CHAPTER 1. INTRODUCTION

1.1 Background

1.1.1 AGV

An Automatic Guided Vehicle (AGV) is an autonomous or mobile robot, guided by markers which could be wires, magnetic tape on the floor, or uses radio waves, vision cameras, magnets, or laser references. These vehicles are increasingly used in the industrial sector to move materials from the production plant to the automated warehouse, or to manage storage more efficiently with a lower risk for the staff [1].

Figure 1.1 Some types of AGV

1.1.2. History of AGV

During the upturn in industry in the 1950s, manual activities, staffed forklifts and conveyor technology at floor level dominated the scene in the production halls. Heavier parts could only be stored at ground level. Many entrepreneurs therefore had the idea of completely replacing human labor with machines. This ignited the inventive spirit of Arthur Barrett. In 1954, in Northbrook, Illinois, he and his company Barrett Electronics invented the "Guide-O-Matic", the world's first driverless vehicle. It was not until around 1980 that the term Automated Guided Vehicle (AGV) became established [2].

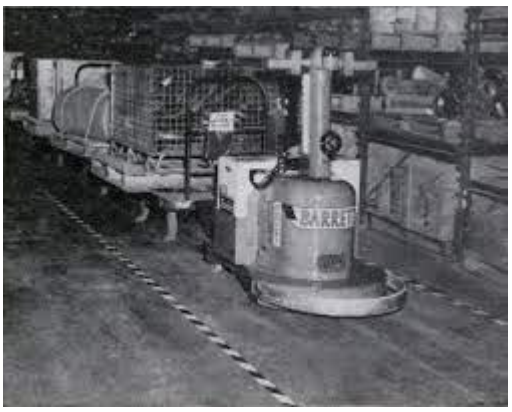

Figure 1.2. One of the first AGV, built in 1954.

The first AGVs had quite simple tracking technology and at first were oriented using colored strips on the floor. The signals from the color strip were transmitted via an optical sensor to a motor on the steering wheel, which moved the wheel accordingly. Sensors were by no means common currently. Mechanical switches, emergency stop devices and bumpers protected the AGV from its surroundings and vice versa.

Later, the vehicles were navigated by a wire mounted below the ceiling, followed by a wire embedded in the floor. Grid navigation worked in an equivalent way. Coupled navigation, which originated in seafaring, was based purely on the departure point and was imprecise. Track guiding using magnets also became a popular method. With the very first technologies, the stopping points and transfer points were marked with floor magnets, which were detected by the vehicle sensor.

Thanks to technological advances, particularly in the IT and scanner sectors, laser navigation was introduced around 1995. Physical guidelines have thus become obsolete. The flexibility of the AGV improved significantly. Redirection could be achieved with minimal effort. Technology quickly became immensely popular. The LGV (Laser Guided Vehicle) thus became synonymous with the AGV.

Over the course of many years, advancements in computing technology have played a significant role in the development of vehicle controlling systems. Recently, there has been a growing interest in using computational techniques for path planning and scheduling in Automated Guided Vehicle Systems (AGVs). These are crucial aspects that directly impact the efficiency of manufacturing systems. Path planning involves finding the best non-conflicting routes for each AGV to perform their tasks, while scheduling deals with ordering and assigning tasks to AGVs for optimal performance, considering factors like energy and time.

Numerous studies have focused on applying metaheuristic algorithms to solve the routing problem in AGVs. Various algorithms have been utilized for optimal path planning [3]. For example, Shan et al. used the Dijkstra algorithm to address the multi-AGV collision-free path planning problem. Additionally, heuristic methods like the Ant Colony Optimization (ACO) [4]

with fallback strategy and the Artificial Bee Colony (ABC) algorithm [5] have been introduced to improve path planning for AGV systems.

The primary objective of these studies has been to enhance the timely delivery of tasks and to ensure collision and conflict-free paths. However, recent research has started considering energy consumption as a novel consideration in AGV path planning. It has been observed that energy-efficient operation of AGV systems can lead to overall improvements in the manufacturing process.

Researchers, such as Zhongwei et al., have developed an energy efficient AGV path planning (EAPP) model and explored solution methods like particle swarm optimization (PSO) to achieve this goal [6]. Additionally, Riazi et al. proposed an optimization method to minimize performance measures like makes pan, maximum lateness, and sum of tardiness in AGV path planning.

Overall, these advancements in computing technology and the integration of energy-efficient strategies are driving noteworthy progress in AGVs control systems.

1.1.3 AGV types

As AGVs are used in many areas of application they can look quite different and have different attributes. There is an opinion that the best way to categorize AGVs is by looking at the loads they transport.

1.1.3.1 The Forklift AGV

This vehicle's load unit is pallets or forklift-compatible containers. It can be used independently or with other AGVs, then managed by an AGV guidance control system.

Figure 1.3. A forklift AGV

1.1.3.2 The Piggyback AGV

This AGV can carry pallets, boxes, or containers. In contrast to the forklift AGVs mentioned in the previous, this AGV cannot lift the load directly from the floor but requires a certain height which must be maintained throughout the loading and unloading areas of the AGV. The advantage of this type of AGV is their lateral

load handling. They can drive up to the loading area and directly transfer the load without turning and maneuvering as a forklift would have to. This results in less space needed for loading operations and can be done quickly using the conveyor belts.

Figure 1.4. A Piggyback AGV

1.1.3.3 The Towing vehicle

Towing vehicles tow several trailers behind it. There are two categories when it comes to towing AGVs. Either they can be specially designed towing AGVs without a space for a human (Figure 1.5) or towing AGVs as automated serial equipment (Figure 1.6).

Figure 1.5. A serial produced towing vehicle.

1.1.3.4 The underride AGV

An underride AGV goes under a roller cart or material wagon and lifts it slightly. They require less space than many other AGVs as the container itself determines

Figure 1.6. An underride AGV

the space required entirely. It also has high maneuverability when loading and unloading.

1.1.3.5 The Assembly Line AGV

Assembly line AGVs carry the assembly object on a path where the object is being constructed during transport. Assembly line AGVs are quite different from transport AGVs. Here the assembly object, its size and weight are important when choosing AGV. The assembly stages are also important to consider. These AGVs usually have simpler navigation systems than other AGVs and move at a slower pace.

Figure 1.7. An assembly line AGV

1.2 Scope of work

In this project, the AGV structures are studied and compared from various points of view and analysis of the AGV structure is done for designing an AGV robot and a basic AGV system. This project presents the mathematical model, control policy and AGV design, based on a mechatronics design methodology proposed by our group. Our design uses Landmarked Based Navigation with line follower. Besides, this project also concerns the design of an automated guided vehicle system, consisting of the AGV, the control and monitoring system and interface, and the communication between the AGV and the control server. Finally, some energy optimization algorithms are studied and simulated.

This project starts with the dynamics models, continues with the mechanical, electronic and control system development, and finally the working of a robot in the system and test results are shown. The design methodology was based on mechanical and mechatronic design methodologies, the principal phases are problem statement, informational and conceptual design, and detailed design.

This project concerns the design of a mini AGV system. The vehicle should be able to follow predetermined paths, determine its location, transmit its parameters to the UI (User Interface) and receive commands from the operator. The AGV is controlled by operators using a user interface (UI), following predetermined paths on the from one point to another, and performing tasks.

A successful system must operate smoothly, without delay and accidents, while being efficient and fast. Such requirements call for the design of both the hardware (the vehicles) and the communication system. Each vehicle in the system should operate reliably together for maximum productivity and communicate with the operator through wireless transmission.

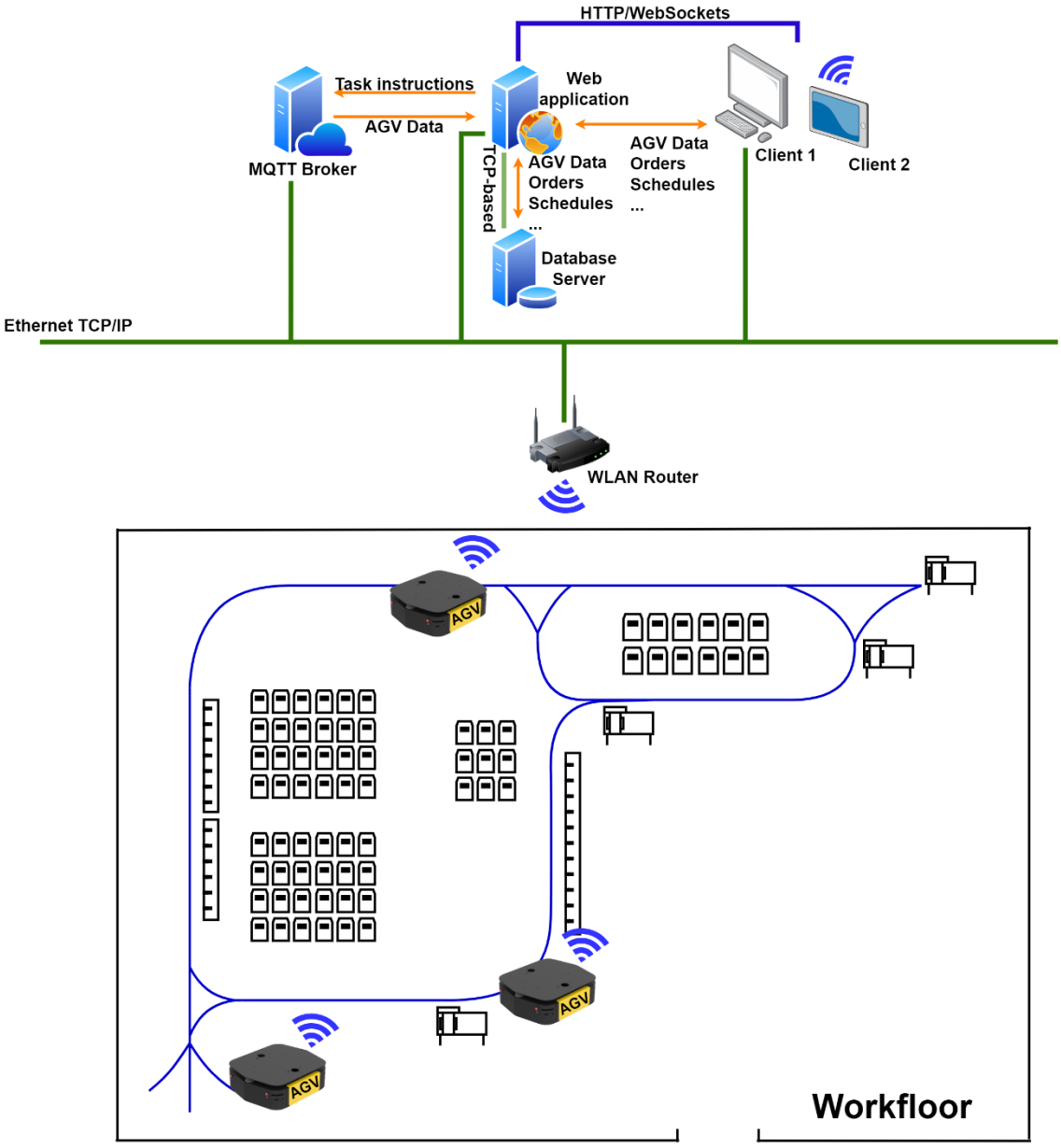

Figure 1.8 System layout in practical

Figure 1.8 presents a comprehensive overview of the practical system layout, featuring three AGVs operating on designated paths within the Work floor. The AGVs establish communication with the Web Application through a WLAN Router, employing the Ethernet TCP/IP Protocol. The Web Application utilizes the MQTT broker to receive data from the AGVs and facilitates the transmission of control signals generated by Clients. Clients interact with the Web Application, Server, and Database through the HTTP and WebSocket protocols, ensuring seamless communication and control over the AGVs. This system layout demonstrates effective data exchange and control mechanisms, contributing to the seamless functioning of the AGVs in the factory environment.

CHAPTER 2. SYSTEM DESIGN

2.1 Technical requirement

The project presents a proposal for the hardware design of an AGV, along with the design of a supervisory and control application for an AGV network. The system must operate smoothly, without delay and accidents, while being efficient and fast.

AGVs can navigate automatically by following colored line, designed readily in manufacture. AGVs can maintain a maximum constant speed among transporting, which does not depend on the weight of goods, and can turn smoothly with a maximum turning constant angular speed. AGVs can also detect and react to obstacles.

The control server is a web application. The application should be able to provide live updates of data sent by the AGVs. It should manage requests, orders, tasks scheduling and the optimal routing problem. The graphical interface should be simple yet functional, providing full controllability and understanding of the system. Communication between the AGV and the control server should be wireless, low latency, and secure.

2.2 Design Analysis

2.1.1 Block Diagram

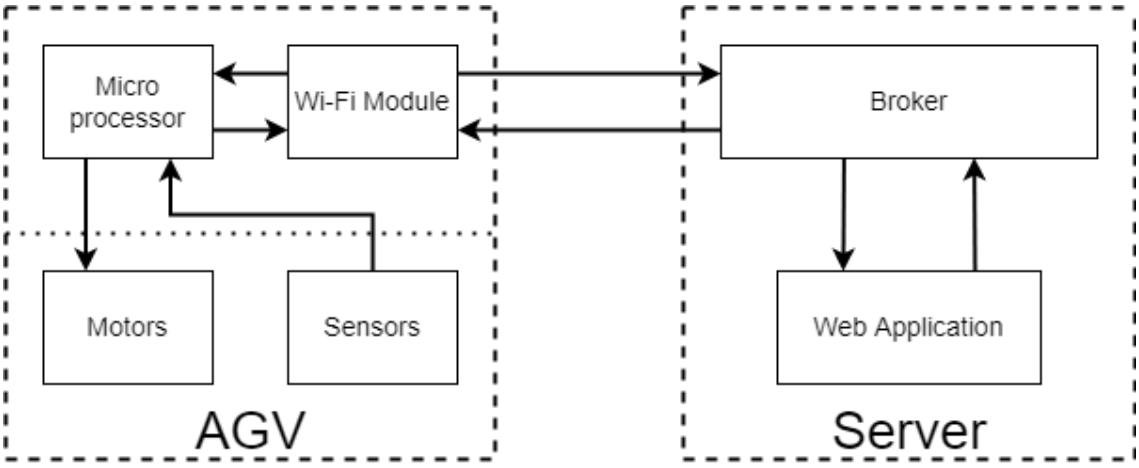

Figure 2.1. Block Diagram of the System

To complete the project, we divided it into two main parts: Hardware Development & Data Transmission. In the Hardware development, we build the AGV actuators and its controller – the STM32. In the Data Transmission part, we will create a Web application (User Interface) that can receive and present the dynamic data from moving AGV, the Web page would also work as an operator to send commands (desired path) for AGV to follow. The Web application will be hosted on a Server which contains the database of AGV. The transmission between AGV and Server & UI will be made by the ESP8266.

2.1.2 Theoretical solution

The AGV uses colored tape for the guided path. The AGV is fitted with the appropriate guide sensors to follow the tape's path. One major advantage of tape

over other types of navigation guidance is the inexpensiveness and accessibility, it can be easily removed and relocated if the course needs to change.

On the path are points which the AGV can scan and proceed with the according instructions, such as navigate to the next point, or performing pick and place tasks for materials. These points are simple RFID (short for Radio Frequency Identification) tags which use electromagnetic waves in radio frequency to transfer data. Corresponding to each RFID tag is a unique ID which can be scanned by the AGV, and the appropriate task at that point is sent to the AGV to perform. To scan the tags, an RFID sensor is used.

To drive the AGV, four motors are used, and the differential speed control method is utilized. In this method, each drive motor is driven at different speeds to turn, or the same speed to allow the AGV to go forwards or backwards. While this method of steering is simple and easy to approach as it does not require additional steering motors and mechanism, it is also suitable and satisfies our requirements in this project.

Forward sensing control uses an array of sensors to avoid collisions with objects on the AGV path, such as other AGVs, human, materials, etc. These sensors include sonic, which works like radar, and optical, which uses an infrared sensor. The two types of sensors have the same working principle: the sensors send a high frequency or infrared signal, which then gets reflected. The sensors can determine the location between the obstacle and the vehicle and help avoid collision.

To monitor and control the AGV, a central control system is used. The AGV connects wirelessly to a server which is managed by a host computer. Information and control interface of the AGV is shown on a web application. The website displays in real time the location of the vehicle. It also gives a status of the AGV, its battery level, unique identifier, current task(s), and material being managed. The user can give command to the AGV through this web application.

AGV must make decisions on path selection to come to the chosen target with the minimum energy consumed by the overall system. In this project, we show that by implementing effective scheduling and routing algorithm, it is possible to improve the energy efficiency of the AGV system. The planned path for each AGV is the input information of multi-AGV path-planning. As single AGV path-planning is the basis of multi-AGV path-planning, a single-load AGV was selected as the research object to facilitate study. To optimize the task assignments, objective functions, such as make span, total completion time, lateness, and tardiness, are used. The optimal route problem, consisting of the shortest path problem, is examined to provide the optimal route in terms of energy and time (cost function). In addition to this, the conflict avoidance requirement is considered to calculate and choose the optimal path to ensure system stability and efficiency. It shows that while the path chosen may not be the shortest path, overall, it allows for the safe travel of the AGV while still outperforming the original solutions in terms of energy cost.

An AGV car itself must contain a control policy to minimize the error of the system and control the car strictly following the predetermined line with a reference velocity from server. In this project, based on the dynamic model of a car, we propose a PID controller. The controller based on PID algorithm restores the measured speed to the desired speed with minimal delay and overshoot by increasing the power output of the engine in a controlled manner. In fact, PID is a control loop mechanism employing feedback widely used in industrial control systems and other applications requiring continuously modulated control. Below is the diagram showing the PID controller method.

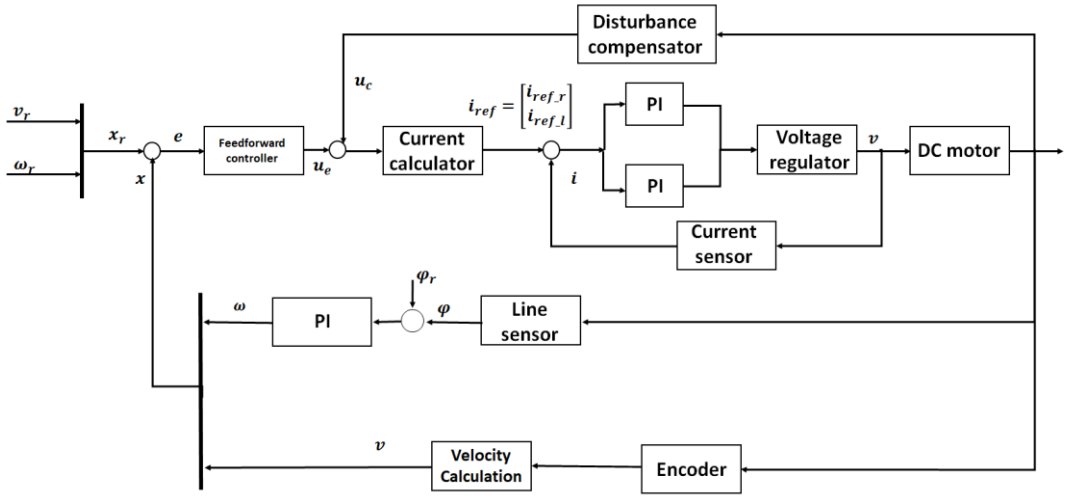

Figure 2.2. Diagram of PID Controller Method

As the picture described, the reference value is obtained from server, the feedback value is gotten from the action of the car, after going through the PID controller, we get the control signal for applying to the motor drive.

At the end of this report, after finishing a model AGV car with available modules in the market, we propose an embedded control system of an AGV to create an AGV in industrial in the future.

CHAPTER 3. HARDWARE DESIGN

This portion discusses the hardware design of the automated guided vehicle (AGV) employed in this project. The AGV's hardware is divided into smaller modules, as illustrated in Fig. 3.1, to facilitate the design process. This modular approach ensures that addressing issues, troubleshooting errors, and replacing components will be simple and efficient. The hardware design is categorized into the subsequent modules:

- Main controller module
- Motor drive module
- Obstacles detection module
- Line detection module
- Location module
- Power supply module
- Wireless communication module

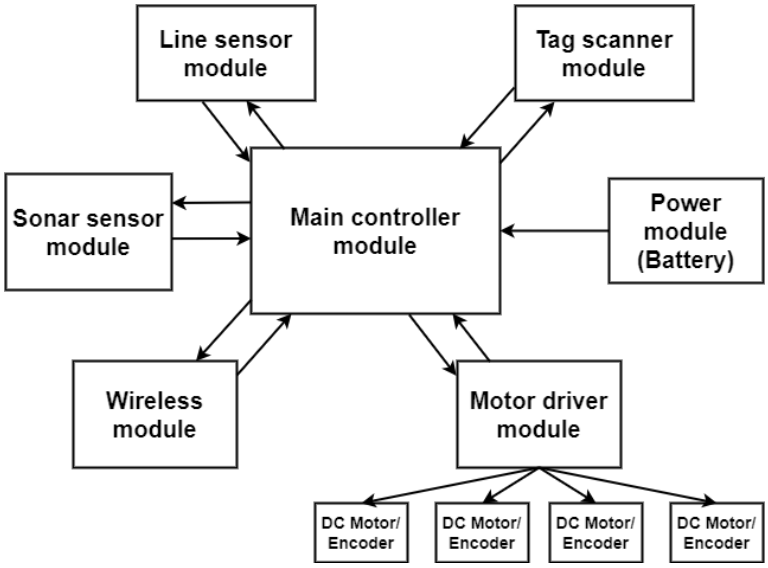

Figure 3.1. Block Diagram of Hardware

Figure 3.2 depicts the interconnections of modules within the AGV (Automated Guided Vehicle) hardware system. This configuration exemplifies a structured and efficient layout of peripheral components, emphasizing seamless integration with the main controller board, STM32 F334R8. The pivotal peripheral modules, encompassing the HC-SR04 Sonar sensor, TCRT5000 Line sensor, RDM6300 RFID reader, ESP8266 Wi-Fi communication module, and LN298N motor driver, together with encoders, are meticulously linked to the central STM32 F334R8 board. Furthermore, the DC motors are interconnected to the LN298N motor driver module. This well-organized arrangement ensures optimal functionality, facilitating smooth and synchronized operations of the AGV system.

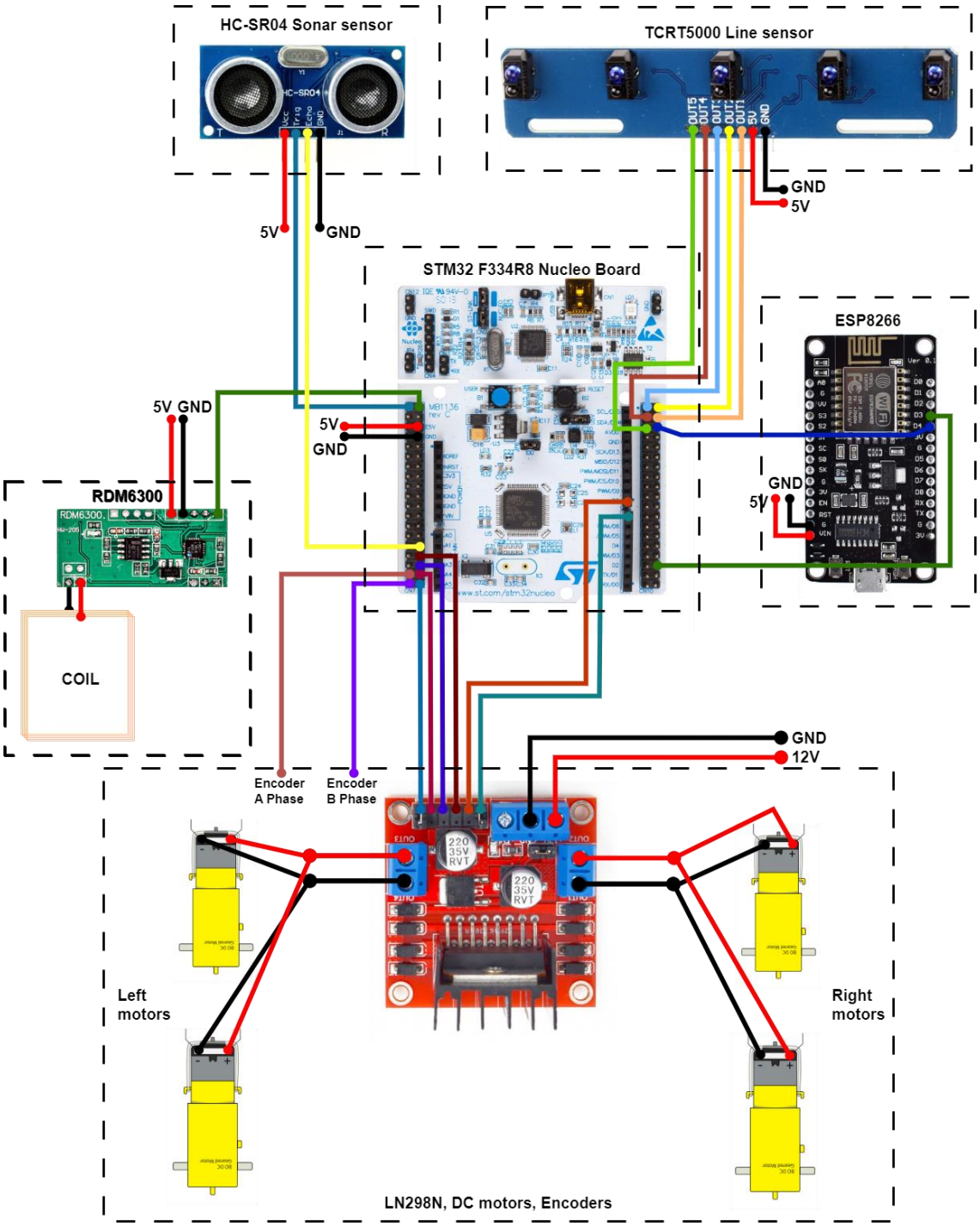

Figure 3.2. Hardware wiring diagram of AGV model

3.1 Hardware design

3.1.1 Main Controller Module

The performance of the AGV relies on the effectiveness of its embedded controller. In the AGV's basic configuration, either a microcontroller or PLC is utilized to perform tasks such as sensor data acquisition, communication with the central control system, and vehicle movement control. On the other hand, more advanced designers opt for a robust ARM chip operating with the Robot Operating System (ROS) to manage these functionalities.

For this project, the chosen microcontroller to control the basic system is the STM32 F334R8, equipped with a powerful Arm® Cortex®-M4 32-bit CPU. This microcontroller fulfills the requirements of high RAM capacity, fast processing, offering up to 51 fast I/O ports, and supporting at least 12 timers and interrupts. [7] [8],

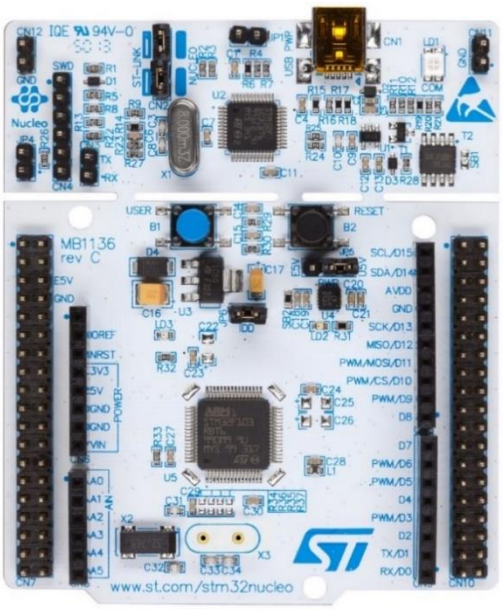

Figure 3.3. STM32 F334R8 Nucleo Board

The primary controller module possesses the following functionalities:

1. *Availability Check for Other Modules:* The main controller board verifies the connectivity with other modules. If any module fails to respond to the "Hello" signal, it indicates an error in that module. The AGV remains inactive until the issue is resolved.

2. *Signal Gathering and Processing from Sensor Modules:* The main controller module receives signals from other sensor modules within the system. These include the line detection module, which provides the vehicle's angular deviation from navigation lines; the proximity module, which indicates the distance between the robot and obstacles (humans, objects, or other vehicles); and the user interface module, responsible for sending direct control signals to the vehicle (e.g., power on/off and emergency stop). Additionally, the main controller module informs the user interface about the availability status of other modules in case they encounter operational errors.

3. *Movement Control Processing and Signal Transmission to Motor Drive Module:* The controller processes signals from the line detection, proximity, and user interface modules. Subsequently, it generates the appropriate control signals and sends them to the motor drive module, ensuring the correct speed is set for each motor drive.

3.1.2 Motor drive module:

The AGV is propelled by four (04) geared DC motors, which offer the flexibility to switch their turning direction, either clockwise or counterclockwise refer to Fig. 13. To measure the travel distance, two encoders are installed on the two rear motors of the AGV. These encoders utilize various technologies to generate a signal, including mechanical, magnetic, resistive, and optical – with optical being the most used method. In the optical sensing process, the encoder provides feedback by detecting interruptions in light. As the encoder shaft rotates, the Code Disk with opaque lines blocks allows light from the LED to pass through, and this modulation is subsequently detected by the Photodetector Assembly.

Figure 3.4. Geared DC Motor (left) & Disk encoder using optical technology(right).

The signal is transmitted to either the counter or controller, which then activates the intended function. Encoder modules are employed to determine the velocity of the AGV. For highly accurate velocity calculations, encoders with a higher number of pulses are preferred. In this project, a 334-pulse encoder with two channels (A and B) is utilized, operating with a voltage source of 5-12 VDC and drawing a current of 20mA.

The motors are controlled by a motor driver module, specifically the L298N module. The L298N Motor Driver is a controller that utilizes an H-Bridge configuration, allowing easy control over the direction and speed of up to 2 DC motors through the PWM (Pulse Width Modulation) method. [9]

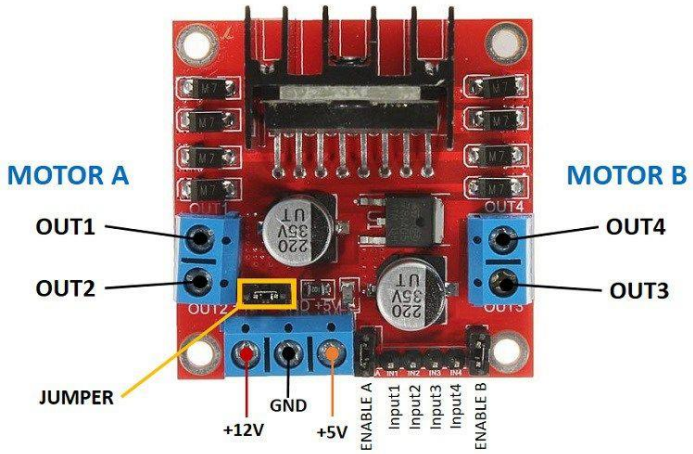

Figure 3.5. L298N Motor Driver Module.

3.1.3 Line detection module and Proximity/Obstacle sensor module

The line detection module serves the purpose of measuring the angular deviation between the AGV robot's path and its vertical axis, ensuring the robot stays on track. For this project, the TCRT5000 line sensor was selected, offering the following features:

- Reliable distance measurement capability ranging from 0.5 to 40mm.
- The sensor board integrates five (05) sensors, each with a dedicated pin.

To detect a wide variety of materials/objects, particularly in industrial settings, the project opted for an ultrasonic sensor, as opposed to others that have limitations based on material types, such as metal. The ultrasonic sensor boasts a long detection range and is suitable for detecting nearby obstacles like AGVs, humans, and packaging. These sensors emit high-frequency or infrared signals, which are then reflected to them, enabling the calculation of the distance to the object.

For this AGV, the HC-SR04 sonar sensor module is equipped. An algorithm is required to convert the feedback signal from this sensor into a distance measurement representing the proximity of the robot to obstacles. This distance data will assist the robot in making informed decisions regarding the appropriate warning level [10].

Figure 3.6. TCRT5000 Line Sensor module (left) & HC-SR04 Sonar Sensor Module (right)

3.1.4 Wireless Communications

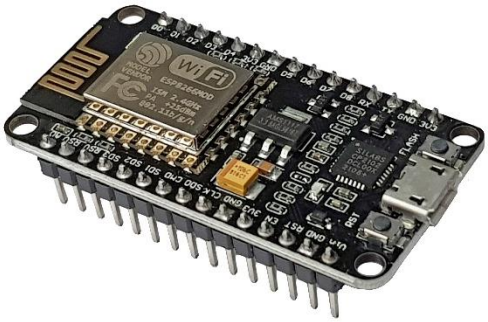

Figure 3.7. ESP8266 for Wi-fi communication

The system incorporates the ESP8266 Wi-Fi module to enable wireless communication functionality. The ESP8266 is an economic development board that comes with integrated TCP/IP networking software and microcontroller capabilities. This module is manufactured by ESPRESSIF Systems in China.

The primary advantage of this micro control unit lies in its wireless networking capability [11]. The Wi-Fi features of this unit encompass:

- Support for 802.11 b/g/n
- Support for 802.11 n (2.4 GHz) with speeds up to 72.2 Mbps.

The ESP8266 was selected due to its widespread adoption. It is well-suited for industrial environments, thanks to its highly integrated on-chip features, requiring minimal external components. Designed for mobile devices, wearable electronics, and IoT applications, the ESP8266 achieves low power consumption through its integrated Tensilica L106 32-bit RISC processor, resulting in energy-efficient performance for the system.

3.1.5 RFID Reader/Location Module

To read RFID tags, the STM32 interfaces with the RDM6300 RFID Reader Module [12]. Operating at a frequency of 125 kHz, the RDM6300 RFID Reader can only read data from 125 kHz compatible read-only tags. This module employs transistor logic, also known as TTL logic, and requires a 5-volt input to function. Communication with the microcontroller occurs through UART serial communication. The key features of the RDM6300 module are as follows:

- Frequency: 125 kHz.
- Reliable reading distance: 20-50mm.

Figure 3.8. RDM6300 RFID Reader Module

Refer to Figure 15 for the RDM6300 RFID Reader module's visual representation. It consists of a driver and coils that are connected to the driver. The module generates and modulates radio signals with a frequency of 125 kHz. When a suitable RFID card is placed near the generated frequency field, it receives energy and transfers power to the RFID module for processing.

3.1.6 Power Unit

Figure 3.9. Lithium-ion Battery 3.7V

The system is powered by a 12VDC source. AGVs can utilize diverse types of batteries, such as flooded lead acid, NiCad, lithium-ion, sealed, inductive power, and fuel cells. Among these options, lithium-ion batteries are favored due to their compact size and superior performance. Lithium-ion batteries offer faster charging and are environmentally friendly, but they do require maintenance.

For this project, the power supply consists of three (03) lithium-ion batteries with specifications as follows: 18650 types, with a voltage range of 3.7V to 4.2V, and a capacity of 1500mAh.

3.2 Firmware design

3.2.1 Tools and languages:

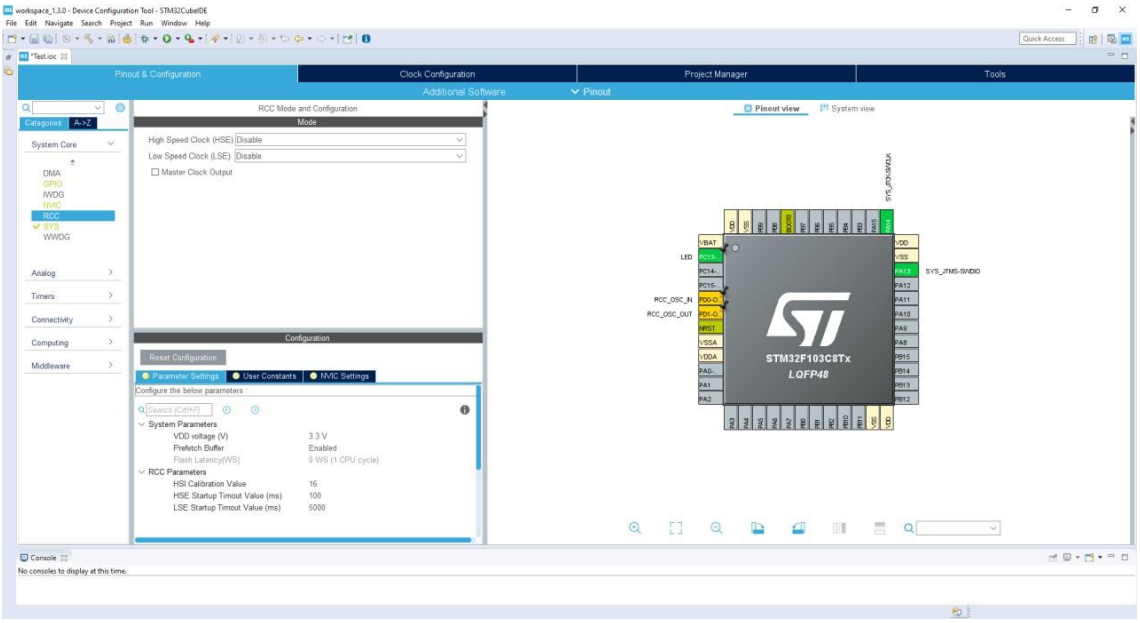

Figure 3.10. STM32 Cube IDE

In this project, with the designing of AGV with available modules the IDE we use to develop firmware for STM32 is STM32 Cube IDE. STM32CubeIDE is an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem. STM32CubeIde Board PhotoSTM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors.

STM32CubeIDE integrates STM32 configuration and project creation functionalities from STM32CubeMX to offer all-in-one tool experience and save installation and development time. After the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller or microprocessor from the selection of a board or the selection of an example, the project is created, and initialization code generated. At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code.

STM32CubeIDE includes build and stack analyzers that provide the user with useful information about project status and memory requirements. STM32CubeIDE also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyzer.

In this project, we use **bare-metal programming method** to deeply understand the construction of STM32 F334R8.

Figure 3.11. Arduino IDE

The Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. It also provides support for various kinds of boards depending on the different microcontrollers used. The IDE application is suitable for different operating systems such as Windows, Mac OS X, and Linux. And it consists of many libraries and a set of examples of mini projects. It supports the programming languages C and C++. Here, IDE stands for Integrated Development Environment.

The program or code written in the Arduino IDE is often called sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '. ino.'

In this project, we use **Arduino IDE** to config ESP8266 boards.

3.2.2 STM32 F334R8 Firmware Development

a) Block diagram

To meet the technical requirements, the car designed with available modules, we have the block diagram for the motion part. In this block diagram, first we generate a pulse at trigger pin of ultrasonic sensor. State_left and State_right is server sending the state's control. Check_Distance() is the function used to enable line sensor after responding to the control signal. Distance is calculated through the ultrasonic sensor, if this is less than 10cm or server control velocity equal to 0, we stop the AGV. If distance is okay and we have the control velocity from server, program move to calculate the PID and drive the motor.

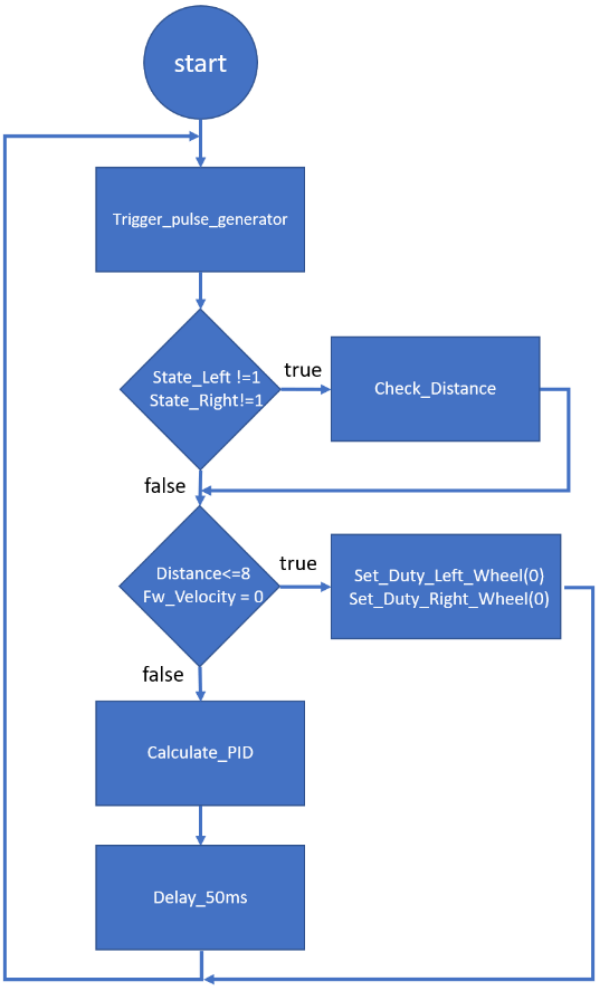

Figure 3.12. Flow diagram of the main function in STM32

b) Instruction set.

First, in the main function, all initialization functions, used to initialize timers, counters, interrupts, GPIO pins, UARTs, are called. Moreover, it has an iteration of distance tracking and reference tracking. Any error, which is raised by IR sensor, or distance warning will be managed in this function.

Hal_Init(), SystemClock_config(), MX_GPIO_Init() are the functions generated by STM32 CubeIDE to config parameters for the microcontroller. We use Config_Line_Sensor () function to config GPIO pin which gets the signal from infrared sensor. Config_timer_PWM() function used to config pins which generate PWM pulse to control the motor where we use output compare mode to generate PWM pulse. The next function is config_encoder() which we used for counting the pulses of encoder and this pulse will serve to calculate the velocity of AGV. Config_Timer3() is the function we set up using input capture mode to capture the pulse and calculate the Distance. We design Config_Timer15() to have time for velocity calculations.

By using a timer, after 100ms, the MCU will calculate velocity for each wheel once time. The equation:

$$velocity_{left} = \frac{count_{left}}{16.7} \quad 3.1$$

$$velocity_{right} = \frac{count_{right}}{16.7} \quad 3.2$$

For PID controller block, From the angle error and velocity error, the controller will calculate control signal to keep the error equals approximately zero. We need to choose k_p, k_i, k_d,coefficient to reduce the overshoot and the transient time.

To receive data from the ESP8266, we employ an interruption mechanism, obtaining the data through UART1. Through the UART1 interrupt handler function, we sequentially acquire the data bit by bit. Subsequently, we receive the data frame, which comprises control signals from the ESP8266. Before interpreting these signals, they are decoded. Additionally, the STM32 transmits an array of twenty-two bytes to the ESP8266 via UART1 transmitter, containing essential information about the AGV, such as car_id, car_state, car_battery, car_speed, previous_node, next_node, etc. This information is obtained from calculation functions within the STM32, or sensors integrated on the AGV.

The RDM6300 module is utilized for reading the RFID when the AGV passes. Upon scanning the RFID tag positioned along the designated path, the tag's ID is converted into a sequence of 14-character bytes and then communicated to the STM32 via the Tx pin of the RDM6300 through serial communication. To receive each RFID tag, the STM32 F334R8 utilizes UART3 with an interrupt handler function.

3.2.3 ESP8266 Firmware Development

The software development for the ESP8266 is done in the Arduino IDE (integrated development environment) SDK (software development kit) for programming the chip directly. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The development for the ESP (Electronic Stability Program) in this project is done in the C language. Many free software libraries are used to augment the project.

The ESP establishes wireless connection between the AGV and the server. The ESP is configured to operate in the Wi-Fi station mode. In Station Mode (STA), the ESP8266 Wi-Fi Module will be connected to a Wi-Fi Network that is already set up by an Access Point, like a Wi-Fi Router.

The code to connect the ESP to a Wi-Fi network is simple and straightforward. It makes use of the ESP8266Wifi library. The SSID and password of the network that needs to be connected to is provided (that is, to define the SSID and passwords as constant in the code).

The “WiFi.begin” function is then called. The ESP will complete the connection after some amount of time, which is affected by several factors such as distance from the access point (router, relay, modem, …), ambient environment, and any obstructions between the module and the access point. If the Wi-Fi connection is lost, then the ESP can automatically reconnect to the defined or strongest network, depending on which version is in use.

The ESP can transmit and receive data to and from the server to log vehicle data and get control instructions. If the wireless connection is successful, the ESP will try to connect to the defined MQTT Broker server, whose concept and design choice will be discussed in a later section. The ESP will attempt to reestablish MQTT connection if it is lost. Transmission of data is halted until an MQTT connection has been secured.

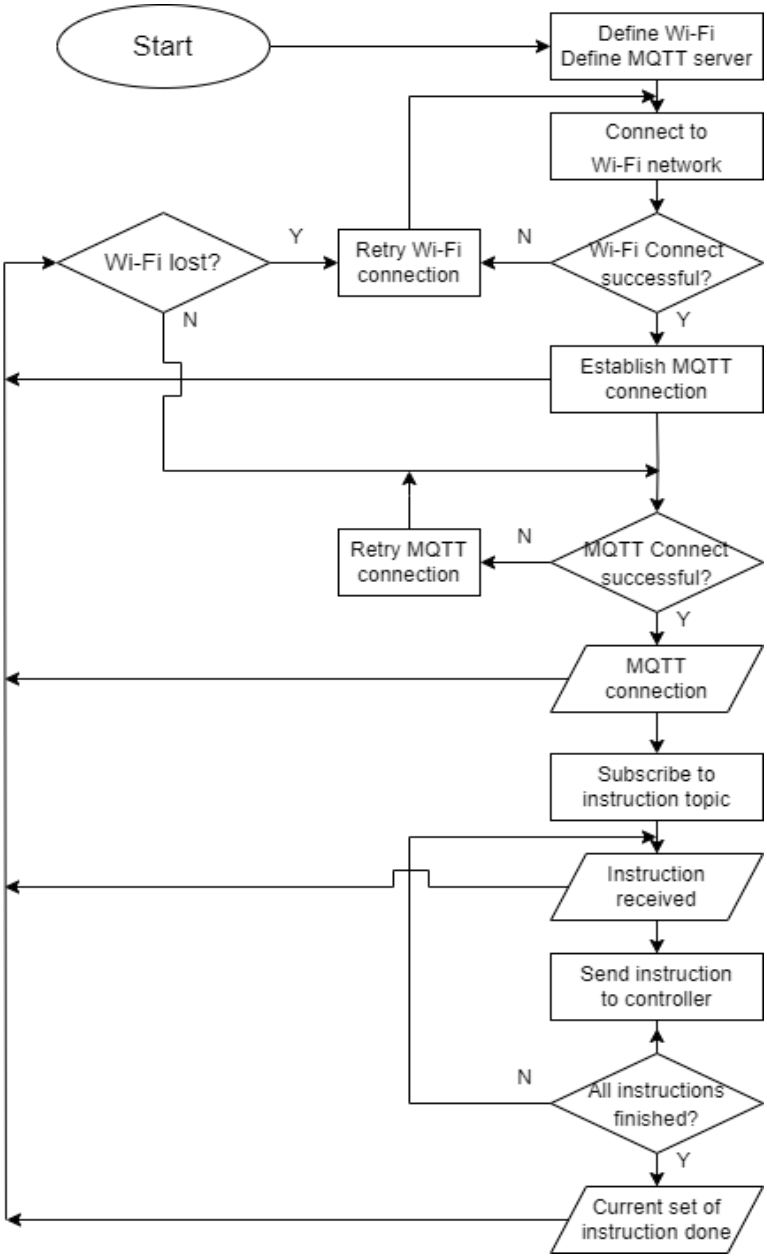

Figure 3.13. ESP8266 Wi-Fi & MQTT connection flow chart.

The ESP8266 receives twenty-two bytes as parameters of AGV (Automated Guided Vehicle) from the STM32 microcontroller through the "SoftwareSerial" pins defined in the Arduino code. When the first byte of the data frame, "0x7A", is received, a "while (1)" loop is initiated to continuously receive all transmitted bytes until the last byte of the frame, which is "0x7F". These 22 bytes are stored in an array of bytes and then sent to an MQTT server using the "client.publish" function provided by the <PubSubClient.h> library.

Additionally, the ESP8266 is responsible for transmitting control signals as a data frame to the STM32, as mentioned earlier. When the data frame is transferred to the MQTT client (ESP8266), a callback function is used to transmit these payload[i] bytes to the STM32 using the "Serial.write" function.

CHAPTER 4. SOFTWARE DESIGN

The system comprises individual AGVs, an MQTT Broker, a Web server, and Web clients. AGVs communicate with the control server using the MQTT protocol, sending real-time parameters and receiving commands. The web server contains the web application, responsible for the server's logical functionality, and a database for data storage. Through a web browser, users can monitor and control the operating process using a web client.

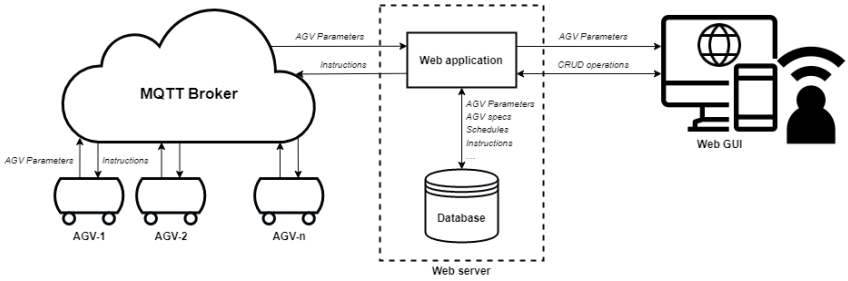

Figure 4.1. System physical structure.

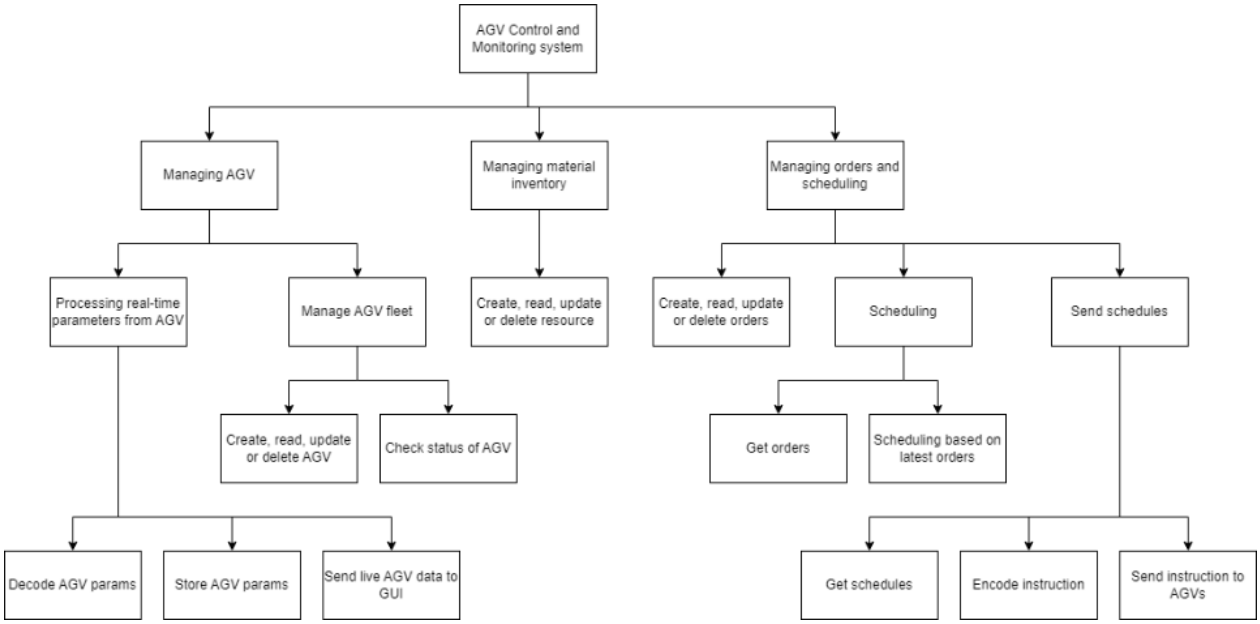

Figure 4.2. System functionality diagram

The web application has three main functions: managing AGV, managing orders and scheduling, and managing the material inventory. The operator can supervise the operating process and live-data update from AGVs, place orders, manage the AGV fleet, and manage the material inventory. Orders are processed by the scheduling function to generate the best possible task order and routes for each AGV.

4.1 Design of the server (back-end)

For the interface to function correctly, and to give functionality to the application, the “back-end” is developed. The server is built using the Python language and the Django web framework, along with the database backend PostgreSQL. Development was done in Microsoft Visual Studio Code.

4.1.1 Tools and Languages - Django web framework and PostgreSQL

Django, a Python-based web framework [13], follows the model-template-views (MTV) architectural pattern, like the model-view-controller (MVC) architecture [14]. The framework consists of an object-relational mapper (ORM) for data models and a relational database ("Model"), a system for processing HTTP requests with a web templating system ("View"), and a regular-expression-based URL dispatcher ("Controller"). It provides various modules for easy development and can be used with PostgreSQL, a powerful relational database management system compliant with SQL [15].

The Python language and Django were chosen for this project due to Python's suitability for mathematical applications, ease of use, and object-oriented nature. Django's multiple readily available packages make it ideal for developing complex projects without starting from nothing. While Django comes with SQLite, PostgreSQL was preferred for its ability to manage complex data and offer better support for various data types and security features.

The server/web-application is a Django project, which can contain multiple applications. Each application serves a specific purpose and interacts with various parts of the framework. Multiple applications were created in this project to efficiently manage and develop the complex system.

4.1.2 Control, monitor and manage AGVs.

This project aims to create a comprehensive system to control, manage, and monitor AGVs (Automated Guided Vehicles). The user/operator will have real-time access to manage and monitor the AGV fleet. Data from multiple AGVs will be processed by the server and displayed on the graphical user interface in real-time. Fleet modifications, such as adding or removing AGVs from the database, as well as activating or deactivating AGVs, can be easily made and will be immediately updated for the user.

The user/operator can register AGVs into the system and subsequently activate or deactivate them as needed. Upon registration, the server will send a special data packet to each registered AGV to check for availability and connectivity. Regular scans will be performed by the server to ensure continuous monitoring of AGV availability and connectivity. If any AGV experiences connectivity issues or errors, the server will promptly update the list of available AGVs that can be assigned to tasks.

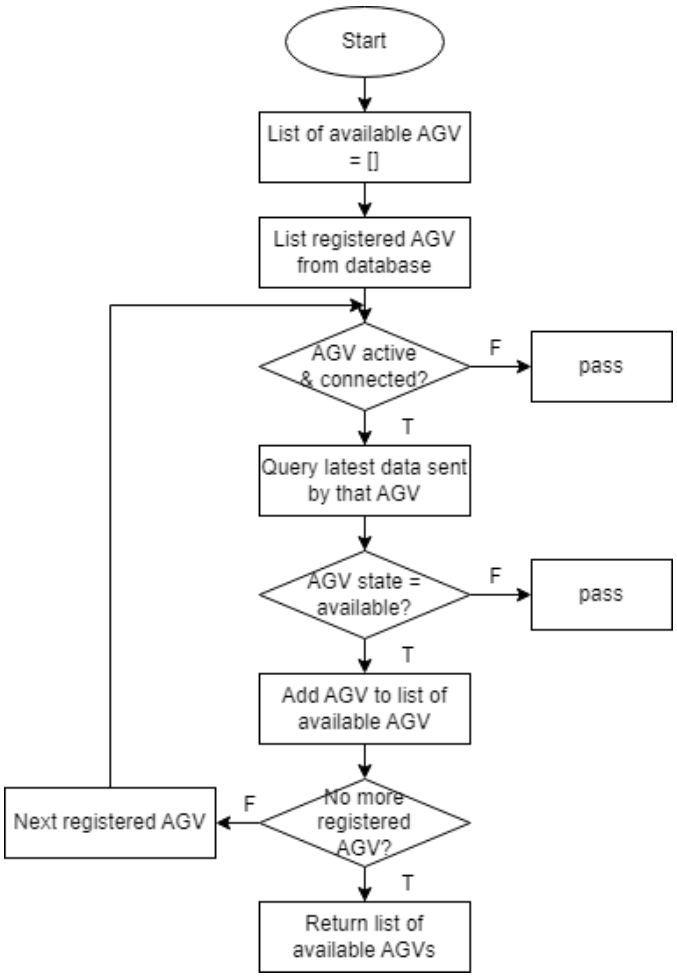

Figure 4.3. Function to get list of available AGVs.

Each AGV sends data packets using unique topics. The server then fetches and decodes this data, saving it into a database, and finally, relays it to the graphical user interface. The data is formatted using a custom data frame structure, which will be discussed further in a later section. This section provides an overview of the process for retrieving and decoding the data transmitted by the AGVs.

4.1.3 Place and update requests/orders

The user can efficiently manage the daily timetable through the graphical user interface, which also allows them to update or remove orders stored in the database. The server regularly queries the database for the latest orders and generates schedules for the AGV fleet. The scheduling process considers numerous factors, such as AGV availability, current states (location and battery capacity), to create an optimal schedule that minimizes energy consumption and lateness of operations while avoiding path conflicts.

The generated schedules, containing material details, assigned vehicles, estimated completion times, sink and source information, and route instructions, are stored in the database. After encoding them according to the specified data frame

structure, the server sends the instructions to the corresponding AGV, and the route instructions are stored on the vehicle for execution.

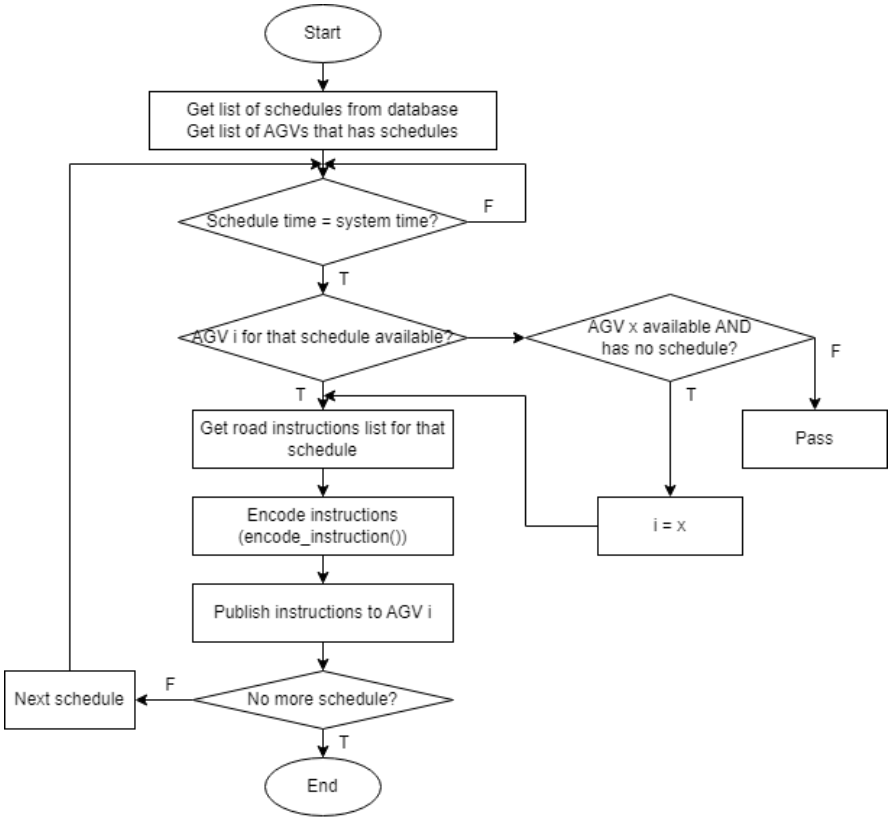

Figure 4.4. Function for sending instructions to AGVs.

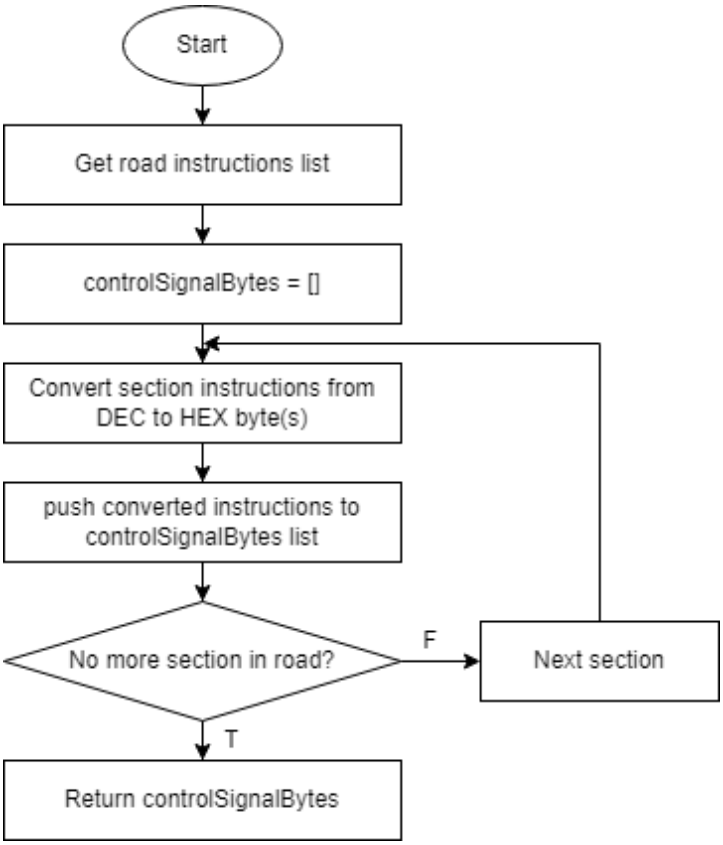

Figure 4.5. Function or encoding instructions.

4.1.4 Manage material inventory.

Through the graphical user interface, the user/operator can manage the material inventory. This includes adding new materials, updating details of existing materials, or removing materials from the inventory. Currently, only these fundamental operations have been implemented. This functionality is essential for placing orders, as users can select materials from the available inventory for transportation. To facilitate these material inventory management operations, the system utilizes the Django REST Framework and REST API, which will be further elaborated.

4.1.5 Structure of the database

The database plays a crucial role in storing various data, including information sent by AGVs, orders, schedules, metadata for each AGV, and material inventory management. PostgreSQL DBMS (Database Management System) is used to manage the database, while Django ORM serves as the intermediary between the web application and the database. ORM is a technique that connects Django's object-oriented programming with the relational database. Django comes with an embedded ORM designed for Python [16]. When using object-oriented programming languages to interact with databases, CRUD operations (Create, Read, Update, and Delete) are commonly performed. Traditionally, these operations are executed using SQL or raw SQL in relational databases. However, ORM simplifies these interactions. For instance, to select the last records from a table in the database, the SQL code would be:

*SELECT * FROM agv_data ORDER BY data_id DESC LIMIT 1;*

where in ORM, the code would be:

agv_data.objects.all(). last ()

Although ORM is perceived to be slower than using raw SQL, this is applicable when dealing with overly complex queries or executing many queries simultaneously (in the range of thousands). Utilizing ORM can accelerate development time since it requires less code compared to writing raw SQL queries. Also, ORM tools are designed to prevent SQL injection attacks, enhancing the database's security.

In relational databases, data in different tables can be interconnected or related by establishing relationships between them. There are three types of database relationships: one-to-one, one-to-many, and many-to-many. In a one-to-one (1:1) relationship, each record in table A is linked to one and only one record in table B, and vice versa. A one-to-many (1: N) relationship allows a record in table A to relate to zero, one, or multiple records in table B, while record(s) in table B can relate to one record in table A. A many-to-many relationship enables multiple records in table A to be associated with multiple records in table B, and vice versa. In the database schema of this project, only one-to-one and one-to-many relationships are utilized. A one-to-one relationship is represented by a direct line connection between a value from table A and table B, while a one-to-many

relationship is indicated by a branching line connection from a value in table A to a value in table B.

Below is the structure of the relational data and the relationships between tables in this project, along with a description of each table.

Figure 4.6. Database schema

4.2 Design of the Web interface (Front-end)

4.2.1 Tools and Languages

In the context of my project, I have created a Web-based interface by employing two primary programming languages: Cascading Style Sheet (CSS) and JavaScript, integrated with the React framework—a specialized JavaScript library dedicated to constructing Web-based interfaces. This combination of technologies allows for the development of a dynamic and interactive user interface that enhances the overall user experience [17].

JavaScript, being a programming language, enables me to implement various functionalities, such as defining actions, conditions, calculations, network requests, and concurrent tasks. JavaScript empowers developers to execute a range of instructions, making it an essential aspect of the programming process [18].

ReactJS, as a declarative, efficient, and flexible JavaScript library, plays a significant role in constructing reusable UI components. It is specifically responsible for managing the view layer of the application. A ReactJS application

is built upon multiple components, each of which is responsible for generating a small, reusable segment of HTML (Hyper Text Markup Language) code. These components serve as the fundamental building blocks of all React applications, and they can be combined with other components to create more complex applications. The use of a virtual DOM (Document Object Model) mechanism in React allows for efficient updating of individual DOM elements instead of reloading the entire DOM each time, resulting in enhanced performance.

Additionally, several tools have aided me in building the UI. The first tool is "Node package manager" or "npm," an online repository for open-source Node.js packages. With npm, I can easily incorporate code written by others without having to develop everything from scratch during the development process. It allows for the convenient installation of libraries and frameworks created by others.

The second tool is Material UI (MUI), a library of React UI components that implements Google's Material Design guidelines. MUI provides a comprehensive collection of prebuilt components ready for use in production applications. These components are extensively utilized in this project to enhance the UI design [19].

Lastly, Axios, a promise-based HTTP client for browsers and Node.js, is another valuable tool utilized in the project. It facilitates communication with APIs (Application Programming Interface) by providing a base URL and enabling the use of HTTP protocols for posting or retrieving data. [20]

4.2.2 Implementation & Development

To set up and run a React application, it is essential to have Node.js and “npm” (Node Package Manager) installed on the host computer. Node.js serves as the runtime environment for JavaScript, and “npm” facilitates the installation of packages and modules required for the application. Once Node.js and npm are installed, a new React application can be created using the `create-react-app` command in the terminal or command prompt. The syntax for creating a new app is as follows: `npx create-react-app <app-name>`, where `<app-name>` represents the desired name for the React application. [21]

After the creation of the app, navigating to the application's directory is achieved using the `cd` command in the terminal. In this project, I would run: “cd my-react-app/learning-react/client.” With the app directory as the working directory, the development server can be initiated, and the React app can be executed locally using the `npm start` command. The development server automatically opens the application in the default web browser, accessible through `http://localhost:3000`. In case the browser does not launch automatically, manual access is possible by entering the provided URL in the browser's address bar. The development server enables hot reloading, ensuring that any changes made to the application's code are immediately reflected in the browser upon saving the files. This feature significantly streamlines the development process, allowing for efficient real-time updates during the coding phase.

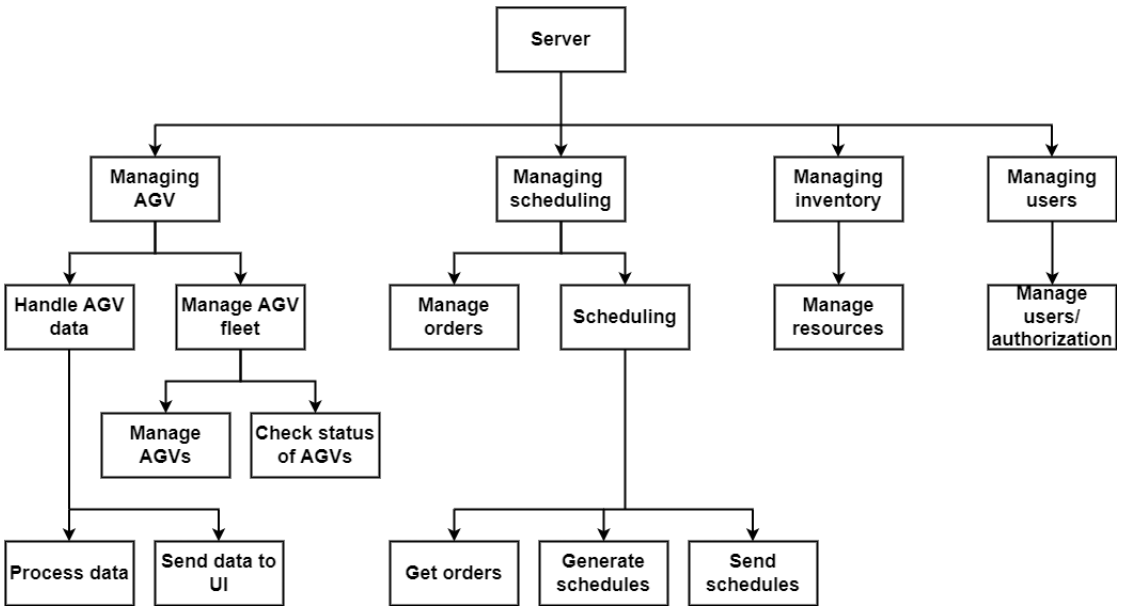

Figure 4.7. Web application functionality diagram

To commence the development of the web application, I initiated the process by creating a theme file named "theme.js." This file comprises a collection of colors, graded from light to dark, that are subsequently exported for utilization throughout different pages. By importing this theme file, each specific color is associated with a corresponding color code, thereby facilitating a consistent and cohesive visual style. Moreover, the use of the "useTheme()" function provided by MUI (Material-UI) ensures seamless implementation of the desired theme across the application.

Next, I designed essential global components, namely the Side bar and the Top bar, integrated into every page after user login. The Side bar is realized using the “ProsidebarProvider” component, an MUI-supported feature. On the other hand, the Top bar comprises a <Box> element featuring a search bar, along with three buttons, namely: switching the theme to dark mode (as facilitated using "useTheme()"), a user icon, and a settings icon. These global components serve to enhance the user interface and provide consistent navigation and functionality across the application.

To manage dynamic data within the application, I utilized two main tools: Websockets and REST API. Websockets enabled the reception of real-time data from the AGVs at regular intervals of one second. Simultaneously, REST API facilitated the interaction with the server by utilizing HTTP protocols such as GET, POST, and DELETE to fetch data from the provided API.

The web-application consists of four distinct pages: "Home," "AGVs Management," "Orders Management," and "Schedule Table." I employed CSS language "gridArea" to construct the page layouts, efficiently dividing each page into smaller areas, each corresponding to a specific key and fulfilling a particular purpose. These areas were created using "box" tags provided by MUI, effectively combining multiple boxes to form a complete web page. Each part of the website is structured as a component in React, and the website is a composition of these individual components. I employed functional components, allowing for

reusability and integration of the same component across multiple pages, thus enhancing code efficiency and maintainability.

In the subsequent sections, detailed descriptions of each page and its components will be presented to provide a comprehensive overview of the web application's functionality and layout.

a. The First Page – Home Page:

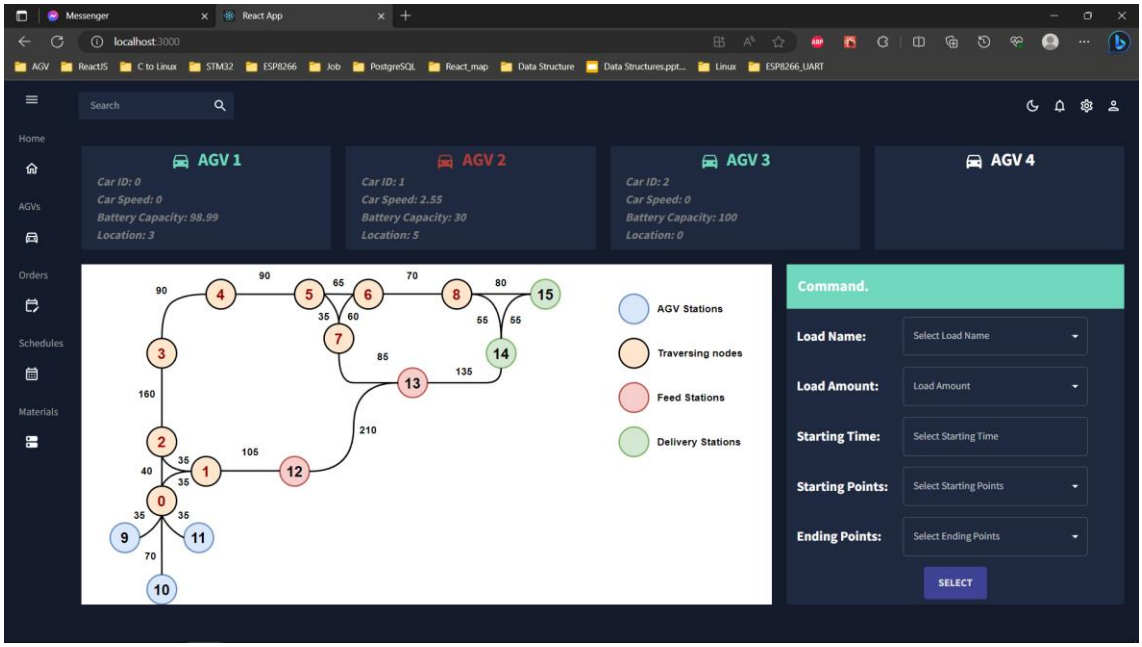

Figure 4.8. Home Page Web-app

The Home page of the web application features a dynamic and interactive layout that provides real-time data on the four most recently operating Automated Guided Vehicles (AGVs). The page prominently displays a Map layout, represented as an SVG file, highlighting the facility's layout, which, in this instance, consists of forty-eight nodes. Alongside the Map, a parameter box is presented, presenting essential information about the AGVs. This parameter box includes details such as the AGV's ID, current speed, battery capacity, and precise location. This live data keeps users informed about the status and activities of the AGVs.

In addition to real-time data visualization, the Home page offers a convenient option for users to swiftly place orders directly from the home screen. To facilitate this, a command box is implemented, constructed using a <Box> element that encompasses an input selection label and a Select button. The Select button is equipped with an onClick function that utilizes Axios.post to send the relevant order information to the API. This information includes details such as the material name, required amount, starting time of the order, as well as the inbound and outbound nodes. The integration of Axios.post streamlines the order placement process, ensuring seamless communication between the user interface and the server.

By encompassing these features, the Home page provides a comprehensive overview of the AGVs' activities and empowers users to initiate order placement quickly and efficiently. The use of dynamic data visualization and streamlined

order placement functionality enhances the user experience and contributes to the overall effectiveness and user-friendliness of the web application.

b. The second page – Manage AGVs Page:

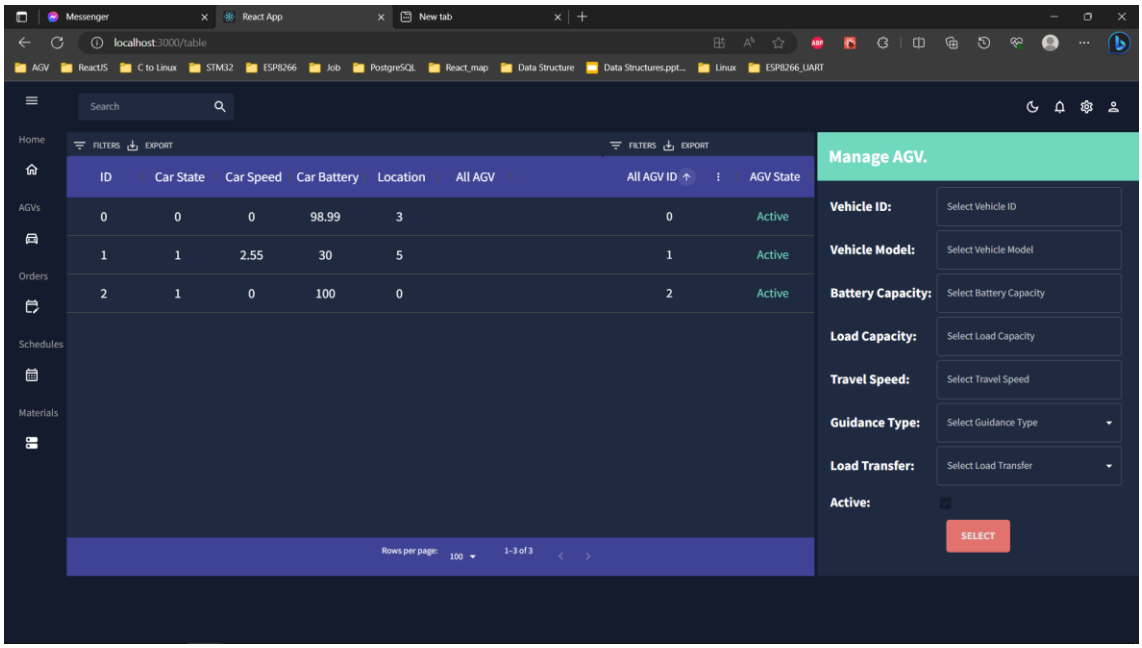

Figure 4.9. Manage AGV Page Web-app.

The Manage AGV page offers users comprehensive control and monitoring capabilities for the AGV fleet. It presents live parameters for all AGVs, enabling real-time tracking of crucial information. The parameters transmitted by AGVs encompass essential details such as the vehicle ID, current state, current speed, battery capacity, and location. To facilitate fleet management, a Command box like the one on the Home page is provided. This Command box is constructed using multiple select input labels and a Select button, leveraging HTTP request methods such as POST to interact with specific APIs.

The Command box empowers the operator to add or remove AGVs from the fleet effortlessly. Adding a new AGV to the fleet involves filling out relevant fields on the right-hand side of the Command box. The fields encompass crucial information, including the numerical ID assigned to the vehicle, essential metadata such as the vehicle model, maximum battery capacity, maximum load capacity, travel speed, guidance type (optical, magnetic, LiDAR (Light Detection and Ranging)), and load transfer type (automatic or manual). Additionally, the operator can activate the vehicle by checking the "active" box, initiating the AGV's participation in fleet operations.

By integrating live parameter monitoring with efficient fleet management features, the Manage AGV page provides users with a powerful and versatile toolset for overseeing and controlling the AGV fleet effectively. The Command box simplifies the process of adding and removing AGVs, streamlining fleet operations and enhancing the overall manageability of the AGV system.

c. The Third Page – Mange Order page:

The Manage s page offers a comprehensive table displaying the orders created for AGVs to execute. The table includes essential information such as Order Number, Order Date, From Node, To Node, Load Name, Load Weight, and Start Time. A user-friendly Command box is provided to facilitate order placement, where the operator must input five key pieces of information: material name, amount, starting time of the order, inbound node, and outbound node.

Additionally, the web application provides a convenient feature that allows the operator to create orders for AGVs by importing CSV files. By selecting the appropriate CSV file from their device, with the correct field structure, the operator can click on the "Send File CSV" button. Using the “Papa.parse” library, the CSV file is efficiently converted to JSON format, which is then sent to the API. Subsequently, the order is successfully created, streamlining the process of generating orders and enhancing overall efficiency.

Furthermore, a feature is implemented to enable the deletion of orders. Each row in the table is equipped with a checkbox, enabling the operator to select specific orders for deletion. By sending a DELETE request to the API provided by the server, the chosen order is promptly removed from the system.

The Manage Orders page offers material inventory management functionalities, enabling the operator to effortlessly add or remove materials as required. The implementation of the Generate button at the bottom of the page serves to demonstrate the ability to generate schedules for AGVs. When the operator clicks on this button, a schedule encompassing all the listed orders is promptly created. Subsequently, the server sends each schedule in the data frame form, allowing the AGVs to receive and execute the orders accordingly.

Upon the successful completion of any action, be it creating an order, deleting an order, or generating a schedule, a notification line is displayed above the Generate Schedule button, promptly informing the operator of the successful action.

The efficient and user-friendly design of the Manage Orders page ensures seamless management of AGV orders and material inventory, streamlining operations and optimizing productivity. The integration of CSV file import, order deletion, and schedule generation functionalities significantly enhances the functionality of the web application, providing a robust and effective platform for AGV fleet management.

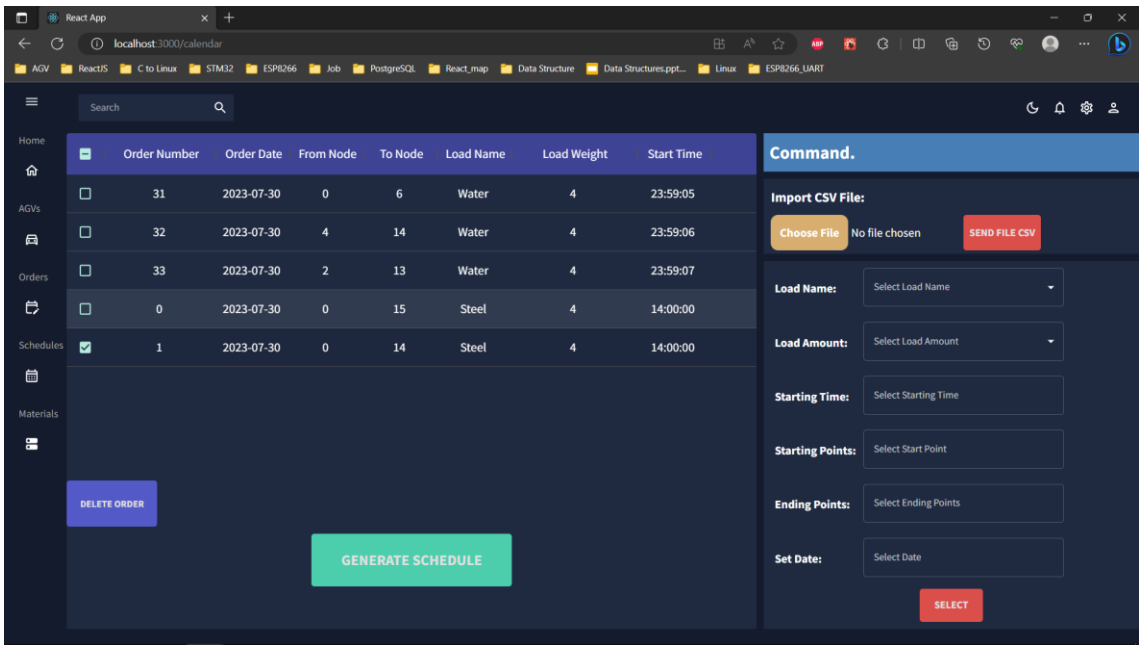

Figure 4.10. Manage Orders Page Web-app.

d. The fourth Page - Manage Schedule Page:

The present web page displays a well-structured table comprising a comprehensive collection of schedules meticulously formulated for the Automated Guided Vehicles (AGVs) to execute their tasks proficiently. These schedules are intricately tied to specific date arrangements, load designations, load weights, car identifications (IDs), and corresponding start times. Notably, these schedules are derived from the Orders created during the preceding stage. The table's immaculate organization enables a holistic understanding of all scheduled tasks, contributing to seamless monitoring and effective management of AGV operations. This valuable resource empowers stakeholders to optimally allocate resources, promptly respond to changes, and enhance overall operational efficiency.

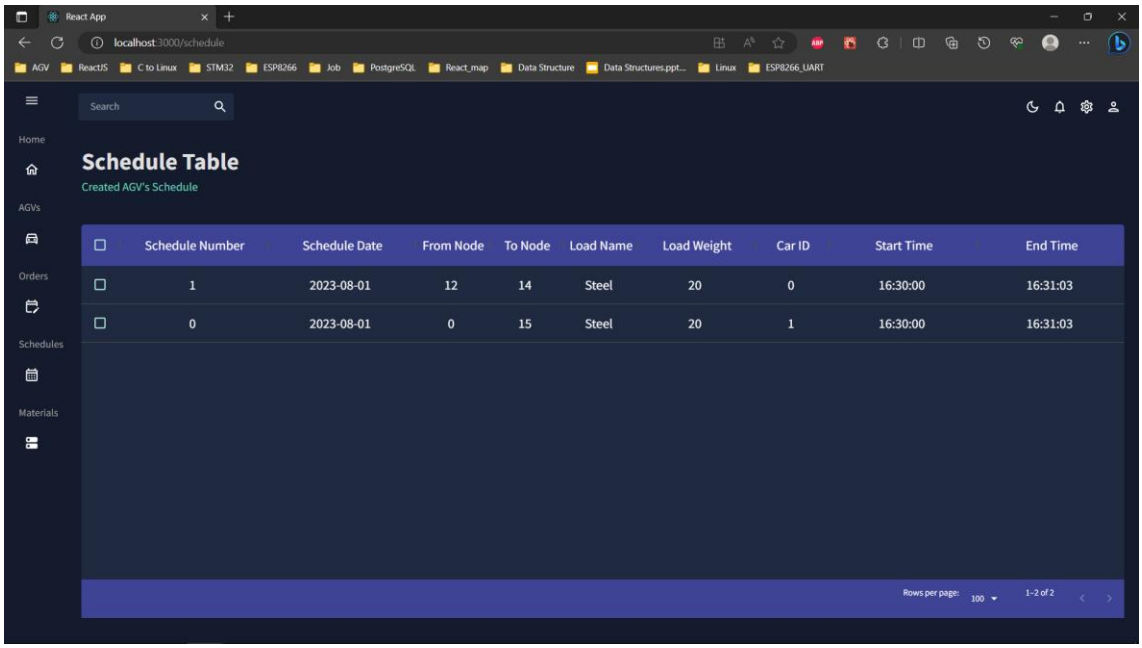

Figure 4.11. Schedule Table Page Web-app

e. Sign In page:

The Sign-In Page of our website offers a secure and user-friendly gateway to access our platform's exclusive features and services. With a sleek and intuitive design, the page presents a clean layout, welcoming users with a well-crafted logo and a concise tagline that embodies our brand's essence. The login form is prominently placed at the center of the page, providing input fields for users to enter their unique credentials, such as email or username and password. Beneath the login form, clear and concise instructions guide users through the process, ensuring a smooth and effortless login experience. To enhance security, we employ robust encryption protocols, safeguarding sensitive information from unauthorized access. Additionally, users can find options for password recovery and account registration if they are new to our platform. With a seamless and secure sign-in process, we prioritize user satisfaction and data protection, fostering a sense of trust and confidence among our valued users.

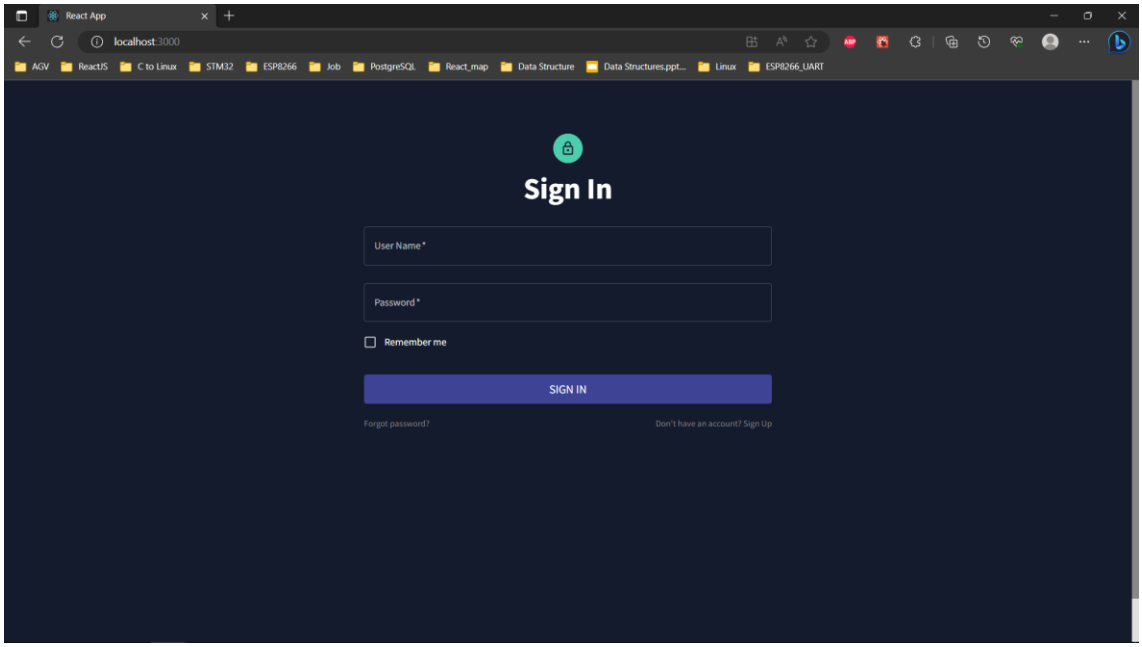

Figure 4.12. Sign In Page Web-app

f. Sign Up page:

The Sign-Up Page on our website is thoughtfully designed to welcome inexperienced users and facilitate their seamless registration process. Upon arriving at the page, users are greeted with an inviting interface featuring our brand's logo and a brief description of the benefits of joining our platform. The registration form takes center stage, with user-friendly input fields where prospective members can provide their essential details, such as name, email address, and preferred password. A well-structured layout guides users through the process, offering clear instructions and error validation to ensure accurate information submission. To streamline the experience, we have incorporated responsive design elements that adjust to different devices, enabling a smooth registration process on both desktop and mobile devices. As part of our commitment to data security, we implement robust encryption measures to

safeguard sensitive user information. Furthermore, to enhance the registration journey, users may receive a verification email to complete the process, adding an extra layer of protection to their accounts. Our “Sign Up” Page embodies a user-centric approach, fostering a positive first impression and encouraging new members to join our growing community with ease and confidence.

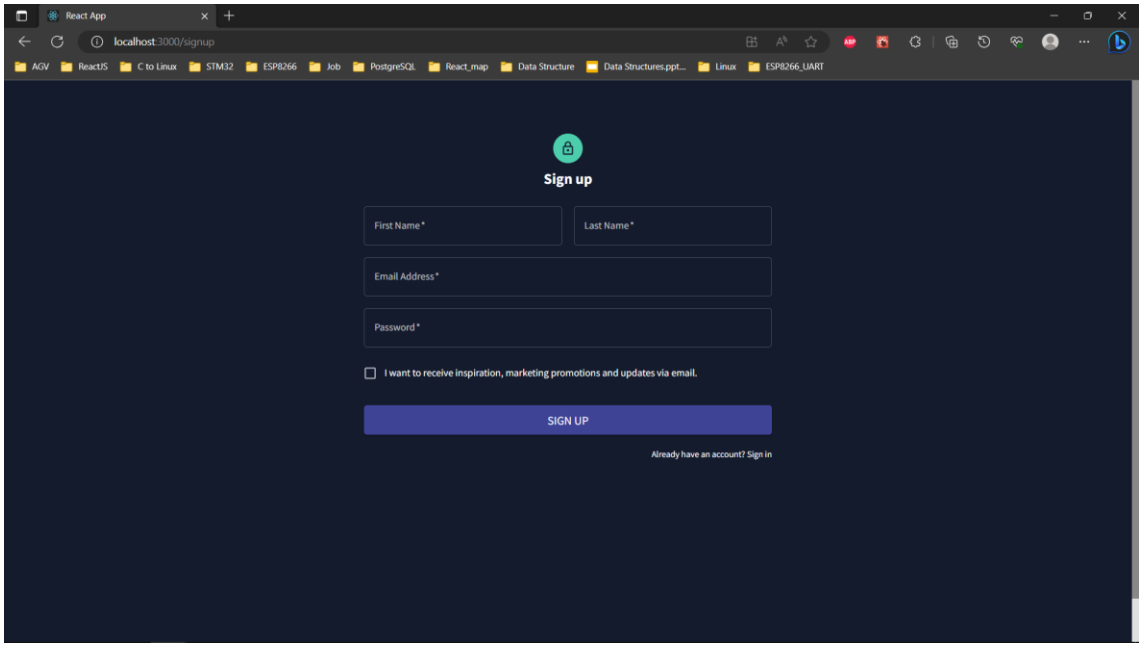

Figure 4.13. Sign Up Page Web-app.

4.3 Design of communications for the system

4.3.1 Communication between the server and vehicles by MQTT

This Internet of Things (IoT) project necessitates bidirectional communication over the network, which cannot be achieved using protocols like HTTP. Hence, the MQTT (MQ Telemetry Transport) protocol was chosen. MQTT facilitates two-way message exchange between clients and servers, while HTTP servers only respond to client requests. MQTT is a lightweight open messaging protocol specifically designed for resource-constrained network clients, making it ideal for distributing telemetry information in low-bandwidth environments and maintaining effective communication even in high latency and unreliable network condition [22].

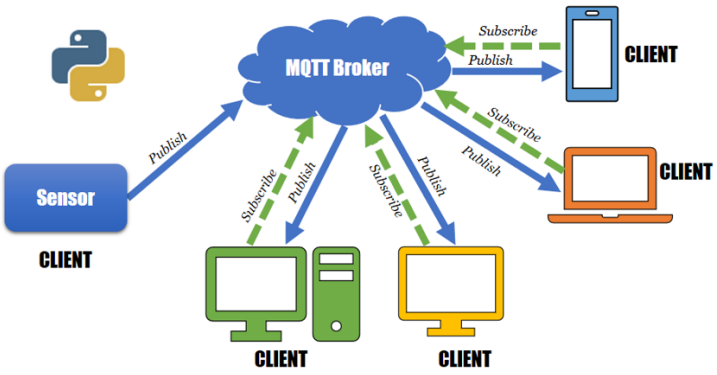

Figure 4.14. Illustration of MQTT protocol

The MQTT protocol operates on a publish-and-subscribe model, where client devices and applications publish and subscribe to topics managed by a broker. In this context, an MQTT broker acts as an intermediary, receiving messages published by clients, filtering them by topic, and then distributing them to subscribers. For this project, the Eclipse Mosquitto MQTT Broker was selected [23]. Mosquitto is an open-source message broker implementing MQTT version 3.1.1 and offers support for multiple ports access. It is lightweight and suitable for a broad range of devices, from low-power single-board computers to full servers. The broker is configured for the project by editing its configuration file to open ports and enable necessary protocols. Additionally, anonymous connections are allowed by setting the "allow_anonymous" property to true.

To enable MQTT capabilities on the server, the Eclipse Paho MQTT Python Client library is utilized. The library is imported as a module in the project for reusability. The server operates as an MQTT client, creating a client instance from the client class and connecting to the broker using the "connect ()" method. Messages are published to the broker using the "publish ()" method, while subscription to topics and message reception are achieved through the "subscribe ()" method. Python functions are used to wrap these methods and specify other actions whenever specific events occur. For instance, when the server successfully connects to the MQTT Broker, the "on_message()" function is called, and it publishes the connection acknowledgment message "CONACK" to the Broker. Using MQTT and these implementation strategies, the IoT project achieves efficient bidirectional communication and reliable telemetry data distribution.

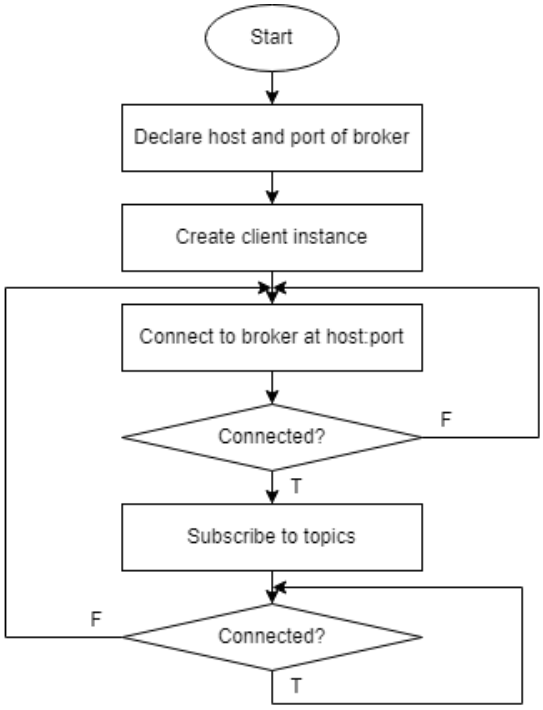

Figure 4.15. Connection from server to the MQTT broker

4.3.2 Live data update with WebSockets protocol and Channels

Several approaches exist for addressing the issue of real-time data updates on the web. One such method employs the HTTP protocol, utilizing a request-response mechanism commonly known as "long polling," where clients continuously send requests to the server, which responds with updated data. However, this approach falls short of true real-time updates and proves resource-intensive, consuming significant processing resources on both the client and server sides.

In contrast, the WebSockets protocol offers a more efficient solution by providing full-duplex communication channels over a single TCP connection [24]. For this project, WebSockets were selected as they excel in real-time applications, delivering low-latency, instantaneous message transmission. To integrate WebSockets into the Django server, Django Channels is employed, a framework built on the asynchronous ASGI specification for Python, while retaining Django's synchronous capabilities. Asynchronous architecture, which enables tasks to be executed independently and concurrently, proves advantageous for serving multiple clients simultaneously, making it a suitable choice for scalability.

Within Django Channels, developers can create "consumers," which are sets of functions triggered whenever an event occurs, like Django "views." These consumers facilitate easy development of ASGI applications, allowing the implementation of synchronous or asynchronous code while managing handoffs and threading automatically. To manage WebSocket connections in the project, a "consumers.py" file is created, utilizing the "AsyncWebsocketConsumer" class to ensure coding and functions operate asynchronously, enabling seamless handling of WebSocket interactions. B leveraging the power of WebSockets and Django Channels, the project achievesefficient and responsive real-time data updates while preserving the

server's scalability and resource utilization. The latest AGV parameters are queried from the database after they are stored, based on the list of active AGVs: if the AGV is not/no longer active, then the data for that AGV will not be queried for. This ensures that the data sent to the graphical user interface is the latest, without unnecessarily overcrowding the data packet sent to the web.

Python file “routing.py” is created to manage consumers. The routing class in Channels allows the combination of consumers to be dispatched based on what the connection is.

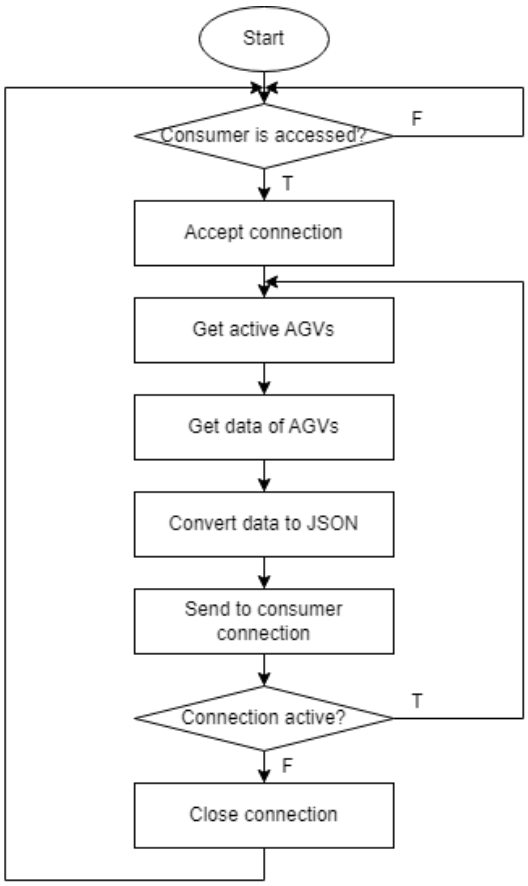

Figure 4.16. Function of WebSocket consumer

4.3.3 Interaction between server and web interface using Django REST framework.

Through the web-based graphical user interface, the operator gains the ability to perform various tasks, including AGV fleet management, material inventory management, and schedule management. These tasks involve fundamental operations such as creating, reading, updating, and deleting records in the database. To facilitate these functionalities, the Django REST framework (DRF) is employed, renowned for its robustness and flexibility in constructing Web APIs [25].

An API (Application Programming Interface) serves as a software interface that enables communication between two or more computer programs. Unlike a user interface that connects a computer to a person, an API connects software components or systems to each other and is not intended for direct use by end users. Web APIs function as services accessed from client devices like mobile phones, laptops, and tablets, connecting them to a web server using the HTTP protocol. Client devices send HTTP requests to the server, and in return, they receive appropriate responses. Web APIs is utilized to query specific data sets from the server. A REST API, or RESTful API, adheres to the principles of the REST architectural style, permitting interactions with RESTful web services. REST,

which stands for Representational State Transfer, employs URLs for making HTTP requests.

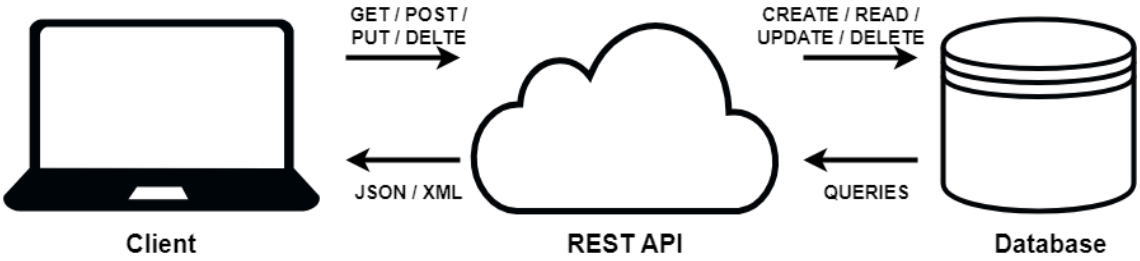

Figure 4.17. RESTful structure diagram

The REST API operates on the foundation of the HTTP protocol and is defined by a set of architectural constraints rather than being a specific protocol or standard. It utilizes HTTP methods such as GET, POST, DELETE, and PUT to communicate with a URL and effect changes in the database. Each method corresponds to a specific HTTP request: GET retrieves a resource or a list of resources, POST creates a new resource, PUT updates an existing resource, and DELETE removes a resource from the database. These methods align with the four fundamental CRUD (Create, Read, Update, Delete) operations.

To construct APIs for the web server and enable interactions between the graphical user interface and the server for resource management, the Django REST framework (DRF) is integrated. APIs necessitate a serializer to facilitate communication with the model and a class to manage the serializer. By setting up a URL, access to the serializer is provided, thereby transforming it into an API. The serializer grants access to the corresponding database, presenting mutable records and fields that can be modified.

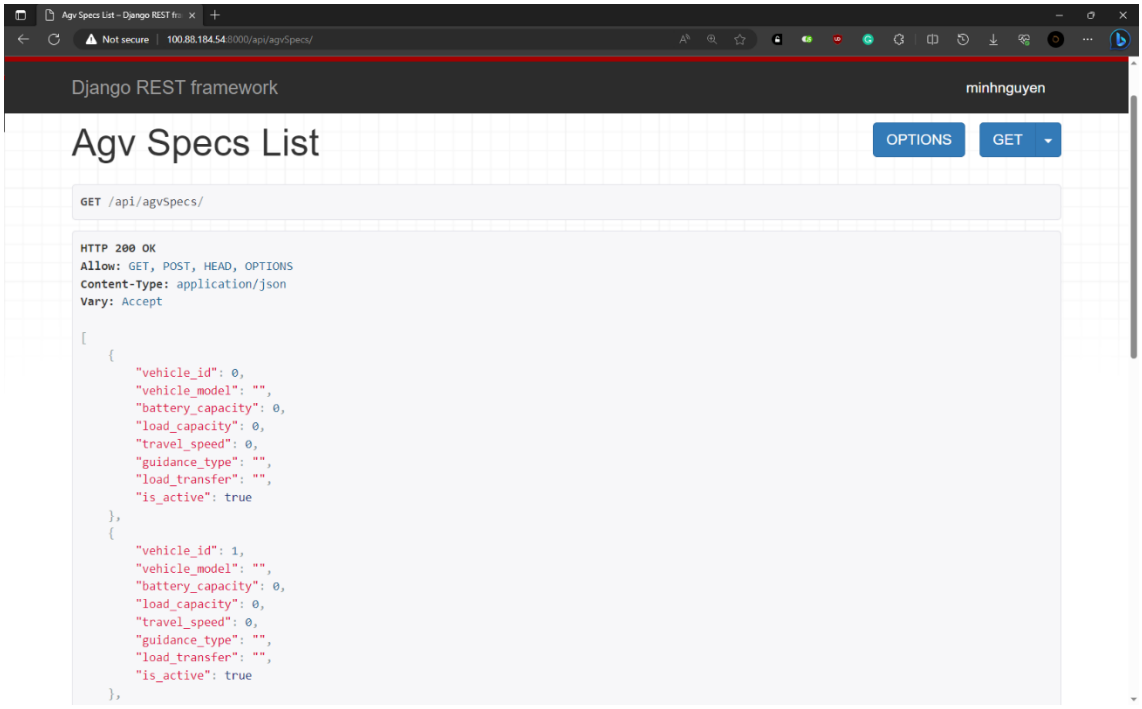

Figure 4.18. DRF API for managing AGV fleet.

4.4 Design of data frame structure

a. General data frame structure

To ensure fast, secure, and stable communication between the server and vehicles, it is essential to have a lightweight data frame structure capable of holding a significant amount of information. The MQTT protocol allows for the transfer of various data formats and encodings. The focus lies in designing a data structure that is both lightweight and informative, minimizing the risk of corruption. In this project, a custom-designed data frame structure in hexadecimal format is implemented. The data is encoded on the transmitting side and decoded on the receiving side. For instance, AGVs can transmit real-time monitoring parameters by encoding the required data into hexadecimal format and sending the packet to the MQTT broker. The server can then subscribe and listen to the AGV parameters' topic, fetch, decode, and store the data in a human-readable format.

The hexadecimal numeral system represents numbers using a base of sixteen, employing "0" to "9" to represent values 0 to 9 and "A" to "F" (or "a" to "f") to represent values from 10 to 15. Each hexadecimal digit signifies four bits, known as a nibble. In this system, an 8-bit byte can have values ranging from 00000000 to 11111111 in binary form, equivalent to 00 to FF in hexadecimal or 0 to 255 in decimal base. When dealing with data types like big integers, which require two bytes to represent a decimal number from 0 to 65535, byte order or endianness must be considered. Endianness refers to the order or sequence of bytes in computer memory. This matters in network and communication, as different machines may have different endianness. In this project, little-endian byte ordering is chosen as the AGV controller and the control server have different endianness, and little endian is preferred. The general data frame adheres to the below structure:

<table>
<tr><td>Frame start</td><td>Frame length</td><td colspan="3">Message type</td><td>Data payload</td><td>Frame end</td></tr>
<tr><td>1 byte</td><td>1 byte</td><td colspan="3">1 byte</td><td>N bytes</td><td>1 byte</td></tr>
<tr><td rowspan="4">0x7A</td><td rowspan="4">N+4 bytes</td><td>0x00</td><td>AGV Hello</td><td>Server sends to all AGV when system starts up</td><td rowspan="4">…</td><td rowspan="4">0x7F</td></tr>
<tr><td>0x01</td><td>AGV send params</td><td>AGVs send parameters to server</td></tr>
<tr><td>0x02</td><td>AGV responses</td><td>AGV response to server when receiving message</td></tr>
<tr><td>0x03</td><td>AGV route</td><td>Server sends to AGV the instructions of the route</td></tr>
</table>

Figure 4.19. General data frame structure

By employing Wi-Fi and an MQTT broker for wireless data transfer, we eliminate the need for certain fields like the sender and receiver addresses, which are necessary in LORA communication. The MQTT broker efficiently manages message routing through a publish/subscribe structure. Each data payload is tailored to its message type, and its purpose is outlined in the figure. The

subsequent section provides a comprehensive explanation and structure for the primary message type devised in this project.

b) Detailed data frame structure

Message type 1: AGV real-time parameters.

Car ID	Car state	Current battery cap	Current speed	Current position			Total energy for route	Total length of route
2 bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
0-65536	0-255 (states and corresponding id are assigned by operator)	0-10000 (divide by 100 to get %)	0-255 (cm/s)	previous node	next node	distance from previous node (cm)	Energy 0-65536 (J)	Length 0-65536 (cm)
Int	Short Int	Int	Short Int	Int	Int	Int	Short int	Int

Figure 4.20. Message type 01 data frame structure

The data packet of Message type 1 consists of real-time parameters of the AGV, including the battery level, speed, and distance values. These values are typically represented as floating-point numbers, which require up to four bytes for storage, leading to a significant increase in data transmission and processing time. To address this issue and optimize the data packet's efficiency, the floating-point values are rounded to two decimal places and then multiplied by one hundred to obtain integer values. By using a 2-byte unsigned integer, which can store values from 0 to 65535, the integer representation is compact and consumes less storage space. During the decoding process, the integer value is divided by one hundred to accurately retrieve the original floating-point value. This approach ensures a lightweight data packet without compromising the essential real-time parameters of the AGV.To ensure the data packet remains lightweight, the battery level, speed, and distance values, which are supposed to be float values, are rounded to two decimal places. Subsequently, they are multiplied by one hundred to obtain integer values, which require significantly less storage and transmission time. An integer value only needs one byte to store numbers from 0 to 255, or two bytes for a two-byte unsigned integer capable of storing numbers from 0 to 65535. During the decoding process, the integer value is divided by one hundred to retrieve the actual float value accurately. This optimization enables efficient data transmission and processing while preserving the crucial real-time information about the AGV's performance.

Frame start	Frame length	Message type	Car ID	Car state	Current battery cap	Current speed	Current position			Total energy for route	Total length of route	Frame end
1 byte	1 byte	1 byte	2 bytes	1 byte	2 bytes	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	1 byte
0x7A	0x10	0x01	00 0A	01	23 28	FF	00 05	00 08	04 2C	0D B5	30 D4	0x7F
	16 bytes	Type 01	10		9000	255 cm/s	Node #5	Node #8	1068 cm	3509 J	12500 cm	

Figure 4.21. Example of message type 01 data frame structure

Message type 3: Instructions from the server to the corresponding AGV:

Section 0				Next sections
Node	Speed	Section length	Action	…
2 bytes	1 byte	2 bytes	1 byte	6 x n-1 bytes
Previous node in current section	0-255 (cm/s)	Section length (cm) 0-65536 (cm)	(actions and corresponding id are assigned by operator)	…
Int	Int	int	Int	…

Figure 4.22. Message type 03 data frame structure

Message type 2 contains route instructions generated by the scheduling function and transmitted to the AGV. The complete route is segmented into sections, with each section defined as the path connecting two nodes on the map and bounded by the two nodes at its ends. The scheduling function determines the speed and action to be executed by the AGV for each section in the route. An illustrative example is presented below:

Start	Length	Message Type	Node Id	Speed	Section Length	Action	Node Id	Speed	Section Length
0x7A	0x14	0x03	01 00	3C	C8 00	01	2D 0D	3C	C8 00
	20 bytes	3	1	60 cm/s	200 cm	1	2	60 cm/s	200 cm

Action	Node Id	Speed	Section Length	Action	Node Id	Speed	Section Length	Action	End
1D	03 00	3C	C8 00	1D	04 00	00	00	0D	0x7F
1	3	60 cm/s	200 cm	1	4	0 cm/s	0 cm	0	

Figure 4.23. Example of message type 03 data frame structure

Message type 0: AGV Hello

Command	
1 byte	
0x00	Server automatically sends to every vehicle on system start up
Int	

Figure 4.24.Message type 0 data frame structure

Message type 2: AGV response

Car ID	Command	
2 bytes	1 byte	
0-65536	0x00	AGV sends response message when receiving message from server
Int	Int	

Figure 4.25. Message type 2 data frame structure

Message type 0 *AGV Hello* is used to check initial connectivity of AGVs to the server. Message type 2 *AGV response* sent by an AGV when they get a message is used to verify that the vehicle received the message; else the AGV will request the server to resend the message.

Message type 6: AGV Error

Car ID	Error code		First node	Second node
2 bytes	1 bytes		2 bytes	2 bytes
0-65536	0x04	AGV low battery	0-65536	0-65536
	0x06	AGV collision		
	0x09	AGV hardware error		

Figure 4.26. Message type 6 data frame structure

AGV sends error message when it encounters error, such as critically low battery, when AGV encounter collision, or when AGV has hardware error.

CHAPTER 5. PATH PLANNING AND SCHEDULING STRATEGY

The goal of the path planning and task scheduling strategy is to achieve the best possible solution for the problem, optimizing both time and energy costs. Previous literature reviews highlight the significance of considering energy consumption (EC) in material handling system path planning [6]. This consideration not only saves energy but also reduces disruptions to the system and the overall manufacturing process, thus significantly improving energy efficiency [26]. The main objective is to minimize the cost function for the system by employing scheduling and path planning techniques. Additionally, subproblems include collision-free path planning, adhering to battery constraints, and prioritizing load weight considerations.

5.1 Master problem

The problem of transportation orders management and scheduling in the company's internal transportation system can be formulated as a head problem in energy and time efficiency. The primary optimization objective is as follows:

$$min(\sum_{k=1}^{n} E_{AGV_k} * T_{task_k}) \qquad (5.1)$$

where E_{AGV_k} is the energy consumption of an AGV assigned to task k, T_{task_k} is the task duration (amount of time an AGV needs to complete task) of task k, and n is the total number of tasks in the schedule of one day. The main objective here is to reduce the whole energy consumption and task duration in the operation.

5.2 Mathematical formulation

5.2.1 Map standardization

Consider a simple map where each node is connected to at least one other node. The topology of the map can be defined as an undirected (path have no direction constraint) graph $G = (V, A)$ where $V = \{1, 2 \dots, n\}$ is the node set and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the edge set. Each edge (i, j) is associated with the distance d_{ij} between node i and node j. an ordered node sets S including the source node and the destination node can be utilized to represent a desired path.

5.2.2 Differential drive

Differential drive is a remarkably simple drive mechanism and is widely used in practice, especially in wheeled mobile robots. Robots with this drive usually have one or more casters to support the vehicle and prevent imbalance. Both main wheels are placed on a common axis. The velocity of each wheel is controlled separately by a DC motor.

Due to (1), the tangential velocity and angular velocity of AGV are calculated as

$$\begin{cases} v_{agv} = \dfrac{v_r + v_l}{2} \\ \omega_{agv} = \dfrac{v_r - v_l}{l_{AGV}} \end{cases} \quad (5.2)$$

where l_{AGV} and W represent the AGV length and width, respectively. R is the turning radius. v_r and v_l is right wheel and left wheel velocity, respectively.

5.2.3 Energy consumption model

The movement of AGV needs to consume energy, so with the aim of calculating the energy consumption for four kinds of motion, namely accelerated motion, decelerated motion, and uniform motion, which can be occurred during for the AGV process, energy consumption formulation needs to be considered.

The AGV has two separate motors. From [27], the electrical energy consumption of an AGV is given as:

$$E_{AGV} = 2 \cdot P_e \cdot t_{run} \quad (5.3)$$

where P_e and T_{run} is electrical power and travelling time of an AGV. When considering the electrical power P_e of the motor, the necessary mechanical power P_m must be considered on the one hand, and the efficiency of the motor η_{Motor} on the other. The electrical power of the motor is formulated as follows:

$$P_e = P_m \frac{1}{\eta_{motor}} \quad (5.4)$$

The mechanical power P_m can be rewritten in combination as follows:

$$P_m = P_{m,trans} + P_{m,rot} \quad (5.5)$$

where $P_{m,trans}$ and $P_{m,rot}$ is translatory movement and rotation movement power of motor, respectively.

The mechanical power $P_{m,trans}$ of the translatory movement is expressed by:

$$Pm, trans = \vec{F} \cdot \vec{v}_F \quad (5.6)$$

For the consideration of an AGV, the driving resistance F_D, can be used, which is composed of the resistance of air, rolling friction resistance F_R, acceleration resistance F_A. Based on the maximum velocity of an AGV is 1.5 m/s, the air resistance can be neglected. Consequently, the mechanical power of the translatory movement is:

$$Pm, trans = F_D \cdot v = (F_R + F_A) \cdot v \quad (5.7)$$

The acceleration resistance F_A can be defined due to Newton's 2nd law calculated as

$$F_A = (m_{AGV} + m_{load}) \cdot a \quad (5.8)$$

Regarding the three movement modules for the translator motion, a case distinction must be made for the acceleration of the AGV:

$$a = \begin{cases} a_0 + \mu_R g & (Acceleration) \\ 0 & (Rolling) \\ a_0 - \mu_R g & (Braking) \end{cases} \quad (5.9)$$

For the mechanical power of the translator movement, the calculation can be combined:

$$P_{m,trans} = (m_{AGV} + m_{load}) \cdot (\mu_R \cdot g + a) \cdot v \quad (5.10)$$

For the modeling of the mechanical power for the rotation of the AGV, it can be started from as

$$P_{m,rot} = \vec{M} \cdot \vec{\omega} \quad (5.11)$$

The relationship results for the torque M in dependence of the rolling friction resistance and the acceleration resistance can be written as follows:

$$M = \frac{F_R + F_A}{2} \cdot l_{AGV} = (m_{AGV} + m_{load}) \cdot (\mu_R \cdot g + a) \cdot v \quad (5.12)$$

This results for the mechanical rotational power can be expressed as

$$P_{m,rot} = \frac{(m_{AGV} + m_{load}) \cdot (\mu_R \cdot g + a)}{2} \cdot l_{AGV} \cdot \omega \quad (5.13)$$

5.2.4 Task duration

A task is delayed when there is no available AGV to service the task at the starting time of the task. The time delay of each task T_{AGV_k} can be calculated as follows:

$$T_{task_k} = T_{start} - T_{end} \quad (5.14)$$

where T_{start} is the starting time of a task; T_{end} is the approximated time that the task is completed as generated from the scheduling algorithm.

5.3 Preliminaries

In the operation of the material handling system, it is important to consider the impact of each element on one another. Behaviors of each AGV will have effects on the path planning and scheduling of another. The following preliminaries were implemented to ensure smooth and collision-free operation of the system as well as simplify the path planning model for the system:

1. The AGV will start executing the next task from the end position of the current task.
2. A charging "task" is automatically assigned to the vehicle with battery capacity lower than a critical threshold.
3. At any time, only one AGV car can move along one direction on a road.
4. The vehicle moves at a constant speed and makes every turn with the same amount of time.
5. The AGV is regarded as a particle with a safe radius, AGVs must keep a safe distance from each other conforming to the safe radius.
6. An optimal set of paths always exists, such that any path considered is a subset of this set. If any path in the set is unable to be conducted, the AGV moves on

to the next optimal set, which has the total distance larger than that of the previous optimal set.

5.4 Methodology

Table 1. Nomenclature

$\mathcal{R}$	Set of requests (tasks)
$\mathbb{Q}$	Set of AGVs
$\mathcal{B}(i,j)$	The set of optimal paths from node i to node j
$(l,m)_r$	Inbound and Outbound node of task r
$E(D_{ij},v)$	Electric energy consumption of AGV go along path D_{ij} at the velocity v
t_x	Time when an AGV x is considered.
T_x^{ij}	Total traveling time of optimal path of AGV x at maximum allowed speed.
d_{ij}	Path length between two nodes in the graph.
t_r^s	Variable representing the start time of a task r.
t_r^e	Variable representing the predicted end time of a task r.
P_x^t	Current position of AGV x at time t.
D_t^{xy}	Distance between AGV x and y at time t.
Γ_r	Critical battery energy threshold of AGV r.
θ	Charging point of map G.

From the head problem description in Section 2, the optimization objective can be obtained from (2) as:

$$\begin{cases} min(P_e) \\ min(t_{run}) \end{cases} \quad (5.15)$$

The head problem is separated into subproblems, the goal of each is to reduce the total traveling time t_{run} and the required electrical power of each AGV for any task. AGVs must be scheduled properly with the optimal path to avoid collision and battery shortage.

5.5 Shortest path problem

The Artificial Bee Colony (ABC) optimization algorithm is utilized to calculate the shortest path between a starting node and a destination node. This algorithm is inspired by the foraging behavior of honeybees and is commonly used for numerical optimization problems [28]. In this study, the ABC algorithm is extended to address the specific task of solving the shortest path problem.

The ABC algorithm employs a colony of artificial bees, divided into three groups: employed bees, onlookers, and scouts. Each food source's position represents a potential solution to the optimization problem, and the nectar amount associated with a food source reflects the quality or fitness of that solution.

An employed bee explores her memory and modifies the position (solution) based on local information (visual information). She then evaluates the nectar amount (fitness value) of the newly generated solution. If the nectar amount of the new solution is greater than that of the previous one, the bee memorizes the new position and discards the old one. This process allows the bees to continuously refine their solutions in search of the optimal path.

Detailed pseudo-code of the ABC algorithm is given below:

1: Initialize the population of solutions xi,j, i = 1 . . .SN, j = 1 . . .D
2: Evaluate the population
3: cycle=1
4: repeat
5: Produce new solutions υi,j for the employed bees by using (2) and evaluate them
6: Apply the greedy selection process
7: Calculate the probability values Pi,j for the solutions xi,j by (1)
8: Produce the new solutions υi,j for the onlookers from the solutions xi,j selected depending on Pi,j and evaluate them
9: Apply the greedy selection process
10: Determine the abandoned solution for the scout, if exists, and replace it
11: with a new randomly produced solution xi,jby (3)
12: Memorize the best solution achieved so far
13: cycle=cycle+1
14: until cycle=MCN

Figure 5.1. Pseudo code of ABC algorithm

5.6 Constraints and subproblems

5.6.1 Battery constraints

Battery shortage problem will occur when an AGV cannot complete its task due to insufficient energy. The battery constraint is implemented to prevent battery shortage and auto-charging planning for AGVs. The AGV c will be selected for task r if it satisfies the condition:

$$d_{ij} \in D((l,m)_r) \vee D((m,\theta)_r): \sum_{i=1}^{n} E(d_{ij}, v) < \Upsilon_r \qquad (5.16)$$

The system sequentially checks available AGVs for any car satisfying the energy demand to perform its assigned task(s), and sufficient energy to travel from its final position to a charging station. If any AGV has its battery capacity lower than a critical battery energy threshold Γ_r, a charging "task" is assigned to the vehicle.

5.6.2 Load priority

The load weight carried by the AGV determines the task's priority. When resolving any conflict or any constraints, the lower payload weight AGV will have its optimal route recalculated until no collision detected.

5.6.3 Conflict constraints

Conflict constraints pertain to the issue of potential physical collisions between AGVs (Automated Guided Vehicles). These collisions can occur when AGVs traverse conflicting routes, leading to interruptions in the manufacturing process, decreased efficiency, and higher operational costs. To address this problem, the scheduling and path planning algorithm incorporates conflict constraints to find the optimal solution that avoids collisions.

The map topology is formulated to divide the map into sections and nodes, which results in two subproblems for the constraints: the section-collision constraint and the node-conflict constraint. Figure (49.a) and Figure (49.b) illustrate the section collision problem, which occurs when two or more AGVs attempt to access the same road section simultaneously. On the other hand, Figure (49.b) and Figure (49.c) depict the node collision problem, wherein two or more AGVs converge on the same node in a short span of time.

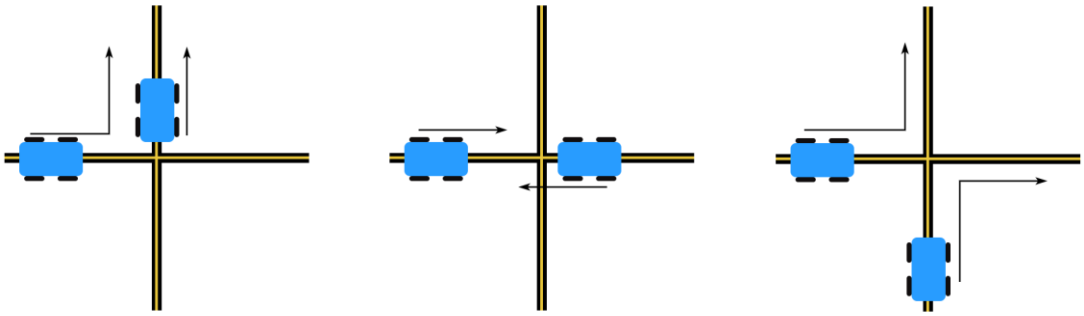

Figure 5.2. Types of physical conflicts AGVs may encounter

a) Path Conflict *b) Path & Node Conflict* *c) Node Conflict*

a) Section-collision constraint

The path constraints ensure that a path collision does not occur. A path collision can occur when two AGVs go into the same road in opposite directions. Consequently, these two AGVs must stop and wait for human intervention, leading to incomplete schedules, and the road where the collision occurs will not be available for other cars.

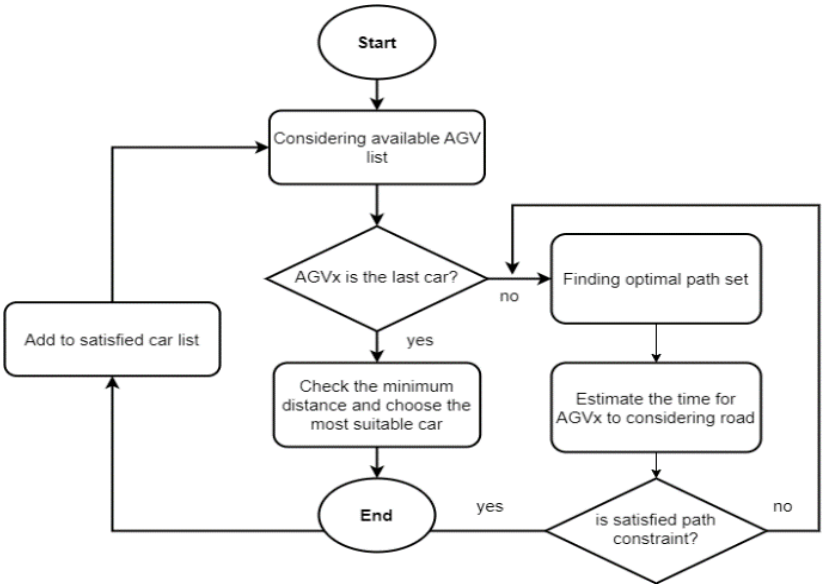

Figure 5.3. Path collision avoidance algorithm

The path constraint is expressed as

$$\forall x \in \mathbb{Q}, t_c \in T_c^{ij} : P_x^t.F \neq j, P_x^t.S \neq i \qquad (5.17)$$

where c is the AGV being considered. Consider an AGV x. At any time that this car wants to choose the road y to run, the algorithm will calculate the predicted time range this AGV goes along in this road. At that time, if existing AGVs go into road y in the opposite direction with x, the value of objective function will be increased sharply telling car x that it needs to choose another road. This work is iterated until a non-conflicting path is assigned to AGV x.

b) Node-collision constraint

Two or more AGV may encounter conflict when they head to the same path in a time span. Consider two AGVs with conflict: AGV x and AGV y. Collision avoidance is achieved by conforming to the constraint:

$$\forall y \in \mathbb{Q} : D_{t_c}^{xy} > S \qquad (5.18)$$

The safe distance between two vehicles is an important constraint when solving conflict problems [4], [29]. The conflict avoidance algorithm checks for the distance and time of arrival to node N for every other AGV at time t_c. If the safety condition is not met, the AGV x lowers its speed until the safety threshold is met and the other AGV passes the possible conflict node without collision. After passing the conflict node, the AGV x will accelerate to its maximum allowed speed on the next section. The flow of the algorithm is described in the figure below. Conflict avoidance method for multiple AGVs by accelerating or accelerating control was shown to solve conflict fast and reduce the possibility of secondary conflict.

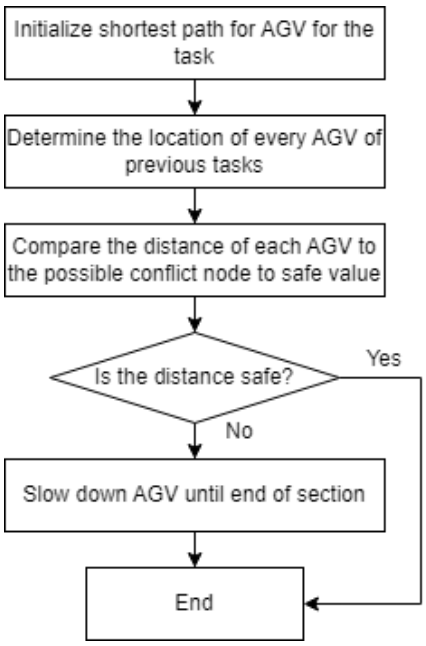

Figure 5.4. Node collision avoidance algorithm

CHAPTER 6. EXPERIMENTAL IMPLEMENTATION AND RESULT

Throughout the project's development, efforts were made to ensure the quality of the production process. At each milestone, careful evaluations and necessary adjustments were implemented to enhance system functionality and incorporate novel features that align with the specified design criteria.

6.1 Experimental method

To assess the effectiveness of the designed system and proposed optimization problem, testing is conducted using both computational simulations (CS) and real-life experiments (IRL). The experiment results provide valuable insights into the supervisory system's performance, the scheduling and path planning algorithm and its associated constraints.

In the CS tests, the algorithm's capability to improve operational costs in both small and large-scale operations is examined. Additionally, the efficiency of the proposed function is evaluated in terms of the time required for the algorithm to generate schedules, considering different numbers of tasks. These results are then compared with the outcomes from the method based on Dijkstra's algorithm, as presented in reference [30].

On the other hand, in the IRL tests, the performance of the server and communication design is evaluated in real-life settings to assess its practicality and effectiveness.

6.2 Computational simulation tests

The system underwent computational simulations to assess how effectively its path planning and task scheduling strategies perform. The tests utilized a plan map inspired by an actual manufacturing setup, depicted in Figure 6.1. The analytical outcomes from these simulations offer valuable insights into the practicality and effectiveness of the implemented path planning and task scheduling approaches within the real-world manufacturing environment. This information is crucial for evaluating the system's performance, accuracy, and applicability, and it also serves as a basis for potential enhancements and optimizations.

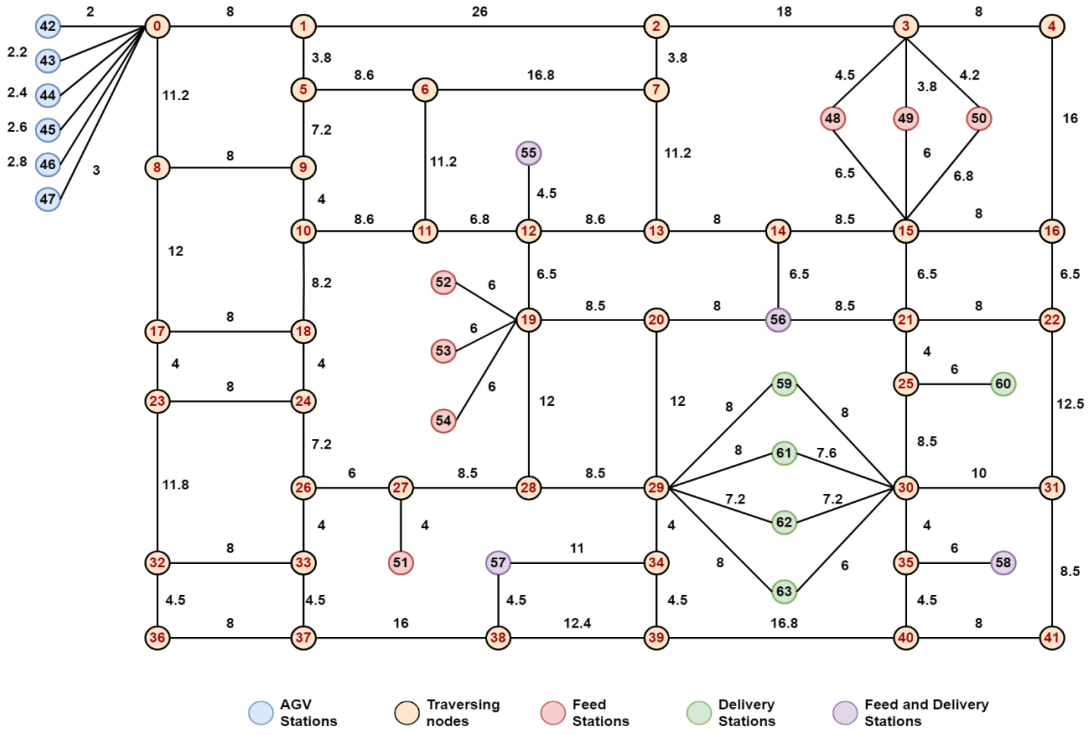

Figure 6.1. Simulation study Map

The map contains bidirectional roads, with each road's length measured in meters (m). The study uses a uniform fleet of AGVs, all with the same characteristics such as weight, battery capacity, maximum velocity, and load-carrying capability. AGVs start their operations from designated stations marked in blue on the map in Figure 6.1, beginning with a full battery capacity of 432 kJ. The simulation setting assumes no errors during AGV operation.

Input for the simulation consists of a timetable containing tasks with varying time and load requirements, as shown in Figure 6.2. The desired output includes complete schedules with generated routes for AGVs, along with metrics such as AGV energy expenditure (SEEX) and task duration (STD) corresponding to each task in the schedules, as presented in Figure 6.3.

For Case study one, the focus is on assessing the operational performance indices of the proposed ABC algorithm with constraints. These indices include the sum of task duration (STDur), the sum of energy expenditure (SEEX), and the sum of travel distance (STDis). These performance metrics are compared against those obtained from the Dijkstra method mentioned in reference [30]. SEEX reflects the total energy expended by each AGV to complete a task, measured in kilojoules (kJ). STDur represents the total time taken to complete all tasks, measured in minutes. STDis represents the overall distance traveled by an AGV to complete all tasks within a single day.

In Case study two, the primary objective is to evaluate the code execution time (CET) of the algorithm used for generating schedules from task inputs. The CET measures the duration of code execution from the beginning of the scheduling and

path planning process until its completion, excluding any time required for file read and write operations.

	A	B	C	D	E	F	G
1	Order	Load Nam	Number	Weight	Start Time	Inbound	Outbound
2	1	Steel	20	43	08:00:05	17	41
3	2	Steel	20	43	08:00:05	34	28
4	3	Steel	20	14	08:00:05	29	32
5	4	Steel	20	38	08:00:05	29	28
6	5	Steel	29	45	08:00:05	5	33
7	6	Steel	20	98	08:01:58	28	43
8	7	Steel	20	57	08:02:49	15	5
9	8	Steel	20	87	08:03:48	41	10
10	9	Steel	20	4	08:03:48	46	25
11	10	Steel	20	21	08:04:23	28	42
12	11	Steel	20	12	08:04:23	36	32
13	12	Steel	20	39	08:04:23	42	33
14	13	Steel	20	58	08:04:49	0	23
15	14	Steel	20	73	08:05:11	25	28
16	15	Steel	29	61	08:06:00	25	16

Figure 6.2. Example of timetable file containing 15 orders

	A	B	C	D	E	F	G	H	I	J	K	L
1	Order	Name	CarId	BatteryCapacity	TotalEnergy	WeightLoad	PlannedTime	ActualTimeSt	TimeEnd	Lateness	Inbound	Outbound
2	1	Steel	4	98.69	5680.038	43	08:00:05	08:00:05	08:02:46	0	39	2
3	2	Steel	1	98.7	5623.386	43	08:06:05	08:06:05	08:08:32	0	33	5
4	3	Steel	2	99.19	3492.18	14	08:11:05	08:11:05	08:13:23	0	17	46
5	4	Steel	3	98.48	6545.003	38	08:20:05	08:20:05	08:23:05	0	25	16
6	5	Steel	2	98.83	1558.948	45	08:26:05	08:26:05	08:26:55	0	39	45
7	6	Steel	1	97.33	5922.659	98	08:33:05	08:33:05	08:34:26	0	10	31
8	7	Steel	0	98.77	5330.177	57	08:38:05	08:38:05	08:40:09	0	9	35
9	8	Steel	4	97.51	5103.632	87	08:46:05	08:46:05	08:47:41	0	0	2
10	9	Steel	1	97.06	1160.331	4	08:51:05	08:51:05	08:52:14	0	18	9
11	10	Steel	0	98.19	2519.162	21	08:58:05	08:58:05	08:59:30	0	29	15
12	11	Steel	4	96.9	2631.86	12	09:06:05	09:06:05	09:07:51	0	2	25
13	12	Steel	0	97.61	2504.836	39	09:16:05	09:16:05	09:17:14	0	21	6
14	13	Steel	0	95.9	7390.657	58	09:22:05	09:22:05	09:24:14	0	2	36
15	14	Steel	4	95.28	6991.021	73	09:30:05	09:30:05	09:31:53	0	34	27

Figure 6.3. Example of generated schedule file containing 15 tasks

a) Case study 1: Operation performance indices

The input consists of ten (10) timetables, each containing thirty tasks with different time and load requirements. The desired output includes complete schedules with metrics such as AGV energy expenditure, task duration, and travel distance for each corresponding task in the schedules.

The results are presented using red bars, which represent the outcomes generated by the proposed ABC algorithm with constraints (ABC-Constraints). In contrast, the blue-dashed bar represents the results obtained from the Dijkstra method (Dijkstra). In general, the results from ABC-Constraints outperform those from the Dijkstra method.

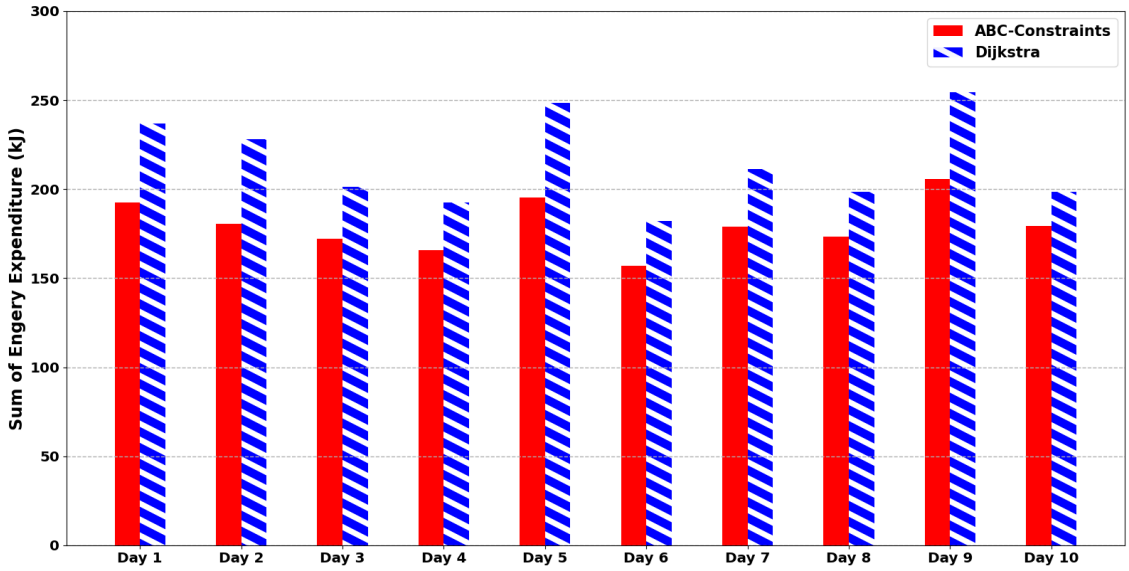

Figure 6.4 Sum of Energy Expenditure of AGVs in a day (kJ)

Figure 6.4 shows a comparison between the SEEX (AGV energy expenditure) generated by the ABC-Constraints and the Dijkstra method. The results demonstrate that ABC-Constraints outperformed the Dijkstra method by significantly reducing the energy consumption of AGVs during task execution. On average, there was a notable 10-20% reduction in energy expenditure for each vehicle while performing tasks. Remarkably, on Day 5 and Day 9, the SEEX decreased by approximately 20%, dropping from around 250kJ to approximately 200kJ.

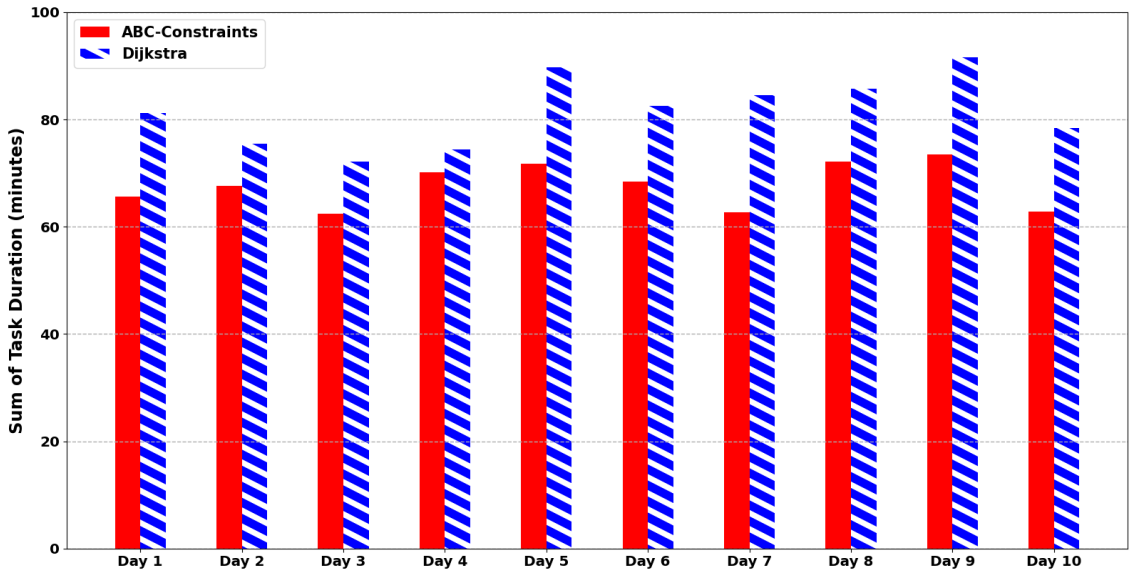

Figure 6.5 Sum of task Duration of all tasks in a day (minutes)

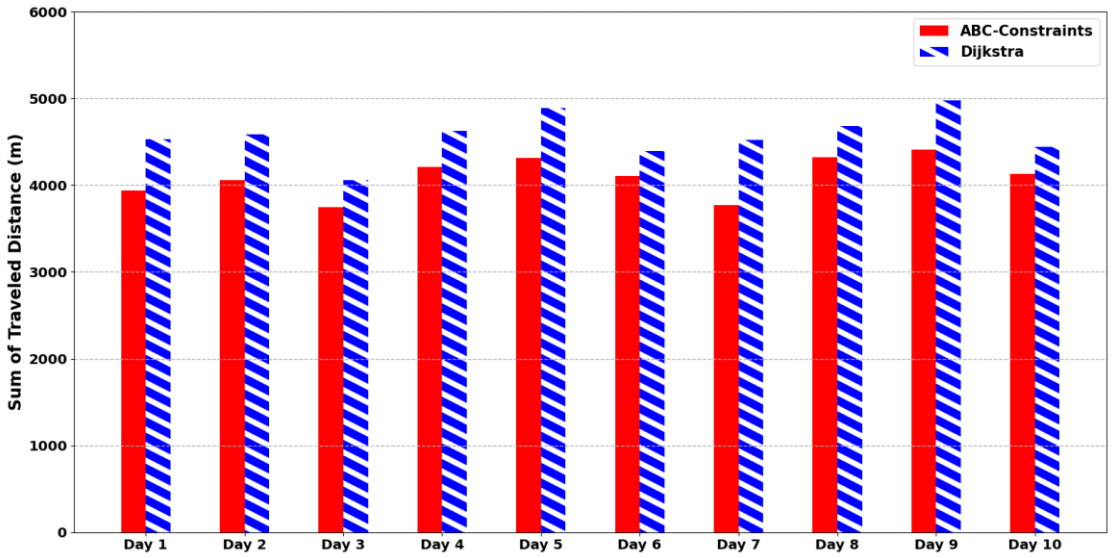

Figure 6.6 Sum of Travelled Distance by AGVs for each task in a day (m)

Figure 6.5 and Figure 6.6 depict the difference in STDur (task duration) and STDis (travel distance) resulting from the two approaches. ABC-Constraints demonstrated superior performance compared to the Dijkstra method in reducing these two parameters. Particularly, on Day 5 and Day 9, there was a noticeable drop in STDur by about 22%, equivalent to approximately 20 minutes (Fig. 6.5, decreasing from around 90 minutes to about 70 minutes). Additionally, STDis showed a decrease of around 10% (Fig. 6.6, reducing from 4800m to 4300m on Day 5 and around 5000m to 4400m on Day 9).

The ABC-Constraints approach successfully improved operational performance indices, including energy consumption, operational time, and travel distance. The reduction in operation duration and travel distance directly contributed to the decrease in energy consumption of the vehicles, as evident from Figure 6.4.

a) Case study 2: Code efficiency CET

The algorithm is subjected to testing using different timetables, each with a varying number of tasks per day (AoT): 1 task/day, 5 tasks/day, 10 tasks/day, 15 tasks/day, 20 tasks/day, 25 tasks/day, and 30 tasks/day. Each AoT is evaluated using 10 different timetables. For instance, AoT 5 is evaluated with ten timetables, each containing 5 tasks with different time and load requirements. The results for each AoT are represented by the code execution time (CET) for each timetable within that AoT.

Figure 6.7 presents a comparison of the CET between the ABC-Constraints algorithm and the Dijkstra method. The red line represents the CET of ABC-Constraints, while the blue-dashed line represents the CET of the Dijkstra method. Overall, the CET of ABC-Constraints is consistently lower than that of the Dijkstra method.

When considering 1 task/day (AoT-1) and 5 tasks/day (AoT-5), both approaches have similar CETs (Code Execution Time), with the Dijkstra method even performing slightly better than ABC-Constraints at AoT-1, taking only about 4 seconds compared to 10 seconds for ABC-Constraints (Fig. 6.7). However, as the

number of tasks increases, the CET of the Dijkstra method begins to exceed that of ABC-Constraints, and at AoT-30, it nearly doubles the CET of ABC-Constraints. In contrast, the CET of ABC-Constraints shows an almost linear trend with increasing AoT, remaining more efficient as the number of tasks grows.

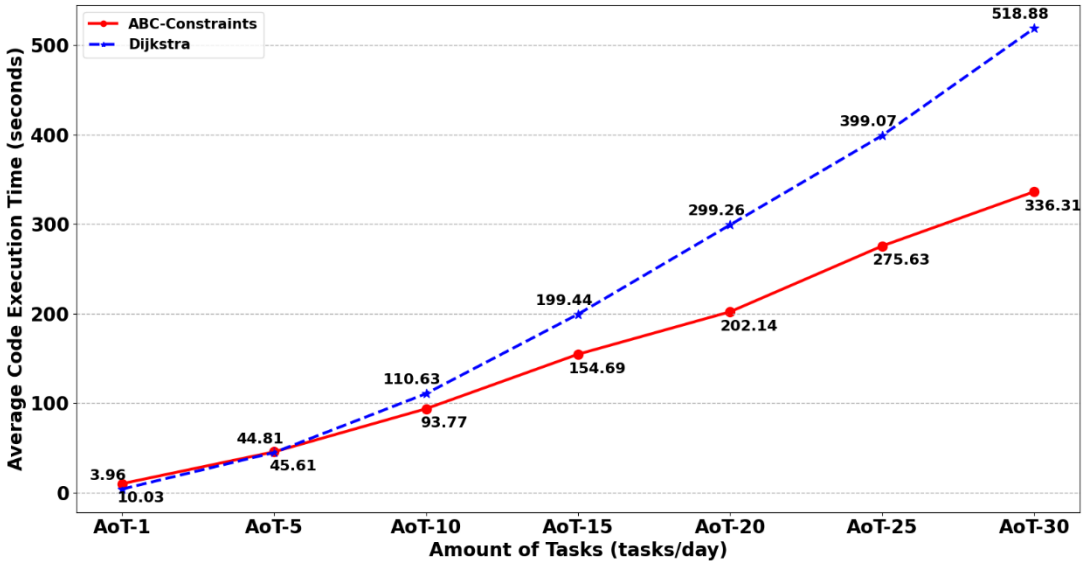

Figure 6.7. Average code execution time for different number of tasks per day

ABC-Constraints outperforms Dijkstra's algorithm for several reasons. First, in the ABC algorithm, the solution's quality is represented by a pheromone value that persists throughout the search until an improved solution is discovered. This approach essentially "snapshots" the solution, ensuring that progress is retained. On the other hand, Dijkstra's algorithm relies on tracking vertices and edges, which necessitates blind scans every time the algorithm runs. Moreover, Dijkstra's algorithm struggles to find the shortest path in acyclic graphs where not all vertices are directly connected by edges. This limitation results in longer processing times as the problem size increases. In contrast, ABC-Constraints handles such scenarios more efficiently, making it a superior choice for certain problems.

6.3 Real life system tests

The practical experiments provide analytical results regarding the performance of the server and communication design. These tests involve using two computers, where one serves as the server and database, and the other functions as the client. Since the system is not fully deployed, the server operates within a development

console. To connect to the server, the client establishes a connection either through LAN or Wi-Fi.

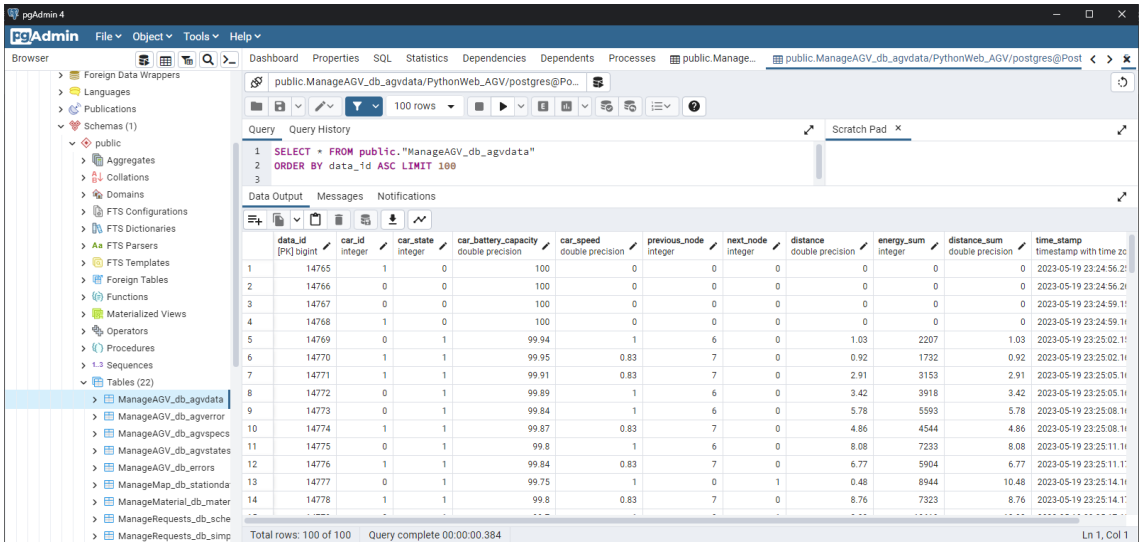

Figure 6.8. AGV live data table in the database

The server machine serves as the host for the Django application, PostgreSQL database server (as depicted in Figure 5.8), and Mosquitto MQTT Broker service. Additionally, the software MQTT Explorer is installed to monitor the MQTT Broker's status during the development process. To run the Django application locally, it is initiated from the development console, and the server becomes accessible through the default local address http://localhost:8000.

On the client machine, the Web-client is run locally by starting the React app from the development console. This automatically opens the application in the default web browser, accessible through http://localhost:3000. In case the browser fails to launch automatically, the application can be accessed by manually entering the URL in the browser.

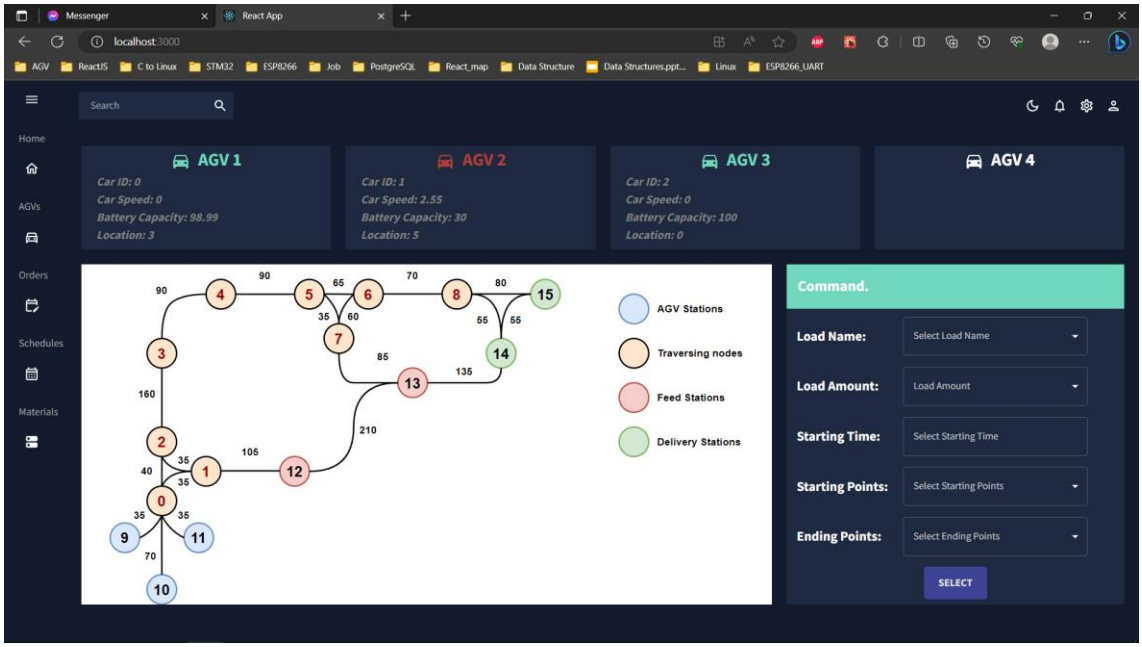

Figure 6.9. The Web application Home page on the client machine

The Web app's Home page displays a map of the plan, a “Command" box for swift order placement, and live data boxes featuring information on the four most recently active AGVs. Additionally, there is a left sidebar that provides access to other sections of the Web application, as illustrated in Figure 6.8.

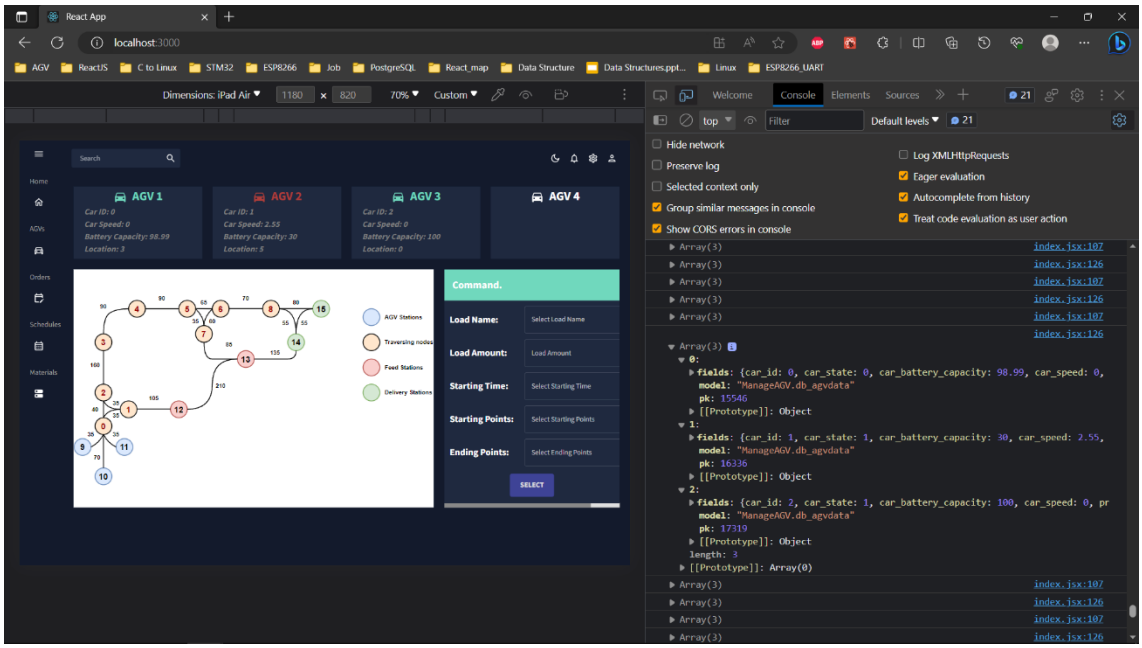

Figure 6.10. WebSocket packets send live AGV data from the server to the client.

The live data from AGVs are sent to the Web client through the WebSocket protocol, in the form of JSON objects. The live AGV data packet from the database can be seen using the development console on the Web browser as shown in Fig. 6.9.

The scheduling and path planning algorithm is also utilized in the real system. The system is evaluated with a small-scale map based on a real manufacturing setting: Fig. 6.9 is the graphical illustration of the map, and Fig. 6.10 is the map in the real-life test.

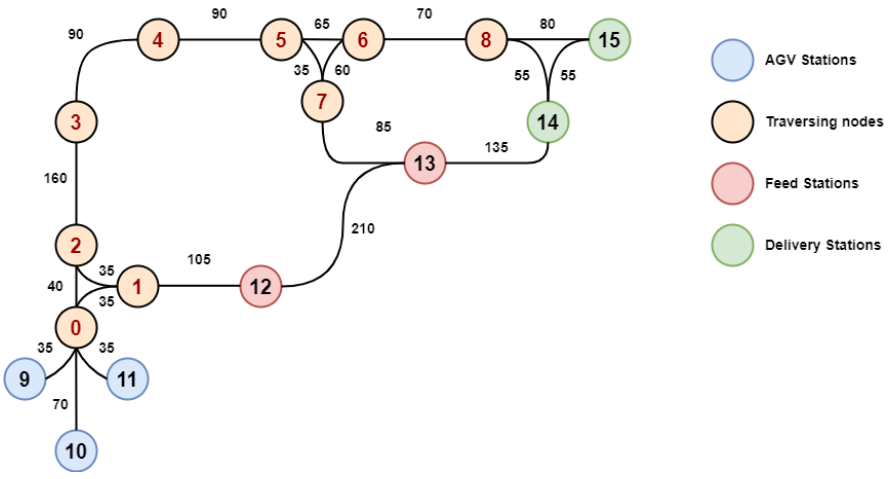

Figure 6.11. Illustration of testing map

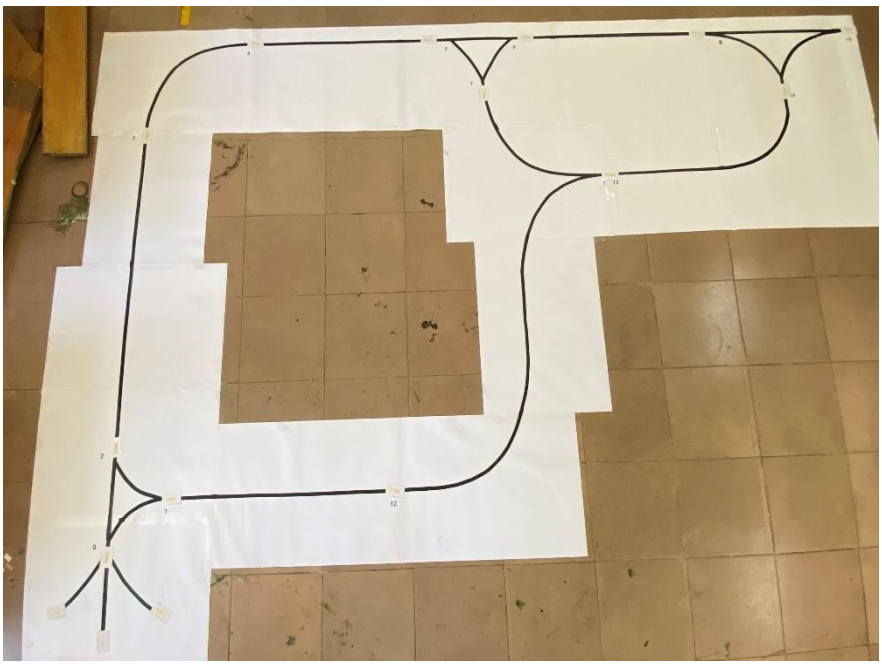

Figure 6.12. Actual test map with taped lines and RFID tags on nodes.

In the graphical representation of the map (Figure 6.11), there are sixteen nodes, numbered from 0 to 15. The roads depicted on the map are bidirectional, and each road's length is measured in centimeters (cm). The AGVs (Automated Guided Vehicles) begin their journey at the starting station, represented by a blue circle, while the other nodes are intended for traversal.

On the actual map, roads are depicted as colored lines, and nodes are represented as RFID tags, also numbered from 0 to 15, which the AGVs follow accordingly.

To facilitate the study, two AGVs with identical parameters (weight, battery capacity, maximum velocity, and load-carrying capability) are used. The system is equipped with a timetable file that contains two orders (Figure 6.12).

csv_test_1.csv

```
1 order_number,order_date,load_name,load_amount,start_time,from_node,to_node
2 0,2023-07-30,Steel,4,16:30:00,0,15
3 1,2023-07-30,Steel,4,16:30:00,0,14
```

Figure 6.13. Content of Timetable

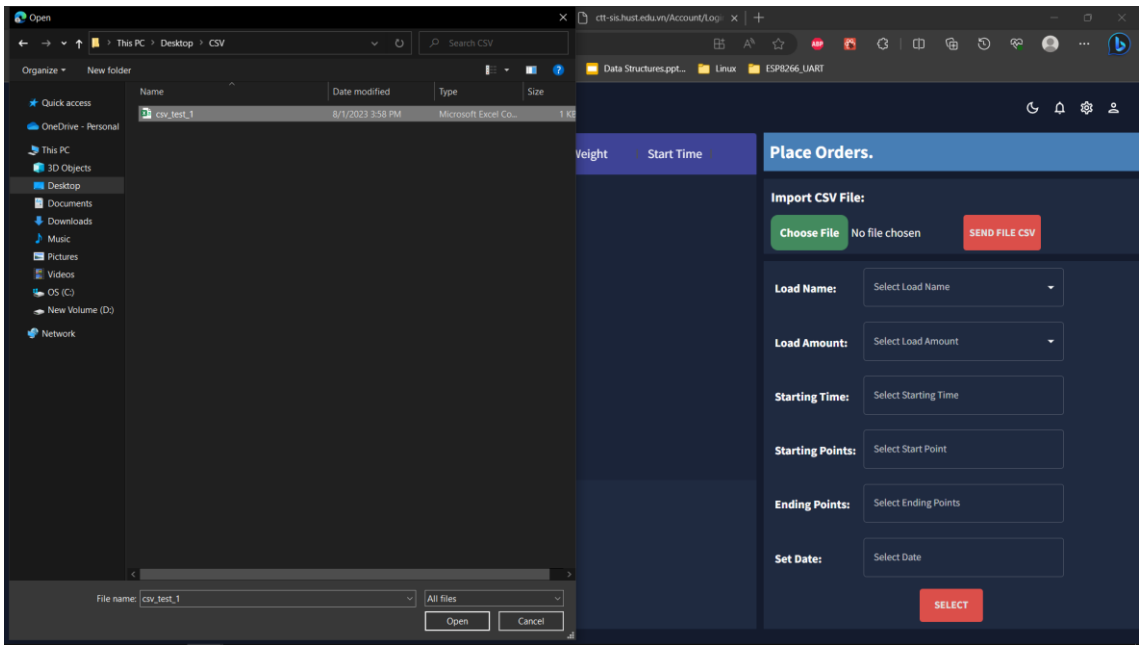

Figure 6.14. Uploading the timetable file to the server

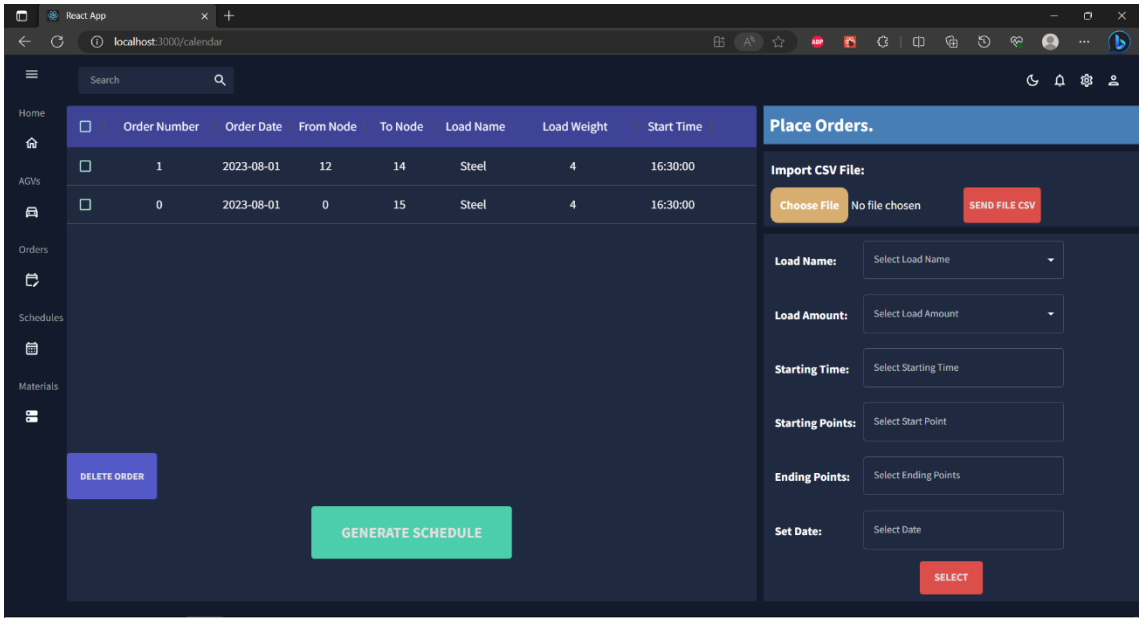

Figure 6.15. The interface shows uploaded orders.

After the file is uploaded to the server (as shown in Fig. 5.15), the client UI displays the contents of each task on the Orders page. The user then uses the “Generate Schedule” button to start the scheduling process.

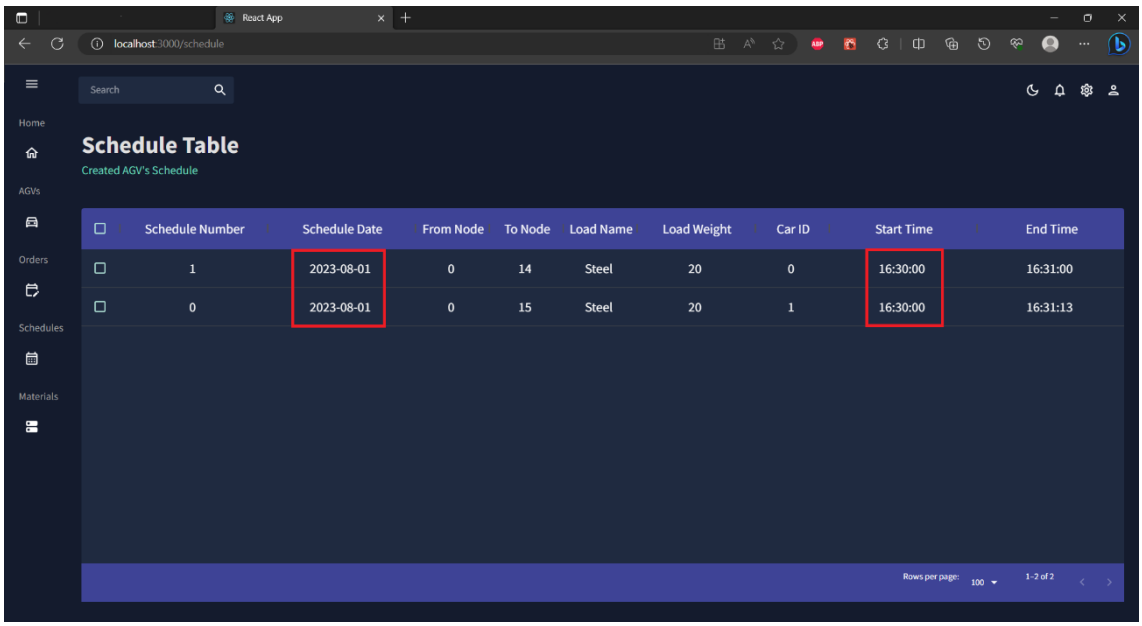

Figure 6.16. Generated schedule

The scheduling process initiates by retrieving task information from the database. Subsequently, the scheduling and path planning algorithm determines the tasks by selecting the appropriate AGV for each assignment, computing a suitable start time (typically matching the corresponding order's start time), and generating route instructions for the AGV. The resulting schedule is then displayed on the Schedule Tasks page, as illustrated in Figure 5.16.

The instruction list is an array containing multiple arrays, first is the ID of the assigned AGV (1), then the instruction for one section of the planned path in the form *[{current node}, {speed (m/s)}, {length of section (m}, {action}]*. For example, this is the instructions for AGV ID 1 for task #0:

[1, [9, 0.1, 0.35, 3], [0, 0.2, 0.4, 1], [2, 0.2, 1.6, 1], [3, 0.2, 0.9, 1], [4, 0.2, 0.9, 1], [5, 0.2, 0.35, 1], [6, 0.2, 1.05, 1], [8, 0.2, 0.75, 1], [15, 0, 0, 0]]

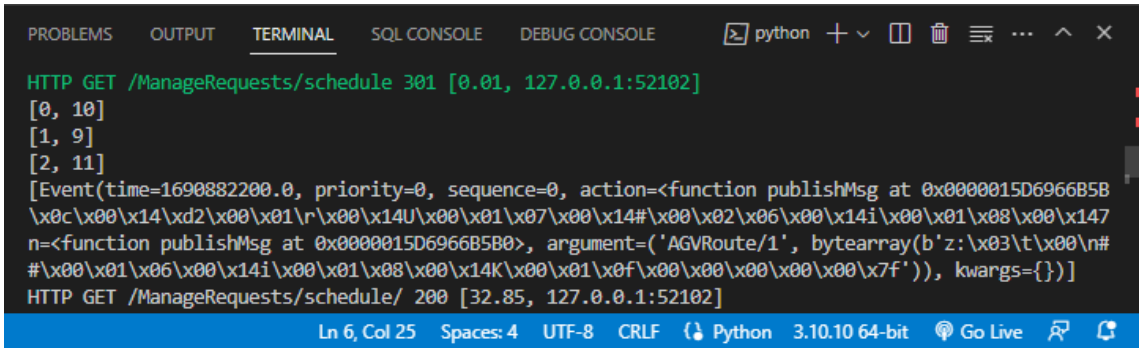

Figure 6.17. The Event Schedule function scheduling to send instructions to AGVs.

The Python Event Scheduler function receives the generated task instructions and arranges them to be encoded and transmitted to the respective AGV precisely at the designated starting time of each task, as depicted in Figure 5.17.

Using the MQTT Explorer interface (utilized for monitoring messages on the system's MQTT Broker), the route instructions for the tasks are sent to the correct AGV precisely at the task's starting time, as illustrated in Figure 5.18.

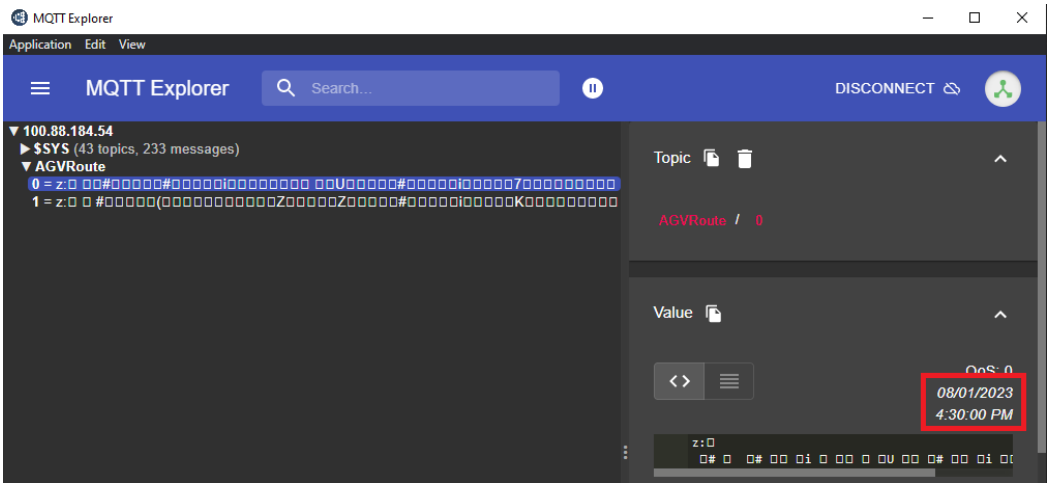

Figure 6.18. The server sends instructions to AGVs at task start time

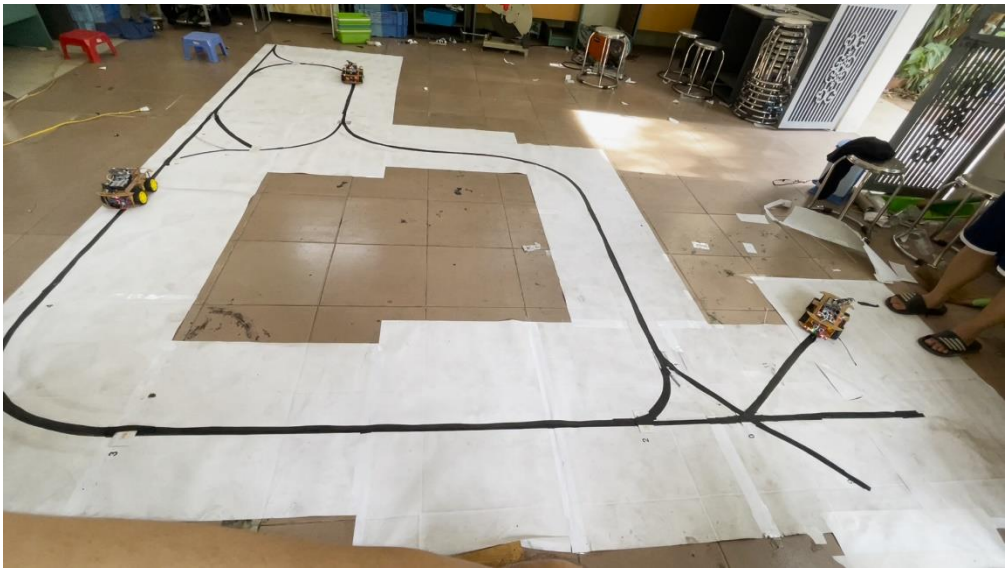

Figure 6.19. Model AGVs in operation in IRL tests

The system performed admirably as anticipated. The Web server was constructed using the Django Web Framework and integrated with fundamental features, including databases, real-time AGV data display for the Web client, AGV fleet and orders/tasks management, and the scheduling and AGV path planning function. The scheduling and path planning function efficiently generated schedules and identified the shortest paths, as verified during testing. The Web client, developed with the ReactJS framework, adhered to standards, boasting a clear and functional interface that exhibited all services provided by the server. The communication structure designed between AGVs, and the server functioned

effectively, ensuring swift data exchange. The model AGVs successfully adhered to route instructions dispatched by the server.

However, there was one challenge arising from the map's low-quality print on paper. This resulted in AGVs occasionally slipping off course, necessitating manual intervention to return them to the designated path. This issue is purely environmental and not related to the system's design. Unfortunately, there is no solution to address this problem, as it depends solely on the map's physical condition. OBJ