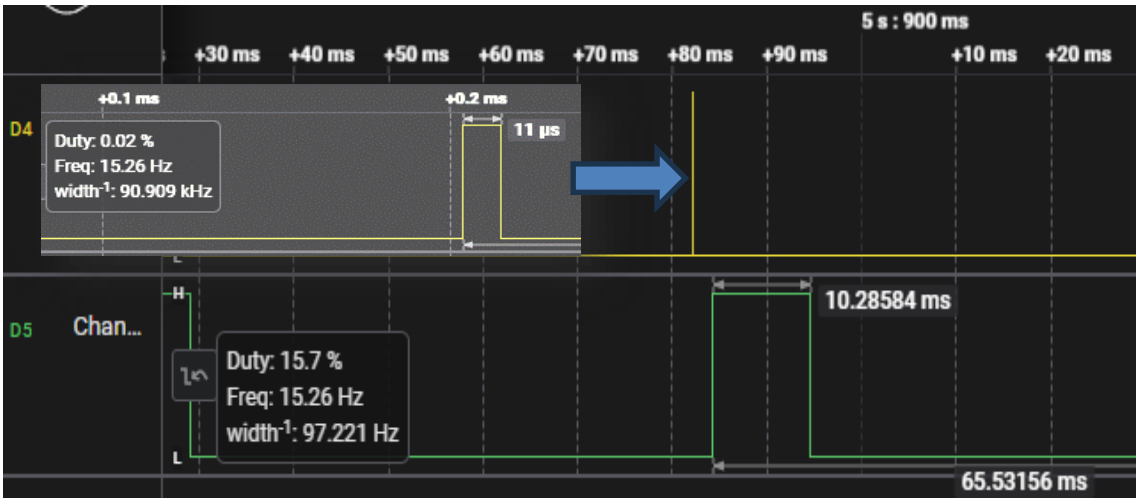


Figure 10. Analysing the trigger and echo signal in a logic analyser

Figure 10 illustrates the trigger signal read into channel D4 in colour yellow and the echo signal into channel D5 in colour green. The width of the pulse during which the echo pin was set high determines the distance to the object. In Figure 10, the echo high signal prolongs 10.28584 milliseconds which calculates as 1.73 meters in distance. The distance is between the ceiling and the HC-SR04 module placed on a table. If no signal is received by the receiver, the echo pin times out and is set low after 38 seconds. The timeout occurs for objects beyond four metres too.

3.3 Motor

A JGB-520 motor is used in this project. It has a top speed of 330 revolutions per minute at 12 volts. Figure 11 shows an image of the wheel, and the motor used in this project.

Figure 11. A DC motor

Figure 11 shows that the motor consists of a magnetic hall effect encoder and a feedback circuit board attached along the shaft. The hall effect sensors work on the principle of hall effect. The principle states that a measurable voltage difference is created across a conductor when a magnetic field perpendicular to the flow of the current is applied to the conductor [31]. Two hall effect sensors placed around a circular magnet pick up the direction of the rotation. As the shaft rotates, the magnet attached to the shaft rotates along with it and the magnetic field changes. This rotation continuously generates pulses based on the direction and speed of the motor. The data transmitted by the encoder can be used in the precise control of the motors.

4 Development Tools

This section introduces the development tools and software used in the project. The project involved the use of a variety of tools, frameworks, and software for programming and configuration to 3D design and printing.

4.1 Development Environment

The integrated development environment (IDE) used in this project was STM32CubeIDE. The development environment was used in the development of the C and C++ programs in the project. STM32CubeIDE is a modern development tool, based on Eclipse and the GCC toolchain, which offers advanced C/C++ support for the development of software for theSTM32 platform. It allows the selection of microcontrollers, the configuration of peripherals, compiling, debugging, and flashing the developed firmware. This Integrated Development Environment (IDE) comes with an Input/Output Configurator (IOC) Graphical User Interface (GUI). The IDE generates code based on the configurations made in the IOC [32].

As seen in Appendix 2, there is a **project_name*.ioc* file that opens a separate tab with several configuration options. The view with the pinout of the chip also allows configuration of each pin. For instance, a General-Purpose Input Output (GPIO) pin must be configured in input mode to interface a button with the pin. Then, it automatically generates code with configurations for clock, the GPIO pin, and interrupt handlers if enabled. With the generated code, custom logic can be added to build a complex system [35]. During the initial phase of the project, individual components were developed and tested in STM32CubeIDE. The serial communication interface between Raspberry Pi and the STM32 controller board, Analog-to-Digital Converter (ADC) readings from the IR Sensor, Pulse-Width Modulation (PWM) and Timer Configurations for the motors were first developed on this platform and later integrated into the project workspace.

4.2 Motor Controller

The code for the motor controller used in this project was developed by using a template project as a starting point. The template project is available as an open-source project in GitHub [33]. The template project is developed with the C language for the STM32F103RCT6 chip. The Common Microcontroller Software Interface Standard (CMSIS) has been used in developing the firmware for the board. This standard approach facilitates software portability and re-usability that reduces the learning curve for developers. The standard embraces certain coding rules and conventions such as data types as in ANSI C standard, macro definition expressions in parenthesis, name of core registers and peripheral registers in uppercase, function names and interrupt related functions in camelCase, and the doxygen documentation generator [34].

The control logic of the template was modified and implemented as per the requirements of this project. Moreover, custom libraries were added to handle the IR sensors. New data structures were introduced to optimise the execution and better handling of data such as encoder reading, current and target position of the motor, and PWM values.

4.3 Robot Framework

Robot Framework enables test automation with easy, readable, and versatile syntax. It is a generic, keyword-driven, and Python-based scripting language written either in plain text, tab-separated values (TSV), or the reStructuredText (reST) format with rich libraries of keywords to make use of the tuples data structure to manage SSH tunnels. Moreover, it is independent of the system under test (SUT) or the platform. The field of software test automation has benefited from the use of this framework. [35.]

4.4 FastAPI

FastAPI is a high-performance web framework powered and served by Starlette and Uvicorn. These are Asynchronous Server Gateway Interface (ASGI) frameworks that offer asynchronous programming for microservices, real-time data processing, and high-speed tasks. FastAPI is easy to learn and use. It generates interactive API documentation automatically contributing to endpoint testing. It is compatible with the OpenAPI which is an open standard for APIs. The command '*pip install fastapi uvicorn*' installs FastAPI and Uvicorn. The server is run with command *'uvicorn main:app –reload'*, given that the main.py is the main source [36]. Listing 1 shows a simple implementation of a FastAPI server that processes GET requests.

```
from fastapi import FastAPI
app = FastAPI()

@app.get("/")
def req_root():
    return {"message": "Hello! Server is up!"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

Listing 1. An example implementation of a FastAPI Server

The code snippet in Listing 1 imports and instantiates FastAPI. The endpoints are defined and implemented as in Listing 1. FastAPI supports OAuth2 for authentication. The server in this project is designed and implemented with simple OAuth2. The endpoints are secured with a bearer token and password flow [37]. The code snippet in Listing 2 is extracted from the Raspberry Pi server.

```
@app.post("/token", response_model=Token)
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    if form_data.username == "username" and form_data.password == "password":
        access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRY_MINUTES)
        access_token = create_access_token (
            data = { "sub": form_data.username }, expires_delta = t_expires
        )
        return {"access_token": access_token, "token_type": "bearer"}
    else:
        raise HTTPException(
            status_code = status.HTTP_401_UNAUTHORIZED,
            detail = "Incorrect username or password",
            headers = {"WWW-Authenticate": "Bearer"}
        )
```

Listing 2. OAuth2 authenticated token endpoint

The username and password in Listing 2 must be successfully authenticated to acquire the token from the server. The token is passed as a header in requests from the client side to access the protected endpoints as shown in Listing 3.

```
Headers = 'Authorization': f'Bearer {TOKEN}'
try:
    response = requests.get(url, headers=headers, timeout=5)
    response.raise_for_status()
    return response.json()
except requests.exceptions.RequestException as e:
    print(f"An error occurred when accessing the protected endpoint: {e}")
```

Listing 3. API client authenticating and accessing endpoint

The token is formatted appropriately as shown in Listing 3 before passing as a header while requesting protected resources from the server. The method raises a ‘Timeout’ exception if the server fails to respond within five seconds due to network issues or internal server errors.

4.5 3D Design

The 3D model of the case for the robot was designed in FreeCAD, an open-source application, which comes with several features. Workbenches in this application allow designers to create and customize the tools available in that workbench [38]. The model is optimized for printing and sliced in another application, UltiMaker Cura that generates a ready-to-print file.

The Ultimaker 3D printers implement a printing process known as Fused Deposition Modeling (FDM) where a filament made up of thermoplastic is fed into a direct drive extruder that deposits the fine strands of the hot material into the plate. The extruder is controlled by multiple stepper motors. The product under development is printed strand by strand. The properties of the strand can be defined before slicing as per the required quality and urgency of the product. Higher quality equals to longer printing time. The FDM method is mostly applicable for fast-paced product prototyping at low cost. The material used in printing is made up of Polylactic acid (PLA) that is biodegradable, odourless, rigid, and strong but brittle and less heat resistive. [39.]

5 Implementation

The previous sections introduced hardware components and software tools used in this project. This section starts with discussing the assembly of the hardware parts. Then, network configurations for the Raspberry Pi and the development of the line following algorithm are introduced. This is followed by elaborating on the motor controller program and the API server. Finally, Robot Framework cases for control automation conclude the section.

5.1 Mechanical Assembly

The mechanical parts consist of four mecanum wheels with an encoder board attached to the motor shaft, the body or chassis, a case for the IR sensors, and a case for the ultrasonic sensor. Figure 12 shows the chassis before other components were 3D-printed and assembled.

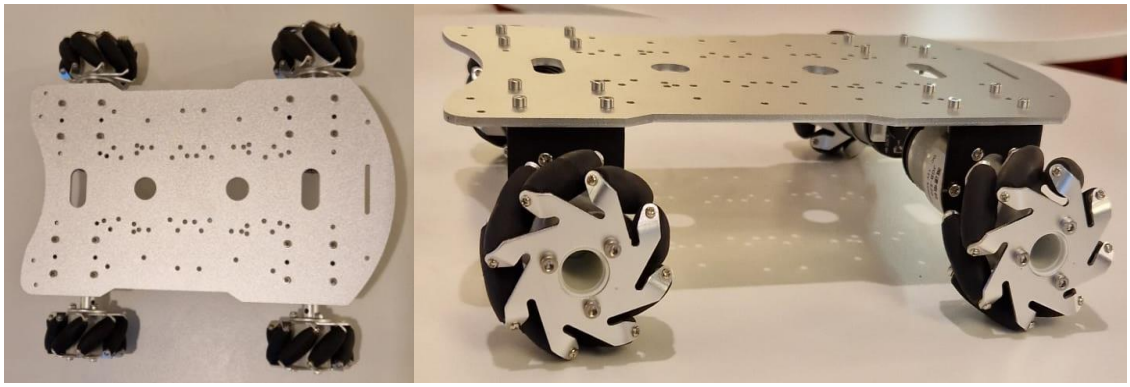


Figure 12. Aerial (left) and side (right) view of the chassis

Motors are first mounted on the motor mounting brackets and screwed to the chassis as seen in Figure 12. There are multiple screw holes on the metal plate that allow the installation of motor brackets of varying dimensions. The chassis is enclosed in a 3D-printed structure as seen in Figure 13.

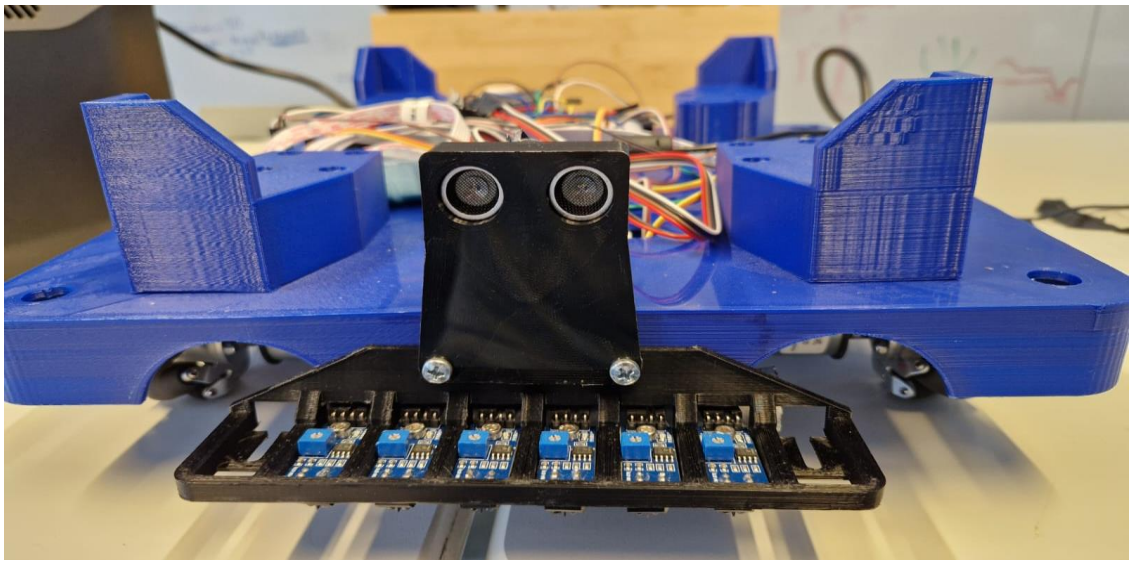


Figure 13. The body and the cases of the robot

As illustrated by Figure 13, four separate support extensions accommodate the battery and provide space for components such as the Raspberry Pi and the STM32 motor controller. These components reside under the battery, protected from any external impacts that could strike while in motion. The body and the case were printed with a 3D printer at one of the 3D printing laboratories at Nokia headquarters in Espoo. The mecanum wheels, unlike a normal wheel with a tyre, have a series of rollers installed at an angle or along the circumference of the wheel as shown in Figure 14.

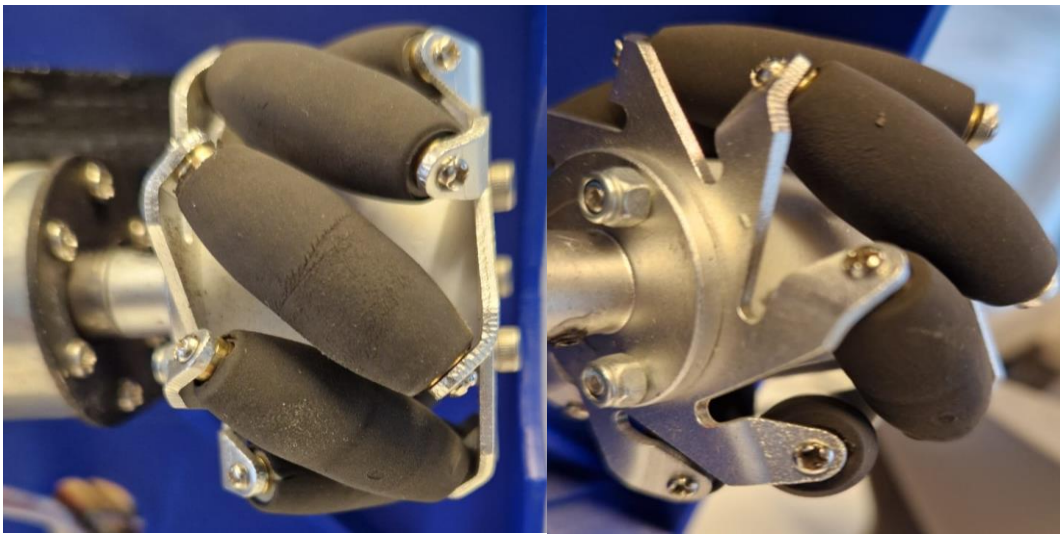


Figure 14. A mecanum wheel design

As seen in Figure 14, each roller is screwed to protruding structures along the circumference. The body is printed in four parts so that any broken part can be printed individually without replacing the whole part. This reduces both material consumption and printing time.

5.2 Network Configuration

The Raspberry Pi is connected to a Wireless Wide Area Network (WWAN) using a USB dongle with a SIM card in it. The network is configured as a test network and requires firewall authentication which restricts the operation of the robot only to authorized personnel. Figure 15 below shows an overview of the implementation.

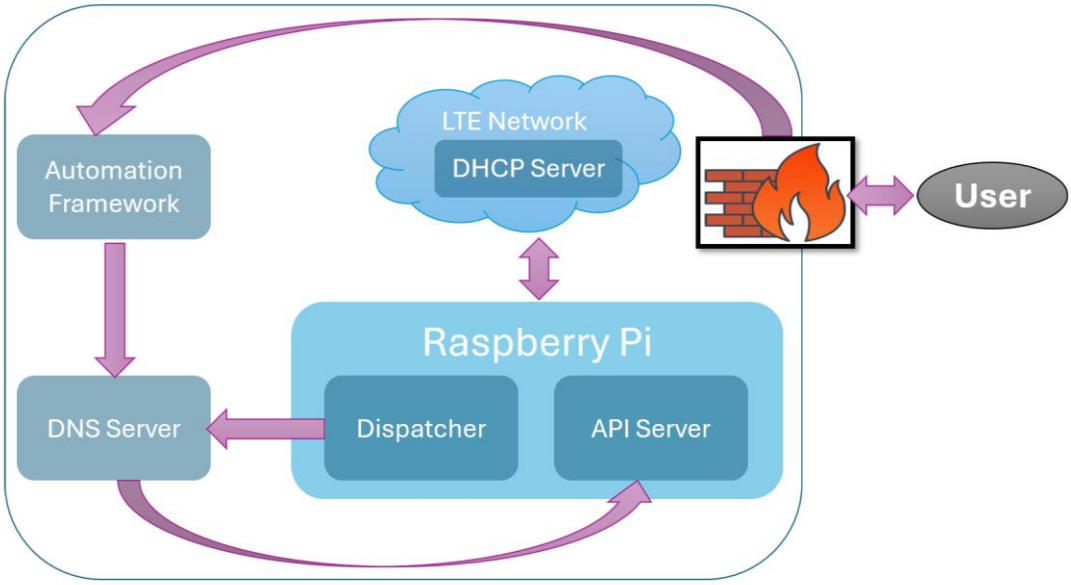


Figure 15. Network configurations on Raspberry Pi

The Raspberry Pi has a domain name but not a static IP address as the DHCP configuration of the LTE network does not allow any static IP address on the network. As seen in Figure 15, a dispatcher script runs on the Raspberry Pi that checks for any changes on the WWAN interface and updates the assigned IP address on the DNS server. This script makes sure that the domain name of the Raspberry Pi is always valid. The script prioritises WWAN for network connectivity but can also connect to Wi-Fi in case of primary network failure. This redundancy of the network connectivity significantly increases the reliability of the system.

5.3 Motor Driver Configurations

The firmware for the STM32 robot controller board consists of configurations for GPIO pins, timers and PWM, and interrupts to drive the motors. This section describes these implementations in detail.

All four channels of timer eight (TIM8) are configured to generate a PWM signal. TIM1 and TIM8 are advanced timers of an STM32F103RCT6 MCU consisting of a 16-bit register for an auto-reload counter. The prescaler value is programmable in these timers [12]. Appendix 3 shows an implementation of the PWM configuration for the motors. The clocks for TIM8 and port GPIOC are configured first. Once the associated GPIO pins are configured for alternate function output, auto-reload (ARR) and prescaler (PSC) values are assigned to the registers as shown in Listing 4.

```
TIM8->ARR = arr;        // Set auto-reload register value
TIM8->PSC = psc;        // Set prescaler value
```

Listing 4. Configuring values for ARR and PSC

The arr and psc values assigned to ARR and PSC registers, as shown in Listing 4, determine the output frequency of the PWM signal. Formula 2 shows the relationship between ARR, PSC, system clock frequency, and output frequency.

$$\boldsymbol{f_{out}} = \frac{\boldsymbol{f_{sys}}}{(\boldsymbol{PSC} + \mathbf{1}) * (\boldsymbol{ARR} + \mathbf{1})} \tag{2}$$

where,

f_{out} is the output frequency for the generated PWM signal

f_{sys} is the system clock frequency.

PSC is the prescaler value for the timer

ARR is the auto-reload value for the timer

The output frequency can be calculated as below with system clock frequency at 72 MHz, the prescaler value of zero, and the auto-reload value set as 7199. Substituting Formula 2 with the stated values, the equation looks like below.

$$f_{out} = \frac{72000000}{(0+1) * (7199+1)}$$

The output frequency is 10 kHz. The period of the PWM signal is inversely of the output frequency calculated above. Thus, the PWM signal has a period of 100 microseconds. A higher frequency provides smoother motor control.

Figure 16 below illustrates the pinout for the TB6612 motor driver chip. The pin numbers in the red colour are the GPIO pins of the STM32 chip, to which the pins of motor drivers are connected.

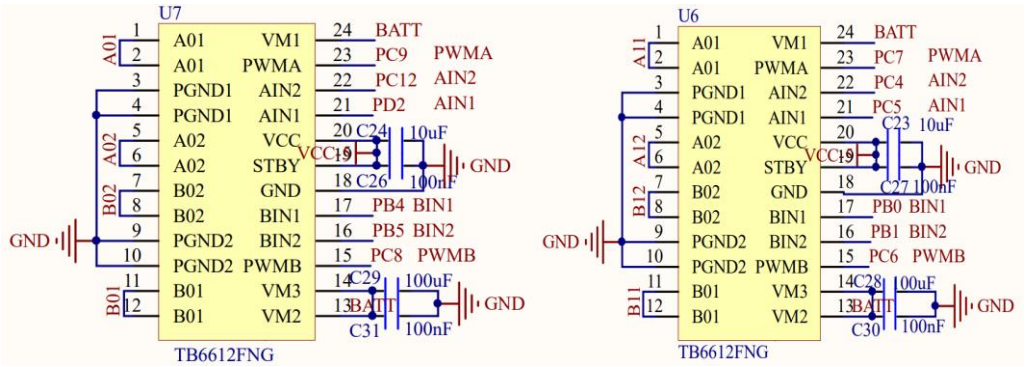


Figure 16. TB6612 motor driver pinout [16]

As seen from Figure 16, there are two PWM input channels in each of the two TB6612 motor drivers. PWMA is connected to pin 23 while PWMB is connected to pin 15 on each motor driver. The PWM output channels of TIM8 from the STM32 MCU are connected to the PWM input channels of the motor driver mentioned above.

5.4 Encoders

The motors used in this project have an encoder feedback circuit board attached along the shaft. This section describes the implementation of encoder feedback logic. Table 4 below shows the connection between the encoder output pins and the GPIO pins of STM32 MCU.

Table 4. Encoder phases, STM32 MCU timers and GPIO pin interfaces

Motors	Encoder phases	STM32 MCU			
		GPIO Pin	Pin mode	Timer	Channel
Motor A	Phase A	PA15	alternate remap	TIM2	Channel 1
	Phase B	PB3			Channel 2
Motor B	Phase A	PA6	alternate default	TIM3	Channel 1
	Phase B	PA7			Channel 2
Motor C	Phase A	PB6	alternate default	TIM4	Channel 1
	Phase B	PB7			Channel 2
Motor D	Phase A	PA0	alternate default	TIM5	Channel 1
	Phase B	PA1			Channel 2

The channels interfaced to encoder phases are from four different timers as seen in Table 4. The GPIO pins are in main function mode or in default state after reset. They must be configured to be in alternate function mode enabling specific function modes such as ADC input channel, USART Tx, I2C_SDA, and TIM2_CH1. All GPIO pins connected to the first and the second channels of TIM3, TIM4, and TIM5 respectively are configured for default alternate functions as seen in Table 4. The GPIO pins for TIM2 must be remapped for an alternate function. The timer is configured in the encoder interface mode that combines channels one and two. Both rising and falling edges of the encoder signals are detected. The counter updates automatically as per the speed and position of the

quadrature encoder. Pin interfaces shown in Table 4 can be further clarified with the schematic in Figure 17.

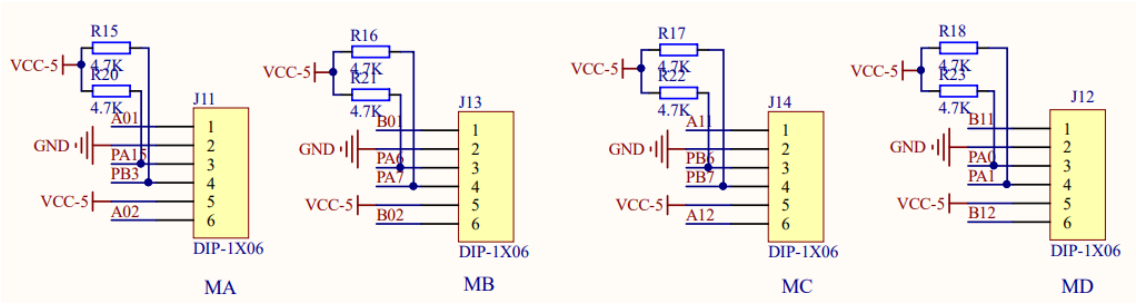


Figure 17. Encoder motor to STM32 interface

As illustrated by Figure 17, the STM32 motor controller has six-pin connectors for all four motors. Pins on the edges of the connector, for instance A01 and A02 on MA, are connected to the A01 and A02 pins of the TB6612 motor driver shown in Figure 16. These pins from the connector connect to pins M+ and M- of the encoder board shown in Figure 18.

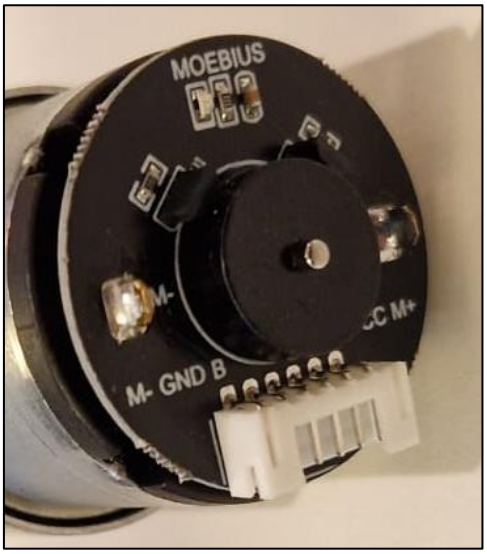


Figure 18. Encoder board attached to the motor

The circular magnet located at the centre of the board in Figure 18 rotates along with the rotor of the motor. Two hall-effect sensors along the circumference of the magnet transmit two different signals, referred to as A and B. These phase

signals depend on the direction of the rotation as the magnet rotates. The input channels reading the encoder signals are configured in edge-aligned mode. The configuration is done by clearing the Centre-aligned Mode Selection (CMS) bits of timer control registers TIMx_CR1 shown in Figure 19 below.

15.4.1 TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 19. Control register of a timer in STM32F103RCT6. Extracted from [36, p. 404].

Bits five and six of the control register are CMS bits as seen in Figure 19 above. In this mode, the counter counts up or down as per the direction bit. The encoder signal sets the direction bit. The fourth bit or the direction (DIR) bit in Figure 19 is set as per the encoder signal. The DIR bit is modified by the hardware at each transition on the input channels [12, p. 392].

5.5 Software Interrupt and Closed-loop Controller

This subsection explains about the external software interrupt and the closed-loop controller configured in the STM32 motor controller. These configurations are implemented to control the robot. An interrupt is triggered by the INT pin of the MPU6050 sensor every five milliseconds. The interrupt handler reads the encoder data, updates the position of the robot, reads USART data from a queue, implements a closed-loop controller, and finally, executes the line following algorithm. As shown in Appendix 4, the kinematic analysis function calculates target velocities based on the encoder reading and the position of each motor. This data is then used to calculate the target position. The proportional-integral (PI) algorithm calculates the appropriate PWM signal as per the bias between the target position and the current position. Then, the line following algorithm takes control and makes a decision based on the readings of the IR sensors.

5.6 USART

The USART Rx interrupt is enabled by setting RXNEIE bit of USART_CR1. RXNEIE is the ‘receive not empty interrupt enable’ bit of the control register one. The USART_CR1 is the control register one of the STM32F103RCT6 MCU. The hardware sets the RXNE bit of the status register as USART_DR is updated. An interrupt is triggered when the RXNE bit is set. Listing 5 shows an implementation of the USART ISR with STM32 Standard Peripheral Libraries [40].

```
void USART3_IRQHandler(void)
{
      if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
      {
            InQueue(USART_ReceiveData(USART3));
      }
}
```

Listing 5. Example implementation of an USART ISR

As seen in Listing 5, the USART number is passed as the first parameter in the if-statement and the RXNE bit is passed as the second parameter. If the RXNE bit is set, the ISR sends the data to a queue. The data is then handled by a queue handler.

5.7 ADC and Reflectance

ADC channels read data from the IR sensors and feed the line following algorithm. This data holds significant value in aligning the robot along the line. In this project, ADC conversion is implemented with a polling mechanism. As shown by the code in Listing 6, ADC conversion starts, and the program waits until the End of Conversion (EOC) flag is set.

```
u16 Get_Adc(u8 ch)
{
    // Start conversion
    ADC1->SQR3 &= 0xFFFFFFE0; // Clear channel bits in regular sequence 1
    ADC1->SQR3 |= ch;
    ADC1->CR2 |= 1<<22;       // Start conversion on regular sequence
    while(!(ADC1->SR & 1<<1));// check for EOC flag
    return ADC1->DR;          // Return ADC value
}
```

Listing 6. Reading ADC values from the IR sensor

The EOC flag is set upon the completion of the conversion. EOC is the second bit or bit 1 of the ADC status register. The function in Listing 6 returns the ADC value upon the completion of the conversion. The ADC data register is cleared upon reading.

5.8 Line Following and Control Algorithm

The code below shows functions to read reflectance values transmitted by the IR sensors. The code consists of relevant lines from both header and source files. Based on the deployment environment, the reflectance threshold values for the sensors must be tested and defined. The sensors are named as per their location on the protective tray; for instance, L1 is the left-most sensor and R1 is the right-most one. A struct, as shown in the code from Appendix 5, maps these names to their respective ADC channels.

IR sensors can be added or removed and the macro ‘NUMBER_OF_CHANNELS’ should be updated before operating the robot. If deployed in a new environment, the reflectance values must be redefined and tested before operating the robot; otherwise, the robot might exhibit unexpected behavior.

As seen from Appendix 5, the function ‘get_channel_number()’ takes the sensor position as a parameter and returns the respective ADC channel number. This return value is passed into another function, ‘read_adc()’, which returns the ADC reading of the channel. Listing 7 shows how the analog values are digitized.

```
/**
  * @brief  Compare ADC value with the threshold
  * @param  IR sensor's position, threshold
  * @retval 0 - less than threshold, 1 - greater than threshold
*/
u8 digital_IR(const char * pos, uint16_t threshold){
          uint16_t val = read_adc(pos);
          if (val < threshold) return 0; // whilte surface
          else if (val >= threshold) return 1; // tape line
          else return 200;
}
```

Listing 7. Converting an ADC reading into a binary value

As seen from Listing 7, the sensors with reflectance less than the threshold return 0 and those above the threshold return 1. This value is used to calculate the mean value as in the code shown in Appendix 6. The function ‘line_state()’ calculates variance of the reflectance values and returns the mean value of the reading. The variance determines which sensors are reading the line and checks for error states. Variance can be used in implementing new features to the movement of the robot, such as a short pause, stop, or turning. Some statements are for debug purpose only.

The code shown in Appendix 4 is extracted from the line following logic of the robot. If the queue is not empty, the mean value of the IR sensor readings is evaluated. The positions of IR sensors are indexed from left to right. As shown in Listing 8, the digital reading of each IR sensor is evaluated in calculation of the mean value.

```
for(i = 0; i < NUMBER_OF_CHANNELS; i++){
                counter += digital_readings[i];
                mean_reading += (i * digital_readings[i]);
     }
     if (!counter) return old_mean;
     if (counter > 4) return 100; //@TODO: implement appropriate logic here

     mean_reading /= counter;
     old_mean = mean_reading;
```

Listing 8. Calculation of mean value for IR sensor readings

The code shown in Listing 8 implements a function that returns the mean value of the IR sensor readings. A mean value of 2.5 indicates that the robot is following the line. The central sensors have indices of two and three. Anything above or below this threshold of 2.5 triggers error correction as shown in Listing 9.

```
static int state = 0;
int nstate = ret < 2.4 ? -1 : (ret > 2.6 ? 1 : 0);
if(state != nstate){ //if mean value changes
      state = nstate;
      if(state < 0) Move_X -= initial_correction;
      else if(state > 0) Move_X += initial_correction;
}
if (ret > 2.6) Move_X += dynamic_correction*(ret-2.5);
else if (ret < 2.4) Move_X -= dynamic_correction*(2.5-ret);

Get_RC(0);
Update_PI();
```

Listing 9. Error correction block of line following algorithm

The code in Listing 9 checks for changes in mean value. In case of changes, it performs correction, incrementing or decrementing the PWM values for each motor. There is also an overcorrection avoidance logic in the block.

5.9 Integration to Automation Framework

The robot is controlled remotely over the network with API commands. Nokia's internal automation framework includes the FiCAT (Field Verification's Centralized Automation Tool) web application that offers an interactive GUI. Among many other features such as navigating to other tools, it allows users to schedule the execution of robot cases as in Figure 20. FiCAT is the end user for the BFAT (Basic Flexible Automatic Test) automation solution, which provides tools to create, manage, execute, analyse, store, report, and automate test setups for tests such as end-to-end, application, simulator, and GUI. The client method is then invoked as BFAT instances execute the cases scheduled in FiCAT.

Status		Case	Start	Tags
✓	☑	CampusRobot-Start	13.08.2024 10:51	
✓	☐	CampusRobot-Stop	13.08.2024 10:52	

Figure 20. Executing Cases in FiCAT Tool

As seen in Figure 20, the start time can be set to execute the cases immediately or in the future. The ‘CampusRobot-Start’ case can be executed now and the stop case can be scheduled to be executed after 24 hours. The cases are implemented with Robot Framework as shown in Listing 10 below.

```
Robot Init
    [Arguments]  ${robot_name}
    Authenticate  ${robot_name}
    ${response}  campusrobot_fastapiclient.GetRequest    ${BASE_URL}    start
    Log  ${response}
```

Listing 10. Robot Framework test case

As mentioned earlier, these cases invoke client methods, i.e. GetRequest in Listing 10, to communicate with the server. The third-party ‘requests’ library of Python is used in this project that offers APIs for making HTTP requests [41]. A code block of the client method can be seen in Listing 11 below.

```
@keyword("GetRequest")
def get_endpoint(base_url, endpoint=None):
    if not endpoint:
        url = base_url
    else:
        url = f"{base_url}/{endpoint}"
    headers = get_headers()
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"An error occurred when accessing the protected endpoint: {e}")
```

Listing 11. Implementation of a GET method

The code block in Listing 11 shows an implementation of a GET method. It takes a base URL (Uniform Resource Locator) and an API endpoint as arguments that are passed from the robot cases to the FastAPI client methods. The endpoints are secured with OAuth2 that was previously explained in Subsection 4.4. Once

authorization succeeds, the API server running on the Raspberry Pi handles the request and initializes the USART communication with the STM32 controller. For instance, in the case of the start command, the USART ISR gets invoked and the STM32 controller executes multiple functions to start the motors and put the robot in motion. The process flow is depicted by Figure 21.

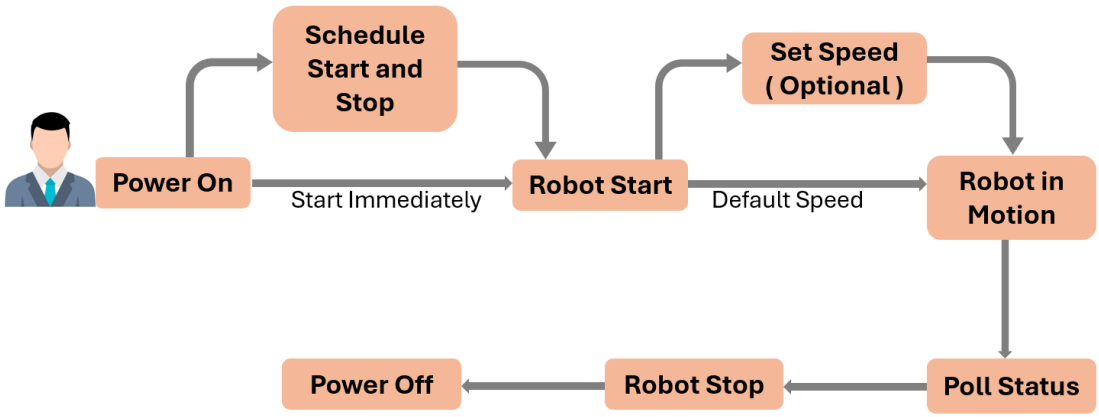


Figure 21. Process flow and automation

The flowchart in Figure 21 shows a high-level architecture of the processes involved in the operation of the robot. The user with access to the FiCAT System Under Test (SUT) of the robot starts with creating a runlist which allows the execution of the cases. As the system is powered on, the start and stop cases can be scheduled with desired operation timespan. If needed, the robot can be polled for the distance to obstacles ahead.

5.10 Raspberry Pi Server

An API server developed with the FastAPI framework is deployed on the Raspberry Pi. In addition to listening for API requests with OAuth2 configurations, the server has obstacle avoidance logic running in a separate thread.

As FiCAT sends a GET start request, the server initializes a daemon thread. The thread calls a method that polls data from the ultrasonic sensor once every second. This is critical for environmental safety while the robot is in motion. The polling takes around 100 milliseconds if the calculated distance is not less than the distance threshold. The main program utilizes the remaining 900 milliseconds in listening for API requests.

In circumstances where an obstacle is detected within the critical range, the callback method of the thread sends a stop signal to the STM32 motor controller. The robot stops until the STM32 motor controller receives a start signal from the Raspberry Pi. As soon as the path is cleared out of any obstacles, the thread sends a start signal to the robot again.

6 System Test and Verification

Testing and verification are an integral part of product development life cycle. Keeping that in mind, the robot was physically tested in the field. The motors were tested iteratively when the motor controller program was developed during the initial phase. The API server was also tested during all phases of development such as the implementation of each endpoint, the integration of the serial interface and the ultrasonic sensor, the implementation of the OAuth2 authentication layer, and the integration with the FiCAT client.

6.1 Robot Control Test

The test environment was created with a black tape that was used to create a test track for the robot. The width of the tape was five centimetres that allowed a maximum of two IR sensors to read the reflectance simultaneously. The track was a rectangular loop with a length of approximately six meters where the robot was placed randomly. An overview of the test environment can be seen in Appendix 7. The robot was able to align its central IR sensors in the line on the straight path. At the corners, the robot occasionally moved over the line. This overcorrection was resolved by the overcorrection correction logic as seen in Listing 9.

The first test was carried out repeatedly for seven hours. The robot exhibited strange behaviour when the IR sensors were exposed to direct sunlight. Instead of following the line, the robot was lost and followed a random path. The solution was to isolate the test field from direct sunlight. As light falls on the tape, the reflectance of the black tape increases and phototransistors come into play as discussed in Section 2.6. After the issue was resolved, the robot was tested for hundreds of hours with successful results. There were obstacles placed at multiple points of the line. The robot was successfully able to detect those obstacles and take the necessary actions.

6.2 API Calls Test

The API server running on the Raspberry Pi was successfully tested with Postman. Initially, each endpoint was tested for expected response. The server successfully handled the start request and executed the required tasks such as initialising the thread and sending a signal to the STM32 motor controller over the serial interface. Figure 22 below shows test results of post requests to acquire an authentication token.

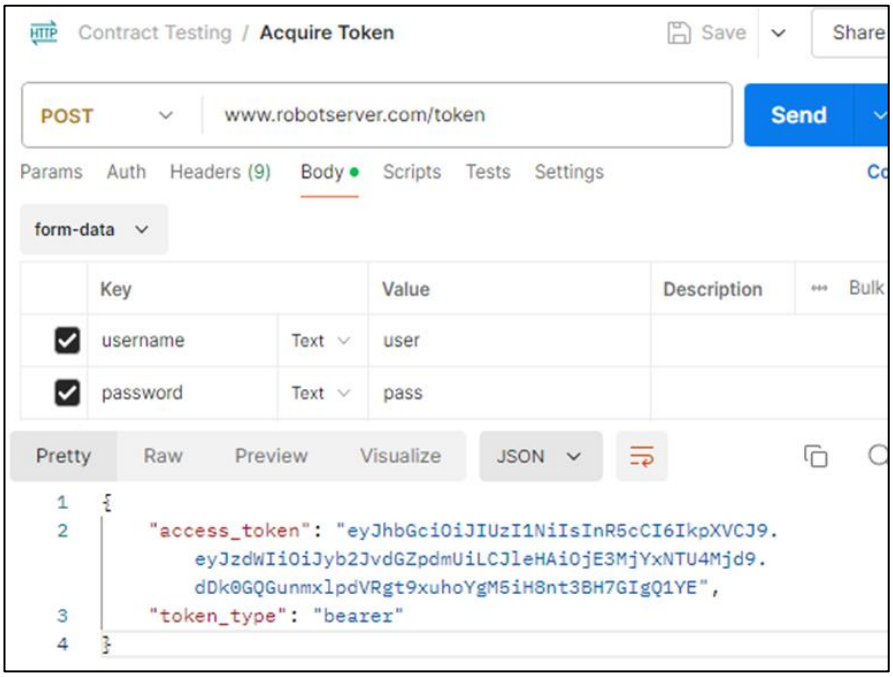


Figure 22. Token acquisition in Postman

Figure 22 shows an implementation of token acquisition. The domain name and token have been modified in this document due to security reasons. The body holds the username and password required in authenticating with the server to acquire the token.

A load test was carried out on Postman with 100,000 requests at regular intervals. A start request was sent every three seconds followed by a stop request. The time elapsed in the full process was six seconds. There were some networks connectivity issues specifically with the firewall; otherwise, the test was successful. The server performed as per the requirement. This test ensured the reliability of the system in case of prolonged testing which can last for days.

One of the major aspects of this project was the integration of the robot with the internal automation framework of the case company. After developing the server and the client with Python libraries, a Robot Framework script was developed for FiCAT. The end-to-end test was executed successfully for 98,968 times out of 100,000. All failed tests were due to network issues due to an expired firewall authentication session.

7 Conclusion

A robotic system consists of several components interacting with each other which can be implemented in different ways. It is of utmost importance to define and recognise the requirements before designing and implementing a solution. This project leveraged an STM32 motor controller board, Raspberry Pi 4, and two sensor modules to develop a line following robot.

This project aimed to develop a robot with a simple design and integrate it into the internal automation framework of the case company. A motor controller template that came with the STM32 motor controller board was used and modified to develop a line following robot. The internal automation framework of Nokia FiVe consists of a tool called FiCAT that schedules Robot Framework cases enabling automation. Raspberry Pi 4 B was configured as an API server to communicate with the STM32 motor controller and FiCAT. The Raspberry Pi was configured with a 4G LTE network protected behind a firewall. The STM32 motor controller and the Raspberry Pi were interfaced with the USART protocol. An array of six IR sensors fed reflectance data to the STM32 motor controller enabling effective line following. An ultrasonic sensor transmitted distance data to the Raspberry Pi to avoid any obstacles.

The robot controller program went through a series of tests during the development phase. The end-to-end test further ensured the reliability of the system. The robot was deployed in a field where it successfully collected data with the help of a UE. It simulated a mobile UE and contributed to a collection of 5G KPIs in a dynamic environment. However, the environment, where the robot is deployed, must be devoid of obstacles to allow the smooth operation of the robot. There can be some obstacles depending on the test setup. For instance, if the robot is deployed in an office environment with people frequently moving around, the robot might take a long time to complete a single loop that could affect the 5G KPI measurement results.

The system is currently at a juvenile stage. This project has opened a huge opportunity for the team to develop an advanced mobile test robot tailored to perform specific tasks. The line following algorithm can be easily modified to deploy the robot in different setups. The sensor handlers are scalable, and the API server is fast, modern, scalable, and user-friendly. A fleet of these robots can be deployed in different locations with similar network configurations. The influence of factors such as altitude, traffic density, weather conditions, and interference from other electromagnetic waves can be evaluated for better understanding of the test results.

References

1 Nokia. Nokia Announces First Phase of Its New Strategy [Internet]. Nokia [cited 2024 September 8]. Available from: https://www.nokia.com/about-us/news/releases/2020/10/29/nokia-announces-first-phase-of-its-new-strategy-changes-to-operating-model-and-group-leadership-team/

2 Nokia. 5G Standard [Internet]. Nokia [cited 2024 September 6]. Available from: https://www.nokia.com/licensing/patents/5g-standard

3 Siciliano B, Sciavicco L, Villani L, Oriolo G. Robotics. Springer London; 2008.

4 Boston Dynamics. Safety Solutions [Internet]. Boston Dynamics [cited 2024 September 8]. Available from: https://bostondynamics.com/solutions/safety

5 International Federation of Robotics. Robot History [Internet]. International Federation of Robotics [cited 2024 September 8]. Available from: https://ifr.org/robot-history

6 IEEE Spectrum. History of Technology [Internet]. Allison March; 2022 [cited 2024 September 8]. Available from: https://spectrum.ieee.org/unimation-robot

7 ABB. Dual-Arm YuMi – IRB 14000 [Internet]. ABB [cited 2024 September 8]. Available from: https://new.abb.com/products/robotics/robots/collaborative-robots/yumi/dual-arm

8 YouTube. Facts about Industrial Robots [Internet]. International Federation of Robotics [cited 2024 September 8]. Available from: https://youtu.be/mtxMYJz4v2Y

9 Progressive Automations. Actuators [Internet]. Progressive Automations [cited 2024 September 8]. Available from: https://www.progressiveautomations.com/pages/actuators

10 Circuit Digest. Building an Easy Line Follower Robot Using Arduino Uno [Internet]. Rajesh; 2021 [cited 2024 September 8]. Available from: https://circuitdigest.com/microcontroller-projects/arduino-uno-line-follower-robot

11 Penttinen JTJ. 5G Explained. Wiley; 2019.

12 STMicroelectronics. STM32F103 Reference Manual [Internet]. STMicroelectronics [cited 2024 September 8]. Available from: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

13 Amazon Web Services. What Is API? [Internet]. Amazon Web Services [cited 2024 September 8]. Available from: https://aws.amazon.com/what-is/api/

14 Peak Sensors. What Is RTD Sensor And How Does It Work? [Internet]. Peak Sensors [cited 2024 September 8]. Available from: https://peaksensors.com/blog/resistance-thermometer/what-is-rtd-sensor-and-how-does-it-work/

15 ElectronicsHub. What Is a Sensor, Different Types of Sensors and Their Uses [Internet]. Ravi Teja; 2024 [cited 2024 September 8]. Available from: https://www.electronicshub.org/different-types-sensors/

16 DigiKey. How to Use a Phototransistor with an Arduino [Internet]. DigiKey [cited 2024 September 8]. Available from: https://www.digikey.com/en/maker/blogs/2022/how-to-use-a-phototransistor-with-an-arduino

17 Elprocus. What Is a Phototransistor: Circuit Diagram & Its Working [Internet]. Elprocus [cited 2024 September 8]. Available from: https://www.elprocus.com/phototransistor-basics-circuit-diagram-advantages-applications

18 WatElectronics. What Is Ultrasonic Sensor: Working & Its Applications [Internet]. WatElectronics [cited 2024 September 8]. Available from: https://www.watelectronics.com/ultrasonic-sensor/

19 Discover Wildlife. What Is Echolocation [Internet]. Jo Price; 2022 [cited 2024 September 8]. Available from: https://www.discoverwildlife.com/animal-facts/what-is-echolocation

20 WatElectronics. What Is IR Sensor: Circuit & Its Working [Internet]. WatElectronics [cited 2024 September 8]. Available from: https://www.watelectronics.com/ir-sensor/

21 Faulhaber. DC Motor Tutorial: Motor Calculations for Coreless Brush DC Motors [Internet]. Faulhaber [cited 2024 September 8]. Available from: https://www.faulhaber.com/en/know-how/tutorials/dc-motor-tutorial-motor-calculations-for-coreless-brush-dc-motors/

22 Renesas. Brushless DC Motor Overview [Internet]. Renesas [cited 2024 September 8]. Available from: https://www.renesas.com/us/en/support/engineer-school/brushless-dc-motor-01-overview

23 Raspberry Pi. Raspberry Pi Documentation [Internet]. Raspberry Pi [cited 2024 September 8]. Available from: https://www.raspberrypi.org/documentation/

24 STMicroelectronics. Datasheet - STM32F103xC, STM32F103xD, STM32F103xE [Internet]. STMicroelectronics [cited 2024 September 8]. Available from: https://www.st.com/resource/en/datasheet/stm32f103rc.pdf

25 SparkFun. TB6612FNG Hookup Guide [Internet]. SparkFun [cited 2024 September 8]. Available from: https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide/all

26 SparkFun. TB6612FNG [Internet]. Toshiba; 2007 [cited 2024 September 8]. Available from: https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf

27 InvenSense. MPU-6000 and MPU-6050 Product Specification Revision 3.4 [Internet]. InvenSense; 2013 [cited 2024 September 8]. Available from: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

28 YouTube. TCRT5000 Infrared Reflective Sensor - How It Works and Example Circuit with Code [Internet]. DIY Machines [cited 2024 September 8]. Available from: https://youtu.be/mtxMYJz4v2Y

29 Vishay. TCRT5000, TCRT5000L [Internet]. Vishay Semiconductors [cited 2024 September 8]. Available from: https://www.vishay.com/docs/83760/tcrt5000.pdf

30 SparkFun. HC-SR04 Proximity Sensor Datasheet [Internet]. SparkFun [cited 2024 September 8]. Available from: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

31 Byjus. FastAPI [Internet]. Byjus [cited 2024 September 8]. Available from: https://byjus.com/physics/hall-effect/

32 STMicroelectronics. STM32CubeIDE User Guide [Internet]. STMicroelectronics [cited 2024 September 8]. Available from: https://www.st.com/resource/en/user_manual/um2609-stm32cubeide-user-guide-stmicroelectronics.pdf

33 GitHub. STM32F103RCT6 Code Template [Internet]. Moebius Tech [cited 2024 September 8]. Available from: https://github.com/MoebiusTech/Stm32f103rct6

34 Arm Ltd. CMSIS [Internet]. Arm Ltd [cited 2024 September 8]. Available from: https://arm-software.github.io/CMSIS_6/latest/General/index.html

35 Robot Framework. Robot Framework [Internet]. Robot Framework [cited 2024 September 8]. Available from: https://robotframework.org/

36 FaspAPI. FastAPI [Internet]. Tiangolo [cited 2024 September 8]. Available from: https://byjus.com/physics/hall-effect/

37 Auth0. What Is OAuth 2.0? [Internet]. Auth0 [cited 2024 September 8]. Available from: https://auth0.com/intro-to-iam/what-is-oauth-2

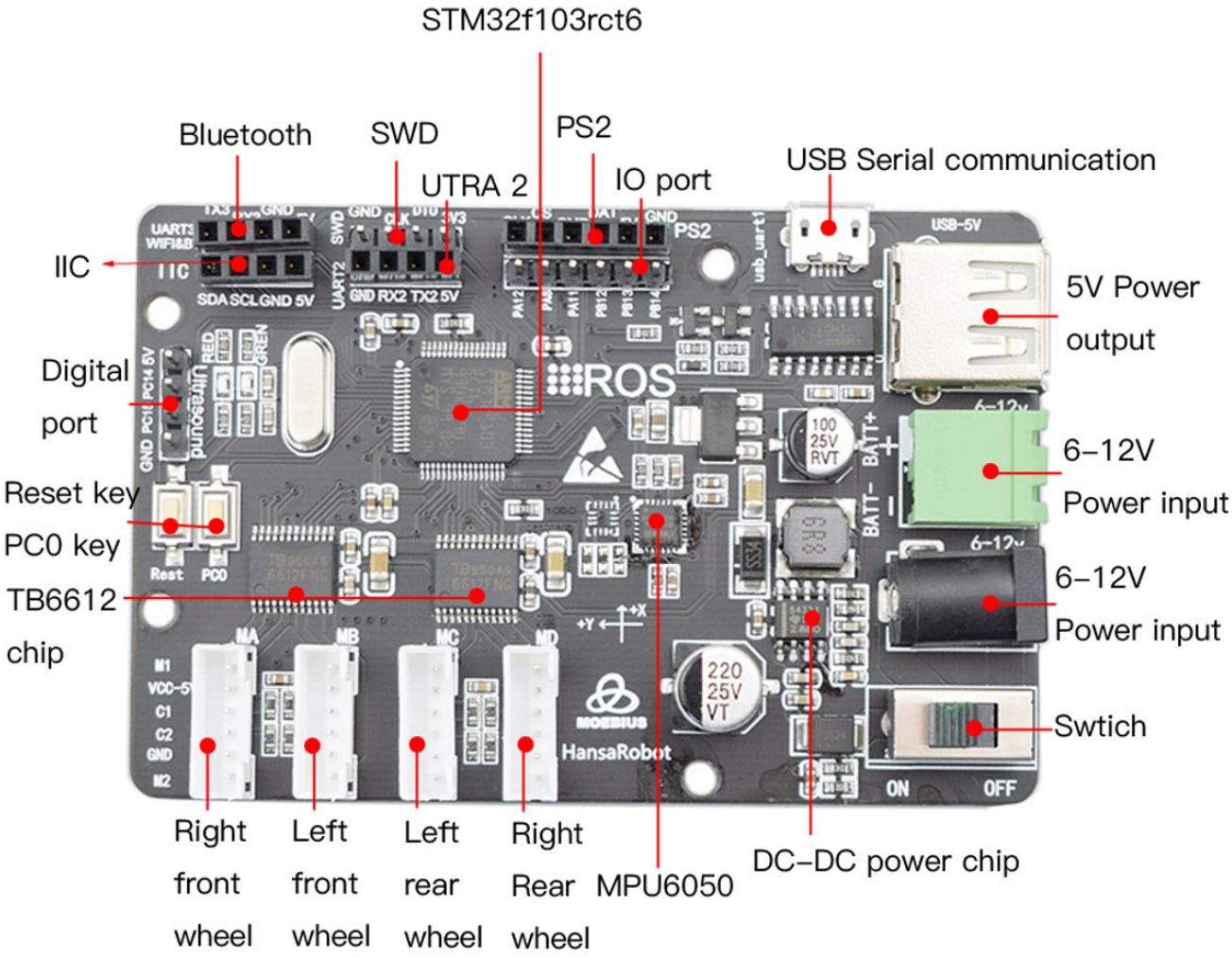
38 FreeCAD. Getting Started [Internet]. FreeCAD [cited 2024 September 8]. Available from: https://wiki.freecad.org/Getting_started

39 Formlabs. Guide to 3D Printing [Internet]. Formlabs [cited 2024 September 8]. Available from: https://formlabs.com/eu/3d-printers/

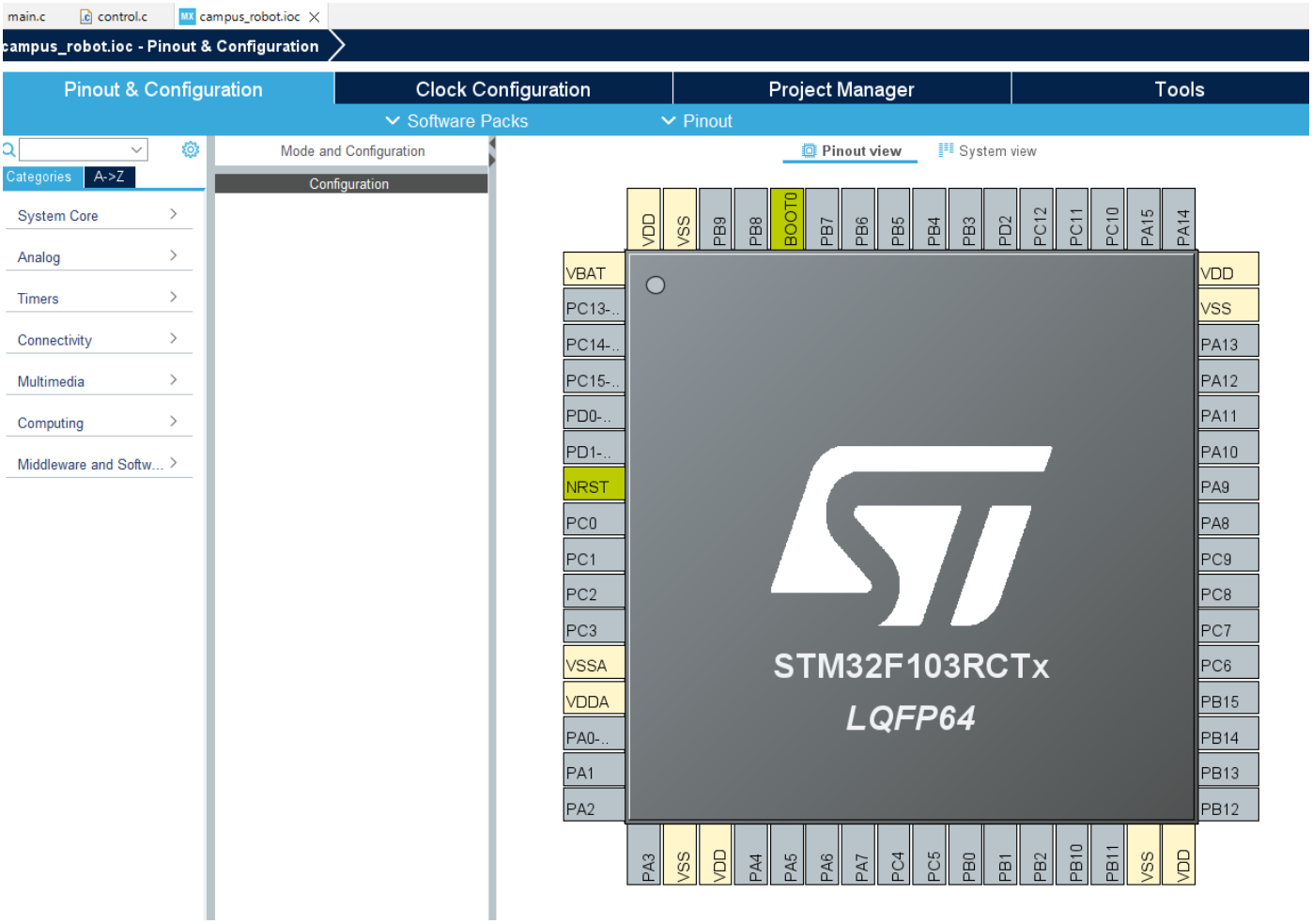
40 STMicroelectronics. STM32 Standard Peripheral Libraries [Internet]. STMicroelectronics [cited 2024 September 8]. Available from: https://www.st.com/en/embedded-software/stm32-standard-peripheral-libraries.html

41 RealPython. Python Requests [Internet]. Alex Ronquillo; 2024 [cited 2024 September 8]. Available from: https://realpython.com/python-requests/

Motor Controller Layout



Overview of STM32CubeIDE Interface



PWM Configuration for Motors

```
// Enable clock for TIM8 and GPIOC
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

RCC->APB2ENR |= 1 << 13;  // Enable TIM8 clock
RCC->APB2ENR |= 1 << 4;   // Enable PORTC clock
GPIOC->CRH &= 0xFFFFFF00; // Configure PORTC8 for alternate function output
GPIOC->CRH |= 0x000000BB; // Configure PORTC8 for alternate function output

GPIOC->CRL &= 0x00FFFFFF; // Configure C6, C7 for alternate function output
GPIOC->CRL |= 0xBB000000; // Configure C6, C7 for alternate function output

TIM8->ARR = arr;          // Set auto-reload register value
TIM8->PSC = psc;          // Set prescaler value
TIM8->CCMR1 |= 6 << 4;    // Set channel 1 to PWM mode 1
TIM8->CCMR1 |= 6 << 12;   // Set channel 2 to PWM mode 1
TIM8->CCMR2 |= 6 << 4;    // Set channel 3 to PWM mode 1
TIM8->CCMR2 |= 6 << 12;   // Set channel 4 to PWM mode 1

TIM8->CCMR1 |= 1 << 3;    // Enable preload register on channel 1
TIM8->CCMR1 |= 1 << 11;   // Enable preload register on channel 2
TIM8->CCMR2 |= 1 << 3;    // Enable preload register on channel 3
TIM8->CCMR2 |= 1 << 11;   // Enable preload register on channel 4

TIM8->CCER |= 1 << 0;     // Enable output on channel 1
TIM8->CCER |= 1 << 4;     // Enable output on channel 2
TIM8->CCER |= 1 << 8;     // Enable output on channel 3
TIM8->CCER |= 1 << 12;    // Enable output on channel 4

TIM8->CCR1 = 0;           // Set duty-cycle as 0
TIM8->CCR2 = 0;
TIM8->CCR3 = 0;
TIM8->CCR4 = 0;

TIM8->BDTR |= 1 << 15;    // Enable main output

TIM8->CR1 = 0x8000;       // Enable auto-reload preload
TIM8->CR1 |= 0x01;        // Enable timer
```

External Interrupt Handler and Closed-Loop Controller

```
void Kinematic_Analysis(float Vx,float Vy,float Vz)
{

#if  AXLE_Z_RESTRAIN
     int temp;
     if(!KEY1)  Gyro_Bias = Yaw;
     temp = Yaw - Gyro_Bias;
     if (temp > 180)
          temp = 360-temp;
     if (temp < -180)
          temp = 360 + temp;
     if(temp > 1 || temp < -1)
          Vz += Gyro_K * temp;
#endif
     motor_a.target   = -Vx+Vy+Vz*(a_PARAMETER+b_PARAMETER);
     motor_b.target   = +Vx+Vy-Vz*(a_PARAMETER+b_PARAMETER);
     motor_c.target   = +Vx+Vy-Vz*(a_PARAMETER+b_PARAMETER);
     motor_d.target   = -Vx+Vy+Vz*(a_PARAMETER+b_PARAMETER);
}

#if DEBUG_CHAR
int do_debug_modify(char c){
     static char buffer[32];
     static int current = 0;
     if(c == ' '){
          buffer[current] = 0;
          *coefficient = (float) atof(buffer);
          //printf("new coef: %f\n\r", *coefficient);
          current = 0;
          return 1;
     }else{
          buffer[current++] = c;
          if(current > 31) current = 0;
          return 0;
     }
}

#endif
```

```
/**************************************************************************
Function description: All control codes are here
         The 5ms scheduled interrupt is triggered by the INT pin of MPU6050
         Ensures time synchronization of sampling and data processing
**************************************************************************/
int EXTI15_10_IRQHandler(void)
{
     float ret = 0.0;
     int Yuzhi=20;
     if(INT==0)
     {
          EXTI->PR=1<<15;        //Clear the interrupt flag bit on LINE5
          if(TimClk)
          {
               TimClk--;
               if(TimClk == 0)
               {
                    TimClk = 200;
                    LED = ~LED;
               }
          }

          Flag_Target=!Flag_Target;

          if(delay_flag==1)
          {
               //Provide a precise 50ms delay for the main function
               if(++delay_50==10) delay_50=0, delay_flag=0;
          }
          if(myCounter >= 999999998) myCounter = 0; //resets in 11.6 days


          motor_a.encoder =+ Read_Encoder(2);  //Read the value of the encoder
          motor_a.position += motor_a.encoder; //Integrate to get position
          motor_b.encoder =- Read_Encoder(3);  //Read the value of the encoder
          motor_b.position += motor_b.encoder; //Integrate to get position
          motor_c.encoder =- Read_Encoder(4);  //Read the value of the encoder
          motor_c.position += motor_c.encoder; //Integrate to get position
          motor_d.encoder =+ Read_Encoder(5);  //Read the value of the encoder
          motor_d.position += motor_d.encoder; //Integrate to get position

          Read_DMP();                          //Update status
          Voltage_All+=Get_battery_volt();     //Accumulate multiple samplings
          if(++Voltage_Count==100) {
               Voltage=Voltage_All/100;
               Voltage_All=0;
               Voltage_Count=0;      //Calculate average value to get battery voltage
          }
```

```
if(InspectQueue())
{

#if DEBUG_CHAR
     static int modify = 0;
     char c = OutQueue();

     if(!modify) switch(c){
          case 'P':
               coefficient = &Velocity_KP;
               modify = 1;
               break;
          case 'I':
               coefficient = &Velocity_KI;
               modify = 1;
               break;
          case 'D':
               coefficient = &Velocity_KD;
               modify = 1;
               break;
          case 'i':
               coefficient = &initial_correction;
               modify = 1;
               break;
          case 'd':
               coefficient = &dynamic_correction;
               modify = 1;
               break;
          default:
               Flag_Direction = c;
               Motors_reset();
               break;
     }
     if(modify){
          while(InspectQueue()){
               if(do_debug_modify(OutQueue())) {
                    modify = 0;
               }
          }
     }
#else
          Flag_Direction = OutQueue();
          Motors_reset();
#endif
          }
```

```
ret = line_state();

if(Flag_Direction != 'Z'){
     if( ret != 254 && ret != 255 && Flag_Direction != 'C'){ //

          static int state = 0;
          int nstate = ret < 2.4 ? -1 : (ret > 2.6 ? 1 : 0);

          if(state != nstate){ // if the mean value changes
               state = nstate;
               if(state < 0) Move_X -= initial_correction;

               else if(state > 0) Move_X += initial_correction;

          }

          if (ret > 2.6) Move_X += dynamic_correction*(ret-2.5);
          else if (ret < 2.4) Move_X -= dynamic_correction*(2.5-ret);
          Get_RC(0);
          Update_PI();

     }

     else if( Flag_Direction == 'C'){
          if (ret > 2.4 && ret < 2.6) {
               Motors_reset();
               Flag_Direction = 'A';
               Get_RC(0);
               Update_PI();
          }
     }

     Limiter_Pwm(7200);
     Set_Pwm(motor_a.pwm, motor_b.pwm, motor_c.pwm, motor_d.pwm);

} //if(Flag_Direction != 'Z')

else {
     //reset everything
     Motors_reset();
     Set_Pwm(motor_a.pwm, motor_b.pwm, motor_c.pwm, motor_d.pwm);
}

}
     return 0;
}
```

```
void Motors_reset(){
     Motor_reset(&motor_a);
     Motor_reset(&motor_b);
     Motor_reset(&motor_c);
     Motor_reset(&motor_d);
     Move_X = Move_Y = Move_Z = 0;
}


void Motor_reset(Motor_PI * motor){
     motor->encoder = 0;
     motor->last_bias = 0;
     motor->pwm = 0;
     motor->target = 0;
     motor->pid_value = 0;
     motor->position = 0;
}

void Update_PI(){
     Incremental_PI(&motor_a);  //Speed closed-loop control calculates the final PWM for motor A
     Incremental_PI(&motor_b);  //Speed closed-loop control calculates the final PWM for motor B
     Incremental_PI(&motor_c);  //Speed closed-loop control calculates the final PWM for motor C
     Incremental_PI(&motor_d);  //Speed closed-loop control calculates the final PWM for motor D
}

/**************************************************************************
Function Description: Assign values to PWM registers
Input Parameters: PWM
Return Value: None
**************************************************************************/
void Set_Pwm(int motor_a,int motor_b,int motor_c,int motor_d)
{
     if(motor_a<0)              INA2=1,              INA1=0;
     else if(motor_a>0)         INA2=0,              INA1=1;
     else                       INA2=0,              INA1=0;
     PWMA=myabs(motor_a);

     if(motor_b<0)              INB2=1,              INB1=0;
     else if(motor_b>0)         INB2=0,              INB1=1;
     else                       INB2=0,              INB1=0;
     PWMB=myabs(motor_b);

     if(motor_c<0)              INC2=1,              INC1=0;
     else if(motor_c>0)         INC2=0,              INC1=1;
     else                       INC2=0,              INC1=0;
     PWMC=myabs(motor_c);

     if(motor_d>0)              IND2=1,              IND1=0;
     else if(motor_d<0)         IND2=0,              IND1=1;
     else                       IND2=0,              IND1=0;
     PWMD=myabs(motor_d);
}
```

```
/*************************************************************************
Function Description: Limit PWM values
Input Parameters: Amplitude
Return Value: None
*************************************************************************/
void Limiter_Pwm(int amplitude)
{
     if(motor_a.pwm < -amplitude) motor_a.pwm = -amplitude;
     if(motor_a.pwm > amplitude)  motor_a.pwm = amplitude;
     if(motor_b.pwm < -amplitude) motor_b.pwm=-amplitude;
     if(motor_b.pwm > amplitude)  motor_b.pwm = amplitude;
     if(motor_c.pwm < -amplitude) motor_c.pwm = -amplitude;
     if(motor_c.pwm > amplitude)  motor_c.pwm = amplitude;
     if(motor_d.pwm < -amplitude) motor_d.pwm = -amplitude;
     if(motor_d.pwm > amplitude)  motor_d.pwm = amplitude;
}

/*************************************************************************
Function Description: Emergency shutdown
Input Parameters: Voltage
Return Value: 1 (emergency), 0 (normal)
*************************************************************************/
u8 Turn_Off( int voltage)
{
     u8 temp;
     if(voltage<2000||EN==0)        //Battery voltage below 22.2V, shut down the motor
     {
          temp=1;
          PWMA=0;
          PWMB=0;
          PWMC=0;
          PWMD=0;
     }
     else
          temp=0;
     return temp;
}

/*************************************************************************
Function Description: Absolute value function
Input Parameters: long int
Return Value: unsigned int
*************************************************************************/
u32 myabs(long int a)
{
     u32 temp;
          if(a<0)  temp=-a;
     else temp=a;
     return temp;
}
```

```
/**************************************************************************
Function Description: Incremental PI controller
Input Parameters: Encoder measured value, Target speed
Return Value: Motor PWM
Formula: Incremental PID formula
pwm += Kp[e(k) - e(k-1)] + Ki*e(k) + Kd[e(k) - 2e(k-1) + e(k-2)]
e(k) represents current deviation
e(k-1) represents previous deviation
pwm represents incremental output
In our speed control closed-loop system, only PI control is used
pwm += Kp[e(k) - e(k-1)] + Ki*e(k)
**************************************************************************/

#define BIAS_LIMIT 1000

void Incremental_PI(Motor_PI * motor)
{
     int Bias = 0;
     Bias = motor->position - motor->target;                // Calculate deviation

     // Incremental PI controller
     motor->pid_value += Velocity_KP * ( Bias - motor->last_bias ) + Velocity_KI * Bias ;
     if ( motor->pid_value > 7200 )  motor->pid_value = 7200;
     if( motor->pid_value < -7200)   motor->pid_value = -7200;
     motor->last_bias = Bias;                     // Save the previous deviation
     motor->pwm = motor->pid_value - Velocity_KD * ( Bias - motor->last_bias );
}


/**************************************************************************
Function Description: Control the vehicle through serial commands
Input Parameters: Serial command
Return Value: None
**************************************************************************/
void Get_RC(u8 mode)
{
     static uint8_t step=DEFAULT_SPEED;  // Set the step size of speed control.
     u8 Flag_Move=1;
     if(mode == 0){
          switch(Flag_Direction)   // Direction control
          {
               case 'A':
                    Move_Y+=step;
                    Flag_Move=1;
                    break;
               case 'C':
                    Move_X+=step;
                    Flag_Move=1;
                    break;
               case 'Z':
                    default:
                    Move_X=0;
                    Move_Y=0;
                    Move_Z=0;
                    Motors_reset();
                    //Set_Pwm(0, 0, 0 ,0);
                    step = DEFAULT_SPEED;
                    break;
               case 'L':  step+=5;   Flag_Direction = 'A'; break;
               case 'M':  step-=5;   Flag_Direction = 'A'; break;
          }
          if (step > 20) step = 20;
          if (step < 3) step = 3;
     }

     // Obtain control target value, perform motion analysis
     Kinematic_Analysis(Move_X,Move_Y,Move_Z);
}
```

Reading Reflectance

```
#define IR_THRESHOLD 4000
#define NUMBER_OF_CHANNELS 6

IR_mapping ir_mapping[] = {
          {"L1", 13},
          {"L2", 12},
          {"CL", 11},
          {"CR", 4},
          {"R1", 3},
          {"R2", 2}
     };

/**
  * @brief  Get ADC channel from the IR position
  * @param  IR sensor's position
  * @retval ADC_channel
  */
u8 get_channel_number(const char * pos) {
          uint8_t i = 0;
          for( i = 0; i < (sizeof(ir_mapping)/sizeof(IR_mapping)); ++i){
                    if(strcmp(ir_mapping[i].ir_pos, pos) == 0){
                              return ir_mapping[i].adc_channel;
                    }
          }
          // error handler if position not found
          return 255;
}

/**
  * @brief  Read ADC value on the associated channel.
  * @param  IR sensor's position
  * @retval Value read on the associated channel, uint16_t, ADC1->DR
  */
uint16_t read_adc(const char * IR){
          u8 channel = get_channel_number(IR);
          if (channel != 255){
                    return Get_Adc(channel);
          }
          else {
                    printf("Error: Invalid IR position '%s'\n", IR);
        return 0;
          }
}
```

Calculate and Return Mean

```
float line_state(void){
     static float old_mean = 2.5;
     u8 counter = 0;
     u8 i = 0;
     float mean_reading = 0.0;
     float variance = 0.0;

     u8 digital_readings [NUMBER_OF_CHANNELS];
     /* on black line = 1, outside = 0 */
     l2 = digital_readings [0] = digital_IR("L2", IR_THRESHOLD);
     l1 = digital_readings [1] = digital_IR("L1", IR_THRESHOLD);
     cl = digital_readings [2] = digital_IR("CL", IR_THRESHOLD);
     cr = digital_readings [3] = digital_IR("CR", IR_THRESHOLD);
     r1 = digital_readings [4] = digital_IR("R1", IR_THRESHOLD);
     r2 = digital_readings [5] = digital_IR("R2", IR_THRESHOLD);

     printf("%d %d %d %d %d %d\n\r", l2, l1, cl, cr, r1, r2);

     for(i = 0; i < NUMBER_OF_CHANNELS; i++){
          counter += digital_readings[i];
          mean_reading += (i * digital_readings[i]);
     }
     printf("counter %d ", counter);
     if (!counter) return old_mean;
     if (counter > 4) return 100;
     mean_reading /= counter;
     old_mean = mean_reading;

     for (i = 0; i < NUMBER_OF_CHANNELS; i++){
          variance += ( digital_readings[i]*(i - mean_reading)*(i - mean_reading) );
     }

     if (counter == 5 ) printf("variance %d, all sensors active\n\r", variance);

     if (variance > 17.4) return -1; // at the edge, turn 180d

     else return mean_reading;
```

Test Environment for the Robot