ROBOT DRIVER DIGITAL AUGMENTED REALITY VIEW

Submitted to the Department of Electrical Engineering,
University of Cape Town, in partial fulfilment of the requirements
for the degree of

BACHELOR OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

at the

UNIVERSITY OF CAPE TOWN

by

Kudzaishe Kadzimu

SUPERVISED BY:

PROF. SIMON WINBERG

October 20, 2024

ABSTRACT

This MSc project focuses on .. describe the system.

Can provide a few paragraphs, want around 300 - 600 words.

ACKNOWLEDGEMENTS

I would like to gladly express my gratitude to

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

- **ADC** – Analogue to Digital Converter
- **ASIC** – Application-specific Integrated Circuit

NOMENCLATURE

Comment: you do not need to have both nomenclature and abbreviations it tends to be redundant.

- **Analogue to digital Converter (ADC):** an electronic device that converts data from its analogue format to its digital form.
- **Very High Speed Integrated Circuits Hardware Description Language (VHDL):** a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as FPGA.

CHAPTER 1

INTRODUCTION

In the past decade, the integration of Augmented Reality (AR) into robotic systems has gained significant traction. Building upon this trend and previous work in robotics, this project aims to develop an innovative AR system for a wheeled robot. By combining AR technology with machine learning, we seek to enhance the robot's functionality and substantially improve the human-machine interaction experience. This fusion of technologies promises to push the boundaries of what's possible in robotic systems, offering new possibilities for both practical applications and user engagement.

1.1 BACKGROUND

The concept of autonomous vehicles dates back to the days of Leonardo Da Vinci's 16th-century designs, although practical implementations only emerged in the 1980s [3]. Radio-controlled (RC) cars, introduced earlier by Elettronica [4], evolved from combustion engines to electric motors, expanding their applications across various industries [5]. As shown in Figure 1.1, Da Vinci's early concept reflects humanity's long-standing fascination with autonomous transportation.

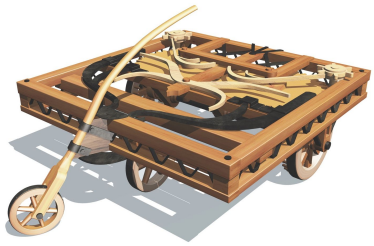


Figure 1.1: Model of autonomous car designed by Leonardo Da Vinci

The field of robotics and autonomous systems has undergone significant evolution since the days of Leonardo, from rudimentary remote-controlled machines to highly advanced, AI-driven systems. These systems are no longer limited to following pre-defined paths or responding to simple commands. Modern autonomous robots can sense, analyze, and adapt to their environ-

ment, making decisions in real-time[6]. This leap in capabilities demands more sophisticated interactions between humans and robots, where ease of communication and control is vital for efficiency and safety.

As we approach the Fifth Industrial Revolution (5IR), the focus shifts to a symbiotic relationship between humans and AI-powered robots, emphasizing workplace efficiency while maintaining a human-centric approach [6]. Augmented Reality (AR) plays a crucial role in this evolution, bridging the gap between humans and machines in areas such as manufacturing, healthcare, and human-robot interaction (HRI) [7].

In the context of autonomous systems, fiducial markers (e.g., ARToolKit, AprilTags, ArUco) have become integral to robotics and AR applications. These markers facilitate robot navigation, environmental mapping, and interaction zone definition. The choice of marker system depends on specific application requirements, available computational resources, and desired accuracy.

The integration of AR and fiducial markers in robotics enhances perception, navigation, and interaction capabilities, pushing the boundaries of human-robot collaboration and making autonomous robots more adaptable and intelligent.

1.2 PROBLEM DESCRIPTION

As robots continue to evolve so does our demand for more intuitive and seamless interaction with these robots. Traditional control methods often fall short when it comes to offering the flexibility and situational awareness needed to integrate robots effectively into dynamic, real-world environments. This project addresses several key challenges in the domain of mobile robot control and environmental interaction:

1. **Limited Contextual Awareness**: Current remote-controlled robots often operate with minimal understanding of their surroundings, leading to inefficient navigation and potential safety hazards in complex environments.

2. **Inflexible Control Mechanisms**: Many existing robot control systems rely on fixed command sets that do not adapt to changing environmental conditions or task requirements, limiting their versatility and usability.

3. **Absence of Dynamic Task Assignment**: Most mobile robots are pre-programmed for specific tasks and lack the ability to receive and interpret new instructions or environmental cues on the fly.

4. **Integration Challenges**: Incorporating augmented reality (AR) elements into robotic systems presents technical challenges in terms of real-time processing, accurate marker detection, and seamless information overlay.

1.3 OBJECTIVES

The primary objective of this project is to improve human-robot interaction (HRI) by integrating Augmented Reality (AR) technology with mobile robotics. Using AR and ArUco visual markers along with object detection measures, the robot will navigate and perform tasks in a more intuitive, human-centered way, enhancing both user experience and robot context-awareness.

This project focuses on developing a system that enables visual-based, real-time communication between the user and the robot. Through AR-enhanced interfaces and marker detection, the robot will perform context-aware tasks, with specific ”control zones” triggering robot behaviors, and providing real-time AR feedback to the user.

1.3.1 Main Objectives

1. **Control Zones Development**: AR-based zones will control robot behaviors, such as speed, task initiation, and directional changes, based on the detection of visual markers.
2. **Real-time Dynamic Instructions**: The robot will use ArUco markers for real-time commands, allowing it to navigate, execute tasks, and respond to environment-specific instructions.
3. **Environmental Mapping and Interaction**: Using visual markers, the robot will map its environment, track landmarks, avoid obstacles, and interact with objects.
4. **Enhanced User Engagement**: AR will allow users to visually monitor and guide the robot in real-time, creating a more interactive and immersive user experience.

1.3.2 Secondary Objectives

Additionally, two autonomous features will improve robot autonomy:

1. **Object Avoidance**: The robot will detect specific markers and avoid obstacles by maintaining a defined safe distance (e.g., 20 cm).
2. **Wheel Slip Detection**: The robot will detect terrain changes that may cause wheel slip and adjust its behavior based on power draw characteristics.

1.4 Scope and Limitations

This project was conducted under several significant constraints that shaped the scope of the research and development process. These constraints were primarily due to limitations in time, budget, and available resources, which directly impacted the scale of the implementation and the range of features that could be developed.

The project had a total budget of R2000, which restricted the procurement of high-end components and forced the use of readily available, cost-effective hardware. Additionally, the project timeline was set at three months, limiting the ability to explore more advanced functionalities and requiring a focused approach to key objectives. These constraints influenced both the design and testing phases of the project.

Furthermore, ethical considerations were taken into account, particularly in ensuring that no invasive or harmful testing methods were used. The project did not involve human or animal subjects, which helped to avoid potential ethical conflicts and the need for additional approvals.

Despite these limitations, the project successfully demonstrated the integration of AR in human-robot interaction. While the constraints limited the full realization of more advanced features, the focused approach allowed the core objectives to be met within the available resources and timeline. Further development could build upon this foundation to explore more complex functionalities and broader applications.

1.5 Plan of Development

This thesis is structured into seven chapters, each focusing on different aspects of the research and development process. The chapters build upon each other to present a coherent narrative from the conceptual framework to the implementation, testing, and conclusions of the project.

Chapter 2: Literature Review

Chapter 2 discusses the foundational technologies, theories, and techniques that underpin this project. It provides an overview of the state of the art in human-robot interaction (HRI), mobile robotics, and the use of Augmented Reality (AR) in robotics.

Chapter 3: Methodology

Chapter 3 presents the research methodology used for this project. It provides detailed explanations of the procedures, experimental setups, and tools used for the development and testing

of the system.

Chapter 4: Prototype Design

Chapter 4 focuses on the design of the system. It provides details on the hardware and software design choices, discussing how the various subsystems (e.g., AR integration, control zones, and dynamic instructions) were conceived and implemented.

Chapter 5: Modular Testing

Chapter 5 Describes the testing process, including independent module testing, subsystem combinations testing, and the full integration phase. It describes how each individual system in the project (hardware and software) was tested.

Chapter 6: Results and Analysis

Chapter 6 presents the results obtained from the overall system. These results will showcase how the robot interacted with AR markers and control zones, as well as its performance in dynamic environments. Data from the acceptance tests will be analyzed to assess the system's functionality and its success in meeting the project's objectives.

Chapter 7: Conclusion

Chapter 7 concludes the thesis by reflecting on the overall project, summarizing the key findings, and discussing the implications of the results. It will cover the challenges faced during the project, how they were addressed, and potential areas for improvement.

CHAPTER 2

LITERATURE REVIEW

This literature review provides a comprehensive exploration of the integration of Augmented Reality (AR) with mobile robotics, focusing on its applications in Human-Robot Interaction (HRI). The review is structured to cover several key areas that are crucial to the development of an AR framework for a 4WD robotic car.

The field of AR-Enhanced Mobile Robotics is a rapidly evolving interdisciplinary domain that combines elements from robotics, computer vision, human-computer interaction, and sensor technologies. This review aims to synthesize current knowledge and identify gaps in the existing literature, providing a solid foundation for the present research project.

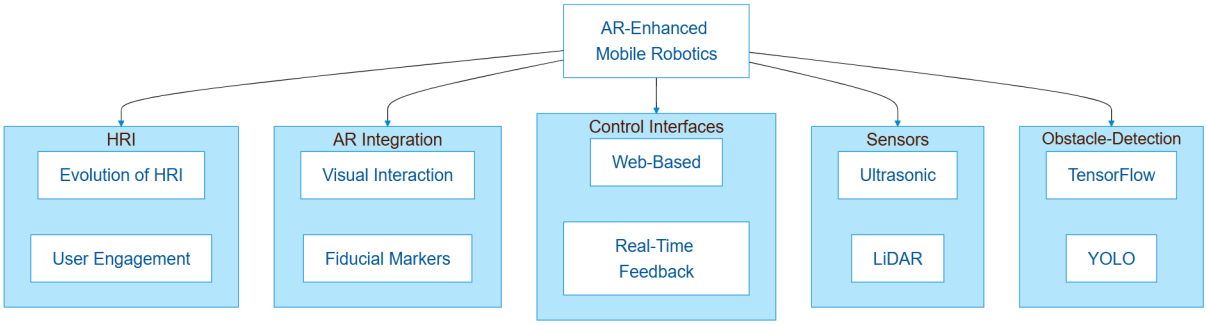


Figure 2.1: Overview of the structure of the literature review

Figure 2.1 illustrates the structure and main topics covered in this literature review. The diagram presents a hierarchical breakdown of the key areas that constitute AR-Enhanced Mobile Robotics Each of these main areas is further divided into subtopics, allowing for a detailed examination of specific technologies, methodologies, and their applications in AR-enhanced mobile robotics.

Throughout this review, I will critically analyze key research papers, industry reports, and state-of-the-art implementations I will discuss current challenges, emerging trends, and potential future directions in AR-enhanced mobile robotics. By the end of this literature review, readers will have a thorough understanding of the current state of AR in mobile robotics, the challenges faced in implementing such systems, and the potential benefits they offer in enhancing human-robot interaction and overall system functionality.

2.1 Introduction to Human-Robot Interaction (HRI) and Augmented Reality (AR)

Human-Robot Interaction (HRI) has seen substantial growth in both research and industrial applications, particularly with the advent of collaborative robots (cobots) that work alongside humans in dynamic environments. These robots, unlike their predecessors, are designed to safely interact with humans in shared workspaces, marking a significant shift in industrial automation.

According to [8], HRI evolved from isolated robot systems in manufacturing to collaborative systems where robots assist humans in complex tasks. This shift is largely due to advancements in sensing, control, and human-machine interface technologies that allow robots to perceive their environment and interact intelligently.

Figure 2.2: Screen Display of work environment with AR additions [1]

Augmented Reality (AR) has emerged as a powerful tool in enhancing HRI by providing a more intuitive and efficient way for humans to interact with robots. AR serves as a bridge that overlays digital information onto the physical environment, enabling users to better understand the robot's actions and the task at hand [9].

2.1.1 Virtual Reality in Human-Robot Interaction: A Comparison with Augmented Reality

Virtual Reality (VR) and Augmented Reality (AR) have both played significant roles in advancing Human-Robot Interaction (HRI). However, their applications and interaction paradigms differ significantly. Understanding these differences is essential when analyzing how each technology enhances the interaction between humans and robots.

VR offers a fully immersive experience where users interact with robots in a completely simulated world. This makes it particularly effective for applications such as robotic training, simulation, and remote teleoperation. For instance, VR is widely used for training operators and robots in new tasks in a risk-free virtual environment [10]. By immersing the user in a fully virtual environment, VR allows for complex robot interactions, including simulated task execution and environment navigation, which is especially useful in fields such as disaster response, industrial automation, and surgical simulations [11].

AR, on the other hand, overlays digital information onto the real world, making it more suitable for real-time, operational tasks involving physical robots. AR enables users to view critical information such as navigational data, sensor outputs, or planned robot trajectories without losing connection to their real-world environment [12]. This makes AR ideal for applications where humans and robots share the same workspace, such as collaborative manufacturing, logistics, and maintenance tasks.

While both technologies are beneficial, this literature review will primarily focus on Augmented Reality due to its practical applications in human-robot collaboration.

2.1.2 Augmented Reality as a Tool in HRI

AR enhances communication between humans and robots by offering real-time visualizations and feedback, thus improving situational awareness and reducing errors in task execution. Suzuki et al. [9] propose a taxonomy of AR-enhanced HRI, highlighting the key areas where AR plays a role, such as task guidance, real-time interaction, and environment mapping. In particular, AR's ability to provide visual feedback significantly enhances the user experience by making robot operations more transparent and reducing cognitive load.

Moreover, AR facilitates dynamic task interaction by enabling users to provide real-time instructions to the robot. This is particularly beneficial in industrial applications where task conditions change frequently. AR not only improves task accuracy but also contributes to a more efficient workflow, as robots can quickly adjust their behavior based on user commands and visual markers. [9].

AR's applications in HRI span various industries, from manufacturing to healthcare. In manufacturing, AR has been utilized to guide robot operations through real-time visual overlays, improving precision and safety in tasks that require close human-robot collaboration [8]. Similarly, in healthcare, AR assists medical robots in performing delicate procedures by providing real-time feedback and guidance, enhancing both safety and operational efficiency [9].

2.2 MOBILE ROBOTICS AND AR INTEGRATION

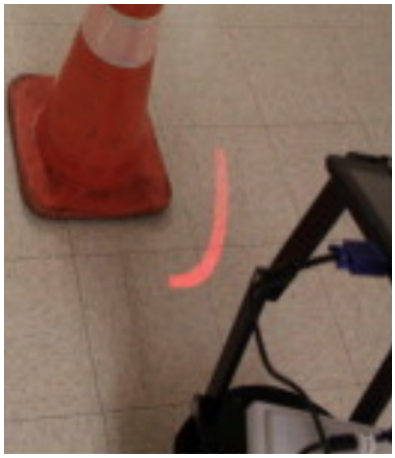
The integration of Augmented Reality (AR) in mobile robotics has transformed how robots communicate their intentions and interact with their environment. AR not only enhances human-robot collaboration but also provides an intuitive interface for users to understand robot behavior in dynamic and complex environments. Several key studies have explored the use of AR to improve robot navigation, task execution, and real-time interaction between humans and robots.

2.2.1 AR as a Communication Tool in Mobile Robotics

One of the primary challenges in human-robot interaction is communicating the robot's intended actions to human operators. Spatial Augmented Reality (SAR) has proven to be an effective method for achieving this. In the study by Green et al. [13], SAR is used to project information about the robot's intended movement directly onto the physical environment. This method allows users to better anticipate and respond to the robot's actions, reducing uncertainty and enhancing collaboration. Michalos et al. [14] demonstrated the application of AR in industrial settings, where visual feedback aids in task coordination between humans and robots, leading to improved task accuracy and faster response times.

Additionally, Coovert et al. [15] explored how spatial AR techniques can improve communication between robots and humans. In their experiment, a robot used visual projections of arrows and simplified maps on the floor to indicate its short-, mid-, and long-term movement intentions. The study found that participants were able to predict the robot's movements with high confidence when the robot projected its intended path. This is especially useful in environments where humans and robots share a workspace, such as hospitals, museums, and factories.

(a) Robot estimating its intended movement. [15].



(b) Robot estimating its intended movement using AR arrows [15].

2.2.2 Multimodal AR Interfaces for Human-Robot Interaction

In addition to visual feedback, multimodal interfaces that combine AR with other modalities, such as voice or gesture recognition, have further improved human-robot interaction. Green et al. [13] developed a multimodal AR interface for mobile robots, allowing users to interact with the robot through visual overlays, voice commands, and gesture-based inputs. This interface improved the efficiency and accuracy of task execution in collaborative environments. The study demonstrated that multimodal AR interfaces allow for more flexible and adaptive human-robot communication, making the interaction process more intuitive and reducing cognitive load on the user.

2.2.3 Applications of AR in Mobile Robotics

AR's integration into mobile robotics is particularly useful in environments that require constant adaptation, such as manufacturing or logistics. Michalos et al. [14] applied AR in human-robot cooperation in industrial settings, where robots must adapt to rapidly changing conditions. AR provided real-time feedback to users, helping them coordinate with robots for tasks such as assembly or quality control. Moreover, AR visualizations helped robots better navigate dynamic environments, improving both safety and efficiency.

Furthermore, AR's ability to create real-time visual overlays in the robot's workspace has significant implications for task planning and execution. In particular, SAR-based interfaces allow users to visually instruct robots by marking control zones or highlighting obstacles, which the robot then interprets for navigation and task completion [15].

2.2.4 Challenges and Future Directions

While AR's role in mobile robotics has demonstrated significant potential, challenges remain in real-time processing, accurate object recognition, and user feedback latency. As Michalos et al. [14] noted, the integration of AR into robotic systems requires further advancements in sensor technology and computational power to ensure real-time responsiveness in dynamic environments. Future research will likely focus on improving AR's scalability in complex settings, refining multimodal interfaces, and enhancing the autonomy of mobile robots to handle more sophisticated tasks.

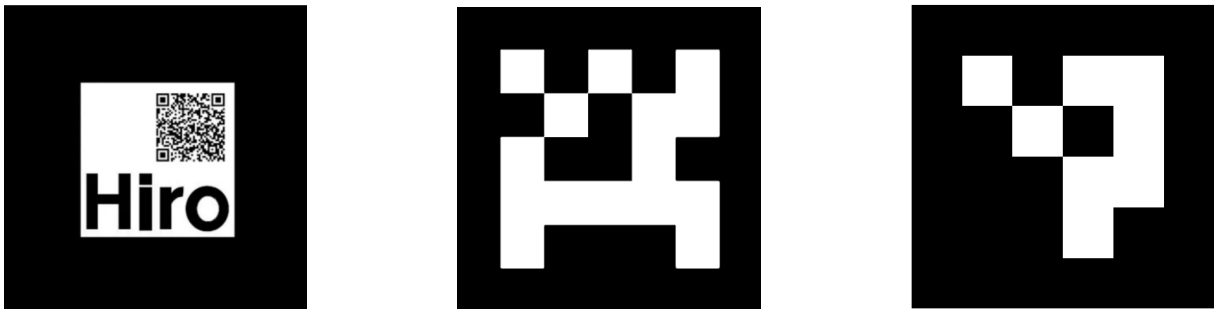
2.3 FIDUCIAL MARKER SYSTEMS IN HRI

Augmented Reality (AR) is increasingly used in robotics to facilitate various types of Human-Robot Interactions (HRI). While multiple interaction modalities have been discussed, the remainder of this literature review will focus specifically on visual and marker-based interactions. Below is a list of the different types of interactions possible with AR:

- **Visual Interaction**: AR overlays visual cues such as control zones, instructions, and real-time status feedback, enhancing human comprehension of the robot's operational environment.
- **Gesture-Based Interaction**: Human gestures can be interpreted by robots through AR-enhanced vision systems, reducing reliance on manual interfaces.
- **Speech and Command Interaction**: AR, combined with voice recognition, allows users to issue commands to robots with visual aids providing additional context.
- **Fiducial Marker-Based Interaction**: Fiducial markers such as ArUco or AprilTags provide robots with a way to interpret their environment, helping with localization, navigation, and object detection.

2.3.1 Fiducial Marker Systems Overview

In AR-enhanced HRI, fiducial marker systems are essential for real-time contextual interactions. These systems use visually distinct markers that robots detect through cameras. The most commonly used marker systems include ARToolKit, AprilTags, and ArUco markers, each with unique characteristics suited to various applications.



(a) ARToolKit (b) AprilTags (c) ArUco markers

Figure 2.4: Visual comparison of different marker systems

ARToolKit is one of the earliest fiducial marker systems developed for augmented reality applications. It was widely adopted in early AR systems due to its simplicity and moderate computational requirements. In HRI, ARToolKit has been used in early experiments involving robot navigation and control. However, its robustness is limited compared to more modern systems, and it struggles under challenging lighting conditions [9].

AprilTags are an improvement over ARToolKit, offering significantly higher robustness and detection accuracy. As [16] highlights, AprilTags are designed to provide accurate detection even in adverse environments with variable lighting and occlusion. AprilTags have been extensively used in industrial applications where precise localization is critical, such as warehouse automation and autonomous guided vehicles (AGVs). The system is computationally heavier than ARToolKit but provides superior reliability, making it suitable for applications that demand accuracy over speed [17].

ArUco markers are designed with computational efficiency in mind, offering high detection speed without sacrificing much in terms of accuracy. This makes them ideal for real-time applications, particularly in mobile robotics where swift responses are critical. [18] notes that ArUco markers have been used in various applications including navigation and task execution in dynamic environments such as warehouses. Their simplicity and widespread software support make them a popular choice for developers working on real-time robotic systems.

Table 2.1: Comparison of Fiducial Marker Systems

Marker System	**Robustness**	**Detection Accuracy**	**Computational Efficiency**	**Applications**
ARToolKit	Moderate	Good	Moderate	Early AR applications
AprilTags	High	Excellent	Moderate-High	Robotics
ArUco Markers	High	Good	High	Real-time robot navigation

2.3.2 Choosing the Best Marker System for the Project

Given the requirements of the project, which involve real-time processing, dynamic navigation, and user interaction, ArUco markers are the most suitable choice. As [17] demonstrates, ArUco markers provide a good balance between computational efficiency and detection accuracy, making them ideal for mobile robotic systems that require fast decision-making. Additionally, the ease of integration with commonly used robotics software such as ROS (Robot Operating System) further justifies their selection for this project.

2.3.3 Visual Interaction in AR-Enhanced HRI

Visual interaction is one of the most common and effective forms of interaction in Augmented Reality (AR)-enhanced Human-Robot Interaction (HRI). It enables users to interact with robots by visualizing contextual information directly within their physical environment. This can include task instructions, control zones, and real-time feedback on the robot's status and performance.

In AR-enhanced visual interaction, the robot's actions and intentions are augmented with overlays that help the human user make informed decisions during interaction. [9] provides a comprehensive overview of how visual interaction enhances HRI by improving situational awareness and reducing the complexity of controlling robotic systems. This is especially useful in industrial applications, where workers interact with robots to execute precise tasks, such as assembly and quality control.

2.3.3.1 Applications of Visual Interaction

Visual interaction in AR has broad applicability across various fields, with significant benefits in improving task accuracy, communication, and overall efficiency. Some notable applications include:

- **Manufacturing and Assembly**: In industries such as automotive and aerospace, visual interaction is used to overlay task instructions on the assembly line, helping workers follow complex procedures. This reduces human error and increases productivity by providing step-by-step visual guidance [16].

- **Healthcare**: In robotic surgery and rehabilitation, visual interaction allows surgeons and therapists to see real-time data overlaid on patients, enhancing precision and safety. AR enhances the accuracy of procedures by showing virtual boundaries and guidelines, ensuring that robots move correctly within the workspace [18].

- **Warehouse and Logistics**: In logistics and warehouse management, AR systems provide real-time visual cues to workers handling robotic systems for item sorting, picking, and delivery. [18] explores the use of AR to improve warehouse efficiency by overlaying instructions on specific zones, helping workers interact more intuitively with robotic arms and autonomous guided vehicles (AGVs).

2.3.3.2 Visual Feedback and Real-Time Interaction

One of the most significant advantages of AR-enhanced visual interaction is the provision of real-time feedback, which facilitates more intuitive robot control. [17] investigates the real-time testing of vision-based systems in AGVs using ArUco markers and AR visualizations. Their research highlights how visual feedback improves decision-making in dynamic environments, helping robots navigate autonomously with greater efficiency.

Visual feedback can take various forms:

- **Control Zones**: Specific areas within the robot's environment are highlighted using AR, guiding the robot's actions and helping the user visualize where the robot can move or interact with objects.
- **Status Updates**: Information such as battery levels, current tasks, or environmental hazards can be projected onto the robot's workspace, allowing users to quickly assess and adjust the robot's behavior as needed.
- **Error Detection**: AR systems can visually highlight potential errors in the robot's movements, such as collisions, off-course navigation or wheel slipping, enabling quick corrective actions.

Suzuki et al. [9] emphasizes that visual interaction not only enhances HRI but also promotes collaboration between human operators and robots by reducing cognitive load. By providing real-time visual cues, AR improves situational awareness and makes controlling robots more intuitive, particularly in environments where precision is critical.

However, there are certain limitations. [16] discusses the challenges of accurately overlaying visual data in real-time, particularly in environments with inconsistent lighting, reflections, or occlusions. Additionally, the processing demands of rendering real-time visualizations may limit the performance of systems with constrained computational resources.

2.4 AR'S ROLE IN IMPROVING USER ENGAGEMENT

Augmented Reality (AR) has significantly enhanced user engagement with robotic systems, particularly in Automated Guided Vehicles (AGVs). By providing intuitive visual feedback and real-time streaming data, AR improves both the psychological and practical interaction between humans and AGVs. The combination of augmented reality and robotics has been a topic of

significant research interest, as AR makes robots more intuitive to control and increases user trust in automated systems.

2.4.1 Enhancing User Engagement through Real-Time Feedback and Visualization

One of the key strengths of AR is its ability to provide real-time feedback and visualizations, allowing users to intuitively understand the robot's actions and status. AR-infused streaming technology improves human-robot collaboration by projecting real-time data, control zones, and navigation paths onto the robot's environment [19]. This continuous stream of real-time data reduces cognitive load, enabling users to anticipate and control AGV movements more effectively in dynamic environments. Such real-time visualization allows users to engage more deeply with robotic systems, increasing their confidence in controlling AGVs.

In particular, [19] explore how AR is utilized across different robotic applications, including medical, industrial, and social robotics. For AGVs, they highlight AR's capability to enhance human-robot collaboration by improving real-time task allocation and decision-making processes. The visual feedback offered by AR systems reduces the need for constant manual adjustments, thereby increasing operator efficiency [19].

In industrial scenarios, [20] emphasize the role of AR in improving user engagement through its intuitive interfaces. They note that AR helps operators visualize tasks more clearly, thus reducing the learning curve for new users while providing advanced feedback for experienced operators. This is important in environments where quick decisions are necessary [20].

2.4.2 Psychological and Practical Benefits

The psychological impact of AR on user engagement is considerable. Real-time visualizations, such as path projections or task instructions, create a more predictable and transparent interaction between humans and robots. Users become more confident when they can clearly see an AGV's next move, which reduces uncertainty about the robot's behavior and enhances user trust. This **increased predictability** improves user satisfaction and promotes wider adoption of AR-enabled robotic systems [15].

AR also offers practical benefits in rehabilitation settings, where users can engage with robotic systems in therapy. As Makhataeva [20] points out, AR interfaces allow users to interact with AGVs or rehabilitation robots in a natural way, improving patient recovery outcomes. The immersive nature of AR encourages patient involvement, which is crucial in therapeutic contexts

[20].

2.5 WEB-BASED CONTROL INTERFACES

Having a functional robot is all well and good but if the robot has an unpleasant or inefficient interface then it ruins the whole user experience. This section will detail different methods of implementing web-based control interfaces, focusing on real-time communication, user interaction, and video streaming.

2.5.1 Webserver Framework Options

Flask is a lightweight Python web framework known for its simplicity and flexibility[21]. It is particularly well-suited for the rapid development of web-based control interfaces. Flask is easy to set up and use, making it ideal for prototyping and small to medium-sized applications. It integrates well with Python libraries commonly used in robotics. However, for real-time features, additional libraries may be required, and its performance can be a concern for large-scale applications.

Node.js is a JavaScript runtime built on Chrome's V8 engine, recognized for its event-driven, non-blocking I/O model. Its asynchronous nature makes it highly suitable for real-time applications [22]. Node.js also benefits from a large ecosystem of packages and libraries and is capable of efficiently handling many simultaneous connections. However, it may require more setup compared to Flask, and its callback-based programming can be complex for beginners.

WebSocket integration is essential for real-time control and data streaming, providing low-latency, bidirectional communication over a single TCP connection [23]. This is particularly advantageous when sending control commands and receiving video streams. WebSockets can be integrated with both Flask and Node.js, though careful implementation is required to handle issues such as connection drops and reconnections, and additional security measures may be necessary.

Apache and Nginx are traditional web servers that are robust, highly scalable, and performant, making them popular choices in production environments. They offer extensive documentation and strong security features. However, they may be more complex to set up and configure, and can be overkill for smaller projects or embedded systems like Raspberry Pi [24]. They are typically used in conjunction with application servers rather than for standalone dynamic content.

Flask or Node.js, combined with WebSocket integration, would likely be the most suitable choices due to their ease of use and real-time capabilities. For this project Flask will be implemented due to its ease of understanding for a beginner.

2.5.2 Security and User Experience

While security is a crucial aspect of any robotic system, especially those controlled through web-based interfaces, it's important to note that for the scope of my project, I've chosen not to delve deeply into privacy and security concerns. This decision allows me to focus more intensively on the core functionality and user experience aspects of the system.

In a real-world application, however, it would be critical to implement robust security measures. These would typically include encryption protocols like TLS (Transport Layer Security) to protect the integrity and confidentiality of data exchanged between the robot and the user interface. Additionally, strong authentication mechanisms would be necessary to prevent unauthorized access to the robot's controls.

2.6 SENSOR TECHNOLOGIES

Sensors serve as the robot's eyes and ears, providing crucial data that allows for effective navigation, obstacle avoidance, and task execution. This section will explore various sensor technologies commonly employed in mobile robotics, highlighting their functionalities, applications, and significance in enabling autonomous operations along with object detection.

2.6.1 Ultrasonic Sensors

Ultrasonic sensors are a common choice in mobile robotics due to their effectiveness in obstacle avoidance and distance measurement. These sensors emit ultrasonic waves and measure the time it takes for the waves to reflect back from an object, allowing the robot to calculate the distance from obstacles [25]. This capability provides spatial awareness that is critical for safe navigation. Below is a basic diagram showcasing the operation of an HC-SR04 Infrared Sensor.

Below is a comparison of some common ultrasonic sensors used in mobile robotics.

Ultrasonic sensors are advantageous because of their simplicity, affordability, and ease of integration. For this project, they could provide an excellent mechanism for detecting nearby obstacles, which is essential for navigation in dynamic environments. Some key properties of

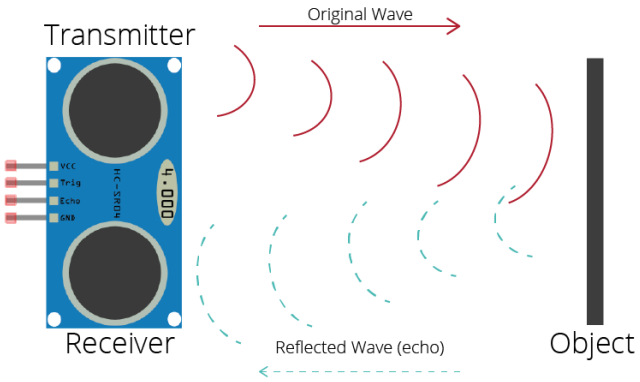


Figure 2.5: Operation of an Ultrasonic Sensor [2]

Table 2.2: Comparison of Ultrasonic Sensors

Model	Range (cm)	Operating Voltage (V)	Accuracy	Field of View	Size (mm)
HC-SR04	2-400	5	±0.3 cm	15 degrees	45 x 20 x 15
Maxbotix MB1000	20-645	2.5-5.5	±1%	42 degrees	22 x 20 x 16
Parallax PING)))	3-300	5	±0.5 cm	20 degrees	22 x 46 x 16

ultrasonic sensors include:

Integrating ultrasonic sensors with augmented reality (AR) offers the possibility of visualizing sensor data in real time. For example, AR can highlight areas where the robot detects obstacles, providing the operator with enhanced spatial awareness, preventing collisions, and improving task efficiency. Additionally, real-time data streaming through a visual interface allows for immediate adjustments to the robot's behavior based on sensor readings.

2.6.2 LiDAR Sensors

LiDAR (Light Detection and Ranging) is an essential technology for mobile robotics that uses laser light to measure distances and create 3D maps of environments. By utilizing LiDAR sensors, mobile robots can detect obstacles, map their surroundings, and navigate autonomously [26].

For this review, two LiDAR sensors will be considered although they will not be considered for the project itself: **RPLIDAR A1** and **TF Mini LiDAR (ToF) Laser Range Sensor V2.0**. These sensors offer capabilities that can significantly enhance the robot's ability to perceive and interact with its environment. Below is a comparison of their specifications [26]:

The **RPLIDAR A1** is a 360-degree scanning LiDAR sensor capable of creating a comprehen-

sive map of the robot's surroundings in real time. This would enable the robot to detect objects and obstacles from all directions, making it ideal for use in complex environments where autonomous navigation is crucial.

Table 2.3: Comparison of RPLIDAR A1 and TF Mini LiDAR (ToF) Sensors

Specification	**RPLIDAR A1**	**TF Mini LiDAR (ToF)**
Max Range	12m	12m
Resolution	0.2cm	1cm
Field of View	360° (horizontal)	3.6° (horizontal)
Sample Rate	8000 samples/sec	1000 samples/sec
Power Consumption	5V, 500mA	5V, 140mA
Interface	UART, USB	UART
Size (mm)	97.5 x 60 x 38	42 x 15 x 16

On the other hand, the **TF Mini LiDAR (ToF)** sensor, although more limited in its field of view, offers a compact and lightweight design, making it suitable for applications requiring focused, high-speed distance measurement. Its small size and low power consumption could be beneficial for tasks where precise distance measurements are needed without adding significant weight or power demands.

Both sensors would improve the robot's environmental mapping and obstacle detection capabilities. Visualizing the LiDAR data in an AR interface would provide real-time feedback to users, enabling them to monitor the robot's surroundings and make informed decisions about its movement.

2.7 Object Detection Algorithms

Apart from using physical dedicated sensors we are also capable of detecting objects and learning information from just using a camera as input. Below I discuss several models that allow us to enable real-time obstacle detection and navigation.

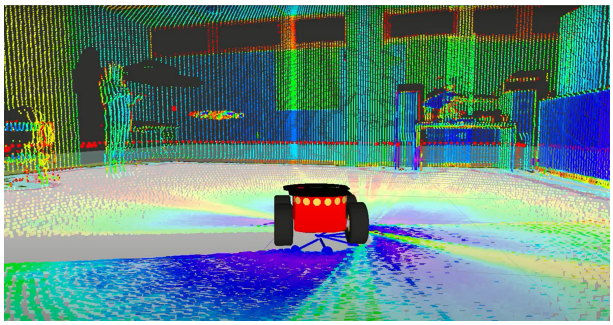


Figure 2.6: LiDAR scanning and mapping process in mobile robotics.

Haar Cascades

Haar cascades are one of the earlier object detection algorithms, initially introduced by Viola and Jones in 2001 [27]. This method utilizes features, such as edges and textures, to detect objects, traditionally used for face detection but also applicable in real-time environments where computational efficiency is a priority. Haar cascades, while fast, suffer from lower accuracy and can generate false positives, especially in complex environments [28]. Despite these limitations, Haar cascades remain an option for applications where speed is more critical than accuracy.

MobileNet SSD

MobileNet SSD (Single Shot Multibox Detector) is a more advanced architecture designed for embedded systems with limited computational power. It strikes a balance between speed and accuracy, making it suitable for real-time object detection on Raspberry Pi platforms [29]. MobileNet SSD has been used extensively in applications like obstacle detection for autonomous robots, offering a compromise between performance and resource efficiency. Its ability to process images quickly without sacrificing too much precision makes it a popular choice in resource-constrained environments.

TensorFlow Lite

TensorFlow Lite models are specifically optimized for mobile and embedded devices. These models perform well in real-time applications due to their efficient inference times and lower memory usage, while maintaining a reasonable level of accuracy [30]. TensorFlow Lite offers additional advantages when quantized models are used, further reducing the computational load while maintaining accuracy suitable for AGV obstacle detection.

YOLOv5

YOLO (You Only Look Once) models are highly regarded for their ability to provide accurate real-time object detection. YOLOv5, especially in its smaller variants such as YOLOv5s and YOLOv5n, has been optimized for faster processing while maintaining high accuracy levels [31]. YOLOv5s provides superior accuracy but may struggle with computational limitations on devices like the Raspberry Pi, making it less ideal for applications where real-time performance is crucial without hardware acceleration. In contrast, YOLOv5n (nano), being a lighter version of the model, offers a good balance of speed and accuracy, which is crucial for resource-limited platforms like the Raspberry Pi [32].

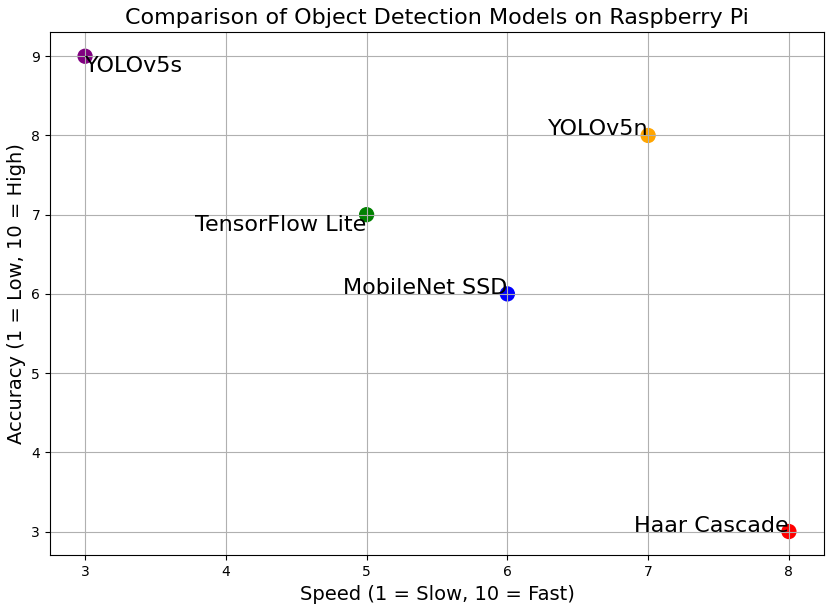


Figure 2.7: Comparison of Object Detection Models on Raspberry Pi

Given the balance between speed and accuracy required for real-time obstacle detection on a Raspberry Pi, YOLOv5n emerges as the most suitable choice. While it does not reach the same accuracy as YOLOv5s, its significantly improved speed makes it a better fit for autonomous navigation tasks where real-time performance is critical [32]. YOLOv5n provides a solid middle ground that enables efficient obstacle detection without introducing latency, which is essential in AGV or robot car applications.

2.8 Conclusion

This literature review has explored the integration of Augmented Reality (AR) with mobile robotics, focusing on Human-Robot Interaction (HRI). Key areas such as fiducial marker systems like ArUco markers, sensor technologies, web-based control interfaces and object detection algorithms were examined to assess their role in improving real-time visualization and user interaction with robots.

While AR shows significant potential for advancing HRI, challenges such as real-time processing limitations and the need for secure web-based interfaces remain. These insights will guide the development of my AR framework for the 4WD robotic car, focusing on real-time visualization, intuitive control, and efficient sensor integration.

CHAPTER 3

METHODOLOGY

This chapter outlines the methodology employed throughout the lifecycle of this project, focusing on the system architecture, design choices, and integration of key subsystems in achieving the overall project objectives. Each design decision is supported by a brief mention of relevant literature and appropriate system requirements, further analysis on component/algorithm selection can be found in Chapter 4.

3.1 OVERVIEW

The mobile robot project is structured around several interconnected subsystems, which work together to achieve real-time autonomous navigation, user interaction through AR markers, and remote control via a web interface. A high-level diagram (Figure 3.1) visualizes this architecture, comprising a Raspberry Pi (as the main processor), a camera module, motor control, AR-based fiducial marker detection, and a web-based user interface.

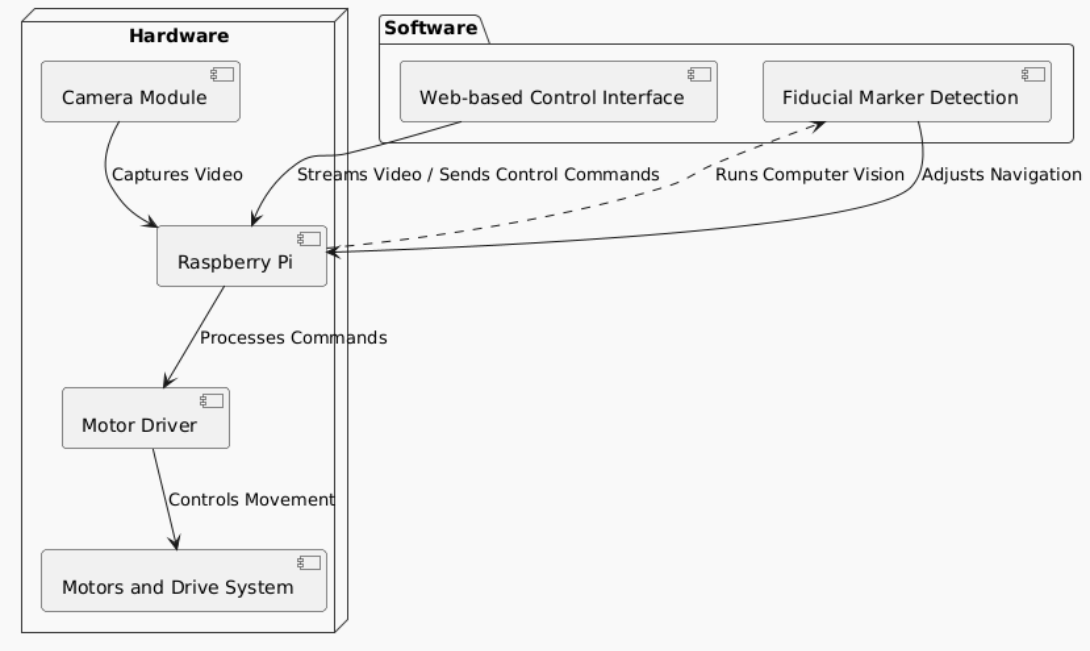


Figure 3.1: High Level Diagram of Project Architecture

The system begins with the camera capturing video streams, processed by the Raspberry Pi for AR marker recognition and navigation. Simultaneously, the web interface allows remote commands to be issued and video feedback to be streamed, closing the control loop between the user and the robot. This methodology draws inspiration from similar studies, such as La Delfa et al. [33] and Jacobsen et al. [34], which explore mobile robots with AR capabilities.

As outlined in Section 1.3, the system needs to meet specific requirements to ensure it performs as intended. These requirements were established through research, discussions with supervisor, and analysis of the intended user environment.

Req. ID	**Requirement Description**	**Spec. ID**	**Specification Description**
R1	The robot must operate in different control zones, defined by AR markers, and adjust its behavior accordingly (e.g., slowing down, stopping, or performing a task).	**F1**	Implementation of AR-based control zones that dynamically adjust robot speed, direction, or initiate tasks based on detected markers.
R2	The system must integrate dynamic instructions based on ArUco marker detection, allowing for real-time task updates.	**F2**	Real-time processing of ArUco markers to update task instructions and provide context-specific navigation and task execution.
R3	The AR interface must provide real-time visual feedback for the human operator to improve task monitoring and control.	**F3**	AR-based visual feedback will provide the human operator with real-time monitoring of the robot's actions and environmental interactions.
R4	The robot must include object avoidance functionality, ensuring it maintains a safe distance from recognized obstacles (e.g., warning cones) based on AR markers.	**F4**	Integration of object avoidance behavior, ensuring the robot maintains a safe distance from obstacles based on AR marker detection.
R5	The system should detect and respond to wheel slip events caused by changes in surface traction, providing stability during operation.	**F5**	Implementation of wheel slip detection functionality that monitors the robot's power draw to identify surface traction changes.

Table 3.1: Requirements and Functionalities for the Robot Control System

The system begins with the camera capturing video streams, processed by the Raspberry Pi for AR marker recognition and navigation. Simultaneously, the web interface allows remote commands to be issued and video feedback to be streamed, closing the control loop between the user and the robot. This methodology draws inspiration from similar studies, such as La Delfa et al. [33] and Jacobsen et al. [34], which explore mobile robots with AR capabilities.

3.2 SUBSYSTEMS OVERVIEW

The robot's subsystems are designed to work in tandem, each playing a crucial role in delivering the overall project's functionality. Below, we discuss the primary subsystems and their respective roles.

3.2.1 Camera Module

The camera module captures real-time video, which is essential for detecting AR markers and obstacles. Positioned to maximize environmental visibility, the camera's data feeds directly into the Raspberry Pi, where real-time processing occurs. High-definition video capture ensures accurate marker detection and obstacle avoidance, critical for achieving precise navigation.

Justification: The camera module is essential for both marker detection and obstacle avoidance, as demonstrated by Vanitha et al. [35], who emphasize the importance of high-resolution visual input in autonomous systems.

3.2.2 Raspberry Pi

The Raspberry Pi serves as the central processing unit (CPU), handling video input from the camera, running image processing algorithms (via OpenCV), and controlling the motors based on user commands and AR markers. It was selected due to its computational power, affordability, and strong compatibility with the system's software and hardware requirements. The Pi also facilitates real-time communication between the robot and the web-based interface.

Justification: The Raspberry Pi was chosen over alternatives (e.g., Arduino) for its ability to handle computationally expensive tasks like image processing while maintaining efficient communication with external interfaces, as supported by Jacobsen et al. [34].

3.2.3 Motor Driver and Drive System

The motor driver receives commands from the Raspberry Pi and translates them into movements. Pulse-width modulation (PWM) enables precise control of speed and direction, which is essential for executing tasks like AR-based navigation or obstacle avoidance.

Justification: PWM control provides the necessary flexibility for dynamic responses, particularly when navigating through AR-defined zones or avoiding obstacles. Previous work by La

Delfa et al. [33] highlights the significance of precise motor control in mobile robots.

3.2.4 Web-Based Control Interface

The web interface allows for remote control of the robot and streams real-time video to the user. It facilitates direct interaction by allowing users to send movement commands and receive immediate visual feedback. This subsystem is integral for monitoring and controlling the robot's behavior from a distance.

Justification: Real-time feedback and control are critical for AR-based robotics applications, particularly in environments where direct line-of-sight control is impractical. The web-based control interface ensures the system's usability aligns with these requirements, supported by findings from Vanitha et al. [35].

3.2.5 Fiducial Marker Detection

AR markers, specifically ArUco markers, are used to guide the robot's navigation. The Raspberry Pi processes the video input using OpenCV, detecting these markers to trigger predefined actions, such as altering the robot's path or stopping at specified zones.

Justification: Fiducial markers are a reliable, low-cost solution for guiding robotic navigation, as demonstrated by La Delfa et al. [33]. Their accuracy in marker detection enables real-time adjustments, which are crucial for dynamic interaction with the environment.

3.2.6 Obstacle Detection

Obstacle detection is critical for ensuring the safe navigation of the robot in dynamic environments. This process leverages computer vision techniques, specifically using a YOLO (You Only Look Once) model, to identify and localize obstacles such as bottles or other potential hazards in real-time. The Raspberry Pi captures video input from a camera mounted on the robot, and the YOLO model processes each frame to detect obstacles.

Justification: Implementing YOLO for obstacle detection allows for efficient processing of video frames and quick responses to dynamic environments. According to Redmon et al. [36], YOLO achieves state-of-the-art performance in object detection by balancing speed and accuracy, making it suitable for real-time applications in robotic systems. This capability enables the robot to navigate autonomously while avoiding collisions, enhancing overall operational safety and efficiency.

3.3 INTERACTION AND DEPENDENCIES

The subsystems work in unison to create a seamless experience for both the robot and the user. The camera module captures the environment, with the Raspberry Pi processing the video stream to detect fiducial markers. The Pi sends commands to the motor driver to adjust the robot's movement while relaying the processed video to the web-based control interface, allowing for user feedback and control. This integration is essential to ensure that navigation, interaction, and feedback are synchronized and responsive to both user input and environmental changes.

3.4 ACCEPTABLE TEST PROCEDURES (ATP)

The success of the project depends on thorough testing of each subsystem. The tests are designed to ensure the system meets its functional and performance requirements. Table 3.2 outlines key test scenarios linked to specific subsystems and project goals, ensuring the robot performs as expected in various operational conditions.

Table 3.2: Enhanced Breakdown of sub-tests to be performed in the acceptance testing.

Test	**Description**	**Expected Outcome**	**Spec. ID**	**Req. ID**
T1	Test the robot's behavior in AR-based control zones	The robot adjusts speed or direction based on AR markers within defined zones (e.g., slow zone, stop zone).	F1	R1
T2	Test real-time ArUco marker detection for dynamic instructions	Robot recognizes ArUco markers in real-time and triggers corresponding task instructions (e.g., task initiation or environmental change response).	F2	R2
T3	Test AR-based visual feedback for real-time monitoring	The user receives accurate and real-time AR feedback on the robot's position, detected markers, and current task.	F3	R3
T4	Test object avoidance system using visual markers	The robot successfully avoids predefined obstacles marked by ArUco codes within 20 cm, halting or rerouting its path.	F4	R4
T5	Test wheel slip detection and response functionality	The robot detects slip on different surfaces (e.g., sand or wet surfaces) and compensates by adjusting motor power or stopping.	F5	R5

CHAPTER 4

DESIGN AD IMPLEMENTATION

This chapter outlines the design and implementation phases of the mobile robot system project. The purpose of this chapter is to describe the system's architecture, the design of each subsystem, the integration of hardware and software, the justification of components chosen and how these components work together to achieve the goals of this project. The design emphasizes ease of navigation, real-time feedback, and effective control via an intuitive interface.

4.1 SYSTEM DESIGN

The design of the mobile robot system was based on the specific project requirements outlined in Chapter 3. Following a detailed review of relevant literature and an analysis of system needs, the robot's design was established. The architecture integrates several key components; Raspberry Pi, camera module, fiduciary markers, motor control system, and wireless communication modules—to ensure optimal system performance.

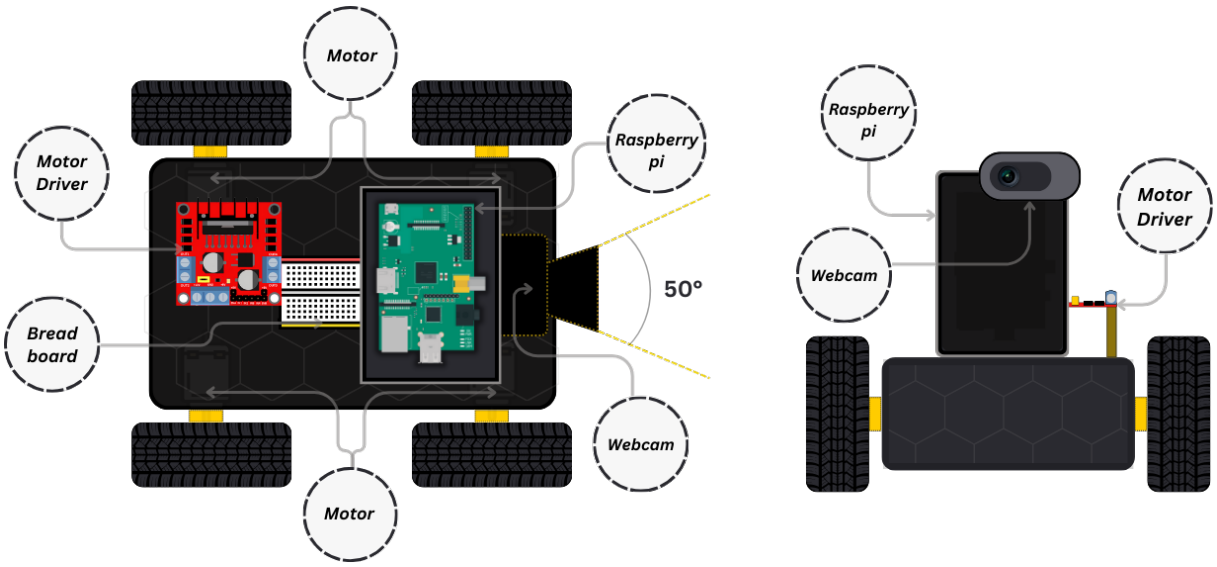


Figure 4.1: Illustration of the hardware layout, showing the arrangement of key components like the Raspberry Pi, motor system, and camera module.

The core decision-making process involved selecting the appropriate technologies and components. Based on the findings from previous research (e.g., Jacobsen et al. [34], La Delfa et al. [33], and Vanitha et al. [35]), which explored various mobile robot control methods using augmented reality (AR) and computer vision, it became clear that a Raspberry Pi-based system with a web interface would provide the required flexibility and ease of control for this project.

After evaluating several options for actuation and navigation, it was determined that motorized control through a motor driver and a four-wheel drive system would provide the most reliable and efficient solution for movement. This decision was supported by studies that demonstrate the performance of motor-driven robots in both laboratory and dynamic environments. Additionally, the use of fiducial markers, such as ArUco or AprilTag markers, was identified as a robust solution for localization and obstacle avoidance (La Delfa et al. [33]; Vanitha et al. [35]).

4.2 HARDWARE SYSTEM ARCHITECTURE

The system architecture provides a high-level overview of how the robot and its subsystems interact with one another to achieve semi-autonomous navigation, real-time feedback, and remote control functionality. The key components of the system include the hardware modules such as the Raspberry Pi, motors, sensors, and camera, alongside software elements such as computer vision algorithms and the web-based control interface.

The project is composed of several key components, below is a high level overview of all the components that are needed in order execute the project properly.

4.2.1 Micro-controller Selection

The Raspberry Pi Model 4B with 4GB RAM was selected as the micro controller for this project due to its versatility and capability to handle multiple complex tasks simultaneously. Its ability to run sophisticated computer vision algorithms while interfacing with other hardware components makes it ideal for this application. The capacity to manage real-time video streaming, marker detection, and motor control simultaneously is crucial for the system's functionality.

Unlike dedicated micro-controllers(processors) such as the Arduino, the Raspberry Pi can run Python-based frameworks like OpenCV, which is essential for implementing the system's computer vision capabilities. This aligns with findings from Jacobsen et al. [34], who demonstrated the effectiveness of using a Raspberry Pi for web hosting.

The use of a single, integrated system for both processing and control, as highlighted in Chap-

ter2, simplifies the overall design, reduces latency, and enhances system performance. In our setup, the Raspberry Pi functions as the central processing unit, responsible for interfacing with all other components. It processes the video feed from the camera, executes computer vision algorithms such as ArUco marker detection, and transmits control signals to the motor driver. The Raspberry Pi runs a Python-based framework that integrates OpenCV for vision tasks and SocketIO for communication with the web interface. This allows it to receive input from the camera, process it using OpenCV, and then send commands to the motor driver based on the detected environment and user instructions received through the control interface.

The selection of the 4GB RAM version was made to ensure sufficient memory for running multiple processes concurrently and handling intensive computational tasks. While the 8GB version offers more memory, the 4GB model provides a balanced compromise between performance and cost-effectiveness for our application. Table 4.1 provides a comparison of the specifications for the Raspberry Pi Model 4B with 4GB RAM:

Specification	**2GB RAM**	**4GB RAM**	**8GB RAM**
CPU	Quad-core Cortex-A72	Quad-core Cortex-A72	Quad-core Cortex-A72
Clock Speed	1.5 GHz	1.5 GHz	1.5 GHz
RAM	2GB LPDDR4	4GB LPDDR4	8GB LPDDR4
Networking	Gigabit Ethernet	Gigabit Ethernet	Gigabit Ethernet
USB Ports	2x USB 3.0, 2x USB 2.0	2x USB 3.0, 2x USB 2.0	2x USB 3.0, 2x USB 2.0
Video Output	2x micro-HDMI	2x micro-HDMI	2x micro-HDMI
Power Supply	5V/3A USB-C	5V/3A USB-C	5V/3A USB-C

Table 4.1: Comparison of Raspberry Pi Model 4B RAM Versions

This hardware configuration provides the necessary processing power and memory to handle the computer vision tasks, video streaming, and robot control required by our project. The Raspberry Pi 4B with 4GB RAM offers a robust platform capable of running complex algorithms, processing real-time video data, and managing the various input and output operations essential for the robot car's operation.

4.2.2 Camera Module

The camera plays a pivotal role in this system as it provides real-time feedback necessary for detecting fiducial markers and controlling the robot's movement. The decision to use a high-resolution web camera, specifically the HD Logitech C270, over a more basic sensor was based on the need for precise marker detection, which demands high-quality image capture. La Delfa et al. [33] emphasized the importance of high-resolution cameras for accurate marker detection, particularly in dynamic environments. A higher-quality video feed improves the reliability of computer vision algorithms and ensures better control over the robot's movements.

The camera module continuously captures the video feed, which is streamed to both the user via the web interface and the Raspberry Pi for processing. The camera plays a critical role in detecting fiducial markers in the environment, enabling navigation and interaction based on visual feedback.

Table 4.2: Comparison of Logitech C270 and Raspberry Pi Camera Module V2

Specification	**Logitech C270**	**Raspberry Pi Camera Module V2**
Resolution	1280 x 720 (HD)	3280 x 2464 (8 MP)
Frame Rate	30 fps (at 720p)	30 fps (at 1080p)
Field of View (FOV)	60°	62.2°
Interface	USB	CSI (Camera Serial Interface)
Price	Moderate	Low
Ease of Use	Plug-and-play	Requires configuration
Compatibility	Universal (compatible with various systems)	Raspberry Pi exclusive

The Logitech C270 was chosen primarily due to it being readily available for use and its ability to seamlessly integrate into various systems via USB, requiring minimal configuration compared to the Raspberry Pi Camera Module. Additionally, while the Raspberry Pi camera offers higher resolution, the Logitech C270 provides sufficient image quality for marker detection. Furthermore, the webcam is universally compatible, making it a practical choice for prototyping and development in this context.

4.2.3 Motor and Drive System

The four-wheel drive system of the DGU ALUM MULTI-CHASSIS 4WD KIT was selected to ensure that the robot can move with precision across various terrains. This chassis provides a sturdy foundation for the robot's movement capabilities. The motors used in this system operate on 3V-12VDC, with a maximum torque of 800gf cm min at 4.5V.

MOTOR DRIVER SELECTION

To control the motors, a motor driver was necessary to interface with the Raspberry Pi and provide effective control over the speed and direction. Several motor driver options were considered before selecting the L298N H-bridge for its compatibility and ease of use. The alternatives explored included the DRV8833 and the TB6612FNG, each offering unique advantages. A comparison of these motor drivers is provided in Table 4.3.

Table 4.3: Comparison of Motor Driver Options

Specification	**L298N H-bridge**	**DRV8833**	**TB6612FNG**
Voltage Range	5V - 46V	2.7V - 10.8V	4.5V - 13.5V
Max Current	2A per channel	1.5A per channel	1.2A per channel
PWM Control	Yes	Yes	Yes
Size	Larger (bulky)	Compact	Compact
Efficiency	Moderate (bipolar transistors)	High (MOSFETs)	High (MOSFETs)
Cost	Low	Moderate	Moderate

While the DRV8833 and TB6612FNG offer more compact designs and higher efficiency due to the use of MOSFETs, the L298N H-bridge was ultimately selected for this project. Its ability to handle higher voltage ranges and currents made it a better match for the motors in use. Additionally, the L298N supports straightforward Pulse Width Modulation (PWM) control, making it ideal for precise speed and direction control via the Raspberry Pi.

According to Vanitha et al. [35], a robust motor and drive system is essential for mobile robots that rely on real-time feedback from sensors for navigation. The choice of using an H-bridge for motor control is well-supported in robotics literature as it allows for both forward and reverse motion, as well as turning, without adding unnecessary complexity. The four-wheel drive system enables the robot's movement and maneuvering, with the L298N motor driver controlling the speed and direction of the motors based on the PWM signals received from the Raspberry Pi. This setup allows the system to adjust motor power dynamically to maintain proper navigation based on sensor input or user commands, achieving precise movement in various scenarios.

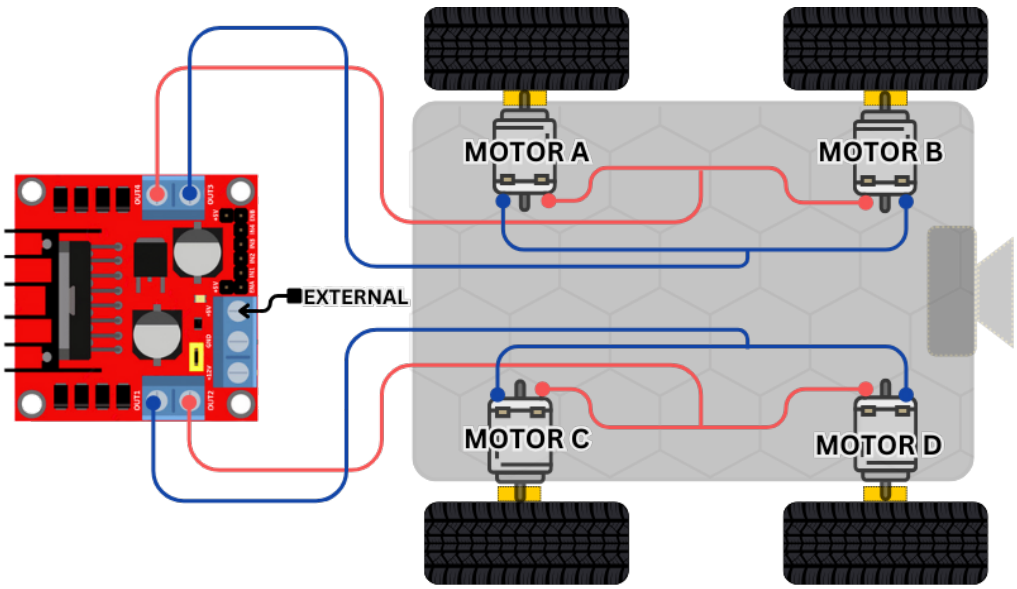


Figure 4.2: Illustration of the H-Bridge connection with the motors.

IN1	IN2	IN3	IN4	Motor A/B	Motor C/D	Robot Movement
0	1	0	1	Forward	Forward	Forward
1	0	1	0	Reverse	Reverse	Reverse
0	1	1	0	Forward	Reverse	Right Turn
1	0	0	1	Reverse	Forward	Left Turn
0	0	0	0	Stop	Stop	Stop
1	1	1	1	Brake	Brake	Brake

Table 4.4: Truth Table for L298N H-Bridge Motor Control

The Raspberry Pi generates PWM (Pulse Width Modulation) signals through its GPIO pins, which are connected to the input pins (IN1, IN2, IN3, IN4) of the L298N H-bridge motor driver. These digital signals control the direction and speed of the motors. The H-bridge then interprets these signals and converts them into the appropriate voltage levels and current flow to drive the motors. For Motors A and B, IN1 and IN2 control their direction and speed, while IN3 and IN4 control Motors C and D. The H-bridge acts as an intermediary, amplifying the low-power signals from the Raspberry Pi into the higher power (by using an external power source) needed to drive the DC motors effectively. This setup allows for precise control over each motor's behavior, enabling complex movements of the robot through simple digital commands from the Raspberry Pi.

4.3 POWER SUPPLY

Although the primary focus of this project is the robot's navigation, control systems, and augmented reality (AR) integration, the power supply plays a vital role in ensuring reliable performance for the entire system. Without a stable and appropriate power source, the Raspberry Pi, motors, and other components would fail to operate consistently, which could result in unpredictable robot behavior. To address the varying power needs of the system components, two separate power supplies were implemented.

4.3.1 Raspberry Pi Power Supply

The Raspberry Pi requires a consistent and stable power supply to handle its computational tasks, including image processing, web-based control, and communication with other subsystems. A power bank was selected for this purpose due to its wide availability, ease of use, and sufficient power output.

- **Power Source:** A commercial power bank with a 2.1A output was utilized to meet the

power requirements of the Raspberry Pi 4.

- **Reason for Choice:** The Raspberry Pi 4 requires a power supply that can provide at least **5V and 3A** for optimal performance. While the chosen power bank offers slightly lower current (2.1A), testing showed that it was sufficient for running the robot's tasks without significant performance degradation. Additionally, the portability and long-lasting capacity of the power bank make it ideal for mobile robotics applications.

4.3.2 H-Bridge and Motor Power Supply

The L298N H-bridge motor driver and the motors require a different power supply than the Raspberry Pi, given their higher power demands during operation, particularly under load (e.g., when turning or climbing). To meet this requirement, an RS PRO 3C 3S1P Li-Ion Battery Pack was chosen, which offers higher voltage and capacity.

The battery pack consists of **three 18650 Li-Ion cells** connected in series (3S1P configuration), providing a total voltage of **11.1V** and a capacity of **2600mAh**.

Specifications:

- Voltage: 11.1V
- Capacity: 2600mAh
- Configuration: 3 x 18650 cells in series

Reason for Choice: The L298N H-bridge can handle up to 46V, so 11.1V is well within its operational range. The motors benefit from higher voltage when performing tasks that require more torque, such as sharp turns or accelerating from a stationary position. The 2600mAh capacity ensures that the robot can operate for extended periods before needing a recharge.

4.4 Software System Architecture

The bulk of the project relied on software for functionality, serving as the brain of the entire system. The software orchestrates communication between hardware components and enables real-time decision-making. This architecture integrates several critical subsystems, including computer vision, motor control, data processing, and network communication.

Python was chosen as the primary programming language for the project, largely due to its versatility and the wide range of libraries it offers, particularly in fields like robotics, image

processing (OpenCV), and web development (Flask). Although other programming languages such as C++ or Java were considered, Python stood out because of its simplicity, ease of development, and extensive library support, the timeline of the project also did not allow for use of a lower level language like C. These qualities made it an ideal choice for rapid prototyping and achieving the project's deliverables within the given time constraints.

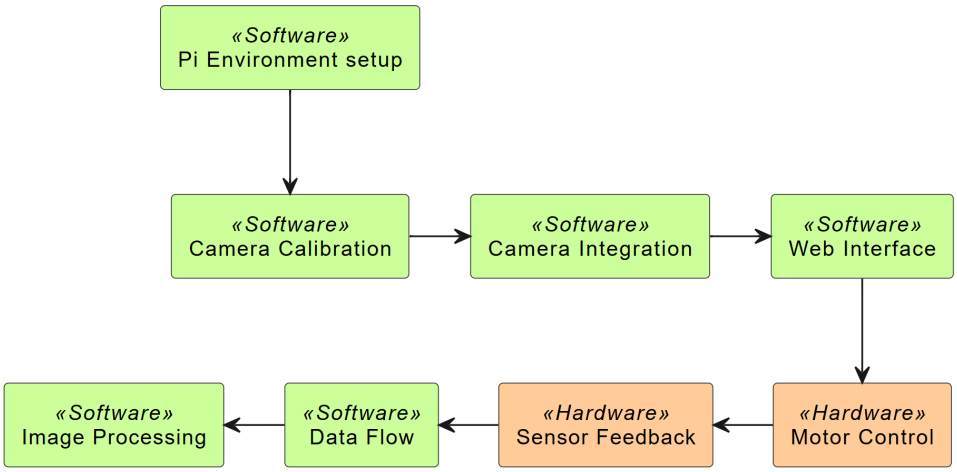


Figure 4.3: Illustration of the project production process.

Figure 4.3 provides an overview of the project production process. The flow begins with setting up the software environment on the Raspberry Pi, followed by camera calibration and integration, which is crucial for enabling accurate image capture and processing. These steps lead to the development of a web interface, facilitating interaction with the system. The software is then responsible for managing data flow between image processing algorithms and sensor feedback, which in turn, directly informs motor control to achieve desired movements and responses. Below I delve into the design of each system.

4.4.1 Development Environment Setup

To begin the development of the robot, I set up a headless Raspberry Pi environment, which means that I connected to the Pi remotely over SSH, avoiding the need for a physical monitor, keyboard, or mouse. This headless setup is common for Raspberry Pi projects where the Pi is used as an embedded device, and allows for easier control via the network from my laptop. I initiated an SSH session using the command:

```
ssh pi@<raspberry-pi-ip>
```

from my development machine to connect to the Pi and began the process of setting up the

environment.

The primary goal of the development environment was to create a clean, isolated workspace that would avoid interfering with the Pi's system libraries and default configurations. To achieve this, I decided to set up a virtual Python environment using `venv`, which would ensure that all libraries required for this project were contained within the environment and did not alter the system's default Python packages.

To create and activate the environment, I ran:

```
python3 -m venv robot_env
source robot_env/bin/activate
```

The project depended on several libraries for various functions. Some of these libraries were for The project depended on several libraries for various functions. Some of these libraries were for **web development**, like **Flask** and **Flask-SocketIO**, which handled real-time communication between the Raspberry Pi and the web interface. Others were for **computer vision**, including **OpenCV** and **ArUco**, which would handle marker detection and processing of the video feed.

To install the required libraries, I used `pip` within the activated virtual environment. Here is the command that I used to install all the necessary dependencies Each of these libraries served a specific purpose:

Library/Tool	**Description**
Flask	Creates the web server for video streaming and control commands.
Flask-SocketIO	Enables real-time communication for responsive robot control.
OpenCV	Handles video capture and ArUco marker detection for navigation.
aiortc	Supports WebRTC for live video streaming from the robot's camera.
pigpio	Controls GPIO pins with high precision for better timing.
Numpy	Manages arrays and matrix operations for image processing.
uuid & **asyncio**	Generates unique IDs and manages asynchronous operations for WebRTC.

Table 4.5: Summary of libraries and tools used in the project.

These libraries were specifically selected to ensure smooth operation of the robot's core functions, including image processing, real-time control, and web interaction.

4.4.2 Camera Calibration

Before proceeding with the implementation of the robot's image processing pipeline, I first verified that the camera was functioning correctly. This involved ensuring that the Raspberry Pi camera module was properly connected and that the Raspberry Pi could capture live video feeds. Once this basic functionality was confirmed, the next step was to calibrate the camera to account for lens distortion, a crucial step for ensuring accurate marker detection later in the project.

To perform the calibration, I ran a **Flask web application** that captured and stored images from the camera for the calibration process. The process of camera calibration involves capturing multiple images of a checkerboard pattern at various angles, which allows for the correction of geometric distortions such as barrel distortion in the captured images. Using the images, I applied OpenCV's camera calibration functionality to compute the intrinsic and distortion parameters of the camera.

The Flask app was set up to continuously stream the live feed from the camera and save frames at regular intervals. Below is a brief overview of the code used to capture images from the camera and store them for calibration:

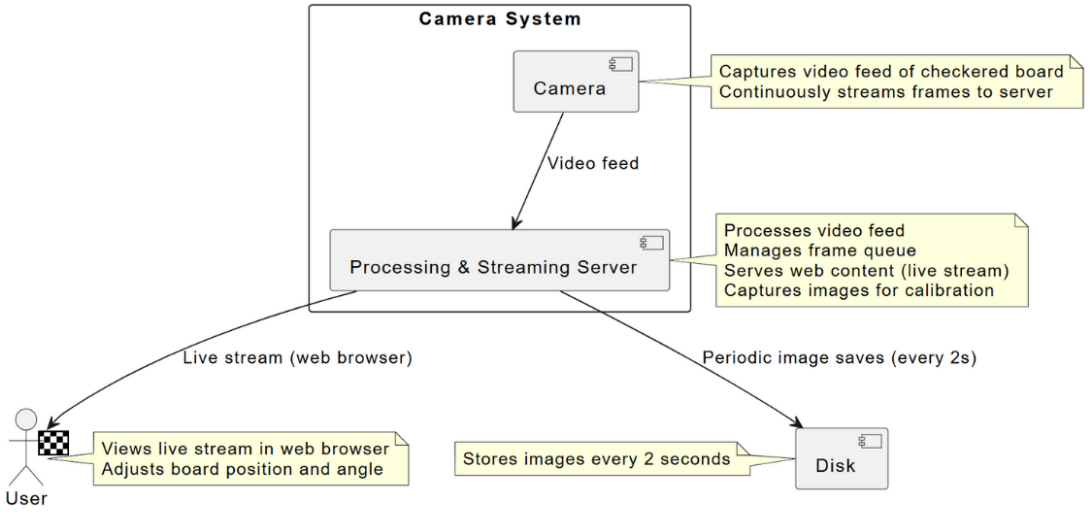


Figure 4.4: Diagram of the functionality of the calibration app.

After confirming the camera's functionality, I captured a series of images through the Flask app, which were then processed using OpenCV's camera calibration tools. The calibration process computes the intrinsic camera matrix and distortion coefficients, which are essential for removing distortion and ensuring the accuracy of subsequent image processing steps.

OpenCV's camera calibration tool was used, with images of a checkerboard pattern captured by

the camera. The following steps were followed for the calibration:

1. **Capture Images**: The Flask app automatically saved images at a regular interval during the video feed, stored in the specified directory.
2. **Camera Calibration**: Using OpenCV, the captured images were processed to detect the checkerboard pattern and calculate the camera's intrinsic and distortion parameters.
3. **Saving Results**: Once the camera was successfully calibrated, the calibration parameters (including the camera matrix and distortion coefficients) were saved to a YAML file, which could be used later for distortion correction in real-time video processing.

The resulting calibration file was stored in calibration.yaml, which contained the necessary parameters for correcting lens distortion during subsequent image processing tasks. This YAML file is crucial for ensuring that the robot can accurately interpret the positions of ArUco markers in real-world coordinates, even when captured through distorted lenses.

4.4.3 ArUco Marker Detection

The implementation of ArUco markers in this project leverages their speed and reliability, crucial for real-time robotics applications. ArUco outperforms alternatives like AprilTag and CC-Tag, with detection times as low as 2 milliseconds on minimalist hardware setups [37]. This efficiency is vital for our Raspberry Pi-based mobile robot system, where rapid visual data interpretation ensures smooth navigation and environmental interaction.

Beyond basic detection, ArUco markers enable advanced features such as distance measurement and speed estimation. The system utilizes the known marker size and camera frame appearance to estimate robot-to-marker distance, forming the basis for speed calculations.

4.4.3.1 Distance Measurement

Distance measurement is implemented using the `estimate_pose` function:

```
distance = self.estimate_pose(corners)
```

This function uses the marker's known physical size and camera intrinsic parameters to estimate the marker's pose relative to the camera. The distance d is then calculated using the translation vector $\mathbf{t}$ from the pose estimation:

$$d = \|\mathbf{t}\| = \sqrt{t_x^2 + t_y^2 + t_z^2}$$

4.4.3.2 Speed Estimation

Speed estimation employs a rolling buffer approach for stable readings. The implementation of a buffer-based speed calculation method, as opposed to a simpler instantaneous approach, was chosen to enhance the robustness and reliability of the system's speed estimates. While a basic speed calculation using only the most recent two data points ($v = \frac{\Delta d}{\Delta t}$) would be computationally simpler, it would be highly susceptible to noise and momentary fluctuations in distance measurements or marker detection. Our buffer-based method, which maintains a rolling window of recent measurements, allows for a more stable and accurate speed estimation by averaging over multiple data points.

The speed calculation occurs in the `calculate_speed` method:

```
def calculate_speed(self, current_distance, current_time):
```

The instantaneous speed v_i between two consecutive measurements is calculated as:

$$v_i = \left| \frac{\Delta d_i}{\Delta t_i} \right| = \left| \frac{d_i - d_{i-1}}{t_i - t_{i-1}} \right|$$

where d_i and t_i are the distance and time at the i-th measurement, respectively.

The average speed $\bar{v}$ over the buffer is then computed as:

$$\bar{v} = \frac{1}{n} \sum_{i=1}^{n} v_i$$

where n is the number of valid speed measurements in the buffer.

The integration of ArUco markers with these distance and speed estimation techniques enables the creation of control zones and the overlay of augmented reality (AR) elements. For instance, upon detecting a "restricted zone" marker, the robot can dynamically adjust its speed or trajectory based on the calculated distance and speed.

The analysis in [37] emphasizes ArUco's superior performance in speed-dependent tasks, which is crucial for our application where the robot must accurately detect and interpret markers while

in motion, facilitating real-time distance and speed calculations.

Given the computational constraints of the Raspberry Pi, the efficiency of the ArUco algorithm, combined with our optimized distance and speed estimation techniques, proves ideal for this project. This setup ensures real-time performance and reliable detection, allowing the robot to respond dynamically to environmental changes based on detected markers and calculated metrics.

4.4.4 Obstacle Detection System

The obstacle detection system in this project utilizes the YOLOv5n (You Only Look Once, version 5 nano) model, a lightweight and efficient neural network designed for real-time object detection. YOLOv5n's compact size and low computational requirements make it ideal for embedded applications like our mobile robot, enabling fast inference on devices with limited processing power such as the Raspberry Pi.

4.4.4.1 Model Configuration

The YOLOv5n model is configured specifically for our use case:

Listing 4.1: YOLOv5n cofiguration

```
1    def setup_yolo_model(self):
2    self.model = torch.hub.load('ultralytics/yolov5', 'yolov5n',
        pretrained=True)
3    self.model.classes = [39]  # Class 39 for bottle detection
4    self.model.conf = 0.1  # Confidence threshold
5    self.model.iou = 0.45  # IOU threshold for NMS
6    self.model.to('cpu')  # Ensure CPU execution
```

This configuration focuses on detecting a single class of objects (bottles, class 39) to optimize processing power and speed. The confidence threshold (`conf`) is set to 0.1, striking a balance between detection sensitivity and false positive reduction. The Intersection over Union (IoU) threshold for non-maximum suppression is set to 0.45, helping to eliminate redundant detections.

4.4.4.2 Detection Process

The obstacle detection process is implemented in the `detect_obstacles` method:

Listing 4.2: YOLOv5n detection Process

```
    def detect_obstacles(self, frame):
    results = self.model(frame)
    self.obstacle_detected = False
    for *xyxy, conf, cls in results.xyxy[0]:
    if conf > self.model.conf:
    # Obstacle detected, mark frame and set flag
```

This method processes each frame through the YOLO model and checks for detections above the confidence threshold. When an obstacle (bottle) is detected, it's marked on the frame and a flag is set.

Notably, the obstacle detection is not continuously active. It's designed to be user-activated via the control interface, optimizing power consumption and reducing potential latency in the video feed. When activated, it processes frames in real-time, providing immediate feedback to the control system if an obstacle is detected.

4.4.4.3 Performance Considerations

The decision to detect only a single object class (bottles) significantly reduces the computational load compared to multi-class detection. This approach is justified by the specific use case of our robot and the need to balance detection capabilities with real-time performance on limited hardware.

The performance of this system can be characterized by its inference time t_i, which is the time taken to process a single frame:

$$t_i = t_p + t_d + t_m$$

Where:

- t_p is the preprocessing time (frame acquisition and conversion)
- t_d is the YOLO model inference time
- t_m is the post-processing time (drawing bounding boxes, updating flags)

By focusing on a single class and using the lightweight YOLOv5n model, we minimize t_d, which is typically the most significant component of t_i.

This design balances detection capabilities with resource efficiency, making the obstacle detection system both responsive and power-conscious. It allows the robot to operate normally under default conditions and only initiates the obstacle detection process when necessary, providing a flexible and efficient solution for real-time obstacle avoidance in mobile robotics applications.

4.4.5 Wheel Interface Design

The wheel interface was designed to control four motors, arranged in pairs, connected to an L298N H-bridge motor driver. Each pair of motors is responsible for controlling the left and right sides of the robot, allowing for differential steering. The system uses the Raspberry Pi's GPIO pins to control the direction and speed of the motors through Pulse Width Modulation (PWM).

4.4.5.1 Motor Control Using GPIO and PWM

Each motor pair is controlled using two GPIO pins for direction control and one PWM pin for speed control. The L298N H-bridge allows the Raspberry Pi to control the motors in both forward and reverse directions by manipulating the GPIO pins. The PWM signal controls the speed of the motors by adjusting the duty cycle of the signal sent to the H-bridge.

- **Forward Movement**: Both pairs of motors are driven in the same direction, with a high signal sent to the forward GPIO pin and a low signal to the reverse pin. The PWM signal controls the speed of both motor pairs, allowing the robot to move forward.
- **Turning**: To turn, the speed or direction of one pair of motors is adjusted. For example, to turn left, the left motor pair is reversed, while the right motor pair continues to move forward.
- **Stop**: The robot is stopped by setting all GPIO pins to low, cutting off the power to both motor pairs.

4.4.5.2 Integration with Web Interface

The web interface allows the user to control the robot in real-time by sending movement commands via Flask-SocketIO. These commands are mapped to motor control functions in the backend, where the GPIO pins are set according to the desired direction and speed. The key inputs (W, A, S, D) correspond to forward, left, backward, and right movements, which are translated into GPIO signals for the motor pairs.

- **W Key**: Moves the robot forward by setting both motor pairs to move in the forward direction.
- **S Key**: Reverses both motor pairs to move the robot backward.
- **A Key**: Adjusts the motor speed and direction to turn left by slowing down or reversing the left motor pair.
- **D Key**: Adjusts the motor speed and direction to turn right by slowing down or reversing the right motor pair.

This design provides precise control over the robot's movements and allows the user to navigate the robot using the web interface in real-time. By using PWM, the robot's speed can be smoothly adjusted based on the user's input.

4.4.6 Web Interface and Server Design

The web interface and server design are crucial components of this project, enabling remote control and real-time monitoring of the robot's movements. The system was designed to create a lightweight and responsive interface accessible through a standard web browser, allowing users to control the robot and view live feedback from the camera in real-time.

4.4.6.1 Server Architecture

The server was built using the Flask framework, a micro web framework for Python, chosen for its simplicity and efficiency in handling web requests, streaming video data, and managing control inputs from the user.

The core architecture of the server includes the following key components:

- **Flask Web Server**: Responsible for serving the main web page and handling HTTP requests. Flask's flexibility allows for seamless integration of both static files (HTML, JavaScript, CSS) and dynamic content (video streaming).
- **SocketIO Integration**: Flask-SocketIO was implemented for real-time, bidirectional communication between the web interface and the robot. This allows the web client to send control commands to the robot without page reloads and enables real-time feedback by pushing updates to the client.

- **Video Streaming**: The robot's camera feed is streamed to the web interface using Flask's response streaming mechanism. Video frames are captured, processed, and sent to the web client for near real-time display.
- **WebRTC Integration**: For improved video streaming performance, WebRTC (Web Real-Time Communication) was implemented using the aiortc library, allowing for peer-to-peer video transmission.

4.4.6.2 Web Interface Design

The web interface was designed to be intuitive and responsive, built using HTML5, CSS, and JavaScript. It allows users to control the robot through both on-screen buttons and keyboard inputs.

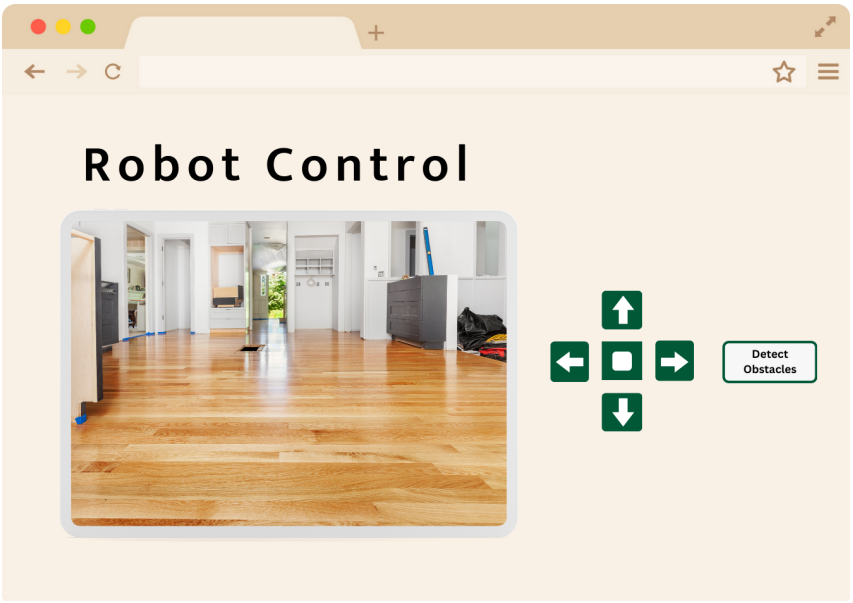


Figure 4.5: Visualization of front-end client side server

Key features of the web interface include:

- **Control Panel**: A set of virtual buttons for controlling the robot's movements (forward, backward, left, right, stop). These controls are mapped to the robot's movement functions through SocketIO.
- **Live Video Feed**: An embedded video player streams the live feed from the robot's camera, allowing users to see the robot's perspective in real-time.
- **Keyboard Control**: Users can control the robot using keyboard inputs (W, A, S, D keys and arrow keys), providing an alternative to on-screen buttons.

- **Bottle Detection Toggle**: A button to enable or disable the bottle detection feature, demonstrating the system's capability for object recognition.

4.4.6.3 Server-Side Implementation

The server-side implementation includes several key functions in a sleek minimalist layout:

- **Route Setup**: Flask routes are defined for serving the main page ('/') and the video feed ('/video_feed').

- **Movement Handling**: SocketIO events handle movement commands, translating them into appropriate robot actions. This includes a safety feature to disable forward movement when a restricted zone is detected.

- **Bottle Detection Toggle**: A SocketIO event handler toggles the bottle detection feature on and off based on user input.

- **WebRTC Offer Handling**: An asynchronous function handles WebRTC offers, setting up peer connections for improved video streaming.

- **UI Interaction Handling**: SocketIO events process user interface interactions, allowing for real-time updates and feedback.

4.4.6.4 Design Considerations

Performance and Latency: Real-time operation necessitated minimizing latency in both video streaming and control inputs. The use of SocketIO for control commands and WebRTC for video streaming helps achieve low-latency communication. The video stream is optimized using a lower resolution and frame rate to reduce the load on the Raspberry Pi and ensure smoother performance.

Responsive Design: The interface is designed to be responsive, adapting to different screen sizes and devices. This ensures a consistent user experience across desktop and mobile browsers.

Modularity: The server architecture is designed with modularity in mind, separating concerns between route handling, video processing, and robot control. This design facilitates easier maintenance and potential future expansions of the system.

4.4.6.5 Testing and Validation

To ensure the reliability and performance of the web interface and server, a series of tests were conducted:

- **Functional Testing**: Each control and feature of the web interface was tested to ensure correct operation.
- **Latency Testing**: The responsiveness of robot controls and video feed was measured to ensure acceptable real-time performance.
- **Cross-browser Testing**: The interface was tested across multiple browsers to ensure compatibility.
- **Mobile Responsiveness Testing**: The interface was tested on various mobile devices to verify its responsive design.

These tests helped refine the design and improve the overall user experience of the web interface and server components. For more details into how these tests were run and the results generated refer to the following chapter.

CHAPTER 5

MODULAR TESTING

The testing process for this project was conducted in a systematic and incremental manner to ensure that each subsystem functioned as expected before being integrated into the final system. The approach involved testing each module independently, followed by creating interfaces between these modules, and finally testing the interactions between them. This strategy helped identify and resolve issues early, ensuring that the final system was robust and performed as required.

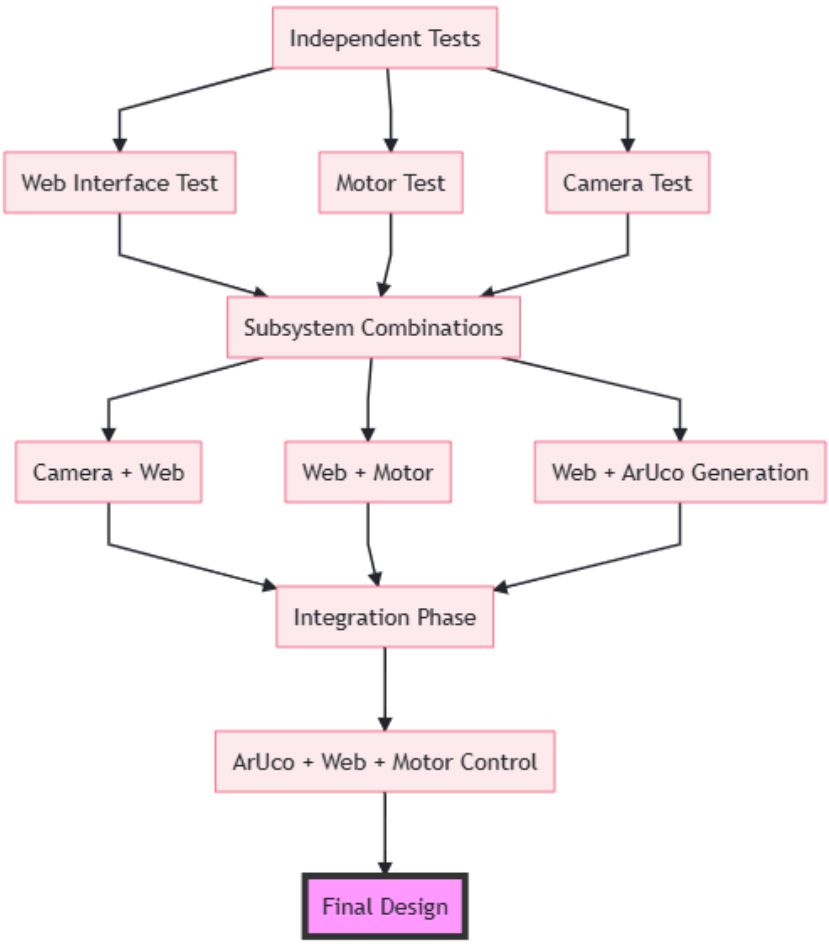


Figure 5.1: Illustration of different levels of testing implemented in the project.

5.1 INDEPENDENT MODULE TESTING

Each subsystem of the robot was tested individually using Python scripts to verify functionality before integration into the full system. This modular testing allowed for focused troubleshooting and ensured that each component operated as expected in isolation.

5.1.1 Web Interface Testing

The first step was testing the **web interface** to verify that the server could host a webpage and be accessed by devices connected to the same network. This involved the following steps:

1. Setting up a simple Flask server.
2. Connecting multiple devices (e.g., laptop, smartphone) to the same network.
3. Attempting to access the server from each device.
4. Verifying the server responded correctly to basic requests.

A script was developed to send ping requests to the server every 5 seconds over a 2-hour period (1440 pings total). Uptime percentage was calculated as (successful pings / total pings) * 100. Packet loss was measured using the 'ping' command with 1000 packets sent from devices at 1m intervals from the server. Significant packet loss ($> 5\%$) was first observed at distances beyond 5 meters.

Results:

- Server connection success rate: **100%** within 2 meters.
- Connection stability: **99.5% uptime** over 2 hours.
- Issues: Packet loss at distances exceeding 5 meters.

The server was accessible from all devices within a **2-meter range** with minimal latency. Stability was measured at **99.5% uptime** over a 2-hour test, with some packet loss beyond **5 meters**. These foundational tests ensured that the control interface could be hosted and accessed remotely, which was essential for subsequent integration.

5.1.2 Motor Control Testing

The **motor control** was tested using a basic Python script with the GPIO library to send PWM signals to the motors via the L298N H-bridge. The following steps were taken:

1. Connecting each motor pair to the H-bridge.
2. Writing a script to send PWM signals to control speed and direction.
3. Placing markers on the wheels for visual tracking.
4. Recording slow-motion video to check rotations per minute (RPM).

Results showed that the motors rotated consistently at **90 RPM** at full speed when using 9V source and lifted above the ground, with a slight deviation of **2 RPM** between motors. Testing revealed that the PWM signal had to be increased to **80%** to enable smooth turning, addressing friction and motor load issues.

Results:

- Maximum wheel speed: **80 RPM** at 100% PWM.
- Wheel speed deviation: **2 RPM**.
- Minimum Required PWM for driving forward and backward: **30%**
- Minimum Required PWM for turning: **70%**.

5.1.3 ArUco Marker Detection Testing

To evaluate the system, we performed tests focusing on key performance indicators: detection capabilities, distance measurement accuracy, speed estimation, and system responsiveness. Experiments were conducted under controlled conditions with varied lighting and distances to simulate real-world environments. Results were analyzed using Python libraries such as NumPy and Pandas.

5.1.3.1 Detection Performance Testing

Data Collection: 100 images were captured at 0.1m intervals from 0.1m to 2m under three lighting conditions. Detection attempts were logged for each image.

Data Collection: A total of 100 images were captured at 0.1m intervals from 0.1m to 2m under three lighting conditions (normal, low light, bright light). Each detection attempt was recorded, and data was gathered for comparison.

Table 5.1: ArUco Marker Detection Performance

Condition	100% Detection	>90% Detection	Avg. Detection Time
Normal (500 lux)	Up to 1.5m	Up to 1.8m	
Low Light (100 lux)	Up to 1.2m	Up to 1.5m	3.2ms (±0.5ms)
Bright Light (1000 lux)	Up to 1.7m	Up to 2.0m	

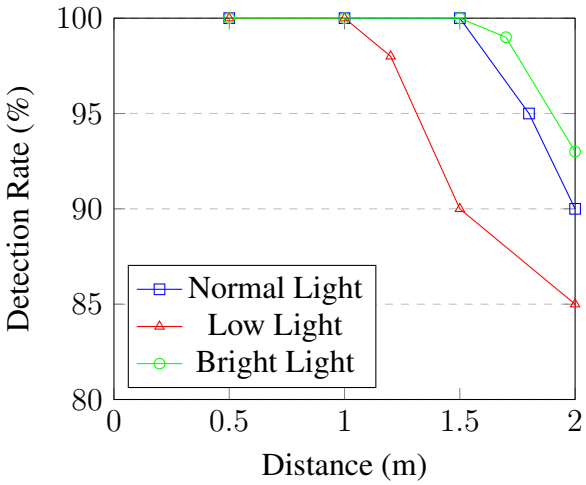


Figure 5.2: ArUco Marker Detection Rate vs. Distance

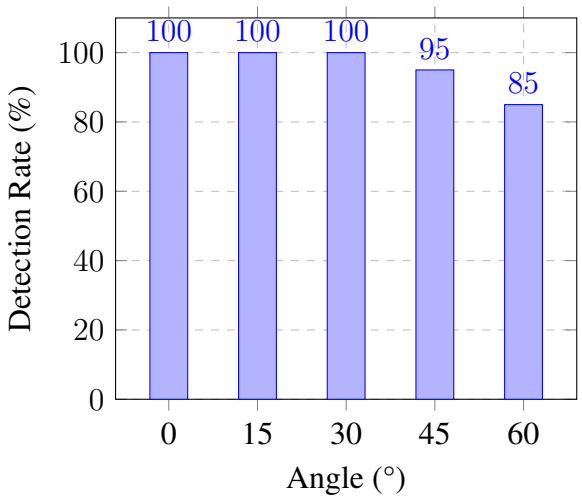


Figure 5.3: Detection Rate vs. Angle for ArUco Marker Detection

Angle Robustness: For angular testing, 100 images were captured at angles from 0° to 60° in 15° increments. Detection rates were analyzed to understand performance under different orientations.

5.1.3.2 Distance Measurement Testing

Data Collection: Measurements were taken using the `estimate_pose` function at fixed distances (0.5m, 1m, 1.5m, 2m), with 100 samples at each point.

Analysis: The Mean Absolute Error (MAE) was calculated between measured and actual distances. Consistency was tested by taking 500 consecutive measurements at 1m, calculating standard deviations and confidence intervals.

Table 5.2: Distance Measurement Accuracy

Actual Distance	Mean Absolute Error	Error Percentage
0.5m	0.015m	3.0%
1.0m	0.025m	2.5%
1.5m	0.045m	3.0%
2.0m	0.070m	3.5%

These comprehensive tests validate the ArUco marker detection system's performance, accuracy, and reliability across various conditions. The results demonstrate the system's capability to provide accurate distance measurements and speed estimations, crucial for the robot's navigation and interaction with its environment. The high detection rates and low processing times confirm the suitability of ArUco markers for our real-time robotics application, aligning with the findings of Patru et al. [37].

The integration testing results confirm that the complete system can operate in real-time, successfully detecting markers, estimating distances and speeds, and responding to control zones. These findings provide a solid foundation for the robot's ability to navigate and interact with its environment using ArUco markers as reference points.

5.2 Subsystem Combinations Testing

After independent testing, the next phase was to combine the subsystems and test their interactions.

5.2.1 Camera and Web Interface Integration

The integration of the camera feed into the web interface was essential for providing real-time visual feedback to the user. The camera was initially connected to the Raspberry Pi, and a Flask-based web interface was used to stream the video feed. The initial implementation focused on setting up the basic functionality of streaming the live video feed through a web page.

5.2.1.1 Testing Video Streaming Performance

Once the camera feed was integrated into the web interface, the system was tested to evaluate its performance. During the initial tests, it was observed that the video stream had a significant lag of **2-3 seconds**. This was primarily due to the high frame rate of **30 FPS** (frames per second),

which caused data transmission to overwhelm the available bandwidth. The delay resulted in a poor user experience, especially when real-time responsiveness was crucial for controlling the robot.

To assess the performance:

- The latency was measured as the delay between the video captured by the camera and its display on the web interface.
- The frame rate, resolution, and bandwidth usage were analyzed to determine the cause of the performance bottlenecks.
- It was also noted that the web browser's buffering behavior contributed to the observed lag.

5.2.1.2 Performance Optimization and Improvements

To improve the streaming performance, several optimizations were implemented. The following changes were made:

- **Frame Rate Reduction**: The frame rate was reduced from **60 FPS** to **30 FPS**. This lowered the amount of data transmitted per second, reducing the load on both the network and the Raspberry Pi, thereby decreasing the lag.
- **Resolution Adjustment**: The resolution of the video feed was scaled down to **640x480**. This significantly reduced the bandwidth consumption, enabling a smoother video feed without affecting the user's ability to interact with the system effectively.

Final Results:

- Initial lag: **2-3 seconds** at 60 FPS.
- Optimized lag: **500ms** at 30 FPS with **320x240** resolution.
- Smooth, reliable video streaming with minimal frame drops.

This phase of testing ensured that the camera feed was effectively integrated into the web interface and optimized to provide responsive, real-time feedback necessary for controlling the robot.

5.2.2 Web Interface and Motor Control Integration

For integrating the web interface and motor control, the following steps were taken:

1. Developing a separate program to listen for key inputs (W, A, S, D).
2. Using SocketIO for real-time communication between the web interface and motor control program.
3. Creating an interface to send PWM signals from the web interface to the motors via the H-bridge.

Listing 5.1: SocketIO event handler for robot movement

```
# SocketIO event to handle robot movement
@socketio.on('move')
def handle_move(direction):
    global forward_disabled
    if direction == 'forward' and forward_disabled:
        print("Forward movement is disabled [restricted zone]")
        robot.stop()
    elif direction == 'forward':
        robot.move_forward()
    elif direction == 'backward':
        robot.move_backward()
    elif direction == 'left':
        robot.turn_left()
    elif direction == 'right':
        robot.turn_right()
    elif direction == 'stop':
        robot.stop()
```

The web interface successfully controlled the robot's movements, with a measured control latency of **300ms**. Minor delays were noted during rapid direction changes, which could require further optimization.

Results:

- Control latency: **300ms**.
- Issues: Slight delay during rapid changes.

5.2.3 Web Interface and ArUco Detection Integration

To integrate ArUco marker detection into the web interface, the camera feed was processed in real time, and marker IDs were displayed on the interface. The system could detect markers with a delay of **100ms** when more than three markers were present in the frame simultaneously.

Results:

- Detection delay: **100ms** with three or more markers.
- Detection accuracy: **100%** for up to three markers.

5.3 INTEGRATION PHASE

The final integration phase brought together the core components: **Aruco marker detection**, **motor control**, and the **web interface**. This stage focused on combining the previously tested individual systems into a unified framework capable of real-time robot control and marker detection.

Key steps included:

1. Merging the motor control program with the ArUco marker detection system to allow the robot to respond to detected markers.
2. Real-time processing of the video feed from the camera for marker detection, alongside real-time control through the web interface.
3. Implementing pre-defined behaviors that adjusted motor actions (such as speed and direction changes) based on the specific markers detected by the camera.

Features such as distance measurement, speed estimation, and marker-based tasks were also developed and tested independently before being added to the full system. These functionalities were integrated step-by-step to ensure proper interaction between components.

A more detailed discussion of the final system's performance, including its compliance with the Acceptance Test Procedures (ATPs), will be covered in the next chapter. There, the results of the fully integrated system will be examined in line with the project's objectives.

CHAPTER 6

RESULTS AND DISCUSSION

This chapter presents the results of the testing and evaluation process outlined in the previous chapter, focusing on how the system performed in real-world conditions. The integration of the web interface, ArUco marker detection, and motor control has been fully implemented, and this chapter aims to assess the performance of these subsystems when working together.

Each aspect of the system, from real-time video streaming to ArUco marker-based navigation, is evaluated according to the Acceptance Test Procedures (ATPs) defined earlier in the project. The discussion will focus on the key metrics such as latency, detection accuracy, and system responsiveness. Any challenges encountered during testing, as well as any deviations from expected outcomes, will be analyzed to provide insight into potential areas for improvement.

In this chapter, I will:

- Present the outcomes of the individual and integrated system tests.
- Analyze how well the system meets the predefined objectives and ATPs.
- Discuss the implications of the findings, identifying both the strengths and weaknesses of the implemented solution.
- Highlight any areas where optimization was required or could be further pursued to enhance system performance.

CHAPTER 7

CONCLUSIONS AND FURTHER WORK

This chapter presents the conclusions and future work for this dissertation.

7.1 CONCLUSIONS

Figure 7.1: Useful concluding diagram

7.2 RECOMMENDATIONS FOR FURTHER WORK

BIBLIOGRAPHY

[1] ResearchGate. Augmented reality types and popular use cases. *ResearchGate*, 2024. [Online; accessed 18-October-2024].

[2] Random Nerd Tutorials. Complete guide for ultrasonic sensor hc-sr04 with arduino. `https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/`, 2021. Accessed: 2024-09-21.

[3] Mobileye. A brief history of autonomous vehicles – from renaissance to reality. *Mobileye Blog*, 2023. Accessed: 2024-08-18.

[4] RC Crush. A complete rc car history. *RC Crush*, Apr 2023. Accessed: 2024-08-18.

[5] "Gregor Klancar", "Andrej Zdesar", "Saso Blazic", and "Igor Skrjanc". *Wheeled Mobile Robotics*. Google Books, 2017. Accessed: 2024-08-18.

[6] S. Samuels. What is the fifth industrial revolution and are we in it? *Skills Portal*, 2023. Accessed: 2024-09-12.

[7] M. Dalle Mura and G. Dini. Augmented reality in assembly systems: State of the art and future perspectives. In Svetan Ratchev, editor, *Smart Technologies for Precision Assembly*, pages 3–22, Cham, 2021. Springer International Publishing.

[8] A. Hentout, M. Aouache, A. Maoudj, and I. Akli. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. *Advanced Robotics*, 33(15-16):764–799, 2019. Accessed: 2024-09-16.

[9] R. Suzuki, A. Karim, T. Xia, H. Hedayati, and N. Marquardt. Augmented reality and robotics: A survey and taxonomy for ar-enhanced human-robot interaction and robotic interfaces. *arXiv preprint arXiv:2203.03254*, 2022. Accessed: 2024-09-16.

[10] E. Coronado, S. Itadera, and I. G. Ramirez-Alpizar. Integrating virtual, mixed, and augmented reality to human-robot interaction applications using game engines: A brief review of accessible software tools and frameworks. *Applied Sciences*, 13(3):1292, 2023.

[11] Yu Lei, Zhi Su, and Chao Cheng. Virtual reality in human-robot interaction: Challenges and benefits. *AIMS Robotics*, 7(4):477–495, 2022.

[12] Michael Walker, Thao Phung, Tathagata Chakraborti, Tom Williams, and Daniel Szafir. Virtual, augmented, and mixed reality for human-robot interaction: A survey and virtual design element taxonomy. *arXiv preprint*, 2019.

[13] S. A. Green, X. Chen, and M. Billinghurst. Collaborating with a mobile robot: An augmented reality multimodal interface. *Robotics and Autonomous Systems*, 122:103–110, 2019. Accessed: 2024-09-16.

[14] G. Michalos et al. Augmented reality applications for supporting human-robot interactive cooperation. *Procedia CIRP*, 100:79–84, 2022. Accessed: 2024-09-16.

[15] Michael D. Coovert, Tiffany Lee, Ivan Shindev, and Yu Sun. Spatial augmented reality as a method for a mobile robot to communicate intended movement. *Computers in Human Behavior*, 34:241–248, 2014.

[16] P. Daponte, L. De Vito, and F. Picariello. State of the art and future developments of the augmented reality for measurement applications. *IEEE Transactions on Instrumentation and Measurement*, 69(2):455–466, 2020. Accessed: 2024-09-16.

[17] K. Filus and P. Klakowicz. Real-time testing of vision-based systems for agvs with aruco markers. *Journal of Intelligent & Robotic Systems*, 105(1):17–32, 2023. Accessed: 2024-09-16.

[18] J. Husar and K. Wrona. Possibilities of using augmented reality in warehouse management: A study. *International Journal of Production Research*, 60(14):4128–4140, 2022. Accessed: 2024-09-16.

[19] J. Fu, A. Rota, S. Li, J. Zhao, Q. Liu, E. Iovene, G. Ferrigno, and E. De Momi. Recent advancements in augmented reality for robotic applications: A survey. *Actuators*, 12(8):323, 2023. Accessed: 2024-09-16.

[20] Z. Makhataeva and H. A. Varol. Augmented reality for robotics: A review. *Robotics*, 9(2):21, 2020. Accessed: 2024-09-16.

[21] Moraneus. Python flask: A comprehensive guide from basic to advanced. *Medium*, Mar 2024. Accessed: 2024-09-28.

[22] Asian Digital Hub. What is node.js and how it work? *Medium*, Dec 2023. Accessed: 2024-09-28.

[23] MDN Web Docs. Websockets and real-time web applications. 2024. Accessed: 2024-09-28.

[24] Tonino Jankov. Nginx vs apache: Web server showdown. *Kinsta®*, Jun 2019. Accessed: 2024-09-28.

[25] N. Maupin and N. Maupin. How do ultrasonic sensors work? `https://howdowork.com/technology/computers-electronics/ultrasonic-sensors-working/#:~:text=How%20Do%20Ultrasonic%20Sensors%20Work%20Ultrasonic%20sensors%20work,they%20are%20partially%20reflected%20back%20to%20the%20sensor.`, September 2023. Accessed: 2024-09-21.

[26] T. Yang et al. 3d tof lidar in mobile robotics: A review. *arXiv*, 2022. Accessed: 2024-09-28.

[27] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2001.

[28] Jason Brownlee. Using haar cascade for object detection, 2024. Accessed: 2024-10-18.

[29] How to Train Your Robot. Training mobilenet ssd on custom objects, 2023. Accessed: 2024-10-18.

[30] TensorFlow. Tensorflow lite for mobile and embedded devices, 2024. Accessed: 2024-10-18.

[31] Glenn Jocher. Yolov5: Pytorch implementation, 2024. Accessed: 2024-10-18.

[32] Muhammad Hussain. Yolov5, yolov8 and yolov10: The go-to detectors for real-time vision, 2024. Accessed: 2024-10-18.

[33] 'G.C. La Delfa', 'V. Catania', 'S. Monteleone', 'J.F. De Paz', and "J. Bajo". Computer vision based indoor navigation: A visual markers evaluation. In *Ambient Intelligence - Software and Applications*, pages 165–172. Springer, 2015.

[34] D. Divani I. Jacobsen, F. Supty. Remote-controlled camera robot. In *BCIT Capstone Project Report*. BCIT, 2018.

[35] R. Selvarasu M. Vanitha, M. Selvalakshmi. Monitoring and controlling of mobile robot via internet through raspberry pi board. In *2016 Second International Conference on Science Technology Engineering and Management (ICONSTEM)*, pages 462–463. IEEE, 2016.

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 06 2015.

[37] George-Cristian Patru, Alina-Irina Pirvan, Daniel Rosner, and Razvan-Victor Rughinis. Fiducial marker systems overview and empirical analysis of aruco, apriltag and cctag. *U.P.B. Sci. Bull., Series C*, 85(2):49–62, 2023.

APPENDIX A

ADC/DAC CORE

A.1 A USEFUL SUB APPENDIX

Put your appendix stuff here.