Gheorghe Asachi Technical University of Iaşi
Faculty of Automatic Control and Computer Engineering
Department of Automatic Control and Applied Informatics
Study Program: Systems Engineering

BACHELOR THESIS

A teleoperated driving system via 5G network

Author
Taradaciuc Nicolae

Advisor
Assistant Prof. PhD eng. Carlos-Mihai Pascal

Iași, 2024

DECLARAŢIE DE ASUMARE A AUTENTICITĂŢII PROIECTULUI DE DIPLOMĂ

Subsemnatul TARADACIUC NICOLAE, legitimat cu CI seria XT nr. 827903, CNP 5000310071973, autorul lucrării A TELEOPERATED DRIVING SYSTEM VIA 5G NETWORK, elaborată în vederea susţinerii examenului de finalizare a studiilor de licență, programul de studii AUTOMATICĂ ŞI INFORMATICĂ APLICATĂ organizat de către Facultatea de Automatică şi Calculatoare din cadrul Universităţii Tehnice „Gheorghe Asachi" din Iaşi, sesiunea IULIE a anului universitar 2023-2024, luând în considerare conţinutul Art. 34 din Codul de etică universitară al Universităţii Tehnice „Gheorghe Asachi" din Iaşi (Manualul Procedurilor, UTI.POM.02 - Funcţionarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române (legea 8/1996) şi a convenţiilor internaţionale privind drepturile de autor.

Data
29.06.2024

Semnătura

Contents

List of Figures

Notations

Abbreviation	Complete Name
SBC	Single Board Computer
CAD	Computer-aided design
PWM	Pulse-Width Modulation
1G - 5G	The first - fifth generation of wireless cellular technology
CSI	Camera Serial Interface
GPIO	General-purpose input/output
GND	Ground
SIM	Subscriber Identity/Identification Module
NoIR	No Infrared filter
GNSS	Global navigation satellite system
DL	Download
UL	Upload
LED	Light Emitting Diode
DC	Direct Current
OS	Operating System
AT Commands	Attention Commands
ID	Identifier
USB	Universal Serial Bus
IP	Internet Protocol
3D	Three dimensional
2D	Two dimensional
RAN	Radio Access Networks
FPS	Frames Per Second
ToF	Time of Flight
LiDAR	Light Detection and Ranging
VR	Virtual Reality
ADAS	Advanced Driver Assistance Systems

1 Introduction

This paper introduces a versatile teleoperated driving system that uses 5G Technology for control and user interactions. Equipped with the 5G Module and a Single Board Computer, it seamlessly operates via an Xbox Series X Controller providing an intuitive and user-friendly interface. The project aims to establish flawless interaction between the car and the user through the controller, utilizing the high-speed transmission capabilities of the 5G technology for consistent communication and live feed transmission of the car's surroundings.

This thesis will cover the integration of the 5G Module with the Raspberry Pi board, the process of reading controller inputs, data transfer and live streaming through 5G, as well as the programming of the DC motors and servo motors. The Raspberry Pi 4 Model B serves as the Single Board Computer, facilitating communication with the Xbox Controller, Raspberry Pi Camera Board that is capable of being repositioned as needed and the car's motor control system which includes components of both steering and propulsion.

The project will be presented in a developmental manner, showcasing the entire process throughout the article. It will detail the successful outcomes of the project alongside the lessons learned from the challenges encountered during the process.

1.1 Context

In recent years, the intersection of embedded systems and gaming technology has given rise to innovative and engaging projects that captivate both enthusiasts and learners alike. This project explores the integration of a 5G-controlled smart car, driven by Single Board Computer, with a focus on user-friendly interaction through a Xbox Series X controller. The unique aspect of this project is the intermediary role played by a SIM8200EA-M2 5G Module, connecting the wireless Xbox controller, via 5G, to the Raspberry Pi 4 Board.

In the domain of connectivity, the 5G technology represents a revolutionary leap forward in wireless communication, enabling devices to exchange data without regard to distance. In this project, the wireless link that connects the Xbox Controller and the vehicle is completely done through a 5G network. For a visual representation, refer to Fig. 1.1, showcasing the teleoperated driving system, its components, and the Xbox Controller.

The project is designed to provide a user-friendly experience. The smart car is a small recreation of a real automobile. The analog acceleration, responding to the degree of trigger pressure, simulates the control everyone will expect in a real vehicle. Similarly, the steering mechanism, executed through the left stick of the controller, allows users to decide the degree of turn, offering a lifelike simulation of steering. This project not only serves as an entertaining and educational purpose, but an actual small-scale introduction to the principles of a real car.

Going further, several subjects like the Overview of Components, Hardware Setup, 5G Module Integration, Software Architecture will provide a detailed exploration of the whole project, explaining the integration of various components and the whole process dedicated to the hardware and software part of the project. This will be followed by a brief presentation of the

Figure 1.1: The developed teleoperated driving system via 5G network

challenges encountered during development with the solutions that overcame them. Additionally, a few planned ideas for the future will showcase the true nature and the whole perspective of the project itself, providing an overall understanding of the project, marking the important steps that were made during the development.

1.2 Understanding the 5G Technology

1.2.1 Overview

The 5G Technology represents the fifth generation of cellular technology, engineered to boost speed, minimize latency, and enhance the flexibility of wireless services. With a potential peak speed of 20Gbps, 5G significantly surpasses 4G, which peaks at only 1 Gbps. This improvement in speed and reduced latency, enhance the performance of various applications such as autonomous vehicles, online gaming, videoconferencing and much more.

Wireless communication has been showing continuous advancements in terms of data capacity, speed, frequency, technology, and latency. These advancements have been categorized into Mobile Wireless Generations. The First Generation (1G) relied on analog technology with network speeds of around 2.4kbps. The Second Generation (2G) transitioned to digital technology, enabling features like text messaging and encrypted communication. The Third Generation (3G) introduced significant improvements in speed and capacity for data transmission, as well as support for multimedia services. The Fourth Generation (4G) integrated internet services to offer wireless mobile internet with enhanced bandwidth and reduced costs[1].

The Fifth Generation (5G) is known as a very big step forward, fulfilling all the requirements for modern communication needs. It offers ultra-fast connectivity speeds and the capability to support a vast number of devices concurrently. This generation brings significant innovations such as software-driven networks, virtualized network functions, and seamless roaming between various wireless access points without requiring user intervention[1].

1.2.2 Benefits

The 5G technology has a lot of advantages that are relevant for smart car applications and one of the primary benefits is the high data transfer rates supported by 5G, which allow real-time communication and data processing, which is crucial for applications like live video streaming from the car or sending live commands to control the car.

Another essential advantage of 5G is its low latency, because the technology drastically reduces latency, offering users near-instantaneous data transmission. This is an very important factor for remote control and real-time response smart car applications.

Additionally, 5G offers high bandwidth, which has the ability to connect a massive number of devices that can send and receive data at any time, making it ideal for smart transportation systems [2].

1.2.3 Relevance in this project

In this project, the 5G technology plays a vital role in facilitating the communication between the smart car and the user. By integrating the SIM8200EA-M2 5G Module with the Raspberry Pi, the project uses the high-speed, low-latency capabilities of 5G to provide real-time control from the laptop to the raspberry and live video feed transmission. This ensures that the car responds correctly to the user commands and offers a smooth, immersive experience.

1.3 Understanding the Teleoperated Driving System Technology

1.3.1 Overview

A teleoperated driving systems refers to a vehicle control system where the driver is not physically present inside the vehicle, but operates it remotely. This system uses advanced communication technologies, like 5G, to transmit control commands from a remote operator to the vehicle and send back real-time data, such as video feed and sensor reading from the vehicle to the operator.

1.3.2 Benefits

The benefits of a teleoperated driving system are numerous and one of the primary ones is the improved safety that resulted from the fact that it allows operators to control vehicles in hazardous environments without being exposed to danger.

Additionally, teleoperated driving systems offers significant accessibility advantages, providing mobility solutions for individuals who cannot drive due to physical limitations.

Furthermore, another benefit is that these systems offer improved efficiency, enabling centralized control of multiple vehicles, which can be advantageous for logistics and transportation industries[3].

1.3.3 Relevance in this project

In this project, the teleoperated driving system is implemented using 5G technology to use the low-latency and high-speed communication between the operator (using the Xbox Series X controller) and the smart car.

This setup allows for real-time control and immediate feedback, making it possible to operate the car as if the user was inside it. The live video feed from the car's camera offers the operator the possibility to check his environment with ease and be precise and safe while navigating.

1.4 Problem statement

The current remote-controlled cars systems often face limitations in range, responsiveness, and user interaction. The challenge is to develop a smart car system that uses 5G technology to overcome these limitations.

By integrating 5G with a user-friendly interface, such as an Xbox Series X controller, the project aims to upgrade the remove operation experience, making it more responsive, realistic, and engaging. Additionally, there is a need to simulate real-world driving, including precise steering and variable acceleration, providing a more practical experience.

1.5 Objectives

These objectives outline the main checkpoints of the project and will be thoroughly explained and detailed thought the document.

1.5.1 The Creation and Assembly Process of the Chassis

This objective involves detailing the design and fabrication of the smart car's chassis. It includes the 3D printing, starting from creating a functional chassis for mounting various components to the actual printing of the parts. Additionally, it covers the assembly process, where the printed parts are put together to create a functional chassis capable of mounting various components.

The design phase includes using a Computer-aided design (CAD) software to modify the layout to accommodate the Raspberry Pi, motors, 5G module and other essential parts. After printing, the parts were assembled using screws, nuts and bolts, resulting in an operational chassis that forms the backbone of the smart car. A lot more details regarding the creation and assembply process of the chassis can be found in Chapters 2 and 3.

1.5.2 Component Selection and Mounting

This objective explains the criteria for choosing various components used in the smart car, such as the Raspberry Pi 4 Model B, the SIM8200EA-M2 5G Module, motors, and the camera. It also refers to the process of mounting these components onto the chassis, making sure to obtain a secure and efficient placement to facilitate optimal performance and ease of maintenance.

Each component was selected based on specific criteria that match with the project's goals. for example, the Raspberry Pi 4 Model B was chosen for its powerful processing capabilities and compatibility with the 5G module. The Mounting process involved placing each component on the chassis in such a manner that it helped to optimize the space used and balance the weight distribution, while also ensuring ease of access to the main components. A lot more details regarding the components selection and their mounting can be found in Chapters 2 and 3.

1.5.3 Configuration of the 5G Module

The focus is on setting up the SIM8200EA-M2 5G Module, making sure that there is a stable internet connection via 5G communication. This objective includes installing the necessary drivers, configuring the module settings and verifying the connectivity.

The configuration process also involves troubleshooting any issues that arise during setup to ensure a reliable connection. A lot more details regarding connectivity tests of the 5G Module under different conditions and environments are presented in Chapters 4 and 7.

1.5.4 Server-Client Communication Setup

This objective involves creating a Server-Client Communication setup model by programming the Raspberry Pi as a server. It is responsible for handling commands and data from the Xbox Controller and laptop. The setup ensures that the server can efficiently receive, process, and execute commands sent from the client, facilitating real-time control and data transmission through a 5G network.

Additionally, it includes setting up the communication protocols and ensuring secure data transfer between the server and client, implementing error-handling mechanisms and optimizing data packets for minimal latency, aiming to maintain a smooth and responsive interaction between the client and the server. More details regarding the Server-Client Communication Setup can be found in Chapters 5 and 6.

1.5.5 Integration of the Xbox Controller

The focus of this objective is to program the Xbox controller's input mappings to suit the control requirements of the smart car and making sure that all inputs are interpreted and executed correctly. It involves programming the controller's buttons and joysticks to fit the car's functions, ensuring responsive and accurate control.

Additionally, it includes calibrating the sensitivity of the analog sticks for precise movement and a smoother acceleration and braking system. A lot more details regarding the Xbox Series X controller and its integration can be found in Chapters 2 and 5.

1.5.6 Live Feed Transmission Setup

This objective is focusing on the integration of the camera module and it also includes programming the camera to provide live feed transmission. It involves setting up the camera, configuring the software to capture and stream video, making sure to have a live, stable feed with a good quality.

Moreover, it includes optimizing the video streaming settings to balance quality and latency, crucial for real-time streaming system incorporated in a teleoperated smart car. A lot more details regarding the Camera and the Live Feed Transmission Setup are presented in Chapters 2 and 5.

2 Overview of Components

This chapter has the objective of providing an overview of the components that were used during the development of the smart car and the process of selecting them based on their suitability for this project. It will detail the criteria for choosing each components and explain how they contribute to the overall functionality and performance of the smart car.

2.1 Printed Chassis

The 3D – printed chassis provides a customized and a structure to attach all the components to. Four bearings were used to ensure smooth movement for the servo direction and wheels. The chassis was selected for its simplicity in assembly, offering a customized structure that functions as a platform[4].

Over time, the chassis was modified to accommodate numerous changes made during the project, necessitating many custom modifications. These modifications involved repeatedly changing and remodeling the chassis design. Most of the parts of the chassis can be seen in Fig. 2.1. The modular nature of the 3D-printed components allowed for quick replacements due to improvements or replacements. More details regarding the Printed Chassis and its creation will be provided in Chapter 3.

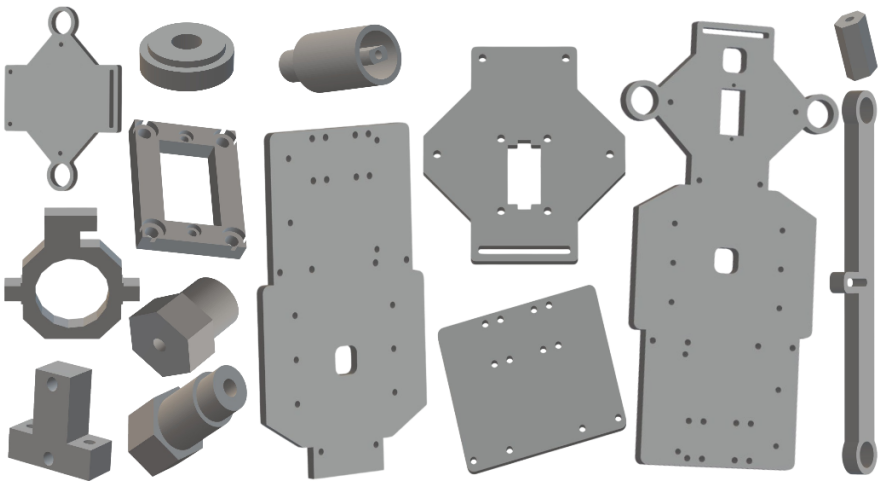


Figure 2.1: The 3D Printing of the Chassis

2.2 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B features a quad-core Cortex-A72 (ARM v8) 64-bit System on Chip running at a clock speed of 1.8GHz. It is equipped with 4GB of LPDDR4-3200 SDRAM, providing ample memory for various applications and projects. This powerful processing unit and memory configuration make the Raspberry Pi 4 an ideal choice for managing the communication between components and rules the overall operation of the smart car.

For a visual representation, the board is displayed in Fig. 2.2 It serves as the central brain of the car and by integrating it with Python programming, resulted in much more flexibility and control over the whole project. The Raspberry Pi was chosen was chosen for this project because it could provide sufficient power for the project and because it is compatible with the 5G module. Additionally, its user-friendly and extensive documentation makes it accessible for beginners and experienced developers[5].

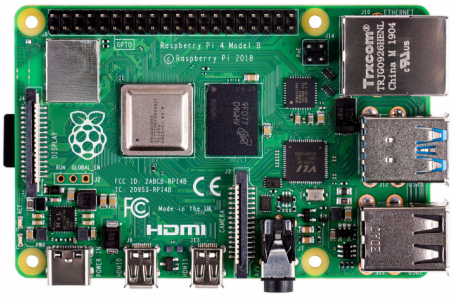


Figure 2.2: Raspberry Pi 4 Model B

2.3 DC Motors and Motor Driver

The smart car's movement is powered by two DC motors and their speed is precisely controlled by the L298N motor driver, both being displayed in Fig. 2.3 for a visual representation. The L289N module is equipped with a dual H-bridge, allowing for speed control through PWM signals and the direction of rotation of the two DC motors. These motors were chosen because they are energy-efficient while they manage to offer plenty of power to move the car's weight.

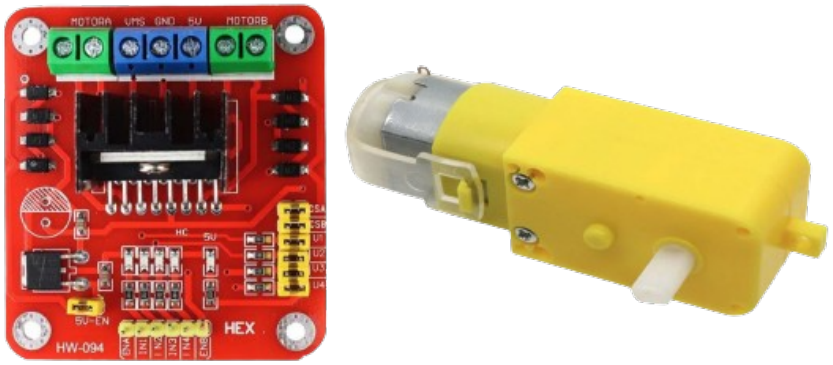


Figure 2.3: L289N Motor Driver and DC Motor 3-6V.

The L298N motor driver module is a component that is controlling the DC Motors of the smart car. It allows the Raspberry Pi to send signals that control the speed and direction of the motors. In the context of motor control, a signal is an electrical pulse sent from the Raspberry Pi to the motor driver. These signals can either be digital (ON / OFF) or analog (Varying levels).

The Pulse Width Modulation (PWM) is a technique that allows users to simulate an analog signal using a digital one, that being the reason why it's used for precise motor control. It is also used to control the power delivered to electrical devices. In this method, a digital signal switches between ON and OFF states at a high frequency. The proportion of time the signal is ON versus OFF is known as the duty cycle which is a percentage of one period in which a signal is active. A higher duty cycle corresponds to a higher power level, meaning the motor runs faster.

The specific technical characteristics of these motors include an operating voltage range of 3-6V, a torque of 0.8 kg*cm, and rotational speeds of 125rpm at 3V, 200rpm at 5V, and 230rpm at 6V. The current consumption for these motors is 60mA at 3V, 100mA at 5V, and 120mA at 6V. These specifications make these motors ideal for this type of project, providing the necessary torque and speed while maintaining low power consumption [6].

2.4 Servo Motors

The MG90S servo motor, displayed in Fig. 2.4, is capable of rotating up to 120 degrees and is controlled by a PWM signal. Its precision and speed make it ideal for steering, making sure that the car would be able to execute precise, fast, and accurate movements, contributing to the overall responsiveness of the vehicle. Additionally, a second servo motor is utilized to control the movement of the camera, allowing for precise observation of the surroundings without the need to reposition the entire vehicle[6].

The MG90S servo motor is very light and powerful, fitting perfectly for this type of project. The motor can rotate up to 180 degrees (90 degrees in each direction). The technical characteristics of this servo motor include a weight of 13.4g, it operates with speeds of 0.1 s/60 degrees at 4.8V and 0.08s/60 degrees at 6V. The power supply ranges from 4.8V to 6V, with a pulse width of 5 µs and an operating temperature range of 0ºC to 55ºC.

Figure 2.4: Servo Motor MG90S

2.5 Xbox Series X Wireless Controller

The Xbox Series X Controller, showcased in Fig. 2.5, known for its ergonomic design and advanced features, is a gamepad developed by Microsoft for the Xbox gaming console series. In this project, the Xbox Series X Controller is chosen due to its intuitive layout, having two adaptive triggers, a precise D-pad, two thumbsticks, and a multitude of buttons, offering extensive input options.

With its versatile interface, the Controller proves to be an ideal choice for fulfilling the project's requirements, providing seamless control over the smart system. Furthermore, its layout ensures compatibility for future developments, eliminating the need for controller replacements.

In addition to its ergonomic design, the Xbox Series X controller features advanced haptic feedback and adaptive triggers. These adaptive triggers can vary the resistance felt by the user, providing a more immersive and responsive control experience. This is particularly useful in applications requiring precise control inputs, such as acceleration and steering a smart car.

Figure 2.5: Xbox Series X Wireless Controller

The Xbox Series X Controller features two thumbsticks, located on the upper left and lower right of the controller's face. In the context of a teleoperated driving system, these thumbsticks are very important because they can be used to provide precise input for controlling the smart car's movements. The left thumbstick is currently used for steering, allowing smooth and responsive directional control, while the right thumbstick can be programmed for camera adjustments or any other additional functionalities.

Moreover, the controller's power management is optimized for extended use. It operates on a pair of AA batteries or a rechargeable battery pack, providing flexibility and convenience. The inclusion of a USB-C port also allows for direct wired connections if needed, ensuring uninterrupted operation during critical testing phases. This combination of features makes the Xbox Series X Controller a suitable input device for the teleoperated driving system.

2.6 5G Module and 5G HAT for Raspberry Pi

The 5G/4G/3G Raspberry Pi communication HAT introduces advanced connectivity features with its SIMCom 5G module, the SIM8200EA-M2, which is presented in Fig. 2.6 and is offering impressive data rates of up to 2.4 Gbps (DL) / 500 Mbps (UL). Powered by the Snapdragon X55, this 5G Module provides versatile support for 5G, 4G, and 3G networks, along with multi-mode, multi-band capabilities [7].

The USB 3.1 port servers multiple purposes including testing AT commands, sending messages, cloud communication, making phone calls, and obtaining GNSS positioning data. It is equipped with a SIM card slot and two LED indicators to monitor the working status of the HAT. Utilizing this module, the entire car seamlessly connects to the 5G network, enabling communication between the controller and the vehicle without any distance limitations. The HAT also features an audio jack and decoder, allowing audio operations like phone calls [7].

The base board includes a standard M2 connector, enabling compatibility with various 4G or 5G communication modules. The 5G HAT is designed for easy integration with the Raspberry Pi, supporting Linux and any other Raspian operating systems (OS). Additionally, it includes an acrylic case and cooling fan to improve heat dissipation.

Utilizing this module, the entire car seamlessly connects to the 5G network, enabling communication between the controller and the vehicle without any distance limitations. This setup ensures high-speed, low-latency communication, which are impotant for real-time control and live video feed transmission, significantly improving the smart car's performance and user experience [7].

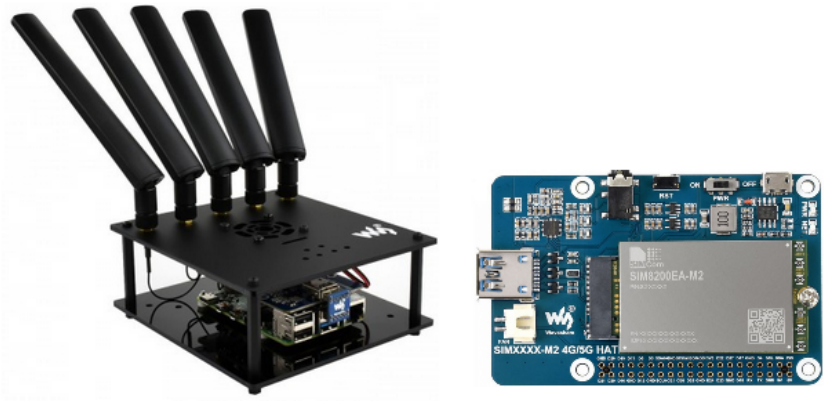


Figure 2.6: The 5G HAT for Raspberry Pi and the SIM8200EA-M2 5G Module

2.7 Raspberry Pi Camera Module

The Raspberry Pi Camera Module 2 is presented in Fig. 2.7 and it serves as an important component integrated into the teleoperated driving system, enabling live streaming over the 5G network to a laptop. Mounted on the car, on top of a servo motor, this camera captures real-time footage of the car's surroundings, being able to be rotated from left to right and vice versa, providing enough visual coverage for the project's requirements.

This particular model is the NoIR version, which stands for "No Infrared filter", making it sensitive to infrared light. This feature is especially useful for our project as it allows the camera to capture clear images in low-light conditions or complete darkness, enhancing the car's ability to navigate in various lighting environments [8].

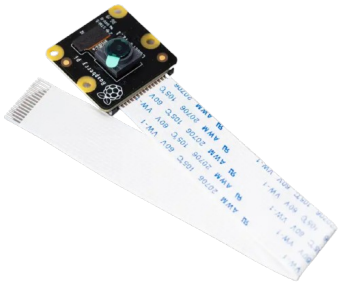


Figure 2.7: Raspberry Pi Camera Module 2 NoIR

In terms of technical specifications, the Camera Module 2 uses the Sony IMX219 sensor, which boasts an 8-megapixel resolution (3280 x 2464 pixels). This high resolution allows for detailed video capture, essential for precise navigation and obstacle detection in the teleoperated system. The camera supports various video modes, providing flexibility depending on the application's bandwidth and latency requirements [8].

The connectivity of the camera to the Raspberry Pi is facilitated via the Camera Serial Interface (CSI) port, which allows high data transfer rates necessary for streaming high-resolution video. The installation process involves securing the flex cable into the CSI port, ensuring a firm connection to avoid signal loss during operation.

Overall, the Raspberry Pi Camera Module 2 NoIR is a great choice for a teleoperated driving system because it's a very small and powerful device, turning it into the perfect solution for a real-time video streaming system. The camera is protected and held by a 3D-printed case and stand that is mounted to the front section of the smart car, providing stability and protection during movement.

2.8 Switches, Battery Holders and Power bank

Two switches have the functions to cut off the power that goes to the Servo Motor that is used for steering and DC Motors. The inclusion of two switches serves a practical purpose, making it significantly more convenient to disable power to a specific component than removing or disconnecting the power source after each use or test of the smart car.

The two battery holders are placed under the smart car, connected to the car's chassis. The positioning of the two battery holders beneath the vehicle serves a dual purpose. This placement helps with the overall design of the car, but mostly with the ease of use. Keeping the batteries securely attached to the car simplifies the process of swapping batteries when needed [6].

Additionally, a Power bank is utilized to power both the Raspberry Pi and the 5G module, ensuring enough power supply to guarantee optimal performance and functionality for both components. Moreover, a buzzer is integrated with the sole purpose of emitting a small signal after the car's startup, indicating that it's ready to connect to the 5G network. All the components mentioned in this section are presented in Fig. 2.8.

Figure 2.8: Switch, Battery Holder, Power Bank and Buzzer

3 Hardware Setup

The current chapter provides a detailed overview of the hardware process, focusing on the creation and the assembly of the chassis, focusing on the arrangement of all the components and their integration.

3.1 The Creation of the Chassis

This design of the chassis improved significantly over time, to accommodate various changes. It was customized to provided a designated space on top of the car that serves as an attachment point for various components, allowing for easier customization. The modular design of the chassis proved to be advantageous, allowing easy replacement of components when they were either damaged or they needed to be remodeled and replaced.

Additionally, another 3D-printed component was integrated into the chassis, the Camera protection box and stand [9] and its components can bee seen in Fig. 3.1. This provided a secure mounting point for the camera, ensuring stability and protection during operation. Most of the parts of the chassis can be seen in Fig. 2.1 from Chapter 2, illustrating the comprehensive and modular design of the chassis's structure.

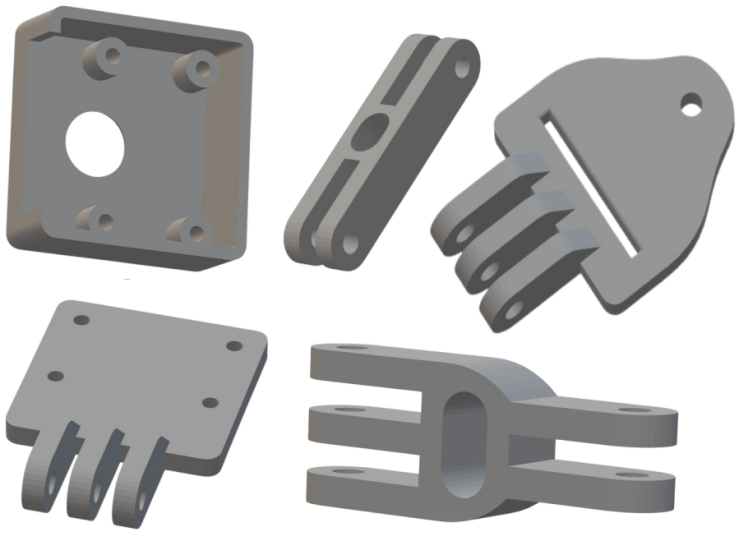


Figure 3.1: The 3D Printing of the Camera Protection Box and Stand

3.2 The Assembly Process

Screws, nuts, bolts, and 3D-printed spacers were used to securely mount the components to the chassis. After the chassis was printed, the bearings were securely glued into their designated spots. The assembly process began with mounting the motors under the chassis. The chassis

consists of two layers held apart by 3D-printed spacers. On the first level, on top of the motors at the rear, the L298 motor driver is positioned, while the Power Bank is located at the front.

On the second level, towards the rear, the 5G Module is mounted and the Raspberry Pi is attached in the middle section, while the servo motor with the camera attached on top is positioned at the very end of the car. Under the car's middle section, the two battery holders are securely fixed to the chassis. Towards the front of the car, the steering system and servo motor are installed beneath.

This arrangement was chosen to help easy modification of different pins and connections. The design allows access to any adjustment regarding the change, without the need to remove any of the parts. All two switches have been securely fixed to the chassis of the car. Each switch serves as a Power Control Switch, being connected to the power supplies of Servo Motor used for Steering and the DC Motors [6].

For further details, the components and wiring configuration Diagram is displayed in Fig. 3.2. This diagram provides a comprehensive visual overview of how each component is connected, showcasing the setup, making it easier to understand the process, the layout, the connections and the overall flow of power throughout the system.

The diagram in Fig. 3.2 illustrates the electrical connections between the components. The input pins IN1 and IN2 for Motor A control its spinning direction and are connected to GPIO 17 and 27 on the Raspberry Pi board. Similarly, the input pins IN3 and IN4 for Motor B are connected to GPIO 23 and 24. The pins ENA and ENB, which enable the PWM signal for Motors A and B, are connected to GPIO 5 and 6. The output pins of Motor A, OUT1 and OUT2, are connected to Motor A, while the output pins of Motor B, OUT3 and OUT4, are connected to Motor B. The L298N ground (GND) is connected to the ground PIN 14 on the Raspberry Pi, and the 12V PIN is connected to a switch seen in the top part of the schematic, powered by the 9V battery next to it.

The Servo motor used for direction has 3 pins that are used for Power, Ground and PWM Signal. The PWM pin is connected to the GPIO 12 pin from the Raspberry Pi board, the ground is connected to the switch from the lower part of the diagram and to the ground PIN 09, while the Power pin is connected to the switch and to the 9V battery next to it.

The Servo Motor used for camera's direction is connected straight to the Raspberry Pi's board, taking power from the PIN 04, connected to the ground PIN 06 and being controlled by the PWM GPIO 14, also known as PIN 08. The Buzzer found in the lower part of the diagram, takes power from PIN 01, being connected to the ground PIN 39 and being controlled by PWM PIN 13.

The Camera is connected directly to the Raspberry Pi board through the Camera Serial Interface port. The SIM8200EA-M2 5G Module is connected to the Raspberry Pi through a USB cable and is powered, along with the Raspberry Pi board, by a Power Bank.

The detailed wiring configuration presented in Fig. 3.2 highlight all the various components of the smart car and each of their connections in the project. This schematic was developed over time, with each component being integrated step by step.

The process began with fundamental elements such as the L298N Motor Driver and the motors, gradually incorporating additional features like the servo motors, buzzer, and 5G module. Each new addition required careful planning and testing to ensure compatibility. This incremental approach allowed for through validation at each stage, ensuring that the entire system functions according to the plan, resulting in an organized layout, as seen in Fig. 3.2.

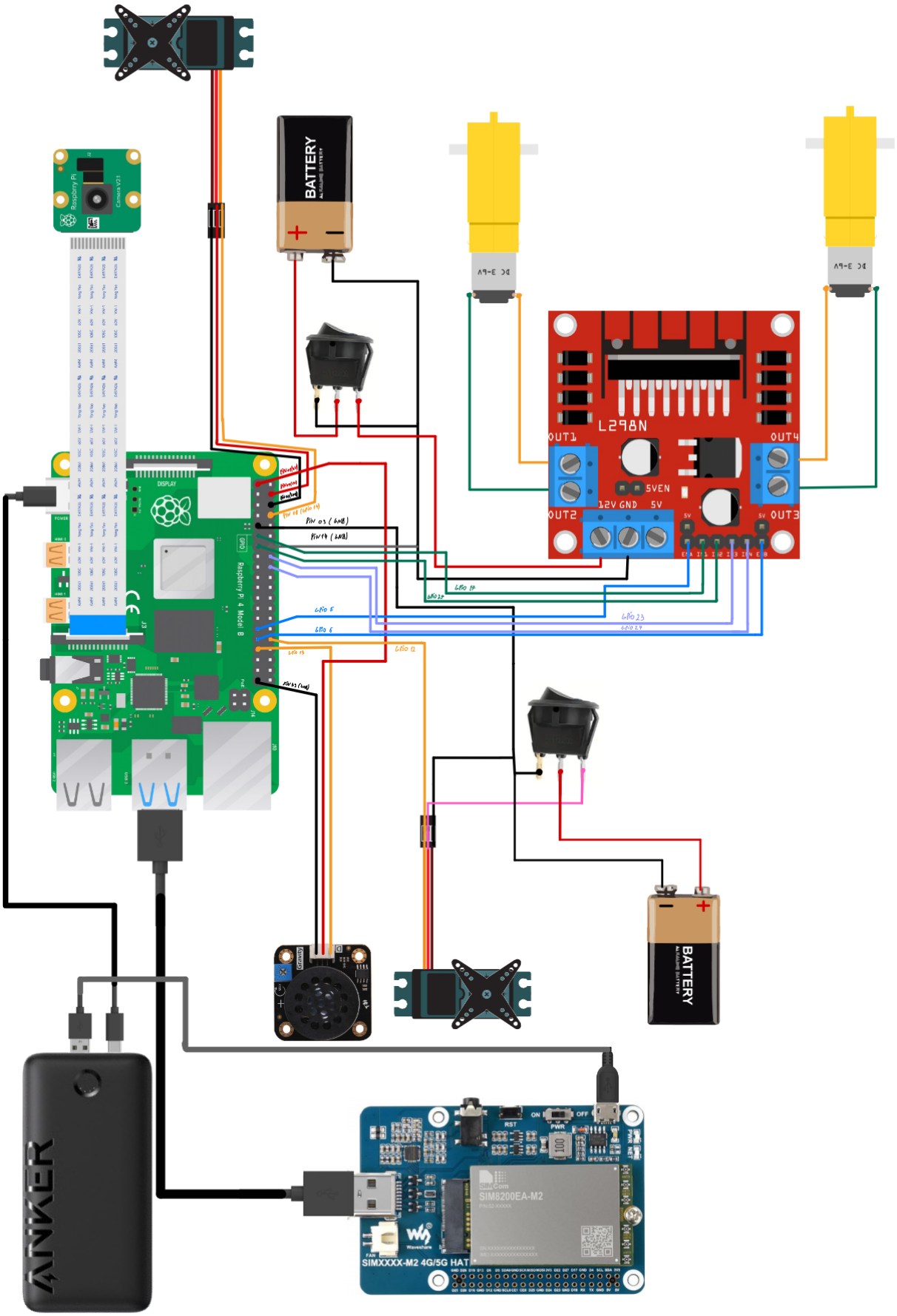


Figure 3.2: Components and Wiring Configuration Diagram

In the process of assembly and choosing components, it was necessary to determine if a 9V battery is sufficient to power the two DC motors and the L298N H-Bridge motor driver. To confirm if it is sufficient or not, several factors must be considered: the voltage drop across the H-Bridge, the current requirements of the motors and the battery's capacity.

The L298N H-Bridge motor driver introduces a voltage drop due to its internal resistance. According to the datasheet [10], this voltage drop can be significant, up to 3.2V and that helps calculating the effective voltage supplied to the motors from a 9V battery, as it is presented in the Equation 3.1.

$$V_{motor} = V_{battery} - V_{drop} = 9V - 3.2V = 5.8V \tag{3.1}$$

The operating voltage range of the motors is between 3V and 6V [11], and the effective voltage of 5.8V falls within this range, making sure that the motors can operate correctly.

Each motor draws 200mA under no-load conditions at 6V [11]. Assuming a no-load current draw of 200mA per motor, the total current draw for both motors would be calculated in Equation 3.2.

$$I_{total} = 2 \times I_{motor} = 2 \times 200mA = 400mA \tag{3.2}$$

Conform the 9V battery's datasheet [12], it has a capacity of 580mAh, offering us the possibility to estimate the operational time under no-load conditions using the formula found in Equation 3.3.

$$T = \frac{I_{battery}}{I_{total}} = \frac{580mAh}{400mA} = 1.45 \text{ hours} \tag{3.3}$$

This estimation shows that the 9V battery can power the motors for approximately 1 hour and 27 minutes under no-load conditions, assuming the motors draw the current continuously, but it is important to note that this is an ideal calculation under no-load conditions. In this project, the car weighs approximately 1.6 kg, which imposes a significant load on the motors, which, combined with the friction between the wheels and the ground, increases the current draw, reducing the effective operational time by a significant amount of time.

During the assembly process, an attempt was made to power the H-Bridge and the two DC motors using the USB-A output from the power bank. The USB-A output specifications are 5V at 3A [13]. However, this configuration did not supply sufficient voltage to the motors. The issue was that the voltage provided by the USB-A output was 5V, which, after taking in consideration the voltage drop of 3.2V from the L298N H-Bridge [10], it resulted in an effective voltage of only 1.8V (5V-3.2V) being supplied to the motors. This voltage is below the operating range of 3V to 6V for the motors, as specified in the datasheet [11] and it was determined that the USB-A output from the power bank could not provide enough voltage for the H-Bridge and the DC motors, making it clear that it was necessary to return to the 9V battery option described earlier in this section.

In addition to the motors, the assembly process also required making sure to have sufficient power supply for the 5G module and the Raspberry Pi. The power bank used in this project has the following specifications: a cell capacity of 20,000mAh, with USB-C and USB-A outputs. The USB-C output can deliver 5V at 3A or 9V at 2.22A, while the USB-A output can deliver 5V at 3A, 9V at 2A, or 12V at 1.5A, with a total output of 22.5W [13].

The 5G module's power supply requirements are an operating voltage ranging from 3.135V to 4.4V and a peak current of 2.7A [14], while the Raspberry Pi board has an operating voltage requirement of 5V with a peak current of 2.5A [5].

To determine if the power bank is sufficient, the power requirements for both devices have to be met. The USB-C output of the power bank, delivering 5V at 3A, is more than enough for the Raspberry Pi, which requires 5V at up to 2.5A [5]. This ensures that the Raspberry Pi will have a stable and sufficient power supply and for the 5G module. The USB-A output of the power bank provides 5V at 3A which is also sufficient, since the 5G module operates within the voltage range of 3.135V to 4.4V and can draw a peak current of 2.7A [14].

Considering the specifications and requirements of the power bank, it is clear that it can provide enough power for both the 5G Module and the Raspberry Pi Board simultaneously, even in peak load conditions, without any issues. Besides the calculation and the requirements found on the data sheets, the project was also tested and no power related issues ever occurred during testing as it can be seen in Chapter 7, confirming that the power bank is a suitable and reliable choice for this project.

4 5G Module Integration

This chapter presents the integration of the 5G Module with the Raspberry Pi, detailing the configuration process and the necessary steps to ensure seamless communication and network connectivity. The system uses high-speed capabilities of 5G technology the enhance the performance and overall functionality of the smart car's system.

4.1 Configuring Raspberry Pi with SIM8200EA-M2

This project was based on the official version of Raspbian Os with the system name "2020-08-20-raspios-buster-armhf"[7]. The following steps outlint the installation and configuration process fo the SIM8200EA-M2 5G Module on the Raspberry Pi. First, the drivers are installed on the Raspberry Pi using the commands found in Box 4.1.1.

Box 4.1.1: Driver Installation Commands

```
sudo apt-get install p7zip-full
wget https://files.waveshare.com/upload/8/89/SIM8200_for_RPI.7z
7z x SIM8200_for_RPI.7z -r -o./SIM8200_for_RPI
sudo chmod 777 -R SIM8200_for_RPI
cd SIM8200_for_RPI
sudo ./install.sh
```

After installing the driver, it is important to verify if the "wwan0" network interface has been generated. The "wwan0" interface is a Wireless Wide Area Network interface, which allows the Raspberry Pi to connect to mobile networks. To check if the interface has been generated, it's needed to take the command presented in Box 4.1.2, and run it in the Raspberry Pi's terminal [7].

Box 4.1.2: List Network Interfaces

```
ifconfig -a
```

This command lists all the network interfaces available on the Raspberry Pi. If the "wwan0" interface is listed, it indicates that the driver installation was successful.

The "ttyUSB" ports are used for USB to serial communication, which is essential for interacting with the 5G Module. To confirm that the device was recognised we need to run the commands presented in Box 4.1.3 and search for any device that is using the "ttyUSB" serial communication port.

Box 4.1.3: Check USB Ports

```
lsusb
ls /dev/ttyUSB*
```

The "lsusb" command lists all USB devices connected to the Raspberry Pi, while "ls /dev/ttyUSB*" lists all serial devices connected via USB. If the output includes entries for

"ttyUSB" ports, it means the 5G Module is properly connected and recognised. Typically, if you have a mouse and keyboard connected, they will occupy the first two ttyUSB ports, so the 5G Module is usually associated with "ttyUSB2"

After ensuring that the "ttyUSB" serial communication port is recognised, the next step to fully configure the network interface with the 5G module. This requires the installation of "Minicom", a terminal emulation program used for communication with devices connected to serial ports [7]. Minicom allows you to send commands directly to the 5G Module and receives responses. It can be installed and opened by running the commands found in Box 4.1.4.

Box 4.1.4: Install and Open Minicom

```
sudo apt-get install minicom
sudo minicom -D /dev/ttyUSB2
```

In this setup, "/dev/ttyUSB2" is used as the communication port for the 5G Module. Minicom opens a terminal session with the 5G module, giving permission for the user to send AT commands for configuration and troubleshooting.

4.2 5G Module Configuration

Notably, this configuration represents the first successful working setup on a Raspberry Pi made in Romania. The initial step in setting up the SIM8200EA-M2 was to activate the network using it's designated AT command (4.1). AT Commands are instructions used to control modems, which allows the user to configure and interrogate modem's settings and status. The first AT command to activate the network is:

$$AT + NETACT = 1 \tag{4.1}$$

After identifying the vendor ID and product ID, another AT command (4.2) has to be used to verify if the data matches. The vendor ID and the product ID are unique identifiers for USB devices, used by operating systems to recognize and interact with the hardware. For this project, the vendor ID is "1e0e" and the product ID is "9001" for Windows devices, and '9011' for Linux and Raspbian OS devices [15]. The command to obtain the IDs is:

$$AT + CUSBCFG =? \tag{4.2}$$

If the returned data does not match the expected vendor and product IDs, it indicates that we are not in the correct mode, necessitating adjustment to meet our requirements. To resovle this, in case we are in the wrong mode we can change it using (4.3) and this way we will connect our OS to the 5G network [15].

$$AT + CUSBCFG = usbid, < vendorid >, < productid > \tag{4.3}$$

This whole process ensures that the Raspberry Pi can properly communicate with the 5G module, enabling it to access the 5G network for high-speed data transmission.

4.3 Private Network Configuration

After configuring the module and the Raspberry Pi, achieving seamless communication between the laptop and the 5G module requires both devices to be connected to the same Local Network. However, this type of connection would defeat the purpose of using 5G, as it would limit the network's range and capabilities.

To overcome this obstacle, ZeroTier's services are used to create new routes and make both IP addresses be connected to each other by sharing the same Local Network, virtually. ZeroTier is a network virtualization services that allows devices to connects as if they are on the same local network, regardless of their physical locations. This enables communication between the two devices, as long as they are connected to the internet [16].

This process involves setting up a ZeroTier network, joining both the Raspberry Pi and the laptop to this network and ensuring that they can communicate with each other over this virtual local network. This setup uses the 5G network's capabilities to provide seamless, high-speed connectivity between the smart car and the control interface, improving the overall functionality and user experience.

5 Software Architecture

The current chapter presents the software Architecture of the smart car's system, highlighting the server-client communication, controller integration, camera feed, motor control, and user interaction. The main goal of this chapter is to provide a detailed understanding of how each component interacts with the system and how it was programmed.

5.1 Server-Client Communication

To establish the connection between the car and the controller, a server-client approach was adopted, using socket programming, for efficient communication. The Raspberry Pi acting as the server and the client being a laptop. This server-client connection grants real-time interaction and control over the car's actions. On the Raspberry Pi (server) side, a socket was hosted on a predefined port, waiting for any incoming connections from the client. Once connected, the server would receive data from the client, processing it and transforming it into commands that would be used to control the car.

The client (laptop) initialized a client socket to establish the communication with the Raspberry Pi server, sending controller commands and other relevant data to the server, facilitating seamless interaction and control over the car's behavior[17]. A comprehensive diagram detailing the Server-Client Architecture, illustrating the communication flow and interaction, is provided in Fig. 6.1 from Chapter 6. The code for the server-client communication can be found in the Code Listing 10.2, 10.3, and 10.5 from the appendix.

5.2 Controller Integration and Data Reading

A controller initialization function was created to detect any controllers connected to the laptop. Once a connection was established or found, the function enables the reading of control inputs from the connected controller. After the detection of the controller, the initialization function granted access for reading control inputs. This facilitates the transmission of control data to the server.

Another function was developed to selectively send data only when new changes were made to the previous data. This approach prevented unnecessary traffic to the server, ensuring a better utilization of resources and avoiding congestion. Any changes detected in the controller's inputs, such as triggers, joysticks, or buttons, were collected in an array and transmitted to the client. Once received, the client processed and utilized this data to maneuver the car[18].

To enhance the accuracy of the transmitted commands, the program rounds the joystick and trigger values to three decimals, before sending them to the server. This reduction in precision minimizes the amount of data being sent while retaining sufficient detail for smooth operation. The system also incorporates a mechanism to handle continuous input, such as holding a joystick in a particular direction, by implementing a feedback loop, which makes sure that the commands are repeatedly sent at regular intervals as long as the input changes. This

method ensures that the car responds correctly to the commands, avoiding the unnecessary transmission of identical commands, while also offering a reliable and consistent control over the smart car. The code for the controller integration and data reading can be found in the Code Listing 10.1 and 10.3 from the appendix.

5.3 Camera Feed and Connectivity

The Raspberry Pi was programmed to stream live video from its camera module over the web using a web server that was created with the Flask framework. This server continuously capture video data and sends it to the IP address '0.0.0.0', making the live stream accessible to any device connected to the same local network[17].

Once the connection is established, users can easily access the live stream by entering the web address that consists the Raspberry Pi's IP address followed by the designated port (5.1)[8].

$$http://172.30.170.224:6700 \tag{5.1}$$

In addition to Flask, the integration utilizes the picamera library, which offers and efficient interface for capturing video with Raspberry Pi camera modules. The combination of Flask and picamera offers a reliable live video stream setup, while managing to maintain a low latency and high resolution signal [19]. The code for the camera feed and connectivity can be found in Code Listing 10.6 from the appendix.

5.4 Motor Control and L298N Module Integration

The L298N is a dual H-bridge motor driver that is designed to allow control over the speed and direction of two DC motors, independently. The H-bridge circuit allows current to flow in either direction through the motor, enabling forward and reverse movements. In this project, the module is used for Motor Connections, PWM and Speed Control, and Direction Control [6].

The software architecture for motor control and the L298N module integration is implemented using Python, using the GPIO library to handle PWM signals and motor control. The code initializes the GPIO pins for the motor driver and servos, setting up PWM for precise control. The speed of the motors is controlled by changing the duty cycle of the PWM signals [20].

In this project, the integration of the L298N motor driver with the Raspberry Pi involves connecting the motor driver's input pins to the GPIO pins from the Raspberry Pi to enable precise control over the motors. The ENA and ENB pins on the L298N module are connected to the Raspberry Pi's PWM GPIO pins, allowing for the modulation of the motor speed. The IN1, IN2, IN3, and IN4 pins of the L298N module are wired to the GPIO pins on the Raspberry Pi, enabling directional control. This setup allows the software to dynamically adjust motor speed and direction by sending appropriate signals through these GPIO connections. This integration of hardware and software components ensures that the smart car can perform precise maneuvers as required through the teleoperated driving system. More details regarding the Motor Control and L298N Module Integration can be view in Fig. 3.2 from Section 3.2. The code for the Motor Control and L298N Module Integration can be found in the Code Listing 10.4 and 10.7 from the appendix.

5.5 Servo Motors Integration

The two servo motors are controlled using PWM, which allows precise control over their position using the duty cycle. The control was organized into six functions: two for initialization, another two for modifying the PWM value and rest for updating the servo direction.

The smart car uses two servo motors that have different purposes: one is for steering and receives data sent from the joystick, while the other takes input from the D-pad. Although both servo motors seem to share similarities regarding the programming methods and functions, they operate with different inputs, data and outputs from the Xbox Controller [6].

Additionally, the integration of the servo motors uses the pigpio library, which provides precise and reliable PWM signal generation. This library is particularly advantageous for handling multiple PWM signals simultaneously, ensuring smooth and accurate servo motor control. The steering servo motor adjusts the car's direction based on joystick inputs, while the camera servo motor, mounted on top of the vehicle, adjusts the camera's angle based on D-pad inputs. This setup allows for dynamic and responsive control, enhancing the teleoperated driving experience. The integration of pigpio is important in achieving the project's precise control over pulse width adjustments, essential for the accurate operation required in teleoperated systems [21]. The code for the servo motors integration can be found in Code Listing 10.8 from the appendix.

5.6 User Interface and Interaction

The control of the car is done through the Xbox Series X controller, a very intuitive interface, which buttons can be seen in the Fig. 5.1.

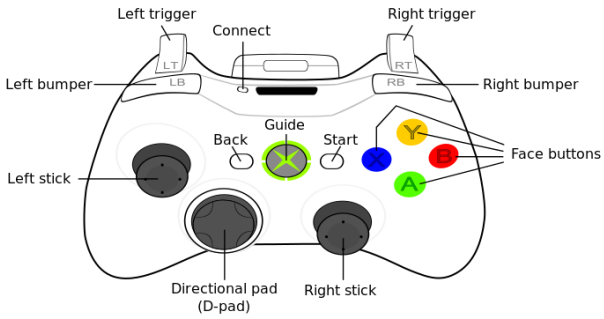


Figure 5.1: Button layout of a wireless Xbox controller [22]

This project is using several buttons for navigation, making sure to have an intuitive and effective control over the smart car. The functions and actions triggered by these buttons are presented in Table 5.1.

Table 5.1: Controller Button Functions

Controller Button	Associated Function
Right Trigger	Move the car forward
Left Trigger	Move the car backward
X-axis of the Left Stick	Steering the car
X-axis of the D-pad	Camera orientation

This integration of the Xbox Series X Controller provides an intuitive and effective user interface for controlling the smart car. By using the triggers for forwards and backwards movement, the left stick for steering and D-pad for camera orientation, the user can easily navigate and control the car. The code for the user interface, interaction and the mapping of the buttons can be found in the Code Listing 10.3 and 10.9 from the appendix.

6 System Architecture

This project is based on a client-server architecture that communicates through a 5G network, as illustrated in the Fig. 6.1. This architecture uses the high-speed and low-latency capabilities of 5G technology to create an efficient interaction between various devices such as Raspberry Pi which acts as the server, and the laptop, which serves as the client.

6.1 System Overview

The system architecture diagram (Fig. 6.1) shows the car circled with a green circle, highlighting components such as the 5G module, Raspberry Pi, camera, motors, and servo motors. The 5G connection that contains the virtual network are circled with a blue circle, indicating the network infrastructure. The client, circled in red, includes the laptop, controller, and the method of connecting to the internet.

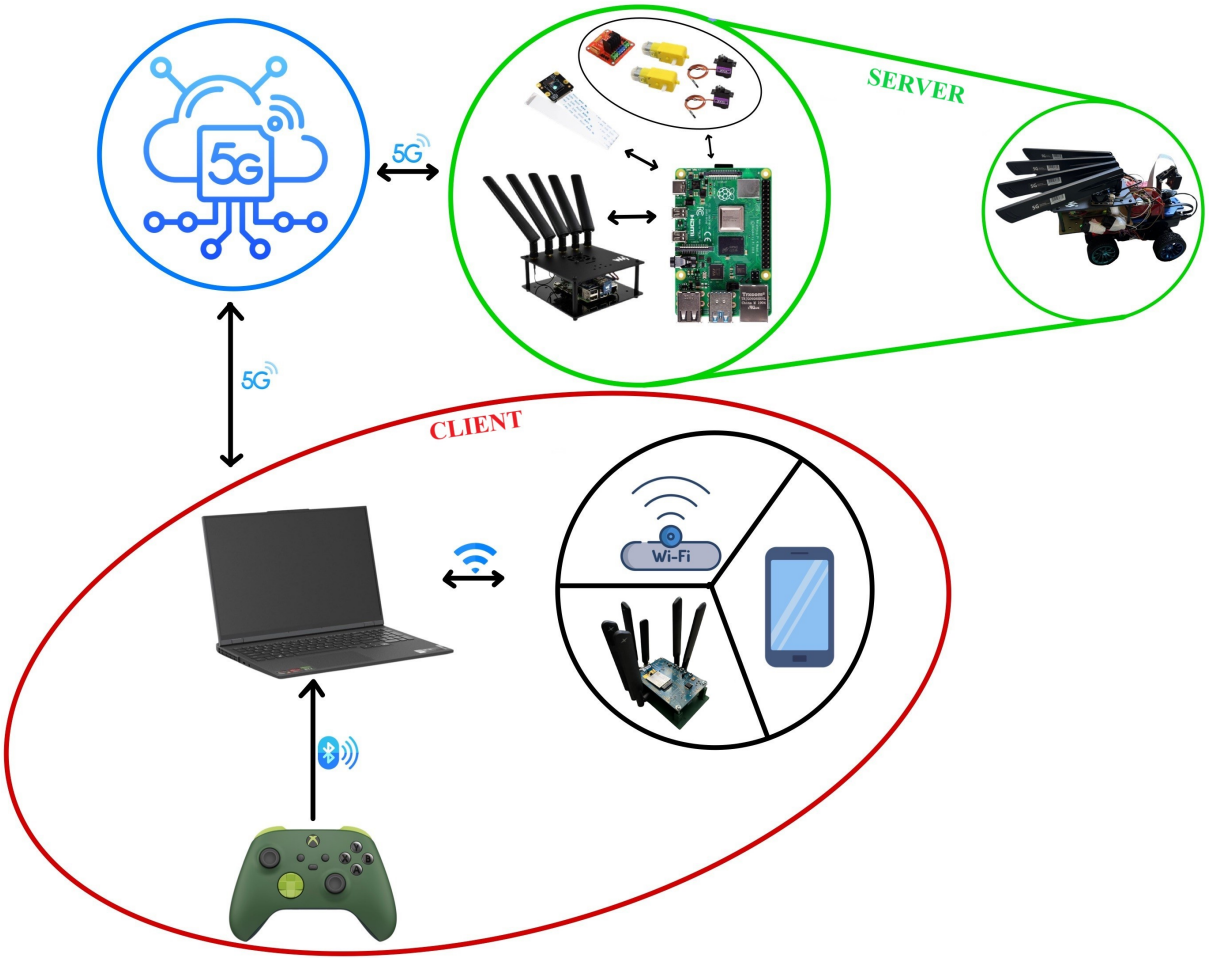


Figure 6.1: System Architecture: 5G Connectivity Between Client and Server

6.2 Server Components and Functionality

The Server is a Raspberry Pi attached to the smart car, as it can be seen in Fig. 6.1. It connects to the 5G network using the SIM8200EA-M2 5G Module. Over this network, a virtual network is established, connecting both the server and the client. The server has a camera attached to it and streams the live feed over to the virtual network.

Additionally, it receives data, interpreted as commands, from the client, via the 5G network. These commands control the motors and servo motors, determining the car's movements. The Raspberry Pi (the server) is configured to be the central control unit of the smart car.

The Server runs a Flask web server to manage and stream live video from the attached camera, ensuring real-time feed transmission over the network. It uses socket programming to handle communication between the client and the server, allowing for efficient and reliable transmission of control commands and feedback data. A lot more details regarding the Server components and functionality can be found in Chapter 5.

6.3 Client Elements and Features

The Client is a laptop connected to the 5G network. The operator uses an Xbox Series X Controller, connected via Bluetooth. The laptop's 5G connection can be achieved through another 5G Module configured for Windows, WiFi, or an Internet hotspot from phone with 5G connectivity.

Additionally, the client runs a Python script that captures the controller's inputs, processes them, and sends the relevant data to the server using socket programming. The integration of the Xbox Series X Controller with the laptop allows for a user-friendly interface, making it easy for the operator to control the car's movements and camera orientation.

The client (the laptop) sends the controller's commands over the 5G network to the server, resulting in real-time control of the smart car. A lot more details regarding the Client components and functionality can be found in Chapter 5.

6.4 Virtual Network and Communication

The virtual network created with ZeroTier's services allows for seamless and secure communication between the server and client, regardless of their physical locations. This network facilitates the transmission of control commands from the laptop to the Raspberry Pi and live video feeds with low latency from the camera to the virtual network. This setup offers the possibility for each device connected to the virtual network to view the livestream at any time [16].

Additionally, the use of ZeroTier ensures a high level of security and reliability in the communication process. The system's architecture supports scalability, enabling the addition of more devices or users without compromising performance. For a more detailed explanation of the virtual network and communication setup, refer to Chapters 4 and 5.

6.5 Performance and Benefits

This setup showed low-latency communication and real-time responsiveness, which are critical for the teleoperated driving system and also being validated through a series of experimental cases and tests, which are detailed in Chapter 7. The use of 5G technology shows a significant improvement in data transfer speeds and connectivity, enabling smooth and efficient operation of the smart car. The integration of these components forms a very innovative system for teleoperated driving systems.

Overall, this is an overview of the system architecture for this project, highlighting the key components and their interactions within the 5G network environment. This setup expands the potential applications of the technology, including teleoperated monitoring and intervention in hazardous environments. The communication setup provided by the 5G network makes sure that the system can handle high volumes of data traffic, maintaining performance under multiple environments and conditions.

7 Experimental Cases and Results

This section has the purpose of presenting the results obtained from several experiments that were made in different environments. Not all experiments were perfect, since in some cases, the area of the city lacked 5G coverage or there too many users connected to the same network and that would result in latency issues and compromised the real-time aspect of the project.

In another experiment, it was observed that the car would turn too sharp, putting too much pressure on the 3D-printed mount of the servo motor which held in place the front wheel, causing it to break. This highlighted the need to program the servo motor to limit its range of motion to prevent such issues.

Furthermore, during these experiments, various factors were analyzed regarding the system's overall performance. Factors such as the stability of the 5G connection, the latency of the command transmission, and the quality of the live feed were evaluated during testing. These factors would eventually determine how reliable the system is and if it fits into the category of a teleoperated driving system in real-world scenarios. The experiments were made to highlight both the strengths and limitations of the current setup, providing helpful information that could be used in future improvements and optimizations of the system architecture and control algorithms.

Two of the main experiments that can demonstrate the project's capabilities were performed in two distinct environments: indoors and outdoors. The detailed results and observations extracted from these experiments are presented in sections 7.1 and 7.2.

7.1 Experiment 1: Outdoor

This experiment was made in an area with few people around and at a time when the network was not heavily overcrowded, as shown in Fig. 7.1. In the same figure, the setup can be seen: the laptop, circled in yellow, the Xbox Series X Controller circled in purple, and the smart car controlled through 5G, circled in red. In this experiment, the laptop was connected to an internet hotspot from a phone with a 5G connection. Fig. 7.1 also shows that the car provides live feedback from the camera on the laptop's screen while being controlled using the controller.

During this experiment, the system demonstrated its capability to maintain real-time responsiveness and smooth control of the smart car. The low-latency communication facilitated precise control over the car's movements and immediate feedback from the camera, enabling the operator to navigate the car effectively. The successful completion of this experiment highlighted the potential of the system to operate efficiently in outdoor environments with minimal interference as long as there is stable 5G connection.

During the outdoors experiment, several factors were monitored to evaluate the system's effectiveness. These include the latency of the control signals, the quality and stability of the video feed, the stability of the 5G connection and the overall responsiveness of the smart car to the user inputs. The smart car's ability to navigate and stream was tested and it highlighted the importance of a stable 5G connections for maintaining the integrity of the control system and ensuring the safety and accuracy of the car's movements.

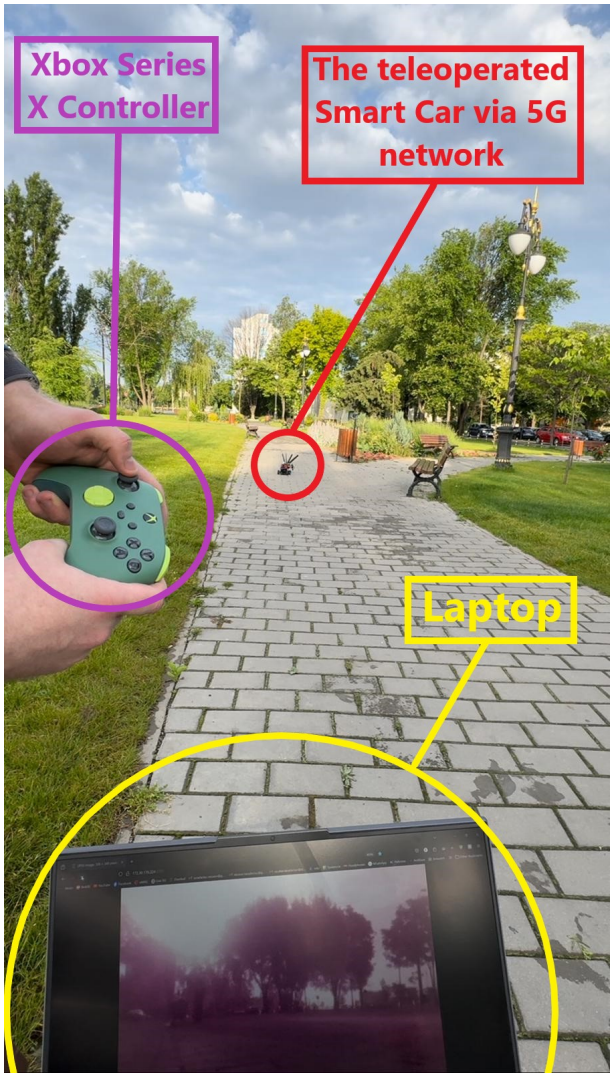
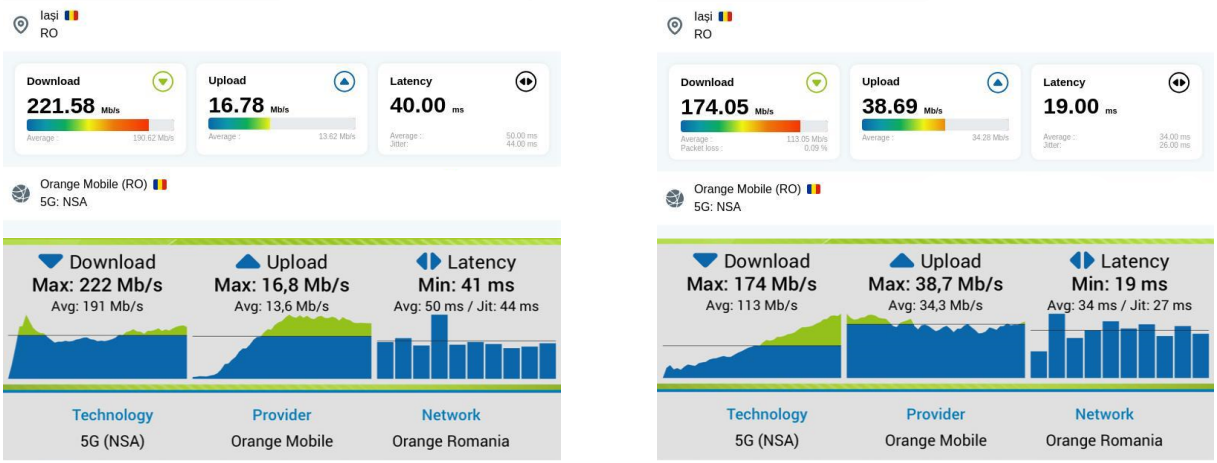


Figure 7.1: Outdoor Experiment Setup with 5G Connectivity

The project was first tested in front area of the student dormitory T1-T2, where the average latency recorded was 50ms and a minimum of 41ms, as shown in Fig. 7.2a. With a camera resolution of 640x480 and 30 frames per second, the latency was too high, resulting in significant delays in command execution and camera feedback. This result demonstrated that the project needs a good latency to avoid excessive delay and the front of the student dormitory T1-T2 is not a suitable area for it.

A solution was attempted by reducing the frames per second and the camera resolution, but it still did not achieve the desired results. Consequently, the project location was moved to the park near the university rectory. In this new environment, the latency improved significantly, with a minimum latency of 19ms and an average of 34ms, as shown in Fig. 7.2b. By reducing the resolution to 320x240, the experiment was successful, as it can be seen in Fig. 7.1.

These results highlight the critical importance of a stable and low-latency 5G connection for the teleoperated driving system to function effectively. The initial test at the student dormitory T1-T2 underlined the challenges posed by high latency, while the successful test at the university rectory park demonstrated the practicability of the system under improved network conditions. The reduction in resolution and frame rate proved to be a practical solution to reduce latency issues, ensuring smooth and responsive control of the smart car.



(a) Latency at the entrance of Student Dormitory T1-T2 [23]

(b) Latency at the University Rectory Park [23]

Figure 7.2: Latency Results in Different Outdoor Locations

7.2 Experiment 2: Indoor

To perform this experiment, access had to be gain to be able to test the project in an indoor laboratory that provided the necessary controlled environment and 5G infrastructure. This experiment was run using a private 5G network, as shown in Fig. 7.3. The controlled environment offered stable network conditions, which were important for accurately testing the system's performance and latency under optimal circumstances.

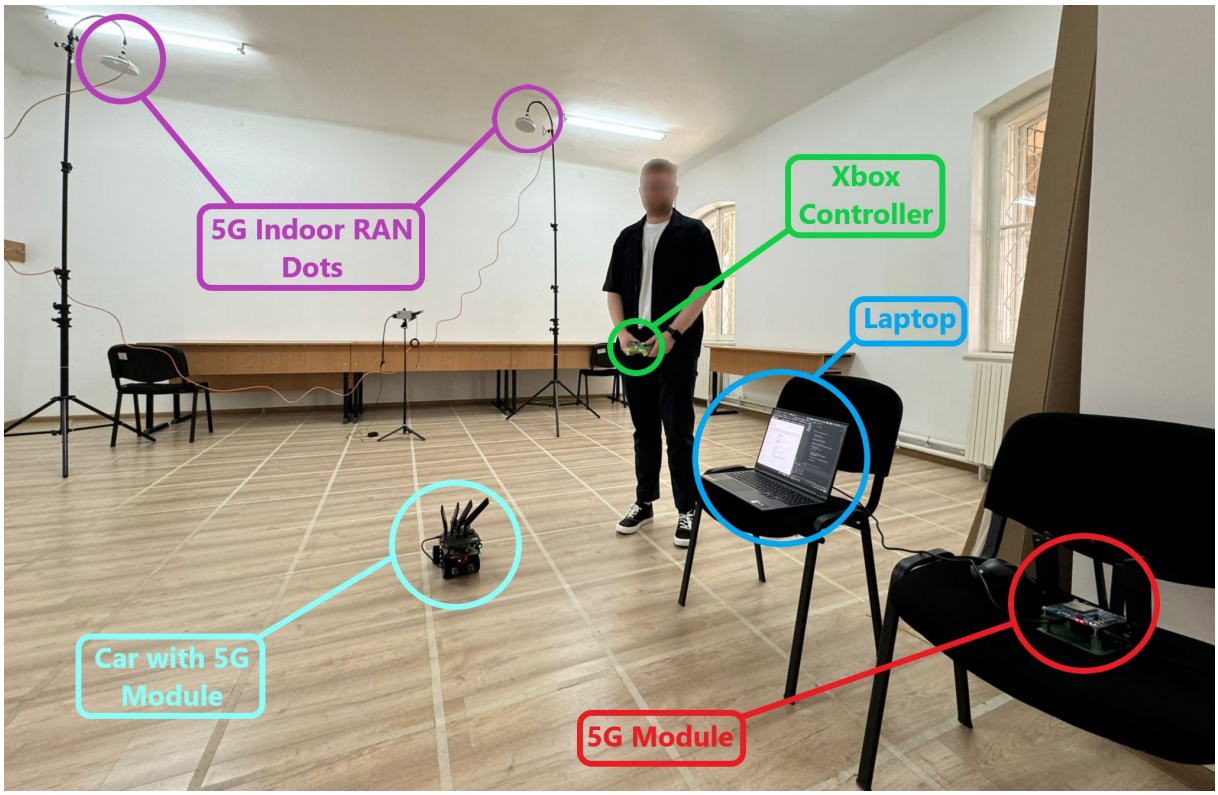


Figure 7.3: Indoor Experiment Setup with 5G Connectivity

In Fig. 7.3, the setup can be seen: the laptop, circled in blue, connected to the private 5G network via a SIM 8200EA-M2 5G Module configured for Windows, which is also visible in

the figure and circled in red. The smart car, connected to the same network, is circled in cyan, the Xbox Series X Controller is circled in green, and the main components of the 5G private network are the two 5G Indoor Radio Access Network (RAN) Dots, circled in purple.

The 5G Radio Access Network plays a critical role in making this setup work because it is a key component for the 5G architecture that provides wireless connectivity to both devices, laptop and raspberry. The system consists of two antennas, known as dots that have the purpose of creating an indoor 5G network, offering this project the solution for the communication problem of our project with the commercial 5G network that did not always have coverage inside buildings. This enables the project to maintain a robust and high-performing network, ensuring optimal performance for real-time teleoperated driving even in an indoor environment [24].

During this experiment, the system demonstrated its capability to maintain real-time responsiveness and low latency, allowing the smart car to successfully respond to various scenarios and commands. This experiment highlights the potential of the system to operate efficiently in indoor environments with minimal interference, as long as there is a stable 5G connection offered by the 5G Indoor RAN Dots.

Several factors were monitored to evaluate the system's effectiveness during the indoor experiment. These include the latency of the control signals, the quality and stability of the video feed, the stability of the 5G connection, and the overall responsiveness of the smart car to user inputs. The smart car's ability to navigate and stream was thoroughly tested, highlighting the importance of a stable 5G connection for maintaining the integrity of the control system and ensuring the safety and accuracy of the car's movements.

8 Challenges and Solutions

The Table 8.1 presents the main challenges encountered along the way and their solutions. Each challenge represents a significant obstacle faced, ranging from power supply issues for DC motors to compatibility concerns with the SIM8200EA-M2 5G Module. The solutions found are a mix of diverse approaches such as hardware changes, software adjustments, adaptability and 5G Module configurations.

Table 8.1: Challenges and Solutions

Title of the Challenge	Challenge and Solution
DC Motor Power Supply	**Challenge:** Initially, the DC motors were provided with 5V from the Raspberry Pi, resulting in a not so optimal speed. **Solution:** Recognized the need for a dedicated power supply for the DC motors to gain more power and speed.
Servo Motor Accuracy and Resetting	**Challenge:** Faces accuracy and resetting issues with the servo motor even code troubleshooting. **Solution:** After various tests, adding a dedicated power supply for the servo motor solved the accuracy and resetting issues.
SIM8200EA-M2 Driver for Raspberry Pi	**Challenge:** Initially, the SIM8200EA-M2 5G drivers would not install correctly on the Raspberry Pi, with the board not recognizing the "ttyUSB" serial communication port. **Solution:** It was discovered that the drivers were specifically configured to work on a few particular series of Raspbian OS Official releases. The issue was resolved after installing the "2020-08-20-raspios-buster-armhf" OS version.
Compatibility Issues with the "ttyUSB"	**Challenge:** In the step where you need to confirm that the SIM8200EA-M2 was recognized, the "ttyUSB" serial communication port would not be found. **Solution:** The "ttyUSB" serial communication port may not be displayed because not all Raspberry Pi operating systems are fully compatible. To make the port appear, you need to run the following commands: sudo apt-get install minicom sudo minicom -D /dev/ttyUSB2
Configuring SIM8200EA-M2 Network with Raspberry Pi	**Challenge:** Initially, the SIM8200EA-M2 5G module would only connect to the 5G network on Windows, refusing to connect on the Raspberry Pi. Although the board would be recognized, it would be unable to communicate. **Solution:** After researching the problem, it was discovered that the product ID is "9001" for Windows devices and "9011" for Linux and Raspbian OS devices, which is necessary for configuring the module.

Continued on next page

Table 8.1: Challenges and Solutions (Continued)

Title of the Challenge	Challenge and Solution
Camera and Server can't run simultaneously	**Challenge:** The functions created for live streaming with the camera and those designed for managing car control data over 5G could not run simultaneously. **Solution:** The solution to make sure that all functions work simultaneously is to create two threads with distinct roles: one for operating the camera and another for managing the server and collecting data from the client.
Data Loss during Transmission	**Challenge:** Occasionally, data transmission issues would show up when the server misinterprets received data, resulting in errors during processing. **Solution:** This issue was caused by data loss occurring during transmission over the internet. The client would send data to the server regarding all the buttons, triggers, and joysticks, even if they have not been moved or used yet. The solution is to create a function that analyzes the data and sends only the information from the inputs that will be modified, rather than sending all of them. Additionally, another improvement was made to reduce precision from 10 decimals to 3 decimals, because such precision was unnecessary.
Insufficient Motor Alimentation via USB	**Challenge:** We attempted to power the motors through the H-bridge using a power bank, but the only available connection was a USB port. It was discovered that USB can only supply 5V, which was insufficient for powering two motors. **Solution:** The solution was to abandon the USB power supply and instead use a dedicated power source that could provide the required voltage and current for the motors.

These challenges and their solutions highlight some important aspects such as correct hardware-software integration, serial port communication, compatibility issues, power management, and adapting to solve the problems encountered along the way. All these challenges present how complex the 5G network is and the fact that there is a lot more knowledge to be explored, especially regarding teleoperated driving systems. This project has demonstrated the potential of 5G technology in enhancing real-time teleoperated systems and has provided a solid foundation for future advancements and optimizations in this field. A few more ideas for future developments and improvements would be presented in the Chapter 9.

9 Future developments and improvements

This project can be expanded with additional features to enhance its functionality. The following proposed improvements include integrating sensors for indoor localization, adding LEDs for better simulation, and developing intelligent navigation systems. These enhancements will improve the smart car's capabilities and provide a more realistic user experience, demonstrating the significant potential of this project.

9.1 Optic Sensor for 2D Localization (Indoors)

The plan is to integrate an optic sensor similar to the one seen in Fig. 9.1, for 2D localization to track the car's position, particularly for indoor use. The optic sensor used is similar to the sensors commonly found in computer mice. This type of sensor will be able to capture precise surface details and movements for the smart car's localization.

In scenarios where the indoor 5G private network is not functional, the system can rely on a less optimal internet connection to transmit some 2D localization data of the vehicle without sending live video output.

This backup approach makes sure that some functionalities remain operational, even if the indoor 5G network created by the dots is unavailable or non-functional. This system has the purpose to improve the reliability of the system, maintaining essential localization capabilities under various network conditions.

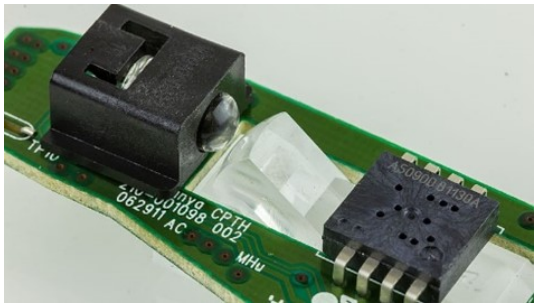


Figure 9.1: Logitech M210 Optical Mouse Sensor

9.2 LEDs for Headlights, Stoplights and Turning Signals

Adding LEDs for headlights, stoplights and turning signals to improve the realism and functionality of the smart car. Two LEDs will be installed as headlights to provide forwards illumination. Stoplights will be implemented with two LED that activate when the car decelerates or comes to a stop.

Additionally, four LEDs will be integrated as turning signals to indicate the direction of intended movement. All LEDs will be controlled via the Xbox Series X Controller, offering the user a more realistic and intuitive feel of a real car.

The LEDs would be wired to the Raspberry Pi, and will be managed by it, based on the commands resulted from the input received from the Xbox Series X Controller through the 5G network. Specific GPIO pins on the Board would have some LEDs assigned to them.

This improvement not only expands the controller's button functionality, but it also makes the project more engaging and makes it suitable to be an educational tool for demonstrating a teleoperated driving system.

9.3 "Going Back to Home" mode and Obstacle Avoidance System

To improve the smart car's navigation capabilities, an Ultrasonic Sensor will be mounted on top of a servo motor to scan its surroundings and provide useful data for an obstacle avoiding algorithm. The sensor can be seen in Fig. 9.2 on the left, and it will help the car to avoid obstacles by sending and receiving sound waves to measure the distance between the car and the nearby objects [25].

Alternatively, a Light Detection and Ranging (LiDAR) system, as presented in Fig. 9.2 on the right, can be used for a more advanced obstacle detection and path planning algorithm. The LiDAR provides a detailed 3D map of the surroundings, allowing the car to create a safe and efficient route to navigate around obstacles [26].

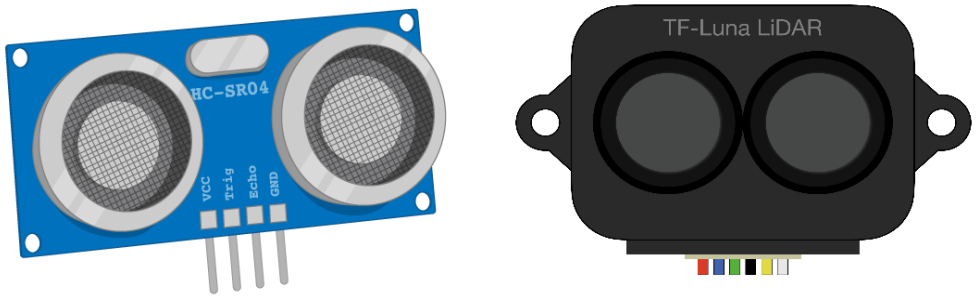


Figure 9.2: Ultrasonic Sensor (left) and LiDAR Module (right)

Additionally, a key feature is to develop a "Going Back to Home" mode. This mode will allow the user to set a "home" position, by pressing a designated button on the controller. After pressing another button, the car will use the obstacle avoidance system to navigate back to the previously save home position.

By integrating either the ultrasonic sensor or the LiDAR system, the smart car will demonstrate the versatility and scalability of this project, showcasing the potential for real-world applications of teleoperated driving systems.

9.4 Virtual Reality Interface

Implementing a Virtual Reality (VR) interface can provide users with an immersive and interactive experience. Using VR glasses similar to the one presented in Fig. 9.3, users can view real-time data from the car's perspective and move their head towards the direction they want, with the servo motor moving the camera in the same direction.

In addition to improving user interaction, the VR interface will also maintain the benefits regarding safety and efficiency. This immersive setup not only makes teleoperated driving more

intuitive but also offers potential applications in remote vehicle operation for hazardous environments, search and rescue missions, and advanced driver assistance systems (ADAS). This setup could bring numerous improvements, such as a system that includes navigation instructions, obstacle warnings, and system status updates. The VR interface will make the teleoperated driving experience more engaging and informative, showcasing the vast improvement possibilities of this project.

Figure 9.3: Apple Vision Pro VR Glasses

10 Conclusions

The implementation of the teleoperated driving system via 5G network had some significant outcomes, showing successful integration of various components. One of the key results is that the smart car responded effectively to the commands from the Xbox Series X controller, offering a very intuitive experience. The analog acceleration functionality allows precise commands and control from the user, offering a more pleasant driving experience. The safety features worked as intended and the servo motor's straightening mechanism in the absence of commands worked as it should. At the same time, the emergency stop functionality with complete controller disconnection did not show any issues during testing.

The configuration process was completed successfully, establishing a seamless connection between the Raspberry Pi and the SIM8200EA-M2 5G Module. All these results mark a successful implementation of the teleoperated driving system via 5G network project, its user-friendly interface being approved even by other people that had the chance of testing it out.

The successful development of the 5G integration with the car's control opened a more broader area for exploration, learning and development. The car's capabilities can be improved and a lot more features can be added to make the project even more advanced than it currently is. This teleoperated driving system via 5G network project showcases a fusion of creativity and technology. From 3D-printed chassis to the Xbox Series X Controller, it merges innovation and user-friendly design and experience. The addition of an analog acceleration elevates the car's functionality, responding dynamically to user input.

The integration of the live stream camera is a very important part of the system, enabling real-time visual feedback and improving the overall user experience. This feature makes the project more than just a simple remote-controlled car, allowing the user to view the car's surroundings and navigate effectively. The smart car's ability to provide a live video feed is essential for real-world teleoperated driving applications, offering the operator precise control and the ability to view the car's surroundings. This factor also makes the project a perfect educational tool for simulating a real driving experience and for showcasing a complete teleoperated driving system through a 5G network.

The project's success in real-world experiments, both indoors and outdoors, validates the system's design and implementation. These experiments highlight the importance of a stable 5G connection for maintaining low latency and high responsiveness, those being some critical factors for teleoperated driving. The use of a private 5G network in an indoor environment demonstrates the potential for creating dedicated networks to ensure reliable performance, even in areas with limited commercial 5G coverage.

Taking in consideration the success of the experiments, all the challenges met along the way had the various future development and improvements ideas, this project only demonstrates its versatility and scalability, showcasing a clear potential for real-world applications of teleoperated driving systems through a 5G network.

Bibliography

[1] Joyce Ayoola Adebusola et al. "An Overview of 5G Technology". In: *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*. 2020, pp. 1–4. DOI: 10.1109/ICMCECS47690.2020.240853.

[2] Amazon Web Services. *What is 5G?* URL: https://aws.amazon.com/what-is/5g/ (visited in 2024).

[3] Johann M. Marquez-Barja et al. "Enhanced Teleoperated Transport and Logistics: A 5G Cross-Border Use Case". In: *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. 2021, pp. 229–234. DOI: 10.1109/EuCNC/6GSummit51104.2021.9482459.

[4] Rodney Majola. *WD RC Smart Car Chassis for Arduino*. URL: https://www.thingiverse.com/thing:2575467 (visited in 2023).

[5] Raspberry Pi. *DATASHEET Raspberry Pi 4 Model B*. URL: https://datasheets.raspberrypi.com (visited in 2024).

[6] Taradaciuc Nicolae. *Bluetooth controlled smart car with obstacle avoiding capabilities*. Ed. by unpublished.

[7] Waveshare Electronics. *SIM8200EA-M2 5G HAT*. URL: https://www.waveshare.com/wiki/SIM8200EA-M2_5G_HAT (visited in 2024).

[8] Raspberry Pi. *Raspberry Pi Camera Module 2 NoIR documentation*. URL: https://www.raspberrypi.com/documentation/accessories/camera.html (visited in 2024).

[9] Dimitar Tanchev. *Holder for Paspberry PI camera module*. URL: https://www.thingiverse.com/thing:2158847 (visited in 2024).

[10] STMicroelectronics. *L298N Datasheet*. URL: https://pdf1.alldatasheet.com/datasheet-pdf/download/22440/STMICROELECTRONICS/L298N.html (visited in 2024).

[11] Handson technology. *Dual Shaft Mini DC Gear Motor Datasheet*. URL: https://handsontec.com/dataspecs/motor_fan/Dual%20Shaft%20Mini%20Gear%20Motor.pdf (visited in 2024).

[12] Varta Consumer Batteries. *Varta Datasheet*. URL: https://www.mega-piles.com/im/VARTA-6LR61-9V-580mAh-HIGH-ENERGY_12.pdf (visited in 2024).

[13] Anker. *Anker 20,000mAh High-Speed 22.5W Power Bank*. URL: https://www.anker.com/products/a1647?variant=42985134686358 (visited in 2024).

[14] Waveshare Electronics. *SIM8200EA-M2 Hardware Design Datasheet*. URL: https://files.waveshare.com/upload/d/de/SIM8200EA-M2_Hardware_Design_V1.03.pdf (visited in 2024).

[15] SIMCom Wireless Solutions Limited. *IM8200 Series_AT Command Manual*. (visited in 2024).

[16] ZeroTier. *ZeroTier Documentationt*. URL: https://docs.zerotier.com (visited in 2024).

[17] Pallets. *Flask Documentation*. URL: https://flask.palletsprojects.com/en/3.0.x (visited in 2024).

[18] Pygame. *Pygame Documentation*. URL: https://www.pygame.org/docs (visited in 2024).

[19] Dave Jones. *picamera Documentation*. URL: https://picamera.readthedocs.io/en/release-1.13/index.html (visited in 2024).

[20] Ben Croston. *RPi.GPIO 0.7.1*. URL: https://pypi.org/project/RPi.GPIO/ (visited in 2024).

[21] Dave Jones Ben Nuttall. *pigpio 1.78*. URL: https://pypi.org/project/pigpio/ (visited in 2024).

[22] Wikipedia contributors. *Xbox 360 controller*. URL: https://en.wikipedia.org/w/index.php?title=Xbox_360_controller&oldid=1224723554 (visited in 2024).

[23] nPerf. *nPerf Application*. URL: https://www.nperf.com/en/ (visited in 2024).

[24] Ericsson. *What is 5G RAN?* URL: https://www.ericsson.com/en/ran (visited in 2024).

[25] Raspberry Pi. *Using an ultrasonic distance sensor*. URL: https://projects.raspberrypi.org/en/projects/physical-computing/12 (visited in 2024).

[26] MakerPortal. *Distance Detection with the TF-Luna LiDAR and Raspberry Pi*. URL: https://makersportal.com/blog/distance-detection-with-the-tf-luna-lidar-and-raspberry-pi (visited in 2024).

Appendix

```
1
2  def controller_init(pygame_controller):
3
4      pygame_controller.init()
5      pygame_controller.joystick.init()
6      joystick_count = pygame_controller.joystick.get_count()
7
8      if joystick_count == 0:
9          print("No joysticks found.")
10         return None
11     else:
12         controller = pygame.joystick.Joystick(0)
13         controller.init()
14         print(f"Joystick found: {controller.get_name()}")
15
16         return controller
17
18
19 def controller_read_input(pygame_controller, controller, prev_axes,
       prev_buttons, prev_hat):
20
21     pygame_controller.event.get()
22     axes = [controller.get_axis(i) for i in
       range(controller.get_numaxes())]
23     buttons = [controller.get_button(i) for i in
       range(controller.get_numbuttons())]
24     hat = [controller.get_hat(0)[0]]
25
26     if (buttons != prev_buttons) or (axes != prev_axes) or (hat !=
       prev_hat):
27         ready_to_send = 1
28     else:
29         ready_to_send = 0
30
31     return axes, buttons, hat, ready_to_send
32
33
34 def controller_disconnect(pygame_controller):
35
36     pygame_controller.quit()
```

Code Listing 10.1: Controller Initialization, Input Reading, and Disconnection Functions

```
1
2  def client_socket_init():
3
4      host = '172.30.170.224'
5      port = 6789
6
7      client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8
```

```
    client_socket.connect((host, port))
    return client_socket


def client_socket_send(client_socket, axes):
    axes_str = ','.join((map(str, axes)))

    client_socket.sendall(axes_str.encode())


def client_socket_disconnect(client_socket):
    client_socket.close()
```

Code Listing 10.2: Client Socket Initialization, Data Transmission, and Disconnection Functions

```

def main():

    pygame_controller = pygame_init()
    client_socket = client_socket_init()
    controller = controller_init(pygame_controller)

    if controller is None:
        return

    prev_axes = [0.0] * controller.get_numaxes()
    prev_buttons = [0] * controller.get_numbuttons()
    hat_state = controller.get_hat(0)
    hat_state_length = len(hat_state)
    prev_hat = [0] * hat_state_length
    loop_on_off = True

    while loop_on_off:

        axes_controller, buttons_controller, hat_controller, ready_to_send
    = controller_read_input(pygame_controller, controller, prev_axes,
    prev_buttons, prev_hat)

        if ready_to_send == 1:

            hat_test = hat_controller[0]

            while hat_test != 0:
                axes_controller, buttons_controller, hat_controller,
    ready_to_send = controller_read_input(pygame_controller, controller,
    prev_axes, prev_buttons, prev_hat)

                just_needed_axes = axes_controller[:1] +
    axes_controller[4:] + hat_controller

                rounded_axes = [round(num, 3) for num in just_needed_axes]
                client_socket_send(client_socket, rounded_axes)
                hat_test = hat_controller[0]
                print(hat_test)
                sleep(0.1)

            just_needed_axes = axes_controller[:1] + axes_controller[4:] +
    hat_controller

            rounded_axes = [round(num, 3) for num in just_needed_axes]
```

```
            client_socket_send(client_socket, rounded_axes)
            print(rounded_axes)
            prev_buttons = buttons_controller
            prev_axes = axes_controller
            prev_hat = hat_controller

            if any(buttons_controller):
                loop_on_off = False

    controller_disconnect(pygame_controller)
    client_socket_disconnect(client_socket)
```

Code Listing 10.3: Main Function: Initialization, Input Reading, and Data Transmission

```

def Start_Buzzer():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Buzzer_pin, GPIO.OUT)
    pwm = GPIO.PWM(Buzzer_pin, 1000)
    pwm.start(50)
    sleep(1)
    pwm.stop()

def servo_init_steering():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Motor_servo_steering, GPIO.OUT)
    GPIO.output(Motor_servo_steering, True)

    global pwm_servo_steering
    pwm_servo_steering = pigpio.pi()
    pwm_servo_steering.set_mode(Motor_servo_steering, pigpio.OUTPUT)
    pwm_servo_steering.set_PWM_frequency(Motor_servo_steering,50)

    global max_servo_range_steering
    max_servo_range_steering = 2300
    global min_servo_range_steering
    min_servo_range_steering = 1200

def servo_init_camera():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Motor_servo_camera, GPIO.OUT)
    GPIO.output(Motor_servo_camera, True)

    global pwm_servo_camera

    pwm_servo_camera = pigpio.pi()
    pwm_servo_camera.set_mode(Motor_servo_camera,pigpio.OUTPUT)
    pwm_servo_camera.set_PWM_frequency(Motor_servo_camera,50)

    global max_servo_range_camera
    max_servo_range_camera = 2500
    global min_servo_range_camera
    min_servo_range_camera = 800

    global camera_angle
    camera_angle=1700
    setAngle_camera(camera_angle)


def motor_init():
```

```
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(MotorA_in1, GPIO.OUT)
    GPIO.setup(MotorA_in2, GPIO.OUT)
    GPIO.setup(MotorA_enable, GPIO.OUT)
    GPIO.setup(MotorB_in1, GPIO.OUT)
    GPIO.setup(MotorB_in2, GPIO.OUT)
    GPIO.setup(MotorB_enable, GPIO.OUT)

    global pwmA
    global pwmB
    pwmA = GPIO.PWM(MotorA_enable, PWM_Frequency)
    pwmB = GPIO.PWM(MotorB_enable, PWM_Frequency)
    pwmA.start(0)
    pwmB.start(0)
```

Code Listing 10.4: Initialization Functions for Motors, Servo Motors and Buzzer

```

def server_socket_init():
    host = '172.30.170.224'
    port = 6789

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    server_socket.bind((host, port))

    server_socket.listen()

    print("Server listening on", host, "port", port)

    client_socket, client_address = server_socket.accept()
    print("Connection from:", client_address)

    return server_socket, client_socket


def server_socket_receive(client_socket):

    data = client_socket.recv(1024).decode()
    return data


def server_socket_close_connection(client_socket):

    client_socket.close()


def parse_data(data):
    parsed_data = data.strip().split(',')
    try:
        useful_data_servo_steering=
    float(parsed_data[0].strip().rstrip('.'))
        useful_data_f= float(parsed_data[-2].strip().rstrip('.'))
        useful_data_r = float(parsed_data[-3].strip().rstrip('.'))
        useful_data_servo_camera =
    float(parsed_data[-1].strip().rstrip('.'))
        direction = 1
        if useful_data_r > -1:
```

```
            direction = 0
            return useful_data_r, direction, useful_data_servo_steering,
    useful_data_servo_camera

        return useful_data_f, direction, useful_data_servo_steering,
    useful_data_servo_camera
    except:
        print("Error while parsing in parse_data")
```

Code Listing 10.5: Server Socket Initialization, Data Reception, Disconnection, and Data Parsing Functions

```

def generate_frames():

    with picamera.PiCamera() as camera:
        camera.resolution = (640,480)
        camera.framerate = 15
        camera.vflip = True
        camera.hflip = True
        stream = io.BytesIO()

        for _ in camera.capture_continuous(stream, 'jpeg',
    use_video_port=True):

            stream.seek(0)
            yield b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' +
    stream.read() + b'\r\n'

            stream.seek(0)
            stream.truncate()

@app.route('/')
def index():
    return Response(generate_frames(),
    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Code Listing 10.6: Frame Generaion and Video Streaming Functions

```

def motor_direction(direction, acceleration):

    if direction == 1:
        GPIO.output(MotorA_in1, GPIO.HIGH)
        GPIO.output(MotorA_in2, GPIO.LOW)
        GPIO.output(MotorA_enable, GPIO.HIGH)
        GPIO.output(MotorB_in1, GPIO.HIGH)
        GPIO.output(MotorB_in2, GPIO.LOW)
        GPIO.output(MotorB_enable, GPIO.HIGH)

        pwmA.ChangeDutyCycle(acceleration)
        pwmB.ChangeDutyCycle(acceleration)

    if direction == 0:
        GPIO.output(MotorA_in1, GPIO.LOW)
        GPIO.output(MotorA_in2, GPIO.HIGH)
        GPIO.output(MotorA_enable, GPIO.HIGH)
        GPIO.output(MotorB_in1, GPIO.LOW)
        GPIO.output(MotorB_in2, GPIO.HIGH)
```

```
        GPIO.output(MotorB_enable, GPIO.HIGH)
        try:
            pwmA.ChangeDutyCycle(acceleration)
            pwmB.ChangeDutyCycle(acceleration)
        except:
            print("Error pwm")


def all_motor_stop():
    pwm_servo_steering.stop()
    pwm_servo_camera.stop()
    pwmA.stop()
    pwmB.stop()
    GPIO.cleanup()


def transform_acceleration(value):
    acc_value = (value + 1) * 50
    return acc_value
```

Code Listing 10.7: Motor Direction Control, Stop and Acceleration Transformation Functions

```

def setAngle_steering(angle):
    duty = angle / 18 + 2
    pwm_servo_steering.set_servo_pulsewidth(Motor_servo_steering,angle)

def setAngle_camera(angle):
    duty = angle / 18 + 2
    pwm_servo_camera.set_servo_pulsewidth(Motor_servo_camera,angle)



def transform_servo_angle_steering(value_servo):
    old_range=(1- (-1))
    new_range=(max_servo_range_steering - min_servo_range_steering)
    angle= (((value_servo -
   (-1))*new_range)/old_range)+min_servo_range_steering

    angle=(max_servo_range_steering - (angle -min_servo_range_steering))

    return angle

def transform_servo_angle_camera(value_servo):
    global camera_angle
    step_size = 70

    if value_servo == -1:
        camera_angle += step_size
    elif value_servo == 1:
        camera_angle -= step_size

    camera_angle = max(min(camera_angle,max_servo_range_camera),
   min_servo_range_camera)
```

Code Listing 10.8: Servo Angle Control and Transformation Functions

```
1
2  def main():
3
4      sleep(60)
5      Start_Buzzer()
6
7      sever_socket, client_socket = server_socket_init()
8      sleep(1)
9      motor_init()
10     servo_init_steering()
11     servo_init_camera()
12
13     var_loop_on_off = True
14     var = False
15     while var_loop_on_off:
16
17         data = server_socket_receive(client_socket)
18         if not data:
19             break
20         try:
21             useful_data, direction, servo_angle_steering,
       servo_angle_camera = parse_data(data)
22
23             acceleration = transform_acceleration(useful_data)
24             motor_direction(direction, acceleration)
25
26             transformed_angle_steering =
       transform_servo_angle_steering(servo_angle_steering)
27
28             setAngle_steering(transformed_angle_steering)
29
30             transform_servo_angle_camera(servo_angle_camera)
31             setAngle_camera(camera_angle)
32             print(camera_angle)
33             sleep(0.1)
34
35         except:
36             print("Error while parsing(main)")
37
38         if var == True:
39             var_loop_on_off = False
40             motor_stop()
41
42     all_motor_stop()
43     server_socket_close_connection(client_socket)
44
45 if __name__ == '__main__':
46
47     t1 = threading.Thread(target=app.run, kwargs={'host':'0.0.0.0',
       'port':6700,'threaded':True})
48
49     t1.start()
50
51     t2=threading.Thread(target=main)
52     t2.start()
```

Code Listing 10.9: Main function: Initialiation, Data Processing, and Thread Management