ENGG2000/3000 Department of Engineering, Macquarie University.

Design Document

Blade Runner, 2024



1. **REVISION LOG**

| **Version** | **Changes Made** | **Contributors** | **Date** |
| --- | --- | --- | --- |
| 1.0 | * Initial draft created. * Sections created: Introduction, Summary, Problem Definition, Requirements, Inclusions and Exclusions, Preliminary Design, and Appendices. | All | 06/08/24 |
| 1.1 | * Added requirements and the respective acceptance criteria   + Functional requirements   + Performance requirements   + Interface requirements * Updated Scope of project * Gantt chart | All | 06/08/24 |
| 1.2 | * Updated Scope of project * Gantt chart * Updated Scope of project * Gantt chart | All | 12/08/24 |
| 1.3 | * Reorganised doc structure * Added to problem and objectives * Added conceptual design and design alternative * Added the detail design * Design traceability | All | 11/10/24 |

#### *Table ii. Document version history.*

1. **Table of Contents**

[**1 Introduction**](#_h09np7fr9w9x) **7**

[1.1 Problem Statement](#_7e9s64dr16xq) 8

[1.2 Subsystems](#_8oxm3t9j5c2) 8

[1.3 Project Scope and Constraints](#_yjizmfh8swcb) 10

[**2 Requirements 12**](#_b1wtrsqz8zu7)

[2.1 Functional Requirements 12](#_3bzmrgzhybq2)

[2.2 Non-functional Requirements 13](#_ojckeuv7188f)

[2.3 Interfacing Requirements 1](#_atefb6psniq6)4

[2.4 Requirements Sign-Off 1](#_atefb6psniq6)5

[**3 Conceptual Design 1**](#_kqr37yfg7293)**6**

[3.1 Initial Analysis 1](#_oaqt5p5w4wvj)6

[3.2 Design Alternative 1](#_bhr7in5d2whw)7

[3.3 Comparison of Design Alternatives 1](#_jkmjcfl7nbt)8

[**4 Detailed Design**](#_vm4pgwtm5uok) **20**

[4.1 Detailed design of CCP](#_ycla4fgo5ab0) 22

[4.2 State machine diagram](#_dj8wcu81606k) 24

4.3 Detailed Design of BR Program26

4.4 Design traceability26

[**5 Technical Performance Measures (TPMs) 2**](#_rt1dyt39pyhs)**8**

5.1 Summary of TPMs28

5.2 List of TPMs29

**6** [**Design Structure Matrix (DSM)**](#_rt1dyt39pyhs) **35**

**7** [**Summary**](#_rt1dyt39pyhs) **35**

[**Appendices**](#_rt1dyt39pyhs) **35**

Appendix A36

Appendix B38

**Gen AI Statement****39**

1. **List of Tables**

[Table i. Glossary](#_apxwaat9erfh) 6

[Table 1.1. List of contributors.](#_wanb5v1ikuff) 8

[Table 1.2: Subsystems for communications system](#_lszpjiu1dsxm) 9

[Table 1.3: Engineering team](https://docs.google.com/document/d/1CyopB6uvuMPpeZ8naO2-DviSIIoWh5l8_WiaXwKxnkQ/edit#heading=h.ao59qcde6f84) 10

Table 1.4: List of constraints [1](#_qnsg624llx6g)0

Table 2.1. Functional requirements [1](https://docs.google.com/document/d/1CyopB6uvuMPpeZ8naO2-DviSIIoWh5l8_WiaXwKxnkQ/edit#heading=h.361wc8pqhcnh)2

Table 2.2. Acceptance criteria for the functional requirements [1](#_r5s7d2wexenq)3

Table 2.3. Non-functional requirements [1](#_5i9lfx5z4178)3

Table 2.4. Acceptance criteria for the non-functional requirements [1](#_e94e4x3s5dd)4

Table 2.5. Interfacing requirements [1](#_uqoa5ngqxet4)4

Table 2.6. Acceptance criteria for the interfacing requirements. [1](#_rzx2f5639zwx)4

Table 2.7. Requirements sign-off by collaborating team. [1](#_i66evkw6p6cq)5

[Table 4.1. Summary of classes in the CCP detailed design.](#_i66evkw6p6cq) 21

Table 4.2. Design traceability27

Table 5.1. TPM Summary28

Table 5.2. Carriage Path Request Management30

Table 5.3. MCP Route Management and Execution31

Table 5.4. Real-time route processing32

Table 5.5. Emergency Braking System Integration33

Table 5.6. LED Status Indicator Accuracy34

Table 6.1. DSM35

1. **List of Figures**

Figure 1.1 Previous UML Diagram17

Figure 1.2: High Level implementation of the system [20](#_wib87rfpjnam)

Figure 2.1 CCP UML Diagram22

Figure 2.2 State Machine Diagram25

Figure 2.3 BR Diagram26

Appendix A: Snippets of CCP code, written in Java.36

Appendix B: Snippets of BR code written for the onboard ESP3238

1. **Glossary**

| Term | Definition |
| --- | --- |
| Acceleration/Deceleration Commands | Commands from the MCP to the CCP that tell the Blade Runner to speed up (FFASTC) or slow down (FSLOWC), as explained in FR03. |
| BR (Blade Runner) | The self-driving carriage or 'train' that runs on the monorail track in the Blade Runner 2024 project, managed by software and hardware systems. |
| BR Program | The software on the Blade Runner's ESP32 microcontroller that reads commands from the CCP and controls things like movement, doors, and status lights. |
| CCIN (Control Communication IN) | A message from the CCP to the MCP to start communication, as explained in Section 4.1. |
| CCP (Carriage Control Program) | The software that connects the MCP and the Blade Runner's ESP32. It manages communication, processes commands, reports status, and handles errors. |
| Collision Detection | The Blade Runner's ability to spot obstacles or other carriages ahead using ultrasonic sensors, to avoid crashes as described in FR07. |
| Communications Team (T3\_C1) | The team that develops the software and communication systems for the Blade Runner project, making sure all parts talk to each other effectively. |
| ESP32 | An affordable, low-power microcontroller with built-in Wi-Fi and Bluetooth. It's used to control the Blade Runner's hardware and enable wireless communication between the CCP and BR Program. |
| FR (Functional Requirement) | Specific things the system or its parts must do, as defined in Section 2.1 of the Requirements. |
| FSM (Finite State Machine) | A model shown in the state machine diagram in Section 4.2, showing the different states and transitions of the CCP based on inputs. |
| Handshake Process | How two devices start communicating in networking. For TCP/IP, this involves a handshake that adds extra steps, as mentioned in Section 3.2. |
| T3\_M1 | Tuesday motion group 1 |
| T3\_S5 | Tuesday structure group 5 |

#### *Table i. Glossary*

# **Introduction**

The client, Macquarie University’s Faculty of Engineering, has tasked teams of student engineers with the construction of a monorail system. This system consists of a circular track and master control program (MCP) that sends commands over WiFi to control several blade runners (BRs, acting as the “trains” on the track) and checkpoints (CPs, acting as the “stations” that carriages stop at). Team T3\_C1, the authors of this document, was tasked with assisting other teams in the making of blade runners. The teams, originally divided via area of expertise (into structure, motion, and communications), were matched with each other to form a larger group consisting at least one of each specialisation. Upon completion of the project, the different groups’ blade runners will be placed on the track and run as a simulated transit system mirroring real-world train systems.

This design document, written by ENGG2000/3000 students in communications team T3\_C1 for the Blade Runner 2024 project, lays out a detailed design of the software system that T3\_C1 has developed to fulfil its part of the project.

T3\_C1 is composed of driven undergraduate software engineering students aiming to deliver effective software solutions for real-world problems. Members include:

| **Name** | **OneID** | **Email** |
| --- | --- | --- |
| Liam Tan | 47505508 | liamisaacching.tan@students.mq.edu.au |
| Pratik Gurung | 47215607 | pratik.gurung@students.mq.edu.au |
| Mihir Sundriyal | 47228164 | mihir.sundriyal@students.mq.edu.au |
| Wasif Zaman | 47422645 | wasif.zaman1@students.mq.edu.au |
| Adya Saanvi | 47525967 | adya.saanvi@students.mq.edu.au |
| Pallika Kafle | 46746218 | pallika.kafle@students.mq.edu.au |

#### *Table 1.1. List of contributors.*

## 

## **1.1 Problem Statement**

Team T3\_C1, in conjunction with motion team T3\_M1 and structure teams T3\_S1 and T3\_S5, has been tasked to produce two blade runners (one for each structure team). As the communications team, T3\_C1 is responsible for developing the software that allows the blade runners to function. In each carriage, the system consists of the ESP32 microcontroller that drives the carriage and the carriage control program (CCP) that acts as an interface between the MCP and the ESP32. The CCP passes commands from the MCP to the ESP32 to execute, and the ESP32 passes status updates back to the MCP through the CCP. The MCP and the ESP32 should never directly interact. The code written for both the ESP32 and the CCP can be used by both carriages with only minor modifications between them.

## 

## **1.2 Subsystems**

The subsystems that comprise the whole of the blade runner system are summarised in table [1].

| **Subsystem** | **Responsibilities** |
| --- | --- |
| **Carriage** | * Manages the physical movement of the Blade Runner. * Controls speed, acceleration, and door operations. * Monitors sensor data to detect obstacles and movement status. |
| **Communications (Server)** | * Handles network communication between the Blade Runner and external interfaces. * Manages WiFi connections and data transmission. * Processes requests from the UI and sends appropriate response |
| **MCP (Master Control Program)** | * Observes and updates the state of the system based on the inputs from the Carriage. * Acts as the central processing unit that coordinates subsystem activities |
| **Logger** | * Logs events and errors occurring within the system. * Provides error handling and debugging information. * Ensures that all significant actions and issues are recorded for analysis. |
| **Route Planning** | * Handles the planning and decision-making for the Blade Runner's movements. * Determines the optimal paths and strategies for navigation. * Interacts with the Carriage to execute the planned routes. |
| **UI (User Interface)** | * Provides an online interface for users to monitor and control the Blade Runner. * Displays real-time system status and allows command inputs and Sends user commands to the Controller for processing. |
| **Controller** | * Acts as an intermediary between the UI and the Server. Processes user inputs from the UI and translates them into system commands. * Manages the system's current state and ensures it is accurately reflected in the UI. |

#### *Table 1.2: Subsystems for Communications system*

Each team in the greater group plays a distinct and vital role in contributing to the holistic development of the Blade Runner. This collaborative effort ensures that Blade Runner operates efficiently and effectively, achieving the project’s overarching goals. The responsibilities of each team can be seen in Table [?].

| **Engineering faculty** | **Responsibilities** |
| --- | --- |
| **Comms** | The system that facilitates the carriage and execution of the program’s instructions, ensuring that commands are transmitted and received accurately across the system. |
| **Motions** | Their goal is to create mechanisms and control systems that enable precise and responsive motions |
| **Structure** | Designing and constructing the physical framework of the Blade Runner. |

#### *Table 1.3: Engineering team*

## 

## **1.3 Project Scope and Constraints**

| **Constraint ID** | **Description** |
| --- | --- |
| C01 | The entire project is to be completed within 13 weeks. |
| C02 | There is a hard budget limit of $100 on the entire project. |
| C03 | The hardware in use is mostly determined by the structure and motions teams according to their designs. |
| C04 | The communications between the components must be carried over WiFi. |

#### *Table 1.4: List of constraints*

#### **Inclusions:**

1. **ESP32 software development**. T3\_C1 is responsible for developing the core functions that control how the BladeRunner moves along the track. This includes algorithms for starting, stopping, accelerating, and decelerating the carriage as needed. This also includes programming the mechanisms that control the opening and closing of doors when the carriage stops at stations.
2. **CCP development and interfaces**. The MCP needs a way to issue commands to the ESP32. Instead of a direct connection, the CCP is needed to mediate between the two. All commands and updates pass through the CCP before reaching an endpoint, whether that be the MCP or the ESP32.
3. **Wi-Fi communication**. The development and testing of wireless communication between the CCP and the MCP. This involves creating a protocol (using JSON) that allows status updates and commands to flow back and forth.
4. **Collision detection and IR sensors**. The ESP32 will integrate sensor data that helps detect nearby carriages and upcoming stations. This data is essential for making sure the carriages avoid collisions and stop at the correct locations. The sensors will send signals to the CCP, which reports to the MCP. The MCP will then issue commands accordingly.
5. **Error handling.** The system must know how to handle errors instead of failing entirely.

#### **Exclusions:**

1. **Master Control Program (MCP)**. Our team will not be developing or testing the MCP itself, but we will focus on making sure our CCP can communicate with it effectively. The MCP’s internal workings are outside the scope of our responsibilities.
2. **User Interface (UI)**. We won’t be working on any user interfaces for controlling or monitoring the BladeRunner. Our work is purely focused on the internal software, ensuring communication between the MCP and the BladeRunner’s hardware through the CCP.
3. **Physical Hardware Deployment**. The responsibility for setting up and installing the actual hardware (motors, sensors, LEDs, etc.) lies with other teams. We will focus on the software side and work in simulated environments to validate the software's functionality.

**2. Requirements**

## **2.1 Functional Requirements**

| **ID** | **Description** |
| --- | --- |
| FR01 | The carriage constantly updates the MCP with information regarding its location and status via WiFi. |
| FR02 | The MCP must be able to issue route commands to the CCP via WiFi. |
| FR03 | The carriage must be able to quickly accelerate from and decelerate to a full stop. |
| FR04 | LED lights on the carriages signify whether the carriage is stopped, in motion, or decelerating. |
| FR05 | The MCP must be able to operate the carriage’s doors via the CCP. |
| FR06 | The CCP should be able to detect any errors (e.g. communication errors) and report back to the MCP. |
| FR07 | The carriage must be able to detect obstacles on the track such as an unexpected object or a carriage either on the front or the back and avoid collision. |
| FR08 | The carriage should be able to support an emergency stop that can be triggered remotely or automatically. |

#### *Table 2.1. Functional requirements.*

| **ID** | **Acceptance Criteria** |
| --- | --- |
| FR01 | At regular intervals, the CCP communicates its location, route, and status.  The status message from the CCP must contain the information: client\_type, message, client\_id, sequence\_number and status. |
| FR02 | Once the MCP assigns a route to a carriage, the carriage must respond promptly and begin moving along the designated route. |
| FR03 | When the appropriate command is called, the carriage must be able to:   * Accelerate (FFASTC): from a complete stop, the carriage accelerates until it reaches a set cruising speed or until a decelerate command is called. * Decelerate (FSLOWC): the carriage decelerates until it reaches a complete stop. |
| FR04 | LEDs on the carriage change colours to reflect the carriage’s current status, as follows:   * Red, if the carriage is stopped * Yellow, if the carriage is decelerating, and * Green, if the carriage is in transit. |
| FR05 | Upon stopping at a station, doors on the side of the carriage must be able to open then close shortly after. |
| FR06 | When an error is detected, the CCP should immediately report it to the MCP. |
| FR07 | The sensors should be able to detect the obstacles within a certain range. When an obstacle is detected the carriage should decelerate and stop without making any contact. Then, send an alert to MCP indicating that an obstacle was detected. |
| FR08 | MCP should be able to issue an emergency stop command that instantly stops the carriage. The carriage should be able to execute a emergency stop in situations like system failure and collision risk. |

#### *Table 2.2. Acceptance criteria for the functional requirements.*

## **2.2 Non-functional Requirements**

| **ID** | **Description** |
| --- | --- |
| NR01 | The system should be able to detect, handle and recover from the error without interrupting the core functionality of the system. It should also be able to handle outages or other failures with minimal interruption. |
| NR02 | All moving parts should operate smoothly. |
| NR03 | The system should be easy to maintain and update. |
| NR04 | The system should be able to provide real time communication between MCP and the carriage with minimal delay. |
| NR05 | The system |

#### *Table 2.3. Non-functional requirements.*

| **ID** | **Acceptance Criteria** |
| --- | --- |
| NR01 | When an error is detected the system should be capable of logging the error and triggering the recovery protocol without having to do a manual reset. In the event of a partial or total system failure, the system should be able to resume its activities as soon as possible. |
| NR02 | The motors on the carriage should behave as follows:   * Track locomotion motors: should accelerate and decelerate gradually, leading to smooth motion without sudden starts or stops. * Door motors: should open and close the doors slowly, without slamming open or closed. |
| NR03 | The code should be designed in a modular design which allows adding new features and debugging without any major refactoring. |
| NR04 | The system should be able to send an status update every 2 seconds without any delay. |

#### *Table 2.4. Acceptance criteria for the non-functional requirements.*

## **2.3 Interfacing Requirements**

| **ID** | **Description** |
| --- | --- |
| IR01 | The CCP must interface smoothly with the MCP. |
| IR02 | The CCP must interface smoothly with the BR. |
| IR03 | The BR must interface smoothly with the hardware provided by the motion team. |

#### *Table 2.5. Interfacing requirements.*

| **ID** | **Acceptance Criteria** |
| --- | --- |
| IR01 | The CCP is able to receive and send packages to and from the MCP. This is done via WiFi and UDP protocols. |
| IR02 | The CCP is able to pass messages originally from the MCP to the BR. In return, the CCP is able to accept updates from the BR. This is done via WiFi and UDP protocols. |
| IR03 | The BR program accurately operates motors, sensors, and LEDs as directed. |

#### *Table 2.6. Acceptance criteria for the interfacing requirements.*

## 

## **2.4 Requirements Sign-Off**

| **Date** | **Collaborating Teams** | **Collaborating Teams** | **Collaborating Teams** |
| --- | --- | --- | --- |
| 01/08/2024 | T3C1, T3M1 | **Motions**: Johann  **Communications**: Adya, Wasif | The communications team codes all communication elements, while the motions team wires and codes the electronics inside the Bladerunner. |
| 08/08/2024 | T3C1, T3M1 | **Motions**: Johann  **Communications**: Adya, Wasif | The communications team will handle sensor coding to trigger the LED indicators, while the motion team will program the LEDs and sensors within the box. |
| 15/08/2024 | T3C1, T3M1, T3S1, T3S5 | **Motions**: Johann  **Structures**: Evan Miles (T3S1),  Allen Lee (T3S5)  **Communications**: Adya, Wasif | Deadlines have been set for the transfer of the box between teams. The structural team will complete work on the box by week 8, the motion team will then take over until week 10, and the communications team will be responsible for the box until the final deadline. |

#### *Table 2.7. Requirements sign-off by collaborating team.*

# 

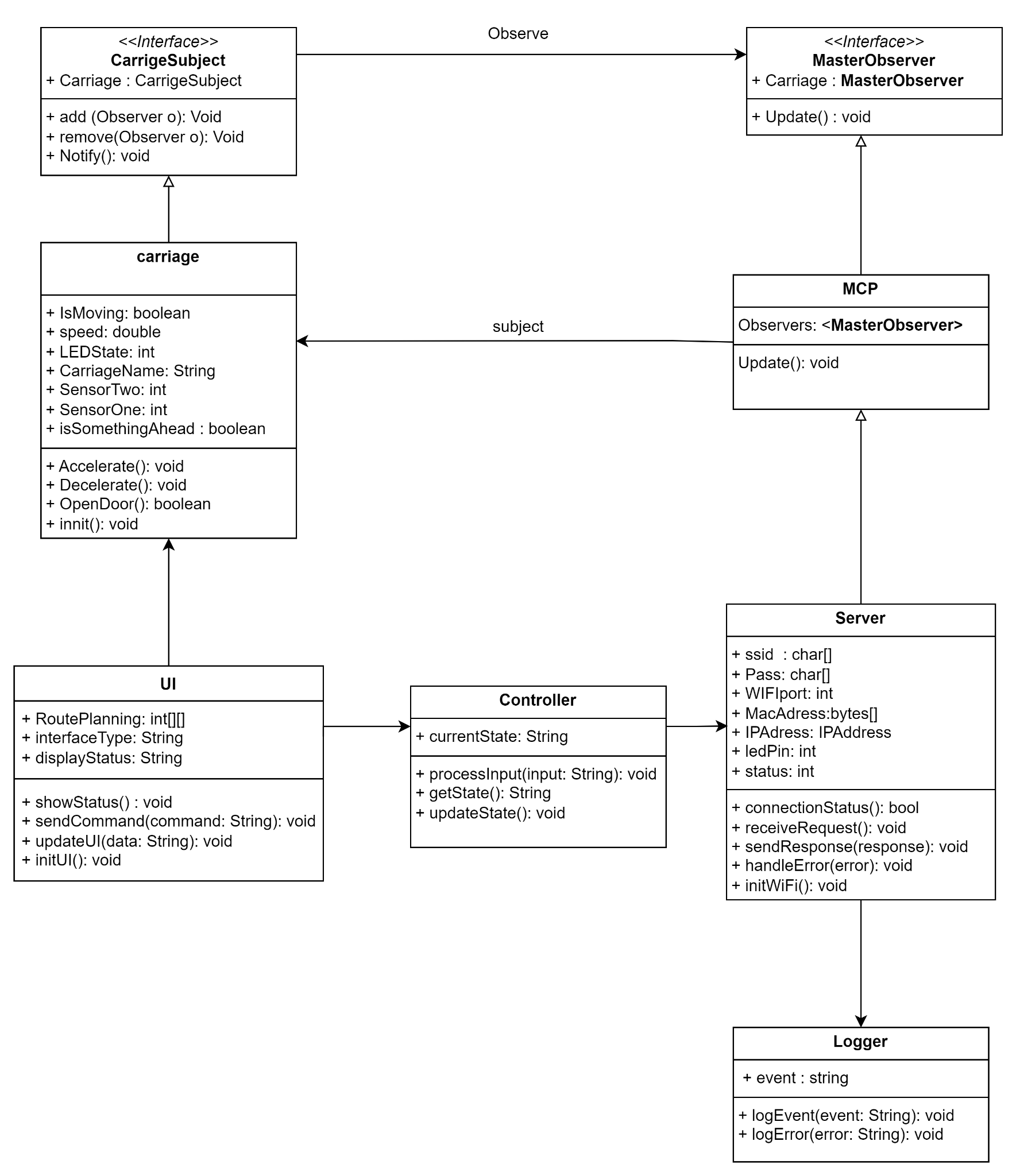
# **3 Conceptual design**

**3.1 Initial Analysis**

Initially, the assumption was that the project meant to include the whole system, which meant the master control program, carriage control program, and the relevant joining software. Due to this, the implementation of the network topology was done since the communication would have to be through internet protocol. Afterwards the clarifications of the project scope was focused only on implementing the CCP. The use of the UDP protocol was chosen for the communication between the MCP and CCP as the priority was MCP and CCP for low-latency and real-time communication. In the initial phase, there is a need for sensor feedback from the ESP32 and hazard detection to be integrated within the CCP design.

#### 

**3.2 Design alternative**

****

*Figure 1.1 Previous UML Diagram*

In the conceptual design phase, there were many alternatives when finalising the design, these alternatives were choices within communication protocols, the structure of the carriage control system and the option of implementing a UI for user control.

For the communication protocol; there were two alternatives, either TCP/IP or using UDP. Comparing them:

* TCP/IP (Transmission Control Protocol/Internet Protocol): This communication protocol would prove to be more reliable considering the connection orientation of this protocol. However this would prove to be a slower transmission since establishing the connection with the handshake process would prove to be higher on the overhead.
* UDP (User Datagram Protocol): This communication protocol is identified as the faster one out of the two communications protocols. Unlike TCP/IP, this protocol is connectionless and does not require the handshake process to establish or stop the connection that makes it a low latency option. Since the connection is not fully established, this protocol does not guarantee the reliability, error correction, ordering or the guarantee of delivery.

In terms of the structure of the code for the CCP, the initial action was to go forward with implementing an Observer Pattern; this would have allowed the program to “observe” changes in other systems such as the MCP and automatically update itself based on the MCP’s changes. However the choice ultimately ended up with going through with a controller class to manage interactions between the CCP, MCP and ESP32.

* Observer Pattern: Implementing this would mean to allow the CCP to automatically react to the changes in other systems by observing them. However this proved to be complex, since it would reduce the flexibility of having to manage multiple observers for just a small system.
* Controller Class: This approach would mean to directly manage the interactions between CCP, MCP and ESP32. It is a simpler design as it allows for better control for communication and reduces the complexities from using the Observer Pattern.

And finally, there were discussions about potentially creating a UI for the Motions team to be able to track and visualise the CCP’s status. This would provide a real-time view of the system and a user-friendly way to monitor the carriage.

**3.3 Comparison of design alternatives**

The decision for the BladeRunner system was to utilise UDP. This was as the transit system required a communication protocol where speed is critical and just enough for the message to be delivered. Although there is security in the fact that the TCP/IP protocol would be reliable and accurate with its communication, it takes a longer time for the message to be sent. The UDP protocol, in this sense, is the suitable option since speed is the priority and the occasional packet loss is bearable.

And for the code structure, the decision was to implement the controller class approach given the complexity and scope of the project. There was no need to scale the project and so the decision went with using this for its simplicity and direct handling of the communication between the systems.

There was consideration to implement a UI to create a user-friendly way to manage this, however considering the amount of work with the time constraints and having to test and deploy. It was left out of our MVP. And given the project’s time constraints, a working CCP was the main priority.

# **4. Detailed Design**

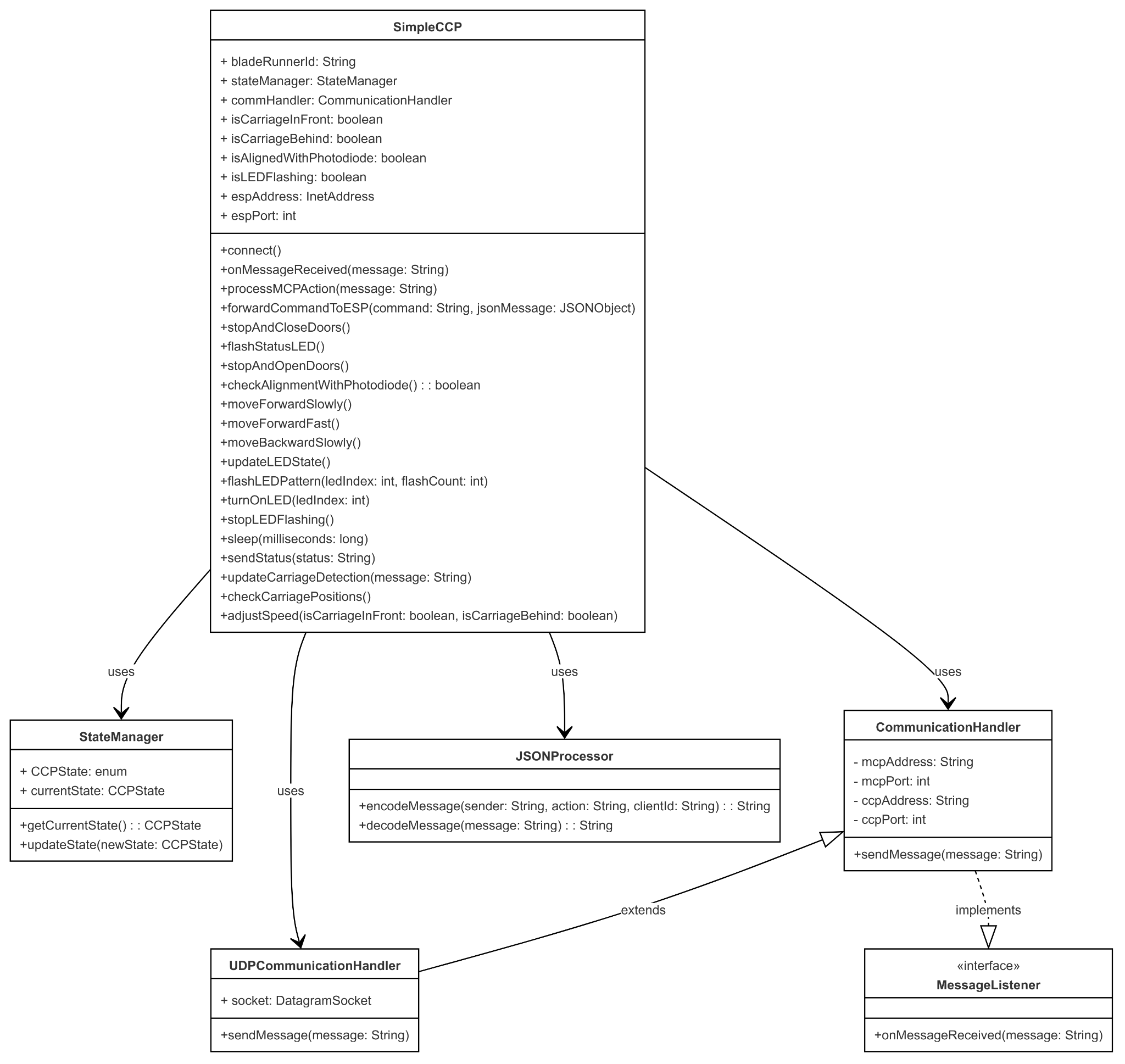
#### *Figure 1.2: High Level implementation of the system*

#### 

| **Class Name** | **Description** |
| --- | --- |
| **Carriage** | Controls the BR physical state and control of the BR, including movement, speed, and sensors. following significant components:   * IsMoving (boolean): This is used to change the state of the Blade Runner's acceleration. * isSomethingAhead (boolean): This is used to detect a possible collision with another Blade Runner. |
| **MCP** | It acts as the central control for observing changes in the carriage and updating other subsystems accordingly. In this instance, the change in the system is the number of carriages and the number of requests for occupying stations. |
| **UI** | Provides a user interface for controlling the Blade Runner, including route planning and status display.   * routePlanning: int[][] is there for all the route stored in coordinates * sendCommand(command: String): void use for movement of BR |
| **Controller** | Manages communication between the UI and other system components, processing user inputs and updating the state. |
| **Server** | Handles network communication via WiFi, managing requests and responses between the system and external interfaces. |
| **Logger** | Responsible for logging system events and errors, aiding in debugging and maintaining system integrity. |

*Table 4.1. Summary of classes in the CCP detailed design.*

**4.1 Detailed design of CCP**

****

*Figure 2.1 CCP UML Diagram*

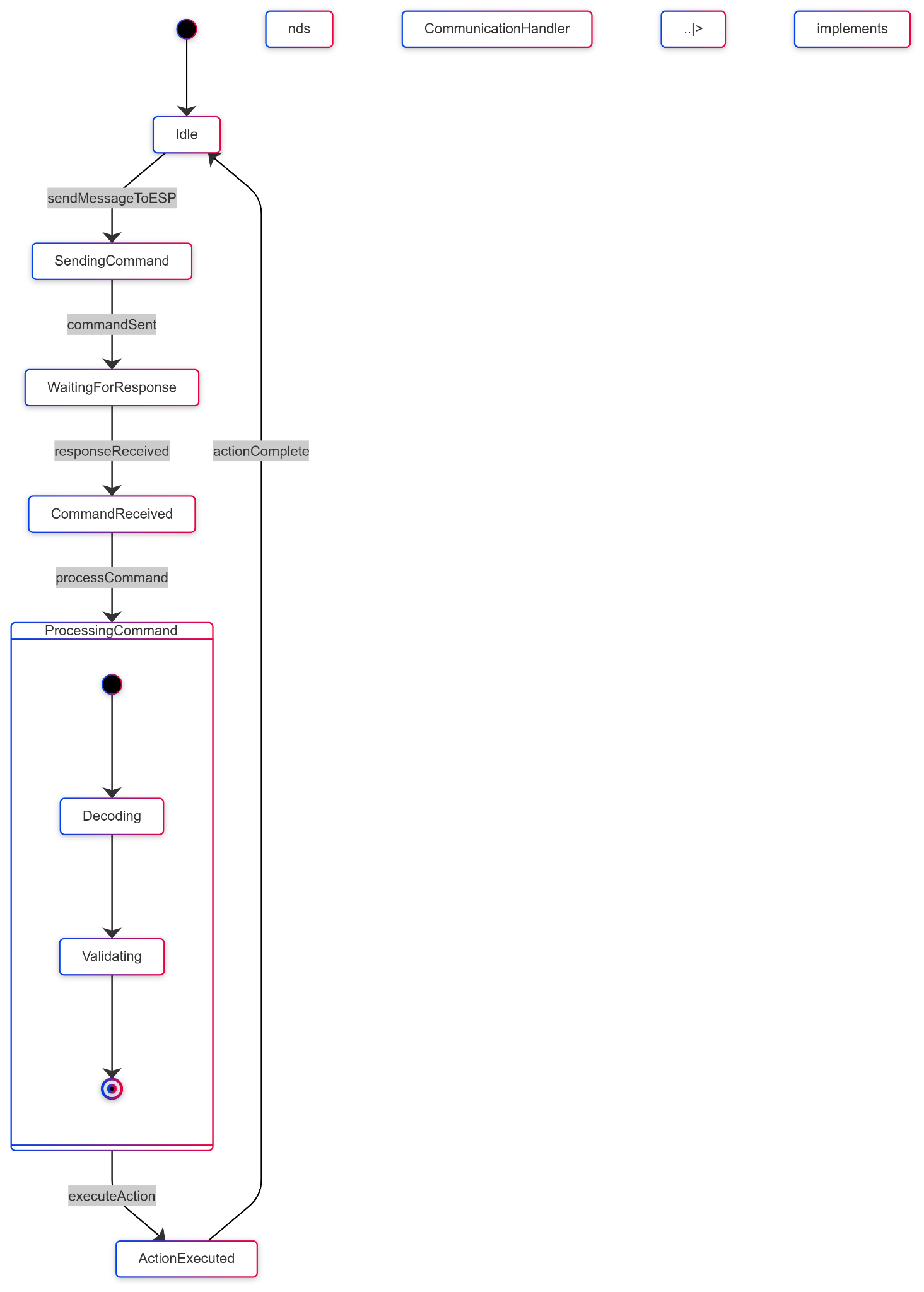
The CCP is responsible for controlling the movement and behaviour of a blade runner. It communicates with the MCP, using the UDP protocol as it listens to the commands from the MCP, processes them and sends them to the BR for execution. As the CCP is responsible for managing communication between MCP, simulating sensors, executing commands and keeping a track of the BR’s state, the code for CCP has been divided into multiple classes: SimpleCCP, StateManager, CommunicationHandler, UDPCommunicationHandler and JSONProcessor. Each of these classes have their own responsibilities and function.

* SimpleCCP: This is the main class so it is responsible to communicate with MCP, manage the state of BR and process incoming commands. The constructor, SimpleCCP, initialises the BR with its ID, MCP’s address and port and the CCP's address and port. Then the connect() sends a CCIN (Control communication IN) message to the MCP to indicate the CCP is attempting to connect. The processMCPAction() method processes the commands sent over to the CCP by the MCP such as opening/closing the door, halting the BR, accelerating/decelerating the carriage. It also has a method named sendStatus(), which sends an status update to MCP, encoded in JSON format.
* UDPCommunication: This class handles the UDP communication between MCP and CCP. It is responsible for listening to the incoming messages and sends messages to MCP. The listenForMessage() method listens for the UDP message and sendMessage() method sends UDP messages to the MCP.
* StateManager: This class is responsible for managing the state of the CCP, whether it be connected, moving, halted, etc. There are multiple enum state that represents different states the CCP can be such as:
  + STOPC
  + STOPO
  + FSLOWC
  + FFASTC
  + RSLOWC
  + ERR
* JSONProcessor: This class is responsible for encoding and decoding JSON messages. The method encodeMessage(), takes a message and encodes it into a JSON object. Then the decodeMessage(), takes the JSON encoded message and returns the message.

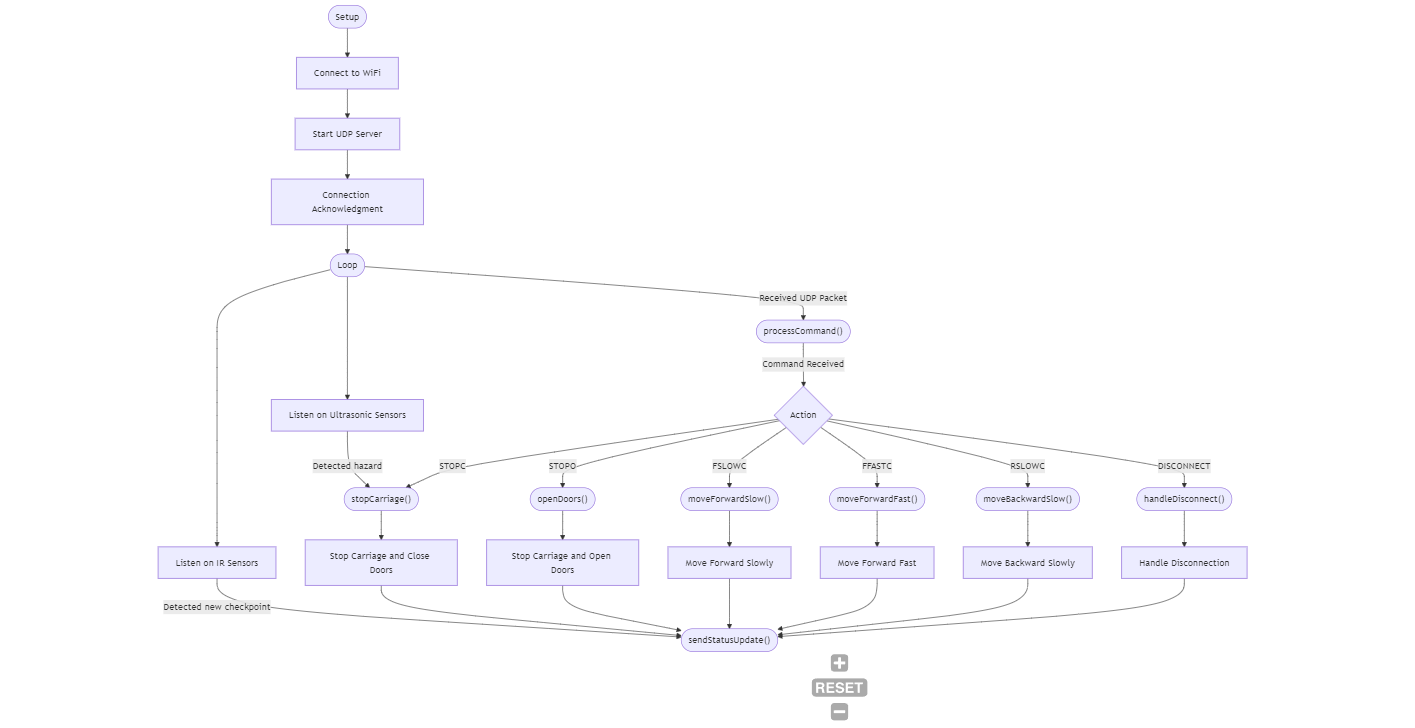
**4.2 State machine diagram**

The state machine diagram demonstrates the flow of states in a system that processes commands, such as sending the commands to an ESP32 and executes actions. The main state for the program is explained below:

* Idle: At first, the system is in the idle state, simply waiting for a command. Only when a command is sent via the sendMessageToESP method does it go on to the next state (SendingCommand).
* SendingCommand: The command being transmitted to ESP32 is represented by this state. The system advances to the following state after the order is sent successfully.
* WaitingForResponse: This state awaits ESP's answer to the previous command. The system changes into a new state upon receiving the answer from the ESP.
* CommandRecevied: The system can transition to a different state once this state verifies that a response has been received.
* ProcessingCommand: This is a composite state as it is further divided into steps: decoding and validation. The system first decodes the command after receiving it and then confirms that it is accurate. It leaves the current state and enters a new one after verifying the instruction.
* ActionExecuted: The system then eventually carries out the command, such as speeding up or shutting the door, after validating it. moves back to the idle state until the system receives another command after the current one has finished.

****

*Figure 2.2 State Machine Diagram*

**4.3 Detailed Design of BR Program**

*Figure 2.3 BR Diagram*

Above is a flowchart summarising the processes in the carriage’s ESP32. In the setup phase, the BR connects to WiFi, starts a UDP server, and initiates a connection to the CCP. In the main loop, the BR listens to its UDP port for commands and to its sensors for information on its surroundings. After any actions performed or checkpoints detected, the BR calls sendStatusUpdate() to update the CCP. In this way, the CCP is always ready to send an accurate heartbeat to the MCP when prompted.

**4.4 Design traceability**

The system’s various requirements are fulfilled by the design as follows:

| **Requirement ID** | **Design Justification** |
| --- | --- |
| FR01 | Fulfilled by the BR calling sendStatusUpdate() whenever changes in state or location are detected. |
| FR02 | Fulfilled via the CCP’s connection to WiFi and UDP socket. |
| FR03 | Fulfilled via BR’s response to MCP commands in states FFASTC, FSLOWC, STOPO, and STOPC. |
| FR04 | Whenever the BR receives a command and executes it, onboard LEDs change to reflect it. Fulfilled via the BR changing onboard LEDs to reflect |
| FR05 | Fulfilled via the BR’s response to STOPO and STOPC commands. |
| FR06 | Fulfilled via the CCP and BR’s checking and handling of errors. |
| FR07 | Fulfilled via onboard ultrasonic sensors that detect obstacles or other hazards in the carriage’s path. |
| FR08 | Once an immediate hazard is detected by the sensors fulfilling FR07, the carriage executes stopCarriage() to enter the STOPC state. |
| NR01 | Fulfilled via the error handling that also fulfils FR06. |
| NR02 | When movement commands are executed, moving parts (drive motor and door motor) exhibit gradual changes in speed instead of sudden large jumps. |
| NR03 | All code for the CCP and MCP are modular, readable, and heavily commented for clarity. |
| NR04 | Fulfilled via UDP protocols. |
| IR01 | Fulfilled via WiFi and UDP protocols and initial connection acknowledgment. |
| IR02 | Fulfilled via WiFi and UDP protocols and initial connection acknowledgment. |
| IR03 | Fulfilled via rigorous code testing with motion team. |

*Table 4.2. Design traceability.*

# **5. Technical Performance Measures (TPMs)**

## **5.1 Summary of TPMs**

The main technical performance measures (TPMs) essential for the dependability and functionality of the system are provided below which have their own individual identifying TPM code and name.

| **TPM Code** | **Summary of Function** |
| --- | --- |
| TPM01 | Carriage Path Request Management |
| TPM02 | MCP Route Management and Execution |
| TPM03 | Real-time Route Processing |
| TPM04 | Emergency Braking System Integration |
| TPM05 | LED status indicator and accuracy |

#### *Table 5.1. TPM Summary.*

#### 

## **5.2 List of TPMs**

The following list of TPMs provided a detailed overview of each TPMs that will be implemented within the system. Each TPMs is linked with a set of functions such as the path for the carriage, route management for carriage, emergency braking system and so on. The table below will overview the main purpose of the TPMs along with their source. It also discusses the method of maintenance of the TPMs depending on their risk level and dependencies.

| **TPM Code** | TPM01 |
| --- | --- |
| **Name of TPM** | Carriage Path Request Management |
| **Purpose** | To ensure that the system efficiently handles multiple path requests from carriages, enabling optimal route planning and management. |
| **Source** | The system shall allow carriages to request different paths for efficient route planning and management. |
| **What should be measured?** | 1. The ability of the system to handle multiple path requests. 2. The accuracy of the provided paths 3. The system's response time to path requests and updates. 4. The system's performance underload |
| **How should it be measured?** | Functionality test: Verify that the system allows at least three different path requests within a session.  Performance test: check the accuracy of the paths, ensuring they stay within a 5% deviation from the optimal route.  Load test: Test the system's ability to handle at least 100 simultaneous path requests without degrading performance. |
| **How often should it be measured** | Regularly during system updates and maintenance |
| **Measure of Success** | The system allows and correctly processes multiple path requests with accurate results within the specified time frames. |
| **Measure of Failure** | The system fails to handle the minimum number of path requests.  Response times exceed the specified limits. |
| **Possible Causes of Failure** | Insufficient processing power of system resources  Bugs or errors in the pathfinding algorithm. |

#### *Table 5.2. Carriage Path Request Management*

| **TPM Code** | TPM02 |
| --- | --- |
| **Name of TPM** | MCP Route Management and Execution |
| **Purpose** | To ensure that the MCP efficiently manages and executes route instructions for the CCP, ensuring seamless operation and coordination. |
| **Source** | The MCP shall manage and carry out routes for the CCP. |
| **What should be measured?** | The MCP must be able to assign, manage, and execute routes for the CCP's.  The accuracy and timeliness of route execution by the CCP under MCP’s control  The MCP’s handling of multiple routes for different CCP’s. |
| **How should it be measured?** | Performing function tests to validate that the MCP can correctly assign and manage routes  Performance test to measure the time taken by the MCP  Reliability test to evaluate the MCP’s ability to handle multiple CCPs |
| **How often should it be measured** | Can be measured during the initial system integration phase |
| **Measure of Success** | The MCP successfully assigns routes to the CP |
| **Measure of Failure** | If the MCP fails to assign or manage routes effectively |
| **Possible Causes of Failure** | Communication problems between the CCP and MCP  Software bugs in the MCP |

#### *Table 5.3. MCP Route Management and Execution*

| **TPM Code** | TPM03 |
| --- | --- |
| **Name of TPM** | Real-time route processing |
| **Purpose** | Ensures that the MCP processes route requests and updates within the time constraints. |
| **Source** | System performance requirements specifically focusing on user interaction and route planning |
| **What should be measured?** | Processing time for route requests and updates |
| **How should it be measured?** | By monitoring the time elapsed from the moment a route request or update |
| **How often should it be measured** | Should be monitored continuously depending on how frequently this system is used. |
| **Measure of Success** | The MCP must process 95% of route requests and updates within 2 seconds so that there is consistency even when it is busy. |
| **Measure of Failure** | The MCP fails to process more than 5% of route requests and has updates that come in more than 2 seconds or shows inaccuracies during busy times. |
| **Possible Causes of Failure** | If a high volume of requests occurs, network latency causes delays and software bugs, including code errors. |

#### *Table 5.4. Real-time route processing*

| **TPM Code** | TPM04 |
| --- | --- |
| **Name of TPM** | Emergency Braking System Integration |
| **Purpose** | To ensure that the CCP braking system is fully integrated and can be overridden in critical situations. |
| **Source** | The CCP emergency braking system must have the ability to be taken over in times of crisis. |
| **What should be measured?** | Whether or not the system is responsive to being overridden by other subsystems |
| **How should it be measured?** | Using simulation tests to test the response time and whether or not it is effective or not. |
| **How often should it be measured** | Should be monitored continuously depending on how frequently this system is used. |
| **Measure of Success** | The system must be responsive within 1 second of initiation of using it. |
| **Measure of Failure** | The system fails to override within 1 second of activation. |
| **Possible Causes of Failure** | * Communication issues between subsystems * Software bugs * Hardware malfunctions within the braking system |

#### *Table 5.5. Emergency Braking System Integration*

#### 

| **TPM Code** | TPM05 |
| --- | --- |
| **Name of TPM** | LED Status Indicator Accuracy |
| **Purpose** | To ensure that LED indicators on the carriage accurately reflect the actual status of the system. |
| **Source** | Interface LED indicators with motion and structural modules to provide visual feedback. |
| **What should be measured?** | The accuracy and visibility of the LED during situations |
| **How should it be measured?** | Simulation tests to see whether the LED indicators change status promptly and are visible |
| **How often should it be measured** | Should be monitored continuously depending on how frequently this system is used. |
| **Measure of Success** | LED indicators accurately and promptly reflect system status changes. |
| **Measure of Failure** | LED indicators show failed accuracy and are not visible. |
| **Possible Causes of Failure** | * Delays in communication from the control system * Hardware issues in the LED |

#### *Table 5.6. LED Status Indicator Accuracy*

# 

# **6. Design Structure Matrix (DSM)**

Design Structure Matrix (DSM) is an effective technique used to visualise the relations between different components in a system and their interaction with one another. The following DSM represents the relation between the different TPMs that were disscussed.

| **TPMS** | **TPM01** | **TPM02** | **TPM03** | **TPM04** | **TPM05** |
| --- | --- | --- | --- | --- | --- |
| **TPM01** | X | X | X |  |  |
| **TPM02** |  | X | X |  |  |
| **TPM03** |  |  | X | X |  |
| **TPM04** |  | X | X | X |  |
| **TPM05** |  |  |  | X | X |

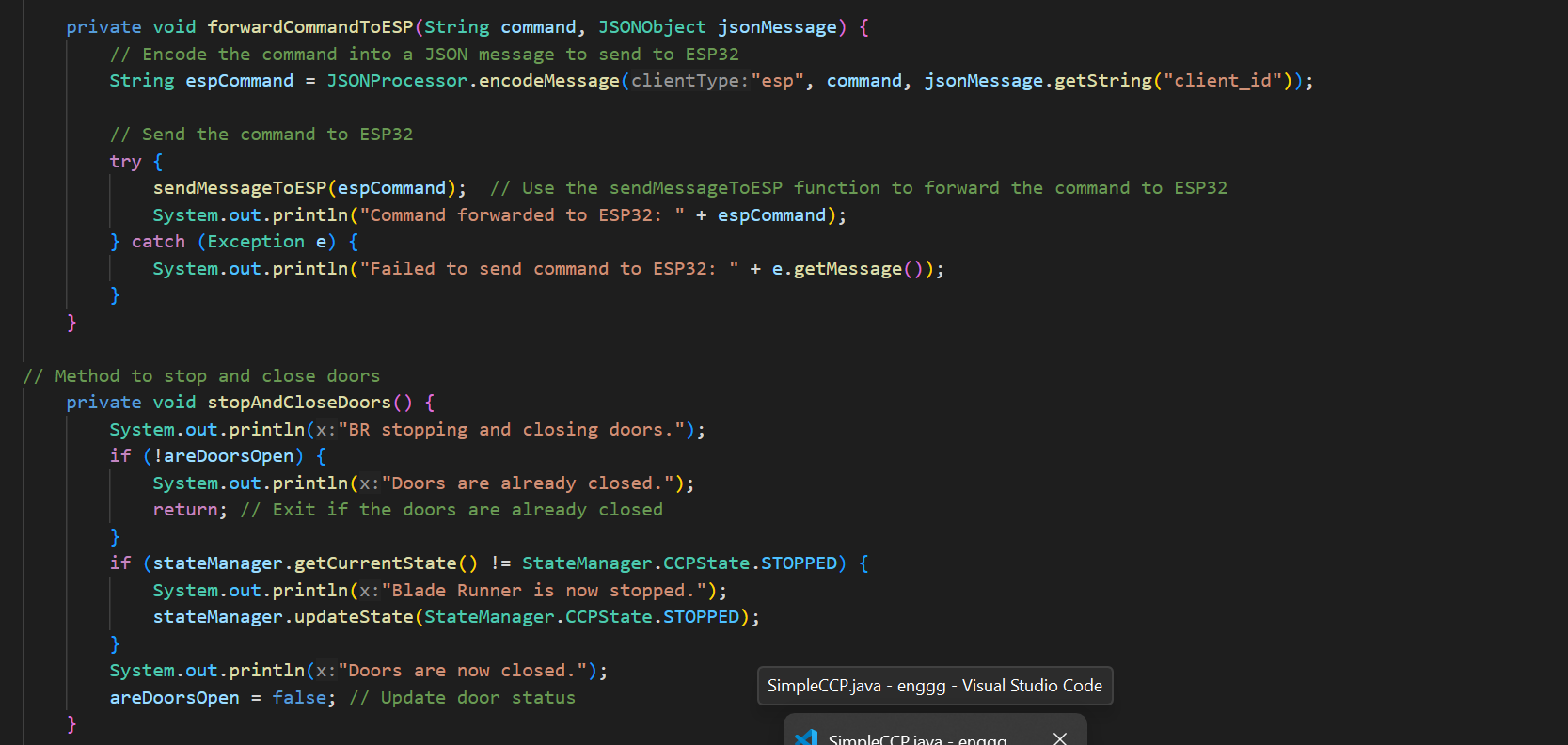
#### *Table 6.1. DSM.*

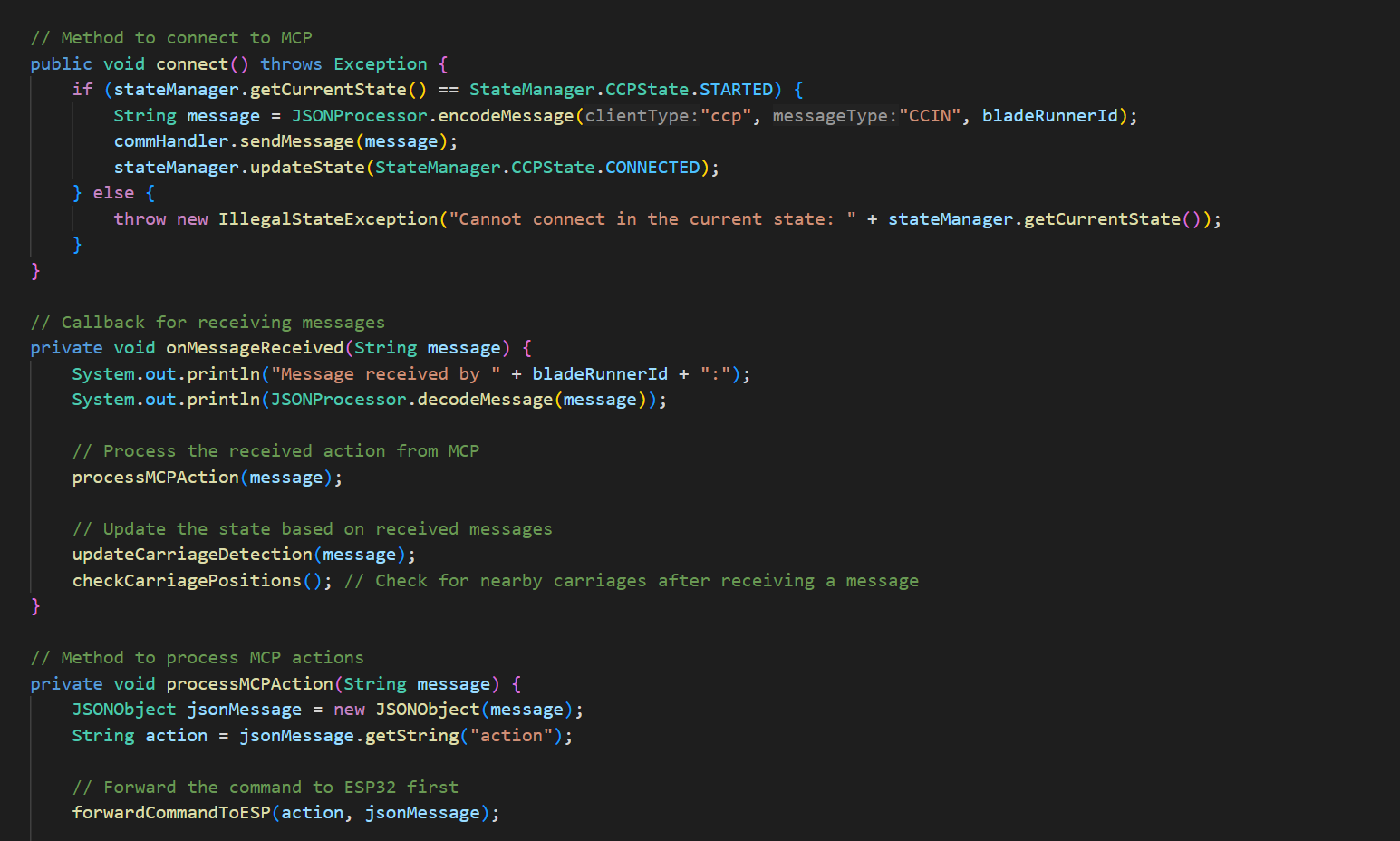
**7. Summary**

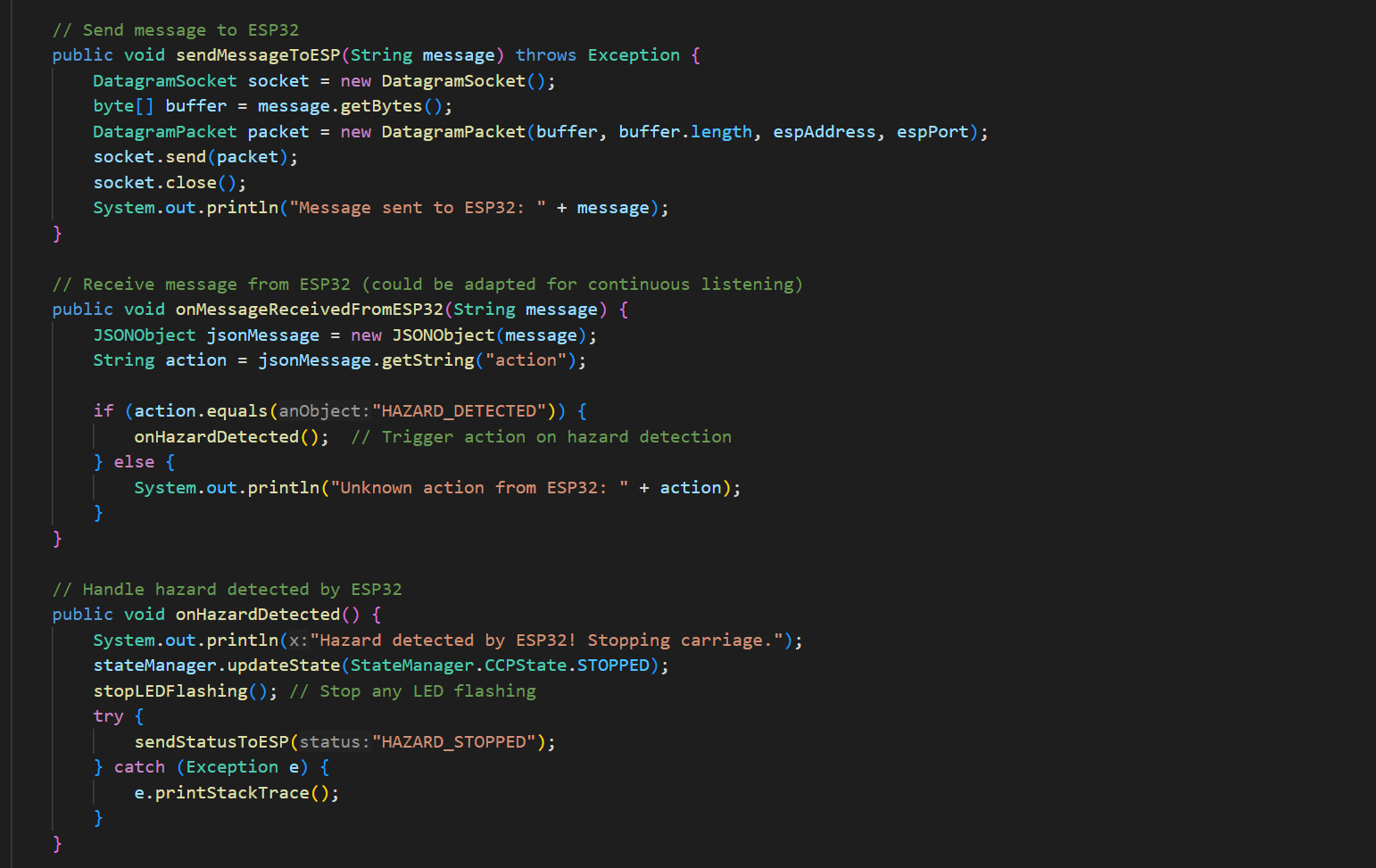
The communication system designed by Team T3\_C1 for the Blade Runner 2024 project manages a monorail system controlled by the Master Control Program (MCP), enabling real-time control and feedback between the MCP and several carriages (Blade Runners) moving on a circular track. At the core is the Carriage Control Program (CCP), which processes MCP commands such as speed, direction, and door operations, while also sending real-time updates, including sensor data like obstacle detection, back to the MCP. The system allows the MCP to communicate with multiple carriages simultaneously with minimal delay, ensuring reliable performance. UDP (User Datagram Protocol) was chosen for its low latency, making it ideal for real-time control, despite the higher reliability of TCP/IP. The CCP's modular design includes a controller class, enabling flexibility and easy updates. Technical Performance Measures (TPMs) monitor critical functions like route management and emergency braking, ensuring the system meets its safety and performance goals. Overall, the design prioritises scalability, reliability, and efficient control of the Blade Runner system

**Appendices**

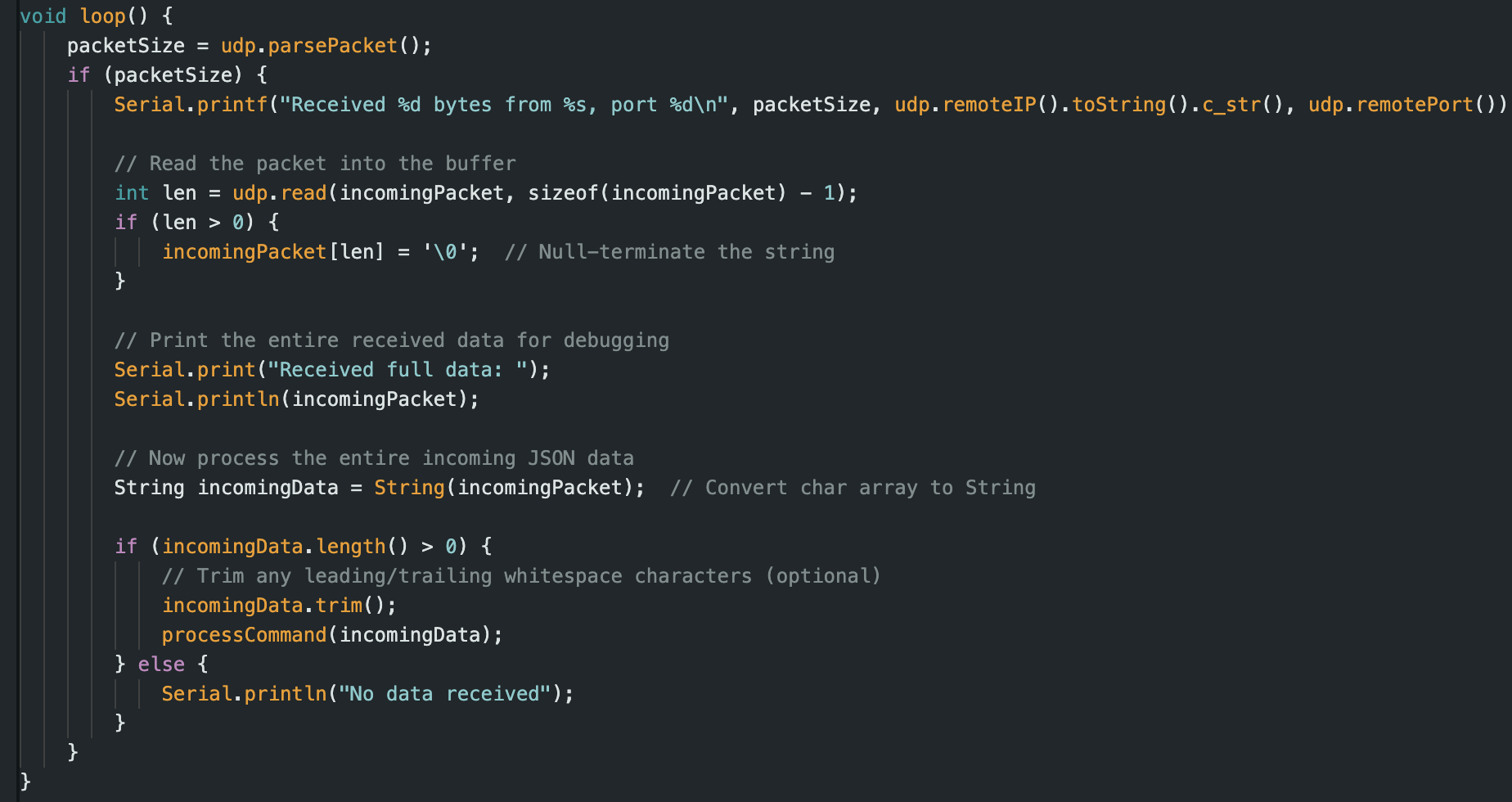
#### **Appendix A:** Snippets of CCP code, written in Java.







#### **Appendix B:** Snippets of BR code written for the onboard ESP32, written in C for Arduino ID





**Gen AI Statement**

Our team, T3\_C1, used generative AI tools during the creation of this design document. The specific use cases and prompts are outlined below:

* **Proofreading and Editing:** We used GEN AI tools to proofread certain sections of the document for grammatical accuracy and clarity. The specific prompt used was: “Check any grammar or spelling errors in the text and correct them.” The GEN AI tool suggested minor editorial changes, which were reviewed and selectively implemented to improve sentence structure and readability.
* **Summary Generation Assistance:** GEN AI was used to assist with drafting portions of the summary in Section 7: "Summary". The following prompt was used to generate the content: "Generate a concise summary of the material provided." This was applied after all content was finished save for the summary itself.
* **Code Documentation Assistance:** Some of the comments in the sections "Design Alternative" (Section 3.2) and "Detailed Design of CCP" (Section 4.1) were inspired by AI-generated content. The prompt used was: "Explain the functionality of a CCP system communicating with an MCP and ESP32 using UDP and state machine logic." The output was used as the basis for the content in the paper. Beforehand, the generated content was reviewed by all team members, and any technical information provided by the AI was cross-verified and adjusted for accuracy.