Université Mohamed Khider de Biskra
Faculté des Sciences et de la Technologie
Département de génie électrique

MÉMOIRE DE MASTER

Sciences et Technologies
Automatique
Automatique et informatique industriel

Réf. : Entrez la référence du document

Présenté et soutenu par :
Houhou Mohamed Dhia Eddine & Haouioui Sid Ahmed

Le : mardi 3 juin 2025

Design and Implementation of a Camera-Controlled Robotic System

Jury :

Mme.	Ouarhlent Saloua	MCB	Université de Biskra	Président
Mme.	Nebbar Hanane	MAA	Université de Biskra	Examinateur
Mme.	Terki Nadjiba	Prof	Université de Biskra	Rapporteur

Année universitaire : 2024/2025

Université Mohamed Khider de Biskra
Faculté des Sciences et de la Technologie
Département de génie électrique

MÉMOIRE DE MASTER

Sciences et Technologies
Département de génie électrique
Automatique et informatique industriel

Réf. : Entrez la référence du document

Design and Implementation of a Camera-Controlled Robotic System

Le : mardi 3 juin 2025

Présenté par :
- Haouioui Sid Ahmed
- Houhou mohamed Dhia Eddine

Avis favorable de l'encadreur :

Prof.Terki Nadjiba

Signature Avis favorable du Président du Jury

Cachet et signature

Abstract:

This work presents the development of an autonomous mobile robot equipped with a 5-DOF robotic arm and a deep learning-based vision system. The project is divided into several tasks. First, the mechanical platform, which forms the foundation for the robot's mobility and manipulation capabilities, is realized. Next, robot control and navigation based on visual information are addressed. Object recognition and manipulation are achieved using a custom-trained YOLOv8n model and ArUco markers, with processing performed on a Raspberry Pi 4. Video input is captured via a Pi Camera. OpenCV is utilized for real-time image analysis, key feature detection, and providing visual feedback to enable precise object localization and effective robot manipulation. The motors and servos are operated using an Arduino Mega, which manages precise motor actuation and servo positioning to ensure smooth movement and accurate object handling. Control modes, both manual and automatic, are facilitated by a custom-developed Android application. Finally, the prototype was demonstrated through a waste sorting task to showcase the integrated perception, control, and manipulation capabilities. **You can find the project files in the github page below:**

https://github.com/SidCodesRobotics/yolo-robot-picknplace

Keywords: YOLOv8n, OpenCV, Raspberry Pi, Arduino Mega, robotic arm, Machine vision, mobile robot, Android app.

:الملخص

يقدم هذا العمل تطوير روبوت متنقل ذاتي مزود بذراع روبوتية بخمسة درجات حرية ونظام رؤية قائم على التعلم العميق. ينقسم المشروع إلى عدة مهام. أولاً، يتم تنفيذ بناء المنصة الميكانيكية التي تشكل أساس قدرات الحركة والمعالجة للروبوت. بعد ذلك، يتم تناول التحكم في الروبوت والملاحة استنادًا إلى المعلومات البصرية. يتم التعرف على الأشياء والتعامل معها باستخدام نموذج YOLOv8n مُدرب خصيصًا وعلامات ArUco، مع معالجة تتم على جهاز Raspberry Pi 4. يتم التقاط الفيديو عبر كاميرا Pi. يُستخدم OpenCV في تحليل الصور في الوقت الحقيقي، واكتشاف الميزات الرئيسية، وتوفير تغذية بصرية تمكن من تحديد مواقع الأشياء بدقة وتحقيق تعامل فعال من قبل الروبوت. يتم التحكم في المحركات والمحركات الصغيرة (السيرفو) بواسطة لوحة Arduino Mega، التي تقوم بتحريك المحركات بدقة وتحديد مواضع السيرفو لضمان حركة سلسة وتعامل دقيق مع الأشياء. تسهل تطبيقات أندرويد المخصصة أوضاع التحكم اليدوي والآلي. أخيرًا، تم عرض النموذج الأولي في مهمة فرز النفايات لإظهار القدرات المتكاملة في الإدراك والتحكم والمعالجة. **يمكنك العثور على ملفات المشروع في صفحة GitHub التالية:**

https://github.com/SidCodesRobotics/yolo-robot-picknplace

الكلمات المفتاحية: YOLOv8n ، OpenCV، Raspberry Pi، Arduino Mega، ذراع آلية، الرؤية الالية، روبوت متنقل، تطبيق أندرويد.

Acknowledgement

We would like to express our sincere gratitude to all those who contributed to the completion of this project.

First and foremost, we want to thank our supervisor, Pr. Terki Nadijba, for her guidance, continuous support, and staying committed to the success of this project. This work could not have come together without her valuable expertise and support. We are also grateful for her generous funding of the hardware components and resources that made this project possible.

We would also like to thank the Lab of Vision and Communication System for providing us with the necessary facilities and computational resources. we greatly depended on our access to their computer and laboratory resources.

Finally, we acknowledge each other as collaborators and partners in this academic journey. The successful completion of this project is the result of our combined efforts, shared knowledge, and mutual support.

Dedication

All praise and gratitude go, first and always, to Allah Almighty, whose mercy, guidance, and blessings have carried me through every step of this journey. Without His light, I would not have reached this point.

To my father, your quiet strength, sacrifices, and belief in me have been the foundation I've stood on.

To my mother, your endless love, gentle words, and late-night prayers were my greatest source of peace and courage.

To my brothers and sisters, your laughter, encouragement, and support in both good times and hard moments meant more than I can ever express.

To every member of my family, near and far, who cheered me on, offered a helping hand, or simply believed in me, thank you for walking this path with me, even from a distance.

To the teachers and professors who guided me, challenged me, and inspired me — your wisdom and patience helped shape this work, and I carry your lessons with pride and gratitude.

This achievement is not mine alone, it is ours. From the depths of my heart, thank you.

Houhou Mohamed Dhia Eddine

First and foremost, To my beloved family, my mother, whose support, heartfelt prayers, have been my greatest source of strength. Thank you for the sleepless nights, for always listening with patience, and for believing in me even when I doubted myself.

To my brother Abdennour, who never hesitated to help running errands, helping the best way he knew how.

To my sister who stood by me throughout my academic path, offering guidance, motivation, and support even before university and long after.

To my friends, fellow researchers, and dedicated teachers, thank you for your encouragement, collaboration, and belief in me. Each of you holds a part of this achievement in your hands.

And to myself this journey was long and often exhausting, but through perseverance, sleepless nights, and unwavering determination, I turned challenges into growth and dreams into reality. I am proud of the person I've become and the work I've accomplished.

Haouioui Sid Ahmed

Table of contents

List of Figures:

LIST OF TABLES:

LIST OF ABBREVIATIONS:

YOLO: You Only Look Once

DOF: Degrees of Freedom

PWM: Pulse-Width Modulation

UART: Universal Asynchronous Receiver/Transmitter

mAP: mean Average Precision

MCU: Microcontroller Unit

OS: Operating System

LiPo: Lithium Polymer

CSI: Camera Serial Interface

NCNN: High-performance Neural Network Inference Computing Framework

IoU: Intersection over Union

IDE: Integrated Development Environment

RAM: Random Access Memory

IC: Integrated Circuit

SDK: Software Development Kit

mAh: milliampere-hour

TP: True Positives

FP: False Positives

FN: False Negatives

TN: True Negatives

FPS: Frames Per Second

PLA: Polylactic Acid

GND: Ground

API: Application Programming Interface

HDMI: High-Definition Multimedia Interface

ROS: Robot Operating System

BLE: Bluetooth Low Energy

SRAM: Static Random-Access Memory

COCO: Common Objects in Context

GPU: Graphics Processing Unit

General Introduction

The rapid advancement of automation technologies has heightened the demand for autonomous mobile robots that can perform complex tasks in dynamic and unstructured environments. Traditional fixed automation systems often lack the flexibility needed to adapt to varying industrial scenarios, making mobile manipulators essential for versatile operations. Industries such as manufacturing, logistics, agriculture, and healthcare increasingly require robots capable of not only navigating challenging terrains but also manipulating objects with precision and intelligence. Moreover, the need for robots that can operate safely alongside humans in cluttered or hazardous environments is critical to improving productivity and reducing workplace injuries. Autonomous robots equipped with advanced sensory perception, including computer vision, enable real-time decision-making and adaptability, which are crucial for handling unpredictable situations. This creates an urgent industrial need for robust, cost-effective, and scalable robotic platforms that integrate mobility, manipulation, and intelligent perception.

Developing such systems can significantly enhance automation capabilities, streamline workflows, and open new possibilities for applications in inspection, maintenance, search and rescue, and waste management.

The objective of this project is to develop a fully autonomous mobile robot capable of integrated navigation, object recognition, and manipulation. Specifically, the robot's design is based on a multi-degree-of-freedom robotic arm combined with a computer vision system that enables real-time environmental perception and decision-making. The project focuses on achieving effective collaboration between mechanical design, embedded control systems, and intelligent software algorithms, utilizing affordable, readily available components and open-source tools. Emphasis is placed on creating a robust system that can adapt to dynamic and unstructured environments, ensuring reliable performance in practical scenarios. For performance validation, a real-world application is adopted: an automated waste sorting task that demonstrates the robot's ability for accurate object detection, precise grasping, and reliable placement in unstructured environments. This application highlights the robot's potential for deployment in industrial and environmental management contexts, where efficiency and adaptability are critical.

This manuscript is organized to comprehensively detail the development process undertaken:

- **Chapter 1** provides a general background on robotics, introducing key theoretical concepts related to mobile robots and manipulators.
- **Chapter 2** focuses on the hardware components of the robot, discussing the selection of actuators, power systems, control boards, sensors, and the overall mechanical design, including the 3D-printed parts that were designed and fabricated.
- **Chapter 3** addresses the software architecture, detailing the vision processing system implemented on the Raspberry Pi, methods for object detection and marker-based localization, the firmware developed for the Arduino responsible for physical actuation and sensor data handling, as well as an overview of the user interface.
- **Chapter 4** presents the results of the project, showcasing the assembled robotic system, evaluating the performance of its vision capabilities, and demonstrating its ability to execute a defined manipulation task from start to finish. This chapter also discusses encountered challenges and limitations and proposes potential solutions for future improvements and system expansions.

Chapter I

Overview about robotics

I.1 Introduction

Robotics is the interdisciplinary field focused on designing and constructing devices that emulate living creatures to perform tasks and exhibit behaviors similar to humans. This fascination dates back centuries, from ancient automata to modern cinematic portrayals. Industrial robots, such as those developed by companies like KUKA, ABB, and FANUC, play a vital role in manufacturing due to their precision and ability to operate in hazardous environments.

Despite high expectations, the adoption of robots in domestic applications has been limited; however, ongoing technological advancements promise continued improvements.

Robotics combines artistic creativity, accumulated knowledge, and practical skills to design, develop, and implement robots for various human applications. This integration aims to create systems capable of performing tasks efficiently across diverse domains.

This chapter delves into the field of robotics, tracing its historical evolution and focusing on mobile robots. It examines localization techniques, application domains, and analyzes the advantages and disadvantages of these robots.

I.2 Definition of robotics

Robotics is a multidisciplinary field that combines engineering, science, and technology—including elements of art—to design, develop, and operate autonomous or semi-autonomous machines capable of performing tasks across various human activities. It integrates mechanical and electronic systems, control engineering, computer algorithms, and artificial intelligence to enable robots to function effectively in both structured and unstructured environments

The discipline also encompasses perception systems, sensor fusion, real-time data processing, and human-robot interaction to ensure efficient and adaptive performance. Robotic systems typically include not only the physical robot, but also complementary devices, sensors, and control mechanisms. Robots are applied in diverse settings such as manufacturing, underwater and space exploration, healthcare, transportation, military operations, education, and entertainment. **[1].**

Drawing from mechanical engineering, electrical and electronic engineering, computer science, cognitive sciences, and biology, robotics stands as an inherently interdisciplinary domain. The effectiveness and versatility of robots across these sectors depend heavily on precise programming and system integration tailored to their operational context.

I.3 Defintion of robots

Robots are characterized by their ability to be reprogrammed to perform various tasks without requiring physical redesign. Definitions and standards for what qualifies as a robot vary across countries, the emphasis is placed on reprogrammability as a key defining feature. For instance, one definition describes robots as systems designed to perform specific tasks, often integrated with other devices.

A key distinction is made between robots and simple manipulators: robots possess a certain level of autonomy and reprogrammability, whereas manipulators typically perform fixed or limited functions without adaptive control. A crane requires continuous human operation to control its movements and perform tasks, lacking autonomy or programmability.

A robotic manipulator, such as those developed by KUKA, can execute complex tasks based on programmable instructions, providing flexibility, adaptability, and a degree of autonomy in its operations.

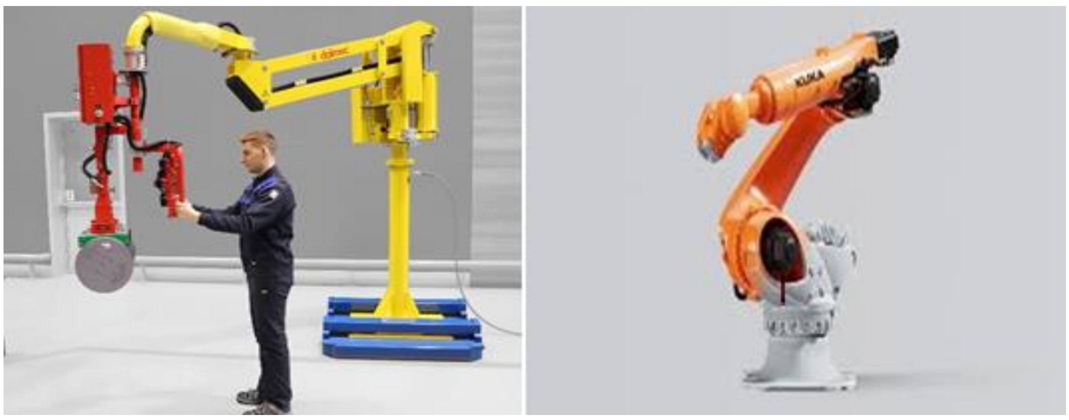


Figure I.1 A Dalmec manipulator on the left, a KUKA robot on the right [1]

An alternative definition would be that of the Robot Institute of America, 1979: “A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks (Jablonski and Posey, 1985)” **[2]**.

I.4 Robots history

The history of robots traces back thousands of years, rooted in ancient mythology, mechanical inventions, and the gradual evolution of automation technologies. Here's an overview:

I.4.1 Ancient and Pre-Modern Era (Before the 1800s)

- **Mythological Origins**

  Ancient myths often depicted mechanical beings. In Greek mythology, Hephaestus, the god of fire and craftsmanship, created automata like Talos, a giant bronze man who guarded the island of Crete. [3]

- **Ancient Automata (3rd Century BCE – 1st Century CE)**

  In the 3rd century BCE, the Greek engineer Ctesibius of Alexandria developed advanced water clocks (clepsydrae) that included moving figures—early examples of automata integrated with timekeeping.

  In the 1st century CE, Hero of Alexandria designed mechanical devices powered by steam and air pressure, including the aeolipile (an early steam engine) and automated theaters that featured moving scenes and figures driven by intricate mechanical systems. [4]

- **Medieval Islamic World (9th–13th Century)**

  Engineers in the medieval Islamic world made significant contributions to mechanical engineering. Notably, Al-Jazari (active around 1206 CE) described a variety of complex machines in his work *The Book of Knowledge of Ingenious Mechanical Devices*. His inventions included programmable humanoid automata, such as a hand-washing robot, which demonstrated early concepts of automation and control systems [5].

- **Renaissance Automata (14th–17th Century)**

  During the Renaissance, advances in mechanics and anatomy inspired the development of automata. Leonardo da Vinci designed a humanoid robot knight with movable limbs. European clockmakers also built intricate, lifelike automata for entertainment[6].

I.4.2 Industrial Revolution and the Age of Mechanization (18th–19th Century)

- **Jacquard Loom (1801)**

  Invented by Joseph-Marie Jacquard, the Jacquard loom used punched cards to control complex weaving patterns. This innovation laid foundational principles for programmable machines and later influenced computer programming [7].

- **Mechanical Robots**

  The 18th century saw the creation of lifelike automata, such as the chess-playing "Turk" (1770) and the Digesting Duck by Jacques de Vaucanson, both of which fascinated audiences with their mechanical realism [8].

- **Rise of Automatons**

  During the Industrial Revolution, machines capable of performing repetitive tasks, particularly in textile manufacturing, emerged. These early automatons foreshadowed the development of modern industrial robotics [9].

I.4.3 20th Century: The Birth of Modern Robotics

- **Robots in Literature**

  The term "robot" was first coined in 1920 by Czech writer Karel Čapek in his play *R.U.R.* (*Rossum's Universal Robots*), where it described artificial workers created to serve humans [10].

- **Early Industrial Robots**

  In 1954, George Devol invented the first programmable robot, named Unimate, which was later used on industrial assembly lines starting in 1961 [11].

- **Artificial Intelligence (AI)**

  AI research began in the 1950s, with pioneers like Alan Turing and John McCarthy laying the groundwork for intelligent machines capable of learning, reasoning, and problem-solving [12].

- **Space Exploration**

  Robotic systems such as the Lunar Rovers (1970s) and Viking Landers were deployed for extraterrestrial exploration, marking a significant milestone in the use of robots beyond Earth [13].

I.4.4 21st Century: Advanced Robotics and AI Integration

- **Autonomous Robots**

  Robots such as the Roomba (2002) revolutionized the consumer robotics market, while Boston Dynamics' robots demonstrated remarkable humanoid and animal-like agility [14].

- **Collaborative Robots (Cobots)**

  Cobots are designed to work alongside humans safely and efficiently. They are widely used in manufacturing, logistics, and healthcare, enhancing productivity and flexibility [15].

- **AI-Powered Robots**

  Modern robots now incorporate advanced AI, enabling capabilities such as natural language interaction (e.g., Sophia by Hanson Robotics) and autonomous navigation, as seen in self-driving vehicles [16].

- **Military and Healthcare Applications**

  Robotics has expanded into fields like defense and medicine, with technologies including drones, robotic exoskeletons, and surgical systems like the Da Vinci Surgical System improving precision and safety [17].

I.5 Types of robots

Robots can be classified into several types based on their functions, design, and applications. The main categories include: industrial robots, service robots, humanoid robots, autonomous robots, mobile robots, underwater robots, military robots, agricultural robots, and space robots.

However, the primary focus of this study will be on:

- Industrial robots (manipulator robots)
- Mobile robots

I.5.1 Industrial robots (manipulator robots)

I.5.1.1 Defintion

Manipulator robots, also known as robotic arms, are fundamental components of modern industrial automation. They are designed to replicate the movements of a human arm, enabling them to perform a variety of tasks such as material handling, assembly, welding, and packaging.These robots typically consist of a series of links (rigid segments) and joints (flexible connections), which allow for precise and versatile movement.

The range of motion and flexibility of a manipulator robot depend on its design—specifically, the type and number of joints it possesses [18].The figure2 illustrates different types of manipulator robots, highlighting variations in structure and joint configuration.

Figure I.2 manipulator robots[20]

Key Features:

- **Degrees of Freedom (DOF):** This refers to the number of independent movements a manipulator can perform. For instance, a 6-axis manipulator has six degrees of freedom, enabling it to execute complex motions such as pitch, yaw, and roll [18].
- **End-Effector:** The tool attached to the end of the manipulator—such as a gripper, suction cup, or welding tool—which directly interacts with objects in the environment [19.The figure3 illustrates the basic structural components of a manipulator robot, showing how links (rigid segments) and joints (movable connections) work together to enable robotic motion.

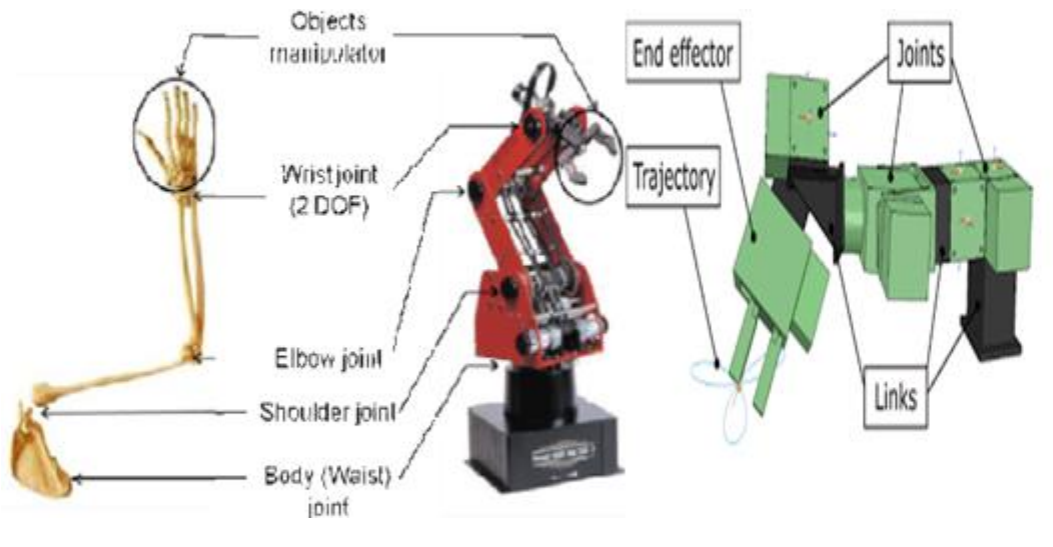


Figure I.3 Links and joints[20]

I.5.1.2 Mechanical structure and motricity:

Three distinct categories of mechanical structures are fundamental in enabling **robotic motricity** (movement and articulation):

- **Serial Robots** — These are **flexible and precise robotic systems** composed of interconnected **links and joints**, forming a chain-like structure. They are commonly used in **manufacturing** and **medical** applications. Serial robots are relatively **easy to control**, but they typically have **lower payload capacity** and **less stiffness** compared to other robotic architectures [21]. The figure4 is a representation of a serial robot, illustrating its chain-like structure composed of interconnected links and joints used to achieve flexible and precise movement**.**

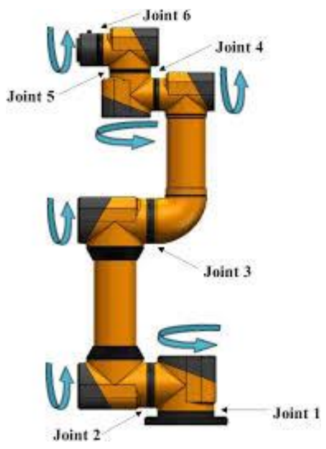


Figure I.4 Serial robot[20]

- **Parallel Robots** — These are **multi-degree-of-freedom robotic systems** characterized by a **closed-loop structure**, in which the **end-effector is connected to the base by multiple independent kinematic chains**. Parallel robots are known for their **high stiffness**, **precision**, and **load-bearing capacity**, making them ideal for tasks that demand **high accuracy and strength**, such as **surgical robotics** and **precision manufacturing**. However, their **complex design and control systems** can present challenges compared to serial robots [22]. The figure5 is a representation of a **parallel robot**, showing its closed-loop structure with multiple kinematic chains connecting the end-effector to the base, enabling high precision and load capacity.

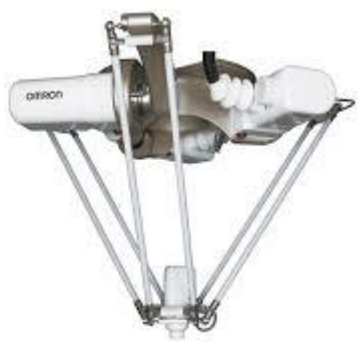


*Figure I.5 Parallel robot***[26]**.

- **Hybrid Robots** — These robots **combine the flexibility of serial robots** with the **precision and load-bearing capacity of parallel robots**. This combination makes them ideal for **complex tasks** that require both **mobility and accuracy**, such as in **surgical procedures** and **advanced manufacturing**. However, their **design and control systems** are typically **more complex** than those of purely serial or parallel robots **[23]**.The figure6 presents an example of a **hybrid robot**, demonstrating the combination of flexibility and precision for complex tasks requiring both mobility and accuracy.

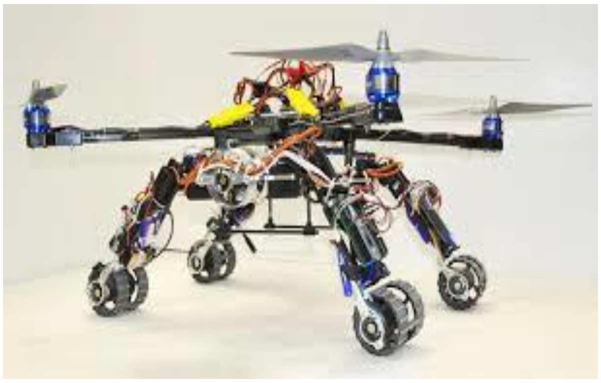


Figure I.6 Hybrid robot[27]

I.5.1.3 Fields of application

- **Industry**

  Manipulator robots are widely used in manufacturing to automate various production processes(see figure7), including assembly, welding, painting, and packaging. These robots can handle heavy loads and perform repetitive tasks with high precision and consistency, thereby increasing productivity and reducing the risk of human error.

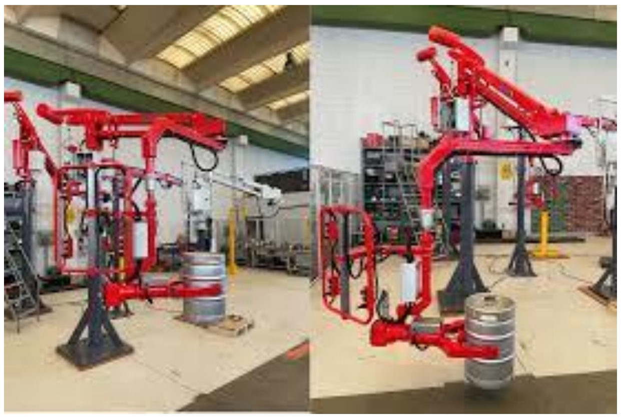


*Figure I.7 Industry manipulators robot***[28]**

- **Healthcare**

  Manipulator robots(Figure 8) are increasingly being integrated into healthcare applications, including surgery, rehabilitation, and assistive technologies for individuals with disabilities. These robots can perform delicate and highly precise procedures, reducing the risk of human error and contributing to improved patient outcomes and enhanced treatment effectiveness

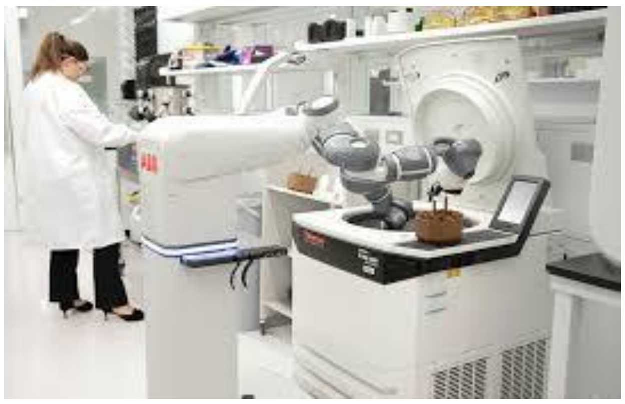


*Figure I.8 Healthcare manipulators robot***[29]**

- **Space**

  Manipulator robots play a crucial role in space exploration, performing tasks such as the assembly, maintenance, and repair of spacecraft and equipment. Designed to operate in zero-gravity environments, these robots carry out operations that would be too hazardous or complex for human astronauts, thereby enhancing the safety and efficiency of space missions**.** (The figure9 is a representation of a **space manipulator robot)**

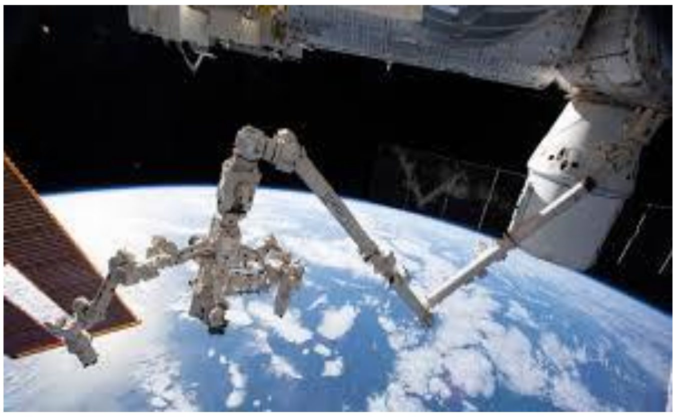


*Figure I.9 Space manipulators robot***[30]**

I.5.2 Mobile robots

I.5.2.1 Definition

Mobile robots are autonomous or remotely controlled machines capable of moving and operating in a variety of environments—including indoor and outdoor spaces, urban and rural areas, as well as hazardous or hard-to-reach locations. These robots are equipped with sensors and actuators, enabling them to perceive their surroundings, navigate obstacles, and perform a wide range of tasks.

Mobile robot(an example is shown in fugure10) are designed for diverse applications such as transportation, exploration, inspection, surveillance, and entertainment. They are powered by various energy sources, including batteries, fuel cells, and solar panels. Increasingly, mobile robots are being adopted across industries like healthcare, logistics, agriculture, and construction to improve efficiency, safety, and productivity [24].

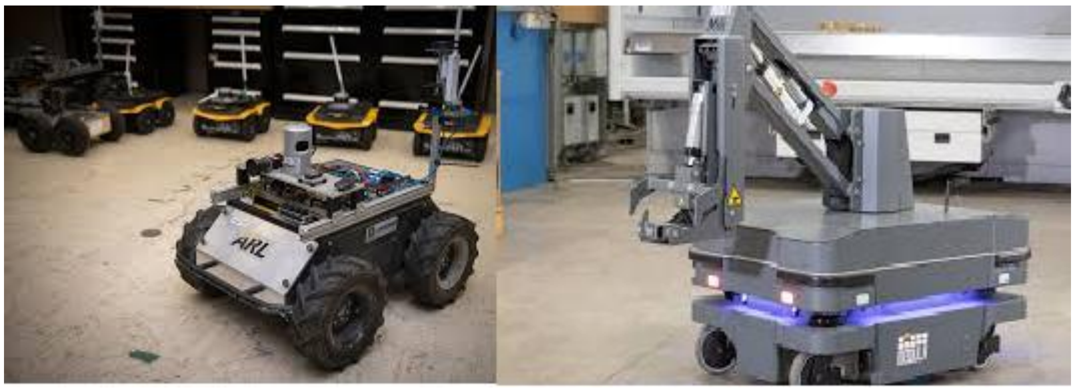


*Figure I.10 Examples of mobile robots***[20]**

I.5.2.2 General Types of Mobile Robots

Mobile robots are a versatile category of machines designed to navigate and perform tasks in a wide range of environments. They are typically classified based on their locomotion mechanisms, including wheeled, legged, crawler, flying (aerial), aquatic, and swarm robots. Among these, legged robots and wheeled robots are two of the most prominent types [24].

I.5.2.2.1 Legged Mobile Robots

Legged robots (see figure 11) use legs as their primary means of movement, mimicking the locomotion of animals such as insects and mammals. They are particularly well-suited for navigating uneven terrain and overcoming obstacles that wheeled robots may not be able to traverse.

These robots can be bipedal (two legs), quadrupedal (four legs), or hexapod (six legs), and typically use hydraulics, pneumatics, or electric motors to power their movement.However, the design and control of legged robots are complex, requiring advanced control systems, as well as significant power and computational resources. These challenges can limit their endurance, range, and energy efficiency.

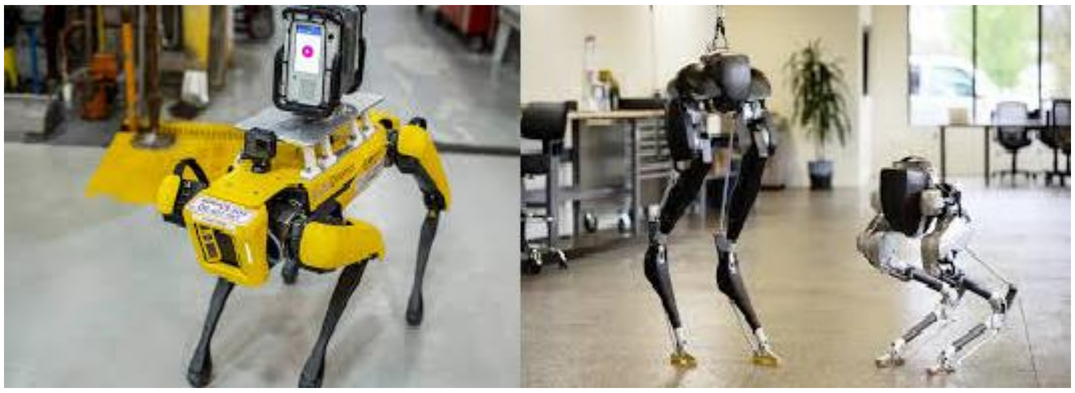


Figure I.11 legged robots[20] [31]

I.5.2.2.2 Wheeled Mobile Robots

Wheeled robots(figure12), by contrast, use wheels as their primary means of movement. They are generally best suited for flat or smooth surfaces and come in a wide range of sizes—from small mobile units to large industrial machines. Wheeled robots are widely used due to their simplicity, efficiency, and relatively low cost of design and control.

Figure I.12 mobile robots with wheels[20]

Differential wheeled robots(figure 15), a subset of tricycle robots, feature two driven wheels powered by independent actuators, with a third passive wheel for support.

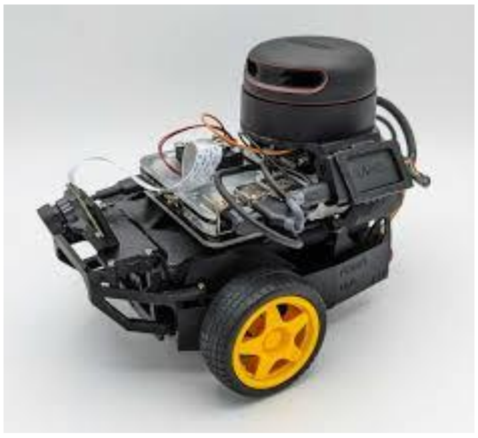


Figure I.13 Differential robot[33]

- **Omni-directional Robots**(figure16) These robots, equipped with three motorized wheels arranged in a triangle, can move in any direction with equal ease, enabling greater manoeuvrability.

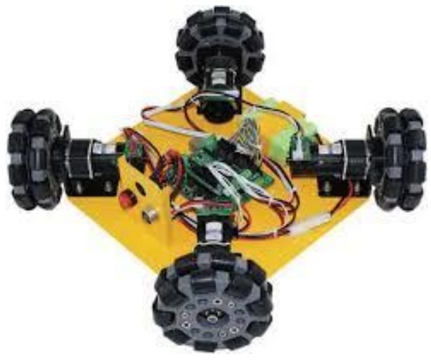


Figure I.14 Omni-directional robot[34]

- **Four-Wheel Drive Robots**: These robots have four or more wheels, providing enhanced stability and mobility, and can operate in various terrains. They are commonly used for remote or autonomous operation in challenging environments.

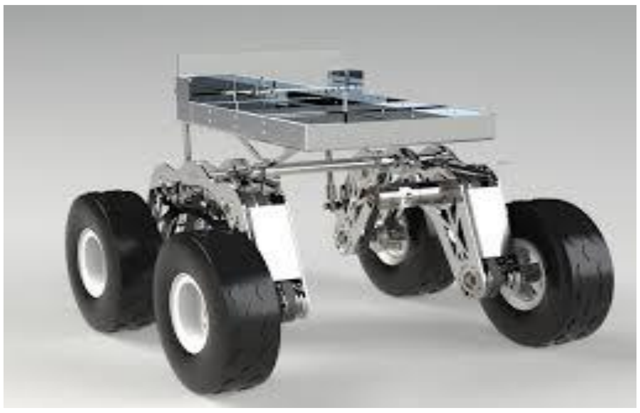


Figure I.15 Four-wheel drive robot[35]

I.5.2.3 Fields of application

Mobile robots have a broad range of applications across different industries:

1. **Agriculture**(see figure 18): Mobile robots are used in agriculture for tasks such as planting, harvesting, and crop monitoring, improving efficiency and reducing labor costs .

Figure I.16 Mobile robots in agriculture[36]

2. **Logistics and Warehousing**: In logistics, mobile robots help move goods(figure 19) through warehouses, navigating narrow aisles and improving operational efficiency.

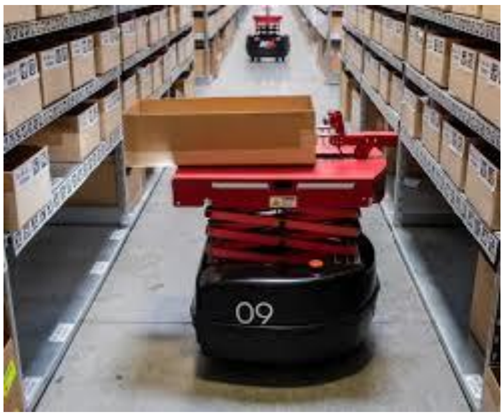


Figure I.17 Mobile robots in warehousing[37]

3. **Military and Defence**(example in figure20) Mobile robots in military applications perform tasks like reconnaissance, surveillance, and demining, operating in difficult terrain and performing dangerous tasks, reducing human risk.

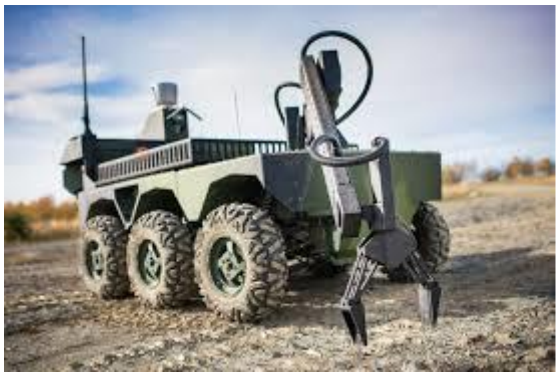


Figure I.18 Mobile robots in military and defence[20]

As technology evolves, mobile robots are expected to expand into even more diverse sectors, continuing to enhance efficiency, safety, and productivity.

I.6 Advantages and disadvantages of robots

Robots offer substantial benefits to workers, industries, and entire nations. When effectively implemented, industrial robotics can significantly enhance the quality of life by relieving workers from tasks that are monotonous, hazardous, dirty, or physically demanding.

However, despite these advantages, the widespread deployment of robots also brings a number of potential disadvantages and challenges that must be thoroughly considered [25].

I.6.1 Advantages of robot

The advantages of robotic systems are numerous and cannot be fully encapsulated in a concise overview. Nevertheless, several key benefits are widely recognized in the literature [25]:

- **Consistency:** Robots demonstrate a high level of repeatability in performing routine tasks, ensuring that operations are carried out with uniform accuracy and quality. This is particularly beneficial in sectors such as manufacturing, where product uniformity is essential.
- **Efficiency:** Robots are capable of executing operations at high speeds over prolonged periods without fatigue, thus minimizing time and effort required for task completion.
- **Increased Productivity:** Robotic systems can operate continuously—24 hours a day, 7 days a week—thereby increasing production rates, improving resource utilization, and boosting overall output.
- **Precision:** Robots perform highly accurate and repeatable movements, enhancing product quality and reducing material wastage due to fewer errors in production.
- **Multitasking:** Robots can execute multiple processes concurrently or switch between tasks with ease. This flexibility enhances workflow efficiency, particularly in dynamic manufacturing environments where adaptability is key.

I.6.2 Disadvantages of Robots

Despite the numerous advantages, the deployment of robotic systems is not without drawbacks. Several significant disadvantages have been identified [25]:

- **Economic and Social Impacts:** The widespread integration of robots into the workforce can lead to economic and social challenges, particularly regarding the displacement of human labor. This may result in job losses, reduced income for workers, and potential dissatisfaction or resentment among displaced employees.
- **Limited Adaptability and Capabilities:** Although technological advancements continue to enhance the functionality of robots, inherent limitations remain. For instance, robots may be incapable of performing tasks that require complex decision-making, creativity, or fine human dexterity. Furthermore, their capacity to adapt to unexpected or unstructured environments is generally less sophisticated than that of humans.

- **High Costs:** The initial investment in robotic systems—including acquisition, installation, maintenance, and repair—can be substantial. These high upfront costs, along with recurring expenses for upkeep and troubleshooting, may place significant financial strain on organizations, particularly small and medium-sized enterprises.
- **Complexity in Programming and Maintenance:** Robots, especially those operating in advanced domains such as artificial intelligence and machine learning, require specialized programming and maintenance. This necessitates highly skilled personnel, and a shortage of qualified professionals may hinder the effective deployment and integration of robotic systems.

I.7 Conclusion

This chapter provides a comprehensive overview of the field of robotics, beginning with a clear definition of robotics and robotic systems. It then traces the historical evolution of the discipline, highlighting key milestones and technological advancements that have shaped modern robotics. The chapter subsequently categorizes the various types of robots, which are broadly divided into two major groups: manipulative robots and mobile robots. An in-depth analysis of the different classifications is presented, offering a structured framework for understanding the wide range of robotic applications across diverse industrial sectors. The chapter concludes with a discussion of the advantages and disadvantages associated with the adoption of robotic technologies.

The primary objective is to explore the design and development of a robotic system that integrates a mechanically actuated manipulator arm with a vision-based camera system. The proposed robot is intended to perform tasks with high precision and autonomous decision-making capabilities, guided by real-time visual feedback. The focus will be on developing a robotic arm capable of executing complex, multi-axis movements, while simultaneously processing visual input for tasks such as object recognition, spatial awareness, and interaction with the environment. This integrated system aims to enhance the robot's operational accuracy and adaptability in dynamic environments through the use of vision-based feedback mechanisms.

Chapter II

Robot Sizing and Components:

II.1 Introduction:

The robot is a four-wheeled autonomous mobile vehicle equipped with a five-degree-of-freedom robotic arm, driven by six servo motors. It is equipped with an "eye-in-hand" type camera for image acquisition. Two microcontrollers are used: the first, a Raspberry Pi 4 Model B, handles image processing, while the second, an Arduino, is responsible for controlling both the robotic arm and the vehicle. Among the robot's applications, a use case is adopted: waste collection through classification.

The objective of this chapter is to introduce and explain the general design of the autonomous waste-collecting robot. It aims to help understand how the different parts of the robot—the mobile platform, the robotic arm, the vision system, as well as the electronic components and power sources—work together to accomplish the task of waste collection and classification. This chapter will serve as a foundation to grasp the technical choices and operating principles of this type of robot.

II.2 Actuation & Power:

II.2.1 Servo Motors (SG90, MG996R):

servo motor is an actuator that provides precise control of rotary or angular position through a dedicated feedback mechanism. It typically consists of an electric motor coupled with a servomechanism that regulates the motor's position, velocity, and acceleration. Servo motors are widely used in automation and robotics applications where accurate and repeatable positioning is required, such as in robotic arms, camera gimbals, and small vehicles.

Servo motors are mainly divided into two categories: positional rotation servos and continuous rotation servos. The differences between these types are summarized in Table 1

Feature	**Positional Rotation Servos**	**Continuous Rotation Servos**
Angular Range	Limited (typically 0-180°)	Unlimited (0-360° continuous)
Control Signal	Determines exact position	Determines direction and speed
Applications	Precise positioning like for robotic arms	Wheels, conveyor systems, continuous motion
Feedback	Position feedback	Speed feedback
Stopping	Holds position when stopped	Requires specific signal to stop

Table II.1 the difference between positional and continuous rotation servos

It is important to note that positional and continuous rotation servos often look physically identical. Their external casings, connectors, and overall dimensions may be indistinguishable by visual inspection alone. A reliable way to determine the type of servo is by testing its response to control signals: a positional servo will move to and hold specific angles, whereas a continuous rotation servo will interpret the same signals as commands to rotate at varying speeds and directions.

In this project, two types of hobby servos are used in the robotic arm. The TowerPro SG90 is a small (9 g) analog micro-servo with a 180° rotation range and plastic gears. According to the datasheet, the SG90 operates at about 4.8 V and provides approximately 1.8 kg·cm stall torque at 4.8 V, with a speed of around 0.10 s per 60° rotation [38]. These lightweight SG90 servos are suitable for the arm's smaller joints, where only modest torque is required.

The MG996R is a larger digital servo with metal gears, designed for high-torque joints such as the base or shoulder. TowerPro specifies that the MG996R (also with 180° rotation) weighs approximately 55 g and can deliver up to 9.4 kg·cm stall torque at 4.8 V (and up to 11 kg·cm at 6.0 V), with a speed of approximately 0.19 s per 60° at 4.8 V [39]. The MG996R's metal gears and wider operating voltage range (4.8–6.6 V) provide greater strength and durability, making it well suited for the heavier parts of the robotic arm.

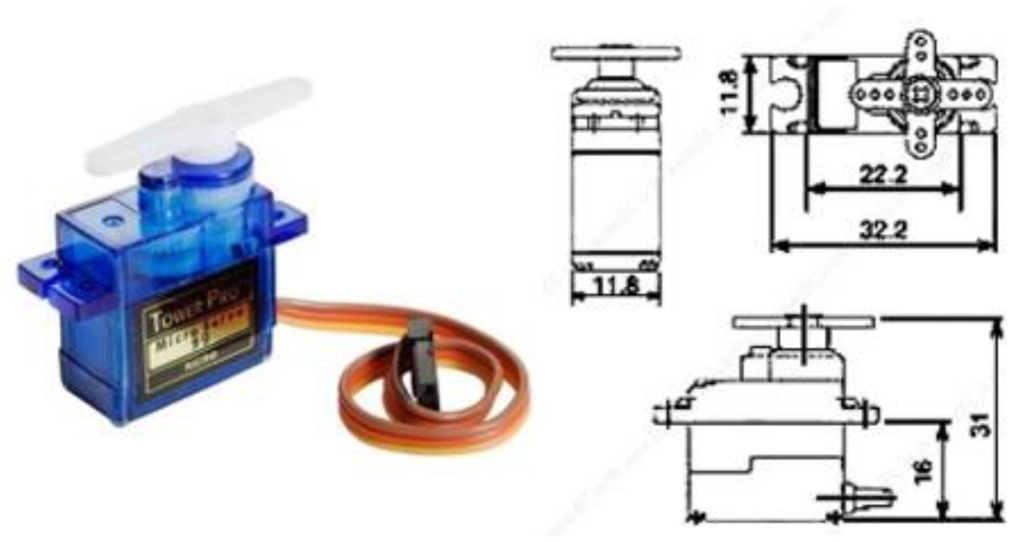


Figure II.1 SG90 9 g Micro Servo [38]

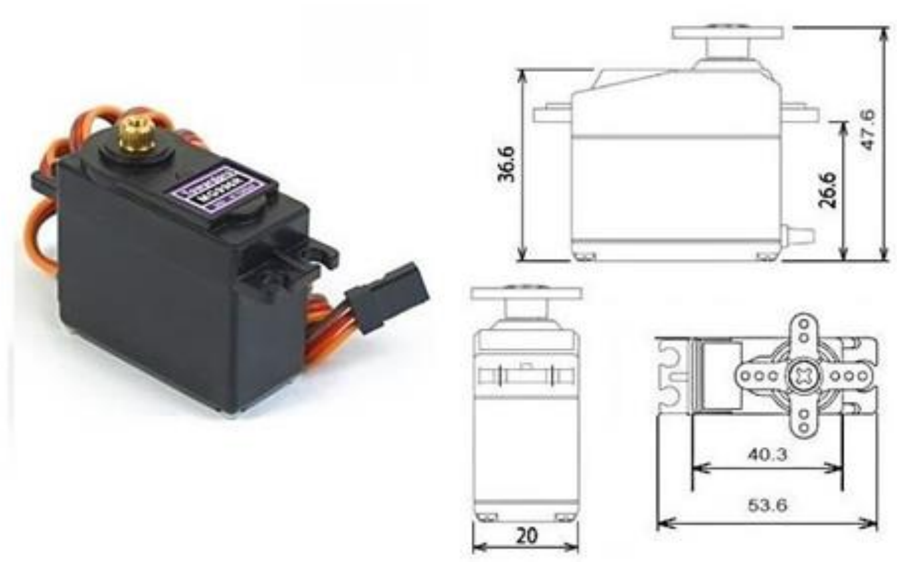


Figure II.2 MG996R servo motor [39]

- **Servo Control Mechanism:** Hobby servos operate using an internal closed-loop control system. A potentiometer attached to the output shaft continuously measures the shaft's position. The built-in controller compares this measured position to the target position defined by the incoming PWM (Pulse Width Modulation) signal. It then drives the motor accordingly until the actual position matches the desired one. This internal feedback loop enables precise positioning and stable holding force without requiring external sensors or complex programming [50].

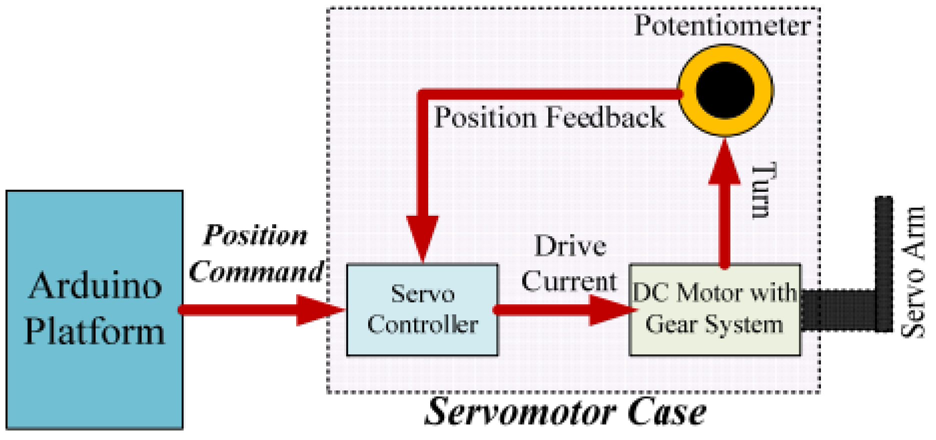


Figure II.3 Block Diagram of Hobby Servo Motor Internal Control System [40]

II.2.2 TT Gear Motors

The robot's chassis is powered by four small DC gear motors commonly referred to as "TT" motors. These are simple two-pole DC motors operating at 3–6 V and equipped with a plastic gearbox, typically with a 1:48 gear ratio. TT motors are inexpensive and compact, making them a popular choice for driving the wheels of small-scale robots. They offer a good balance of speed and torque for wheel propulsion at a low cost [40].

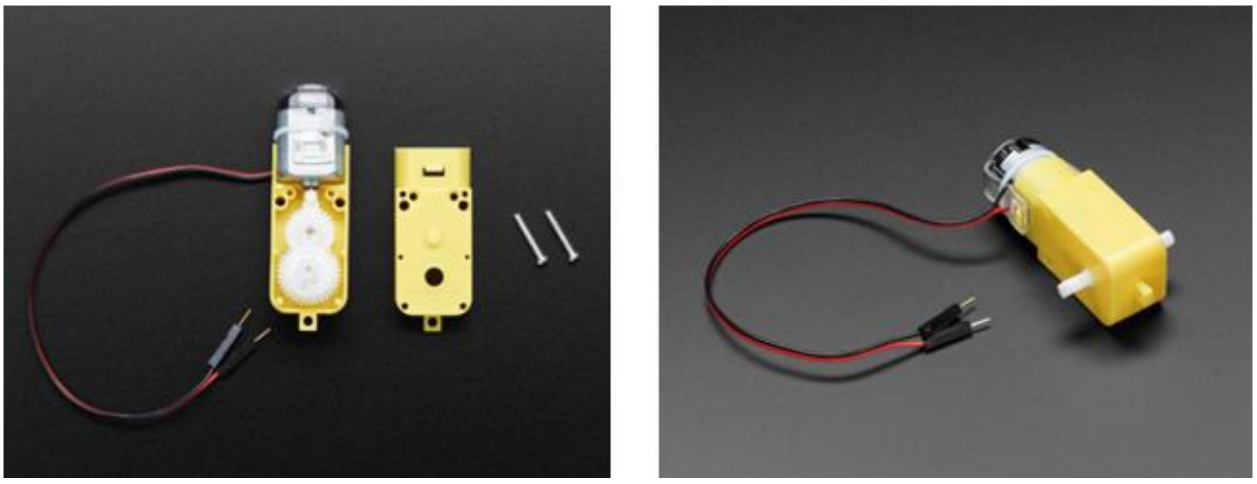


Figure II.4 TT motors [4]

II.2.3 Battery System (Rechargeable Pack and Individual Cells)

After evaluating the energy requirements of each component, two distinct power sources were implemented to ensure stable and efficient operation:

- **Rechargeable Battery Pack:** The Raspberry Pi's power system is based on a ZOP Power 7.4 V 3000 mAh 35C LiPo battery pack (2S configuration). This lightweight pack delivers a nominal voltage of 7.4 V and features a high 35C discharge rate, which ensures stable current supply even during peak loads, such as when the Raspberry Pi 4 is performing image processing tasks.
  To meet the Raspberry Pi's voltage input requirements, a buck converter is used to step down the 7.4 V to 5.1 V instead of exactly 5.0 V. This slight increase provides a safety margin to compensate for potential voltage drops under high-load conditions.
- **Individual Battery Cells:** The robot's drivetrain and auxiliary components are powered by three QOOP LIR-18650 rechargeable lithium-ion cells (3.7 V, 2800 mAh each), arranged in series within a battery holder to provide a total of 11.1 V. This voltage feeds directly into the L293D motor driver shield mounted on the Arduino Mega. The shield's Vin pin is connected to the Arduino's Vin, enabling the onboard regulator to step down the voltage to appropriate levels required by the microcontroller and connected peripherals, including the HC-05 Bluetooth module, the ultrasonic sensor (HC-SR04), and all TT and servo motors.
  To power the PCA9685 PWM driver board safely and reliably, a dedicated buck converter is used to reduce the 11.1 V from the battery pack to a regulated 6 V. This ensures that the PWM driver operates within its recommended voltage range, protecting it from overvoltage and guaranteeing consistent power delivery.

Figure II.5 Li-poly 3000mah battery

Figure II.6 qoop 2800mah battery cells

II.2.4 Charging System

- **Main Pack Charger:** The primary battery pack (2S LiPo) is charged using a balance charger. For this purpose, the SkyRC iMAX B6 Mini, a professional-grade 60 W charger, is employed. The B6 Mini supports charging 1–6 cell LiPo/Li-ion batteries with a maximum current of 6 A (up to 60 W output), and includes integrated cell balancing. It accepts an input voltage range of 11–18 V DC and charges the 7.4 V (2S) pack using a constant-current/constant-voltage (CC/CV) charging profile for precision and safety. In addition to charging and discharging, the charger supports multiple battery chemistries, making it ideal for maintaining the health and performance of the main battery pack [41].
- **Individual Battery Cell Charger:** The HD-232650 is a dual-slot lithium-ion battery charger designed for 3.7 V rechargeable cells such as the 18650. It delivers a constant charging voltage of 4.2 V with a maximum output current of 2000 mA. Safety features

include reverse polarity protection and overcharge protection, which help to extend battery life and ensuvre safe operation

Figure II.7 imax b6 mini charger

Figure II.8 Figure II.8 hwd battery cells charger

II.2.5 Buck Converter

A DC-DC buck (step-down) converter is a switch-mode regulator that produces a lower output voltage from a higher input voltage. This allows the 7.4 V battery pack that powers the raspberry pi to be stepped down to 5.1v and the 11.1v battery cells that power shield driver l293d with its components to be also efficiently stepped down to a stable 5v, ensuring the correct voltage for those components to function properly.**[42]**

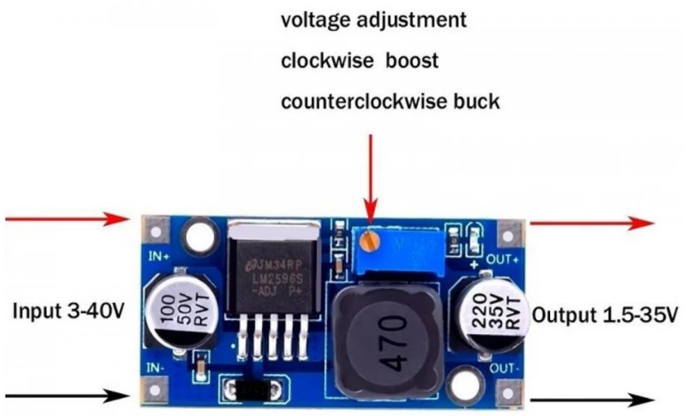


Figure II.9 lm2596 DC-DC Buck Converter[70]

II.3 Control & Driver Boards

II.3.1 Raspberry Pi 4 Model B (8 GB RAM)

The Raspberry Pi 4 Model B is a single-board computer powered by a 64-bit quad-core ARM Cortex-A72 processor (Broadcom BCM2711) running at 1.5 GHz. It features 8 GB of LPDDR4 RAM in this configuration. The board supports dual-band 802.11ac Wi-Fi, Bluetooth 5.0, and Gigabit Ethernet connectivity. It operates on a Linux-based operating system. Key hardware specifications include 2 USB 3.0 ports, 2 USB 2.0 ports, dual micro-HDMI outputs capable of 4Kp60 resolution, and a CSI camera interface port. The board is powered via a 5 V USB-C input, requiring approximately 3 A of current [43].

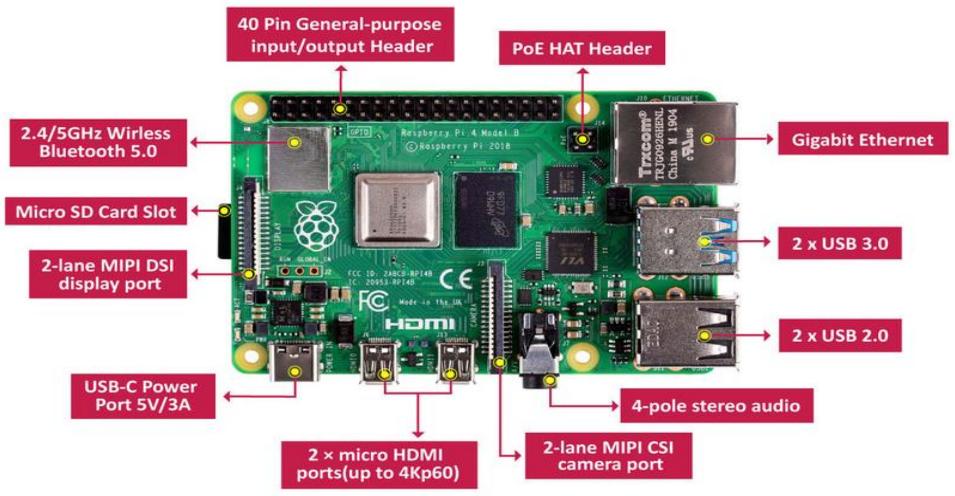

40 Pin General-purpose input/output Header
PoE HAT Header
2.4/5GHz Wirless Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2-lane MIPI DSI display port
2 x USB 3.0
2 x USB 2.0
USB-C Power Port 5V/3A
4-pole stereo audio
2 × micro HDMI ports(up to 4Kp60)
2-lane MIPI CSI camera port

Figure II.10 Raspberry pi 4 model b[71]

- **Cooling Fan and Heatsink Kit for the Raspberry Pi:** We employed a cooling kit comprising a single fan and a set of black anodized aluminum heatsinks specifically designed for the Raspberry Pi 4 Model B. The fan is equipped with thermal conductive adhesive to ensure efficient heat transfer between the components and the heatsinks. This cooling solution effectively manages the thermal load generated during intensive tasks, such as image processing and object detection, where both the CPU and RAM experience high utilization and consequently produce significant heat.

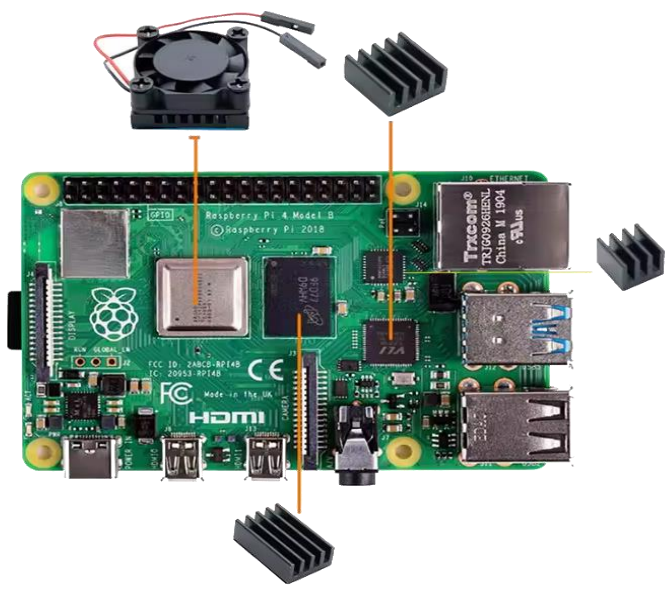


Figure II.11 Raspberry Pi 4B Cooling Fan and Heatsink Kit

II.3.2 Arduino Mega 2560

The Arduino Mega 2560 is an 8-bit microcontroller board based on the ATmega2560 MCU. It operates at 16 MHz and has 256 KB Flash and 8 KB SRAM. The board provides 54 digital I/O pins (15 of which support PWM), 16 analog inputs, and four hardware serial ports. In this project, the Mega 2560 handles real-time tasks like motor driver control signals, servo commands, and sensor interfacing. It communicates with the Pi (via UART) to receive high-level commands and send status. The wide I/O array allows direct connection to motor drivers, encoders, and other peripherals [44].

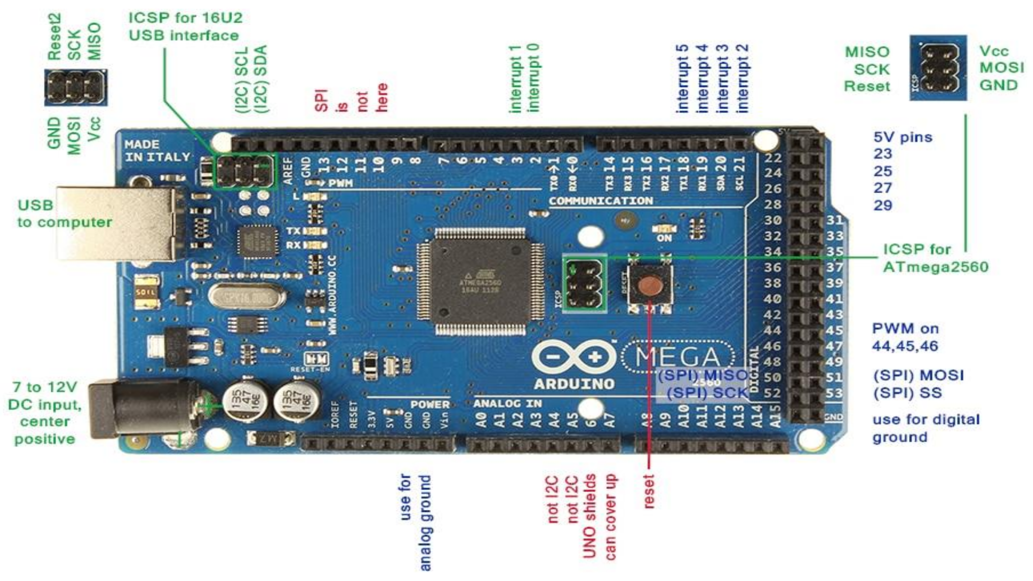


Figure II.12 Arduino Mega[72]

II.3.3 L293D Motor Driver Module

The L293D is a quadruple half-H-bridge driver integrated circuit (IC) designed for controlling DC motors. It provides up to four outputs, equivalent to two full H-bridges, with a continuous current rating of 600 mA per channel and a peak current capacity of 1.2 A. The L293D receives logic input signals from the Arduino and utilizes an external motor power supply, supporting voltages up to 36 V, to drive the motors.

In this project, a single L293D IC controls all four TT wheel motors, with two motors managed per chip and drivers enabled in pairs. The IC integrates built-in diodes to protect against inductive kickback from the motors. This driver facilitates bidirectional motor control and is capable of handling the stall current of the TT motors, which is typically around 1.2 A [45].

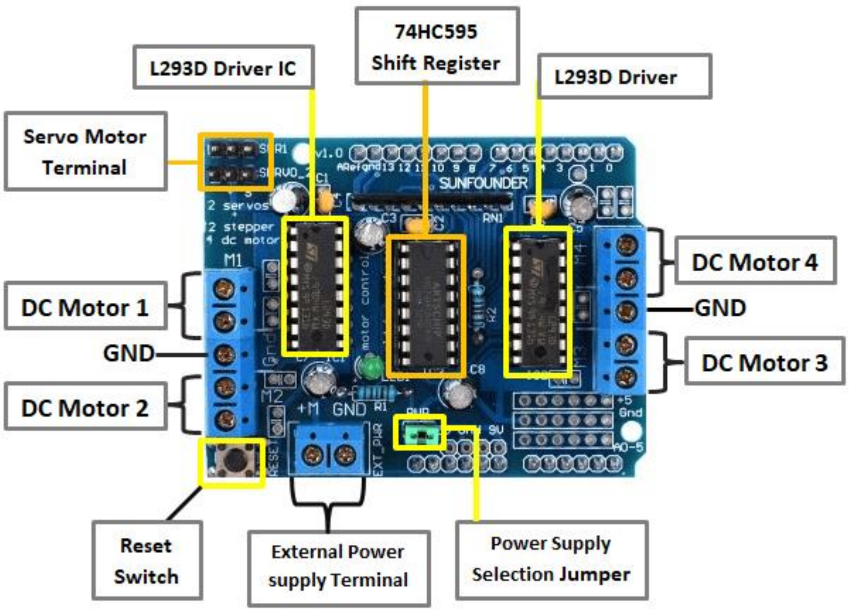


Figure II.13 L293d motor shield driver[73]

II.3.4 PCA9685 PWM Driver Board

The PCA9685 is a 16-channel, 12-bit pulse-width modulation (PWM) controller designed to address common limitations in microcontroller-based systems requiring multiple PWM outputs. This integrated circuit serves as a dedicated PWM signal generator, enabling precise and independent control of up to 16 channels.

In complex systems involving coordinated motion or lighting control, the PCA9685 offloads the timing-critical task of PWM signal generation from the main processor, allowing it to focus on other computational tasks. Communication with the PCA9685 is established via the I2C protocol, which minimizes the number of pins required regardless of the number of controlled outputs.

The PCA9685 supports programmable PWM frequencies up to approximately 1.5 kHz and allows customization of duty cycles to match various servo motor specifications. In this robotic system, the PCA9685 board is employed to control six servo motors of the robotic arm, providing consistent and precise PWM signals while relieving the Arduino from timing-sensitive PWM generation. The availability of 16 channels and fixed frequency ensures stable servo control without overloading the main controller [46].

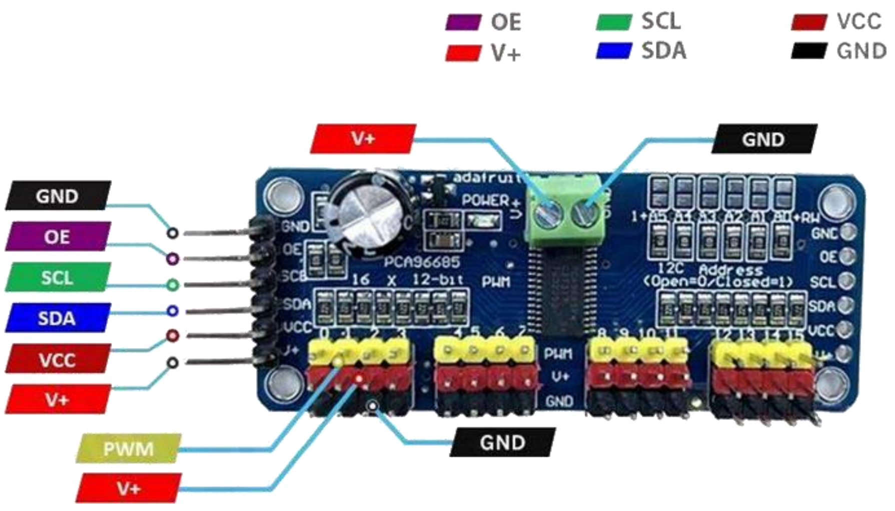


Figure II.14 Pca9685[74]

II.4 Sensing & Communication

II.4.1 HC-SR04 Ultrasonic Sensor

The HC-SR04 is an ultrasonic distance sensor comprising an ultrasonic transmitter, receiver, and an integrated control circuit. It measures distances ranging approximately from 2 cm to 400 cm (4 m) with an accuracy of about ±3 mm. The sensor operates by the microcontroller sending a trigger pulse to initiate an ultrasonic burst; the sensor then emits a pulse whose duration corresponds to the time taken for the echo to return after reflecting from an object. In this robotic application, the HC-SR04 is employed to detect obstacles and verify proximity to waste bins and other objects. It interfaces with the microcontroller via digital trigger and echo pins and is powered by a 5 V supply **[47].**

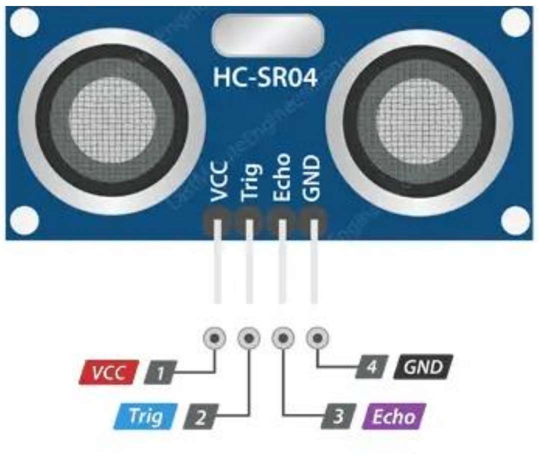


Figure II.15 HC-SR04 Ultrasnoic sensor[75]

II.4.2 Raspberry Pi Camera Module v2

The Raspberry Pi Camera Module v2 is a compact image sensor board featuring an 8-megapixel Sony IMX219 sensor, specifically designed for compatibility with Raspberry Pi devices. It connects via the CSI camera interface port. The module can capture still images with a maximum resolution of 3280 × 2464 pixels and record video at 1080p resolution with 30 frames per second, or higher frame rates at reduced resolutions. Equipped with a fixed-focus lens and measuring approximately 25 × 23 × 9 mm with a weight of around 3 grams, it is well-suited for mobile and embedded applications. In this project, the camera module provides the visual input necessary for waste detection [48].

Figure II.16 Raspberry Pi Camera Module v2

II.4.3 HC-05 Bluetooth Module

The HC-05 is a serial Bluetooth transceiver module compliant with Bluetooth v2.0 + EDR standards, designed for short-range wireless communication. Operating at the 2.4 GHz frequency band, it typically provides a wireless range of approximately 10 meters (Class 2, +4 dBm output power). The module interfaces with microcontrollers via UART (serial communication) and supports power supply voltages ranging from 3.3 to 5 V.

In this project, the HC-05 module is utilized for remote debugging, configuration, and manual control override through a paired smartphone or PC. Upon pairing, it establishes a wireless serial communication link, enabling command and telemetry exchange. Its low cost and straightforward UART integration make it a popular choice for adding Bluetooth connectivity [49].

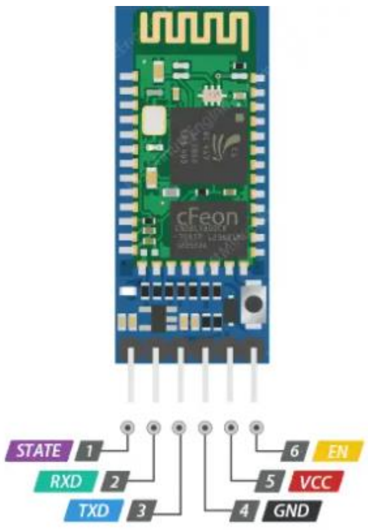


Figure II.17 HC-05 Bluetooth Module[49]

II.5 Mechanical design:

II.5.1 The chassis:

We examined every component to create the ideal shape for our robot, ensuring it could perform all required functions and to achieve balance and proper placement of each part.

- **The Base:** The base is one of the most crucial elements in the design. We built it to support several important components:
  - DC TT motors
  - Electronic parts (Raspberry Pi 4, Arduino Mega 2560, L293D motor driver shield, ultrasonic sensor, the battery pack and the battery cells holder…)
  - The robotic arm

We designed the base to carry all these components while maintaining proper balance and organizing each part in a well-arranged manner.

Due to size limitations of the 3D printer used, we decided to split the base into two connecting pieces that fit together to form a single unified structure. This approach was necessary because printing the entire base as one piece was not feasible.

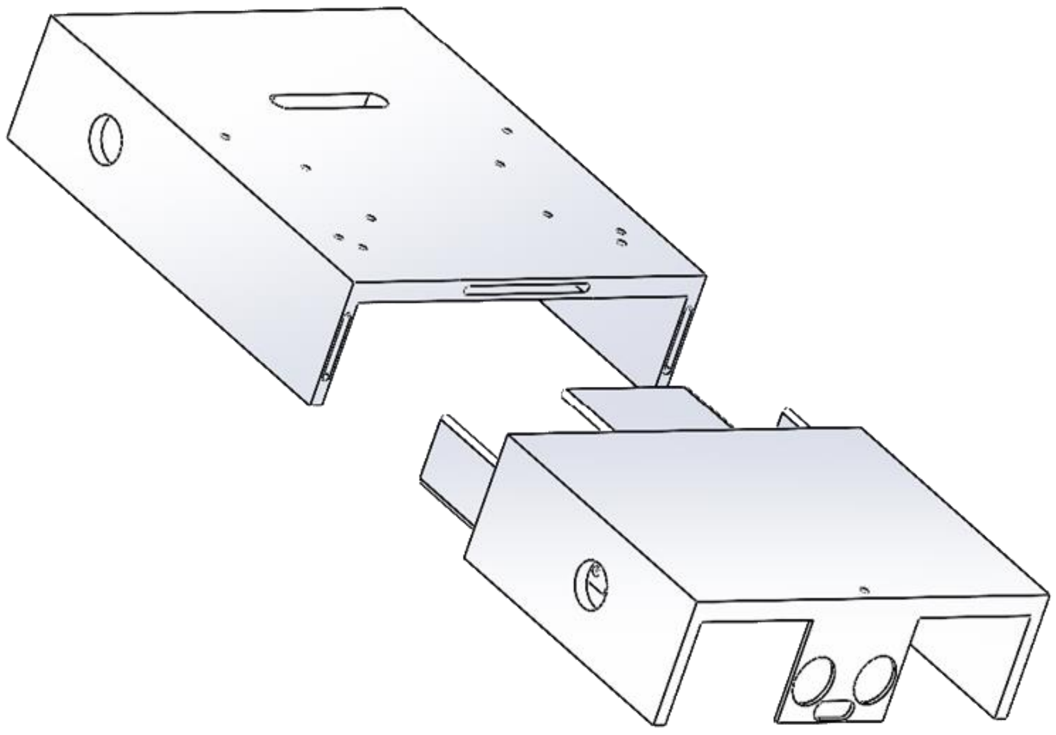


Figure II.18 Top View of the Chassis

We designed mounting areas for the DC TT motors. These motor brackets firmly attach the motors to the robot and reduce vibrations, ensuring stable and efficient movement. Additionally, we included mounting holes for the battery holder and a bracket for the ultrasonic sensor.

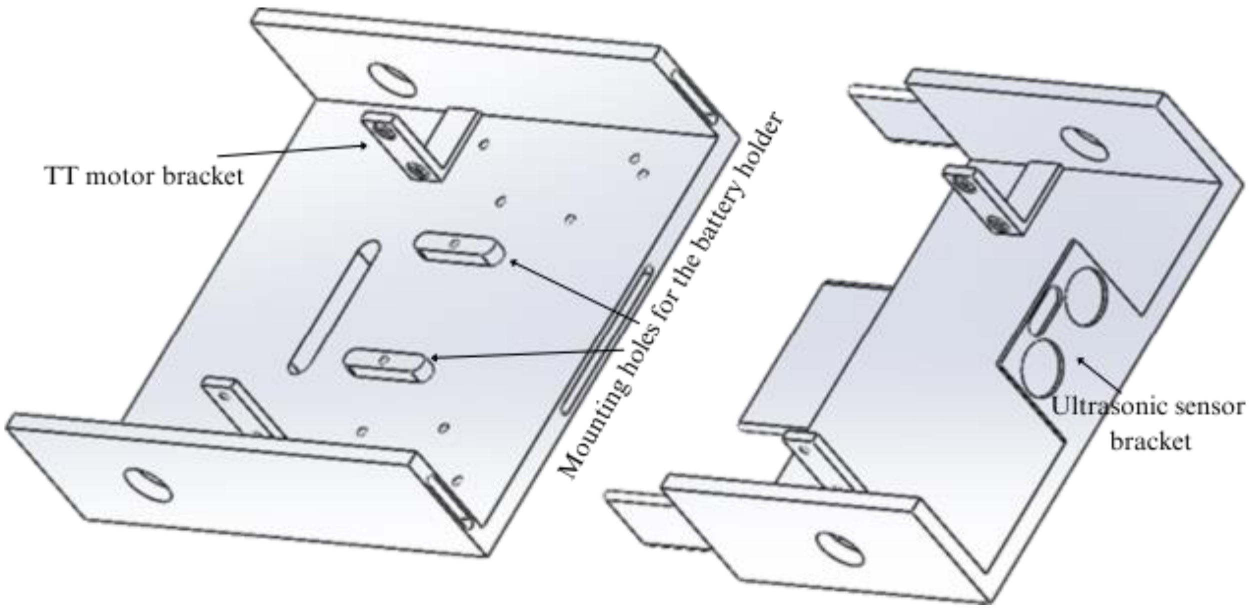

TT motor bracket
Mounting holes for the battery holder
Ultrasonic sensor bracket

Figure II.19 bottom view of the chassis

- **The Motor Coupler :** We designed a special coupler to create a gap between the wheels and the base. This spacing reduces friction, allowing the wheels to rotate freely without resistance. This design helps prevent excessive friction that could increase the electrical current drawn by the DC TT motors, potentially causing them to overheat and fail. Our coupler is precisely engineered to connect securely to the wheels on one side and to the motor's rotation axis on the other, ensuring a smooth and reliable drive system for the robot.

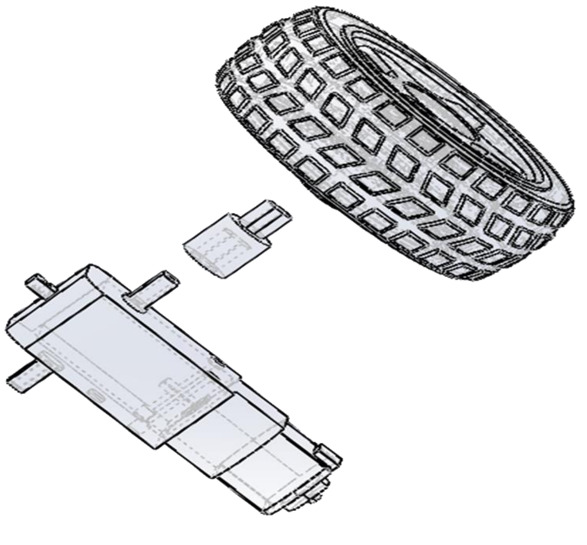


Figure II.20 The Motor's Coupler in between a tt motor and a wheel

We then designed

- **The holes for vertical standoffs** to support the case
- **The holes for mounting the Arduino Mega** securely onto the chassis.

- **Cover Support standoffs:** To support the robot's enclosure, four vertical standoffs (pillars) are mounted onto the main chassis. The chassis includes pre-drilled holes at precise locations to accommodate these standoffs, providing a stable support for the top cover or protective casing.,
- **Arduino Mega Mounting Holes:** The chassis also features a mounting layout aligned with the standard hole pattern of the Arduino Mega board. These mounting holes enable secure attachment of the microcontroller using screws and spacers.

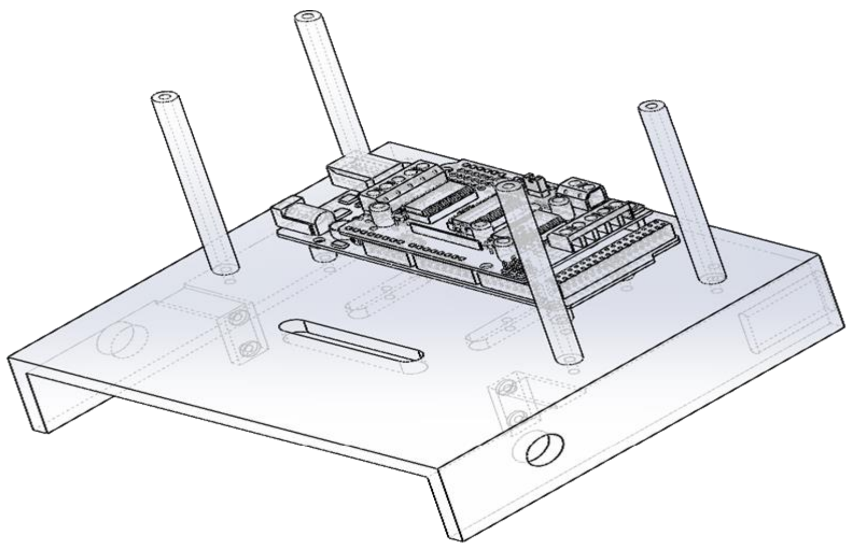


Figure *II.21 Chassis Mounting Holes and Pillars for Arduino and Cover*

The L293D motor driver shield is mounted directly onto the Arduino Mega via its pin headers. The shield is precisely designed to fit the Arduino's header layout. To complete the control system, a Raspberry Pi is positioned above both the L293D shield and the Arduino Mega.

This arrangement is supported by L-shaped and vertical brackets that align with the Arduino Mega and the motor driver shield. Mounting the Raspberry Pi in an elevated position also ensures adequate airflow for its cooling fan. The vertical clearance allows the fan to operate without obstruction, which is crucial for maintaining optimal thermal performance during image processing tasks.

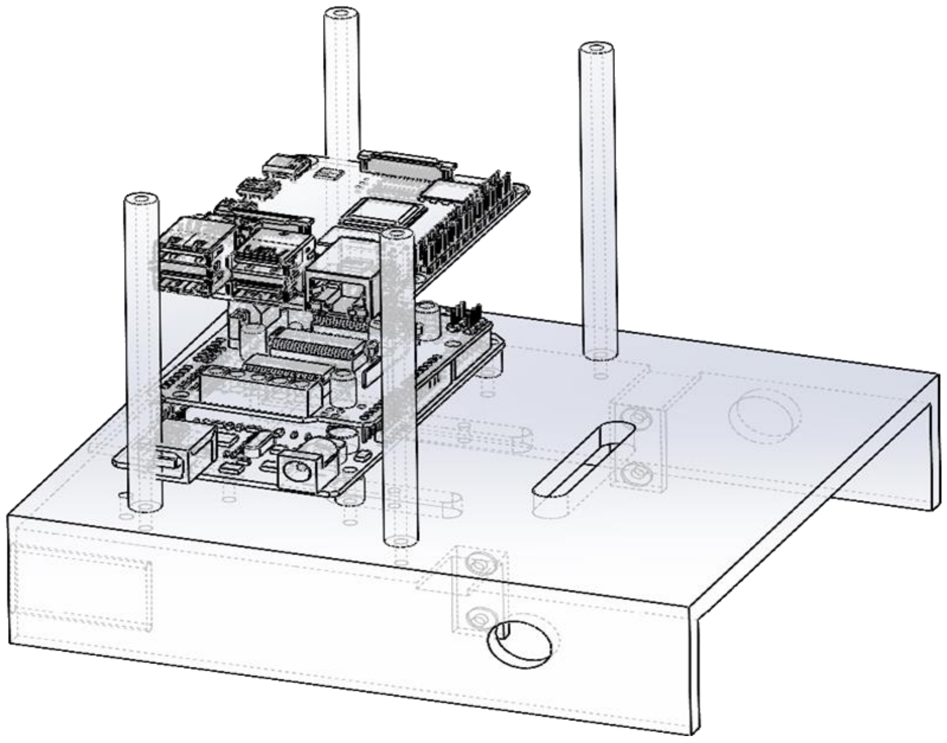


***Figure** II.22 Full Electronics Stack Overview*

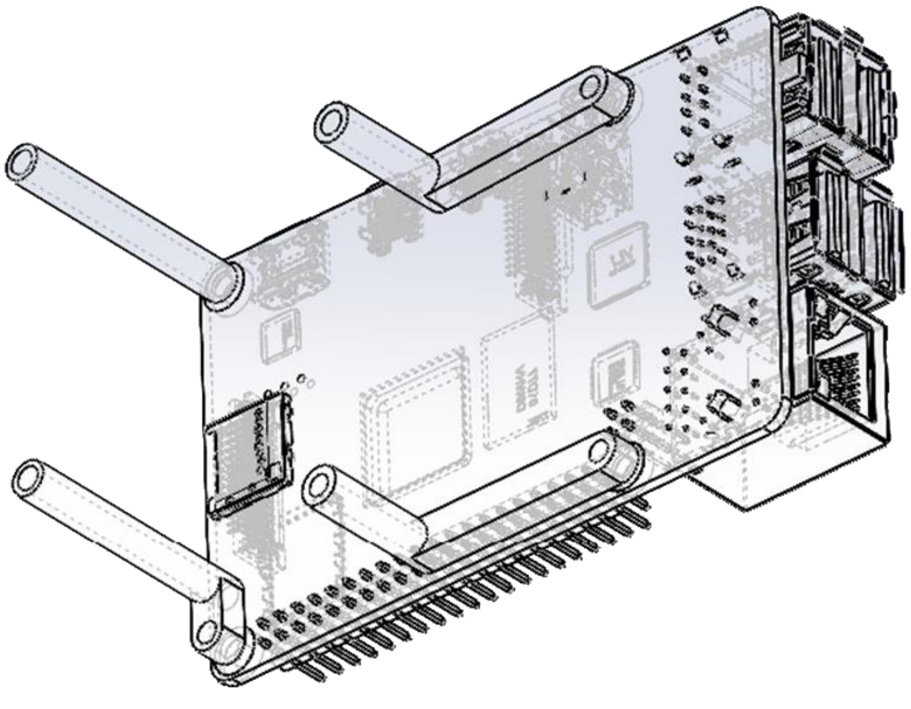


***Figure** II.23 Raspberry Pi Supports*

- **Case Design and Battery Accessibility:** To protect the internal components while ensuring easy access for recharging the battery, we designed a protective case to enclose the chassis and electronics. The case shields sensitive components—including the Arduino Mega, L293D motor driver, Raspberry Pi, and wiring—from physical damage and environmental factors. Since the Raspberry Pi is powered by a LiPo battery, convenient access for recharging is essential. To accommodate this, the case features an openable side panel located at the battery compartment. This design allows quick access to the LiPo battery for charging without requiring disassembly of the entire case.

- **Design Considerations:**
  - ➢ The side panel is secured with removable screws to allow easy opening and closing.
  - ➢ Ventilation slots are included to prevent heat buildup.
  - ➢ Two slots are provided for switches to power on and off the Raspberry Pi and the Arduino Mega.

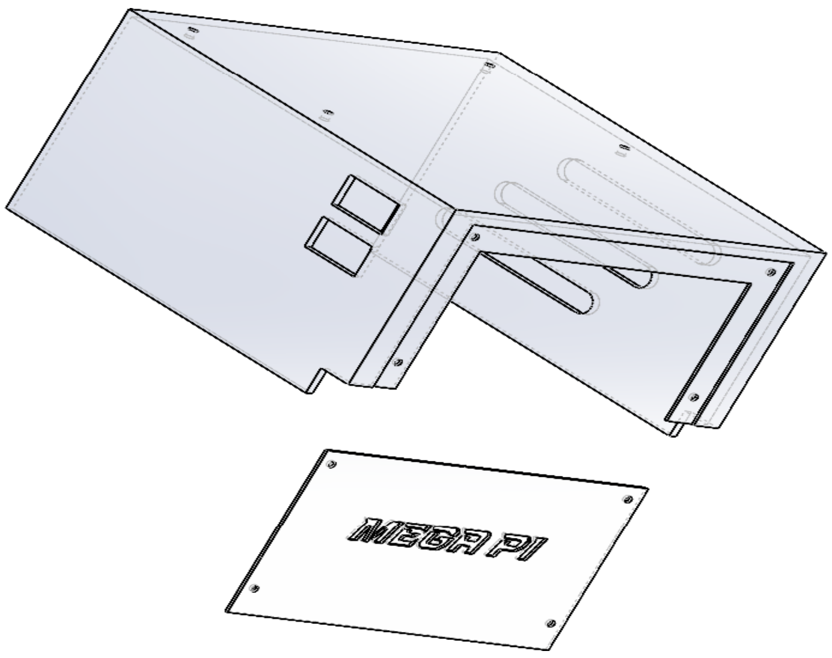


Figure II.24 Openable Side Panel case

II.5.2 The robotic arm

The implemented robotic manipulator is a 5 Degrees of Freedom (DOF) arm based on an open-source design **[51]**. The arm uses 3D-printed components for its structural elements and servo motors for actuation. Control of the six servos is managed by an Arduino Mega via a PCA9685 PWM driver board. Each joint performs a specific movement and function.

The joints are actuated by six servo motors: three SG90 servos and three MG996R servos. The MG996R servos are installed in the first three joints (base, shoulder, and elbow) due to their higher torque and capability to carry the load of these links while performing required movements. The remaining joints (wrist and gripper) are controlled by SG90 servos, which are better suited for lighter loads and provide precise control for fine manipulation tasks such as grasping light objects.

Joint 1: Base The base joint provides rotational movement about the vertical axis. It uses an MG996R high-torque servo mounted underneath the base platform.

The rotating upper part of the base is supported by a large ball bearing, which reduces friction and enhances stability during rotation.

The servo horn is fixed directly to the rotating base structure, enabling effective torque transfer. This joint serves as the foundation for the entire arm, supporting the combined weight of the upper joints and allowing the entire arm to rotate left or right within the horizontal plane.

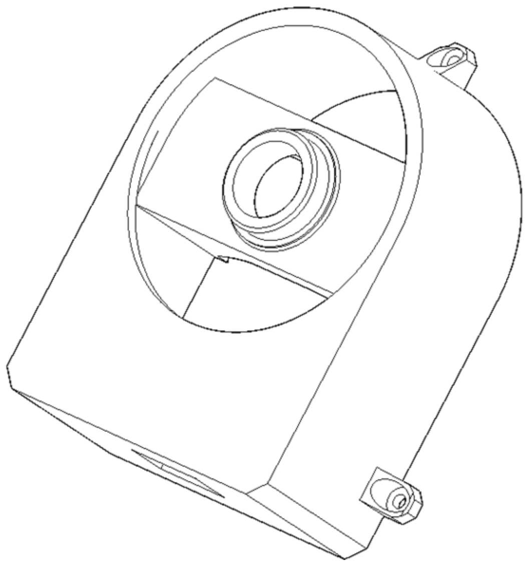


Figure II.25 top view of the base design [51]

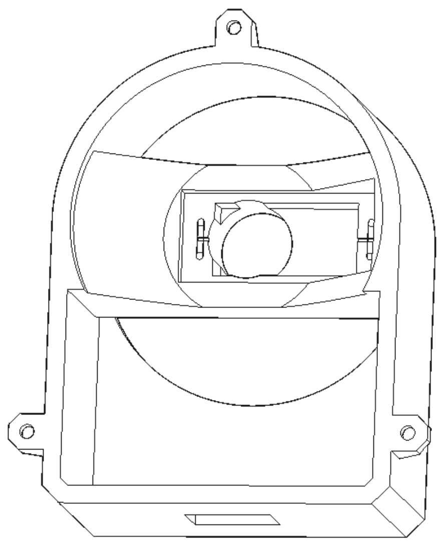


Figure II.26 bottom view of the base design [51]

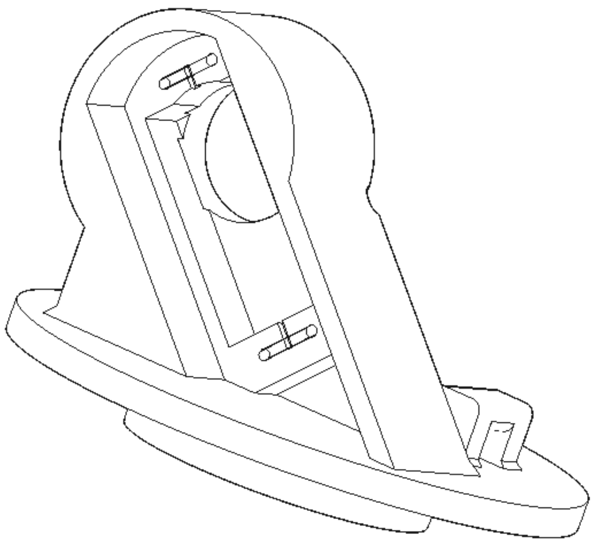


Figure II.27 upper base design [51]

Joint 2: Shoulder

The shoulder joint provides vertical movement for the first arm link. It uses an MG996R servo motor, mounted directly on top of the base's rotating section (the upper base), which transmits torque to raise or lower the link. Since this joint supports the load of all subsequent joints, the high torque of the MG996R servo is essential. This joint enables changes in elevation and is crucial for positioning the arm at different heights.

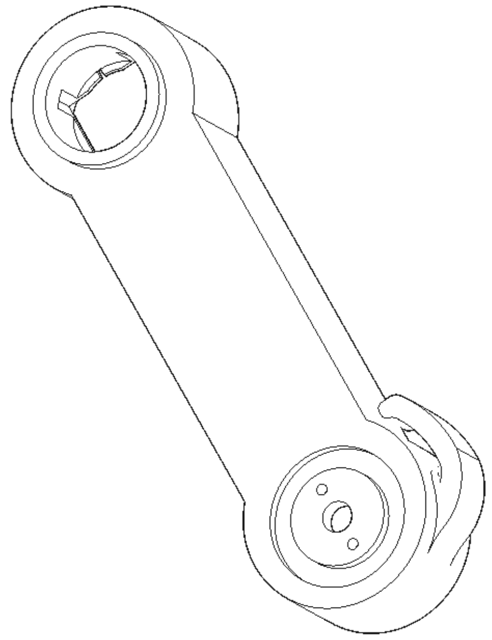


Figure II.28 front view of the shoulder joint [51]

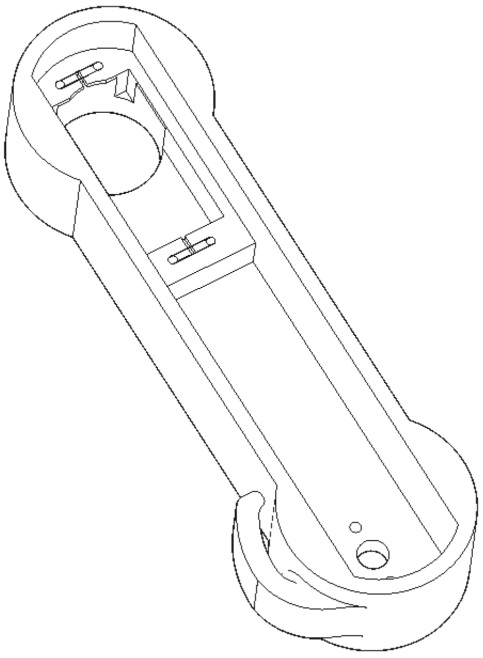


Figure II.29 back view of the shoulder joint [51]

Joint 3: Elbow

The elbow joint connects the first and second main arm links. It uses an MG996R servo motor mounted at the end of the shoulder joint, which drives the lower arm link via a horn linkage. Movement occurs in the same vertical plane as the shoulder joint.

This joint supports part of the payload weight and enables extension or retraction of the arm, making it critical for adjusting the arm’s reach.

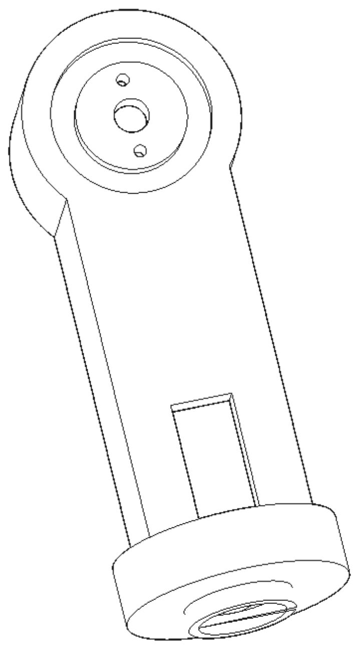


Figure II.30 front view of the elbow joint design [51]

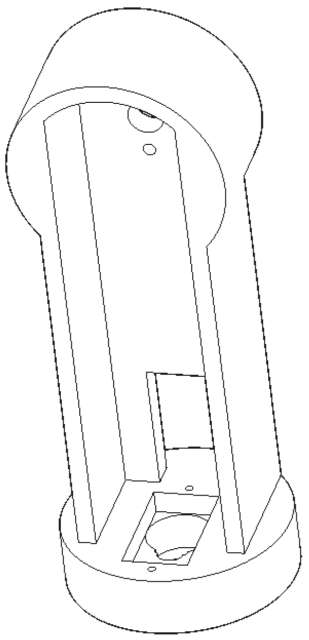


Figure II.31 back view of the elbow joint design [51]

Joint 4: Wrist Up-Down

This joint controls the up-and-down tilting of the wrist. An SG90 micro-servo is mounted at the end of the elbow joint, with its horn connected to the wrist structure. The rotation axis is perpendicular to that of the elbow, allowing precise pitch control. Because the distal end is lighter, the SG90 provides sufficient torque for this movement. This joint enhances dexterity and enables proper vertical orientation of the gripper. Its motion range covers a full sweep to support various gripping angles.

Joint 5: Wrist Rotation

These joint enables rotation of the end effector around the forearm's axis. An SG90 servo is mounted just behind the gripper, with its output shaft directly connected to the rotating part of the wrist. This joint allows the gripper to roll and change its orientation, which is useful for aligning with objects. The design permits approximately 180° of rotation. It is mechanically simple and relies on the precise control of the SG90 servo for fine rotational adjustments.

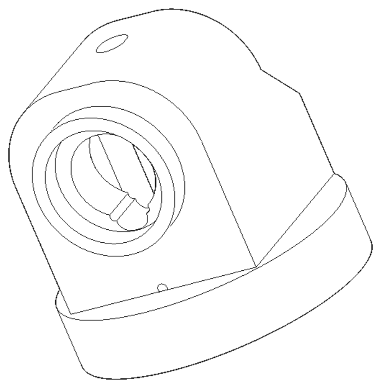


Figure II.32 front view of the wrist design [51]

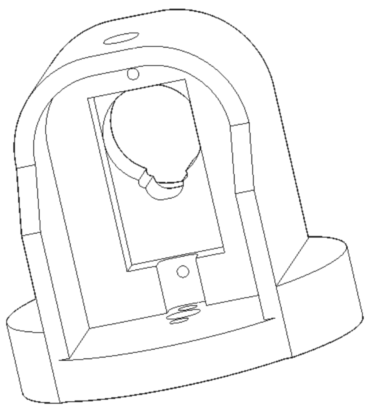


Figure II.33 back view of the wrist design [51]

Joint 6: Gripper:The gripper is the final joint and the arm's end effector. It is actuated by an SG90 servo mounted inside the gripper base. The servo drives two small interlocking gears, each attached to a finger of the gripper. When the servo rotates, the gears open or close the fingers symmetrically. This mechanism allows grasping light objects with adequate force. The mall size and precision of the SG90 servo make it ideal for controlling this mechanism. crews. and spacers ensure smooth movement and proper alignment of the fingers

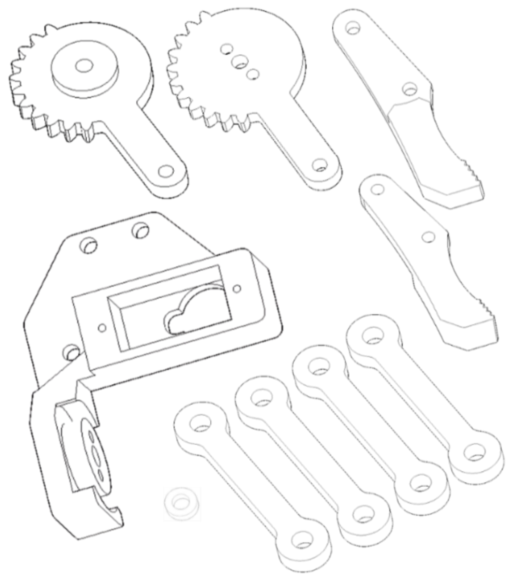


Figure II.34 The gripper design [51]

Camera Holder: The camera mount is a key component, and its placement on the robot requires careful consideration. It must be strategically positioned to provide a clear and wide field of view, which is essential for accurate object recognition and identification.

This component is designed to be compatible with the Raspberry Pi Camera Module v2, allowing it to be securely mounted using screws. It is intended for installation at the optimal position—specifically, at the end of the robotic arm just before the gripper. This configuration is commonly referred to as an "eye-in-hand" setup, where the camera moves along with the manipulator, enabling an unobstructed and close view of the surrounding area for precise visual feedback and manipulation.

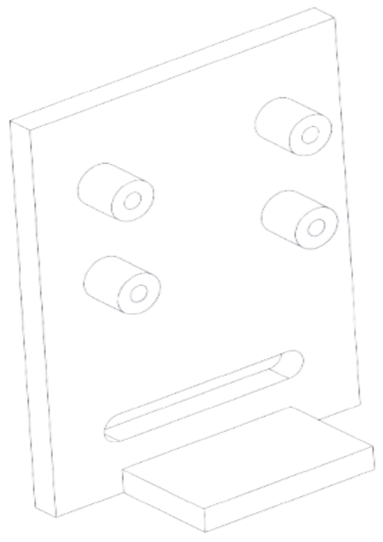


Figure II.35 camera holder

II.6 Conclusion

This chapter provided a detailed overview of the design and key components of an autonomous mobile robot equipped with a four-wheeled platform and a five-degree-of-freedom robotic arm. The selection of servo motors balanced the requirements for torque and precision across the arm's joints, while the dual power system ensured reliable operation of both computational and mechanical elements. The control architecture combined a Raspberry Pi for image processing and an Arduino for real-time actuation, supported by dedicated motor and PWM driver boards. Essential sensors enabled environmental perception and visual feedback, with the "eye-in-hand" camera configuration enhancing manipulation accuracy. The mechanical design emphasized stability, balance, and modularity, utilizing 3D printing for manufacturability. Overall, this chapter establishes a solid technical foundation for further development of control strategies and experimental validation.

Chapter III

Software Tools & Code Development

III.1 Introduction

This chapter offers an overview of the software ecosystem developed for the project. It covers the development environments, key tools, Arduino Mega firmware for low-level control, Python software on the Raspberry Pi 4 for high-level tasks like computer vision and decision-making, and the user interface on the app for remote operation and monitoring.

III.2 Computer-Aided Design (CAD) and 3D Printing

III.2.1 Solidworks

SolidWorks is a 3D computer-aided design (CAD) software developed by Dassault Systèmes. Engineers can quickly sketch concepts, experiment with feature dimensions, and create detailed 3D models, assemblies, and 2D drawings
As a parametric, feature-based program that offers modeling, simulation, and documentation tools, SolidWorks is a common platform for product development in the design and engineering domains [52].

We used SolidWorks for all mechanical design aspects detailed in the previous chapter, this powerful CAD software was instrumental in designing the robot's chassis, and various custom brackets and mounts for sensors and electronic components. SolidWorks allowed us to create precise 3D models, ensuring accurate dimensions, proper component integration, and evaluating the overall mechanical integrity before any physical assembly. This digital prototyping phase was crucial for identifying potential design flaws early.

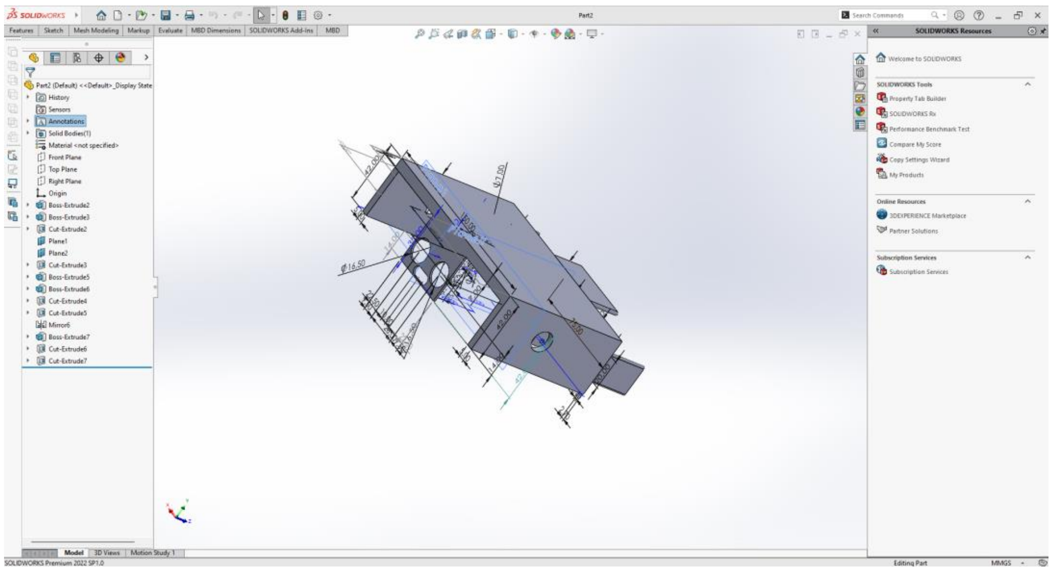


Figure III.1 Modeling of the chassis in SolidWorks

III.2.2 Creality Slicer

Creality Slicer is a desktop slicing prgoram for Fused Filament Fabrication (FFF) 3D printing, provided by Creality 3D Technology. It converts 3D CAD models into G-code instructions that can be sent to a printer. The software includes printer profiles optimized for Creality machines (e.g. Ender, CR series) and offers a user-friendly interface with common slicing controls (layer height, infill, supports) [53].

All the parts we designed in SolidWorks were fabricated using a 3D printer. we utilized this tool to prepare our 3d models for printing.

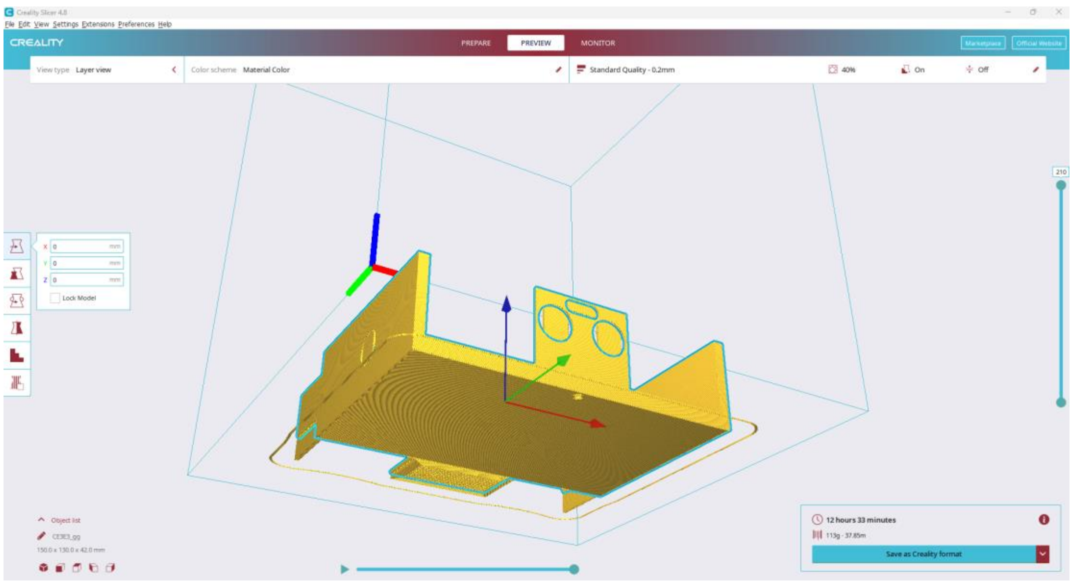


Figure III.2 Chassis model preview in Creality Slicer

III.3 Overall Robot Software Architecture

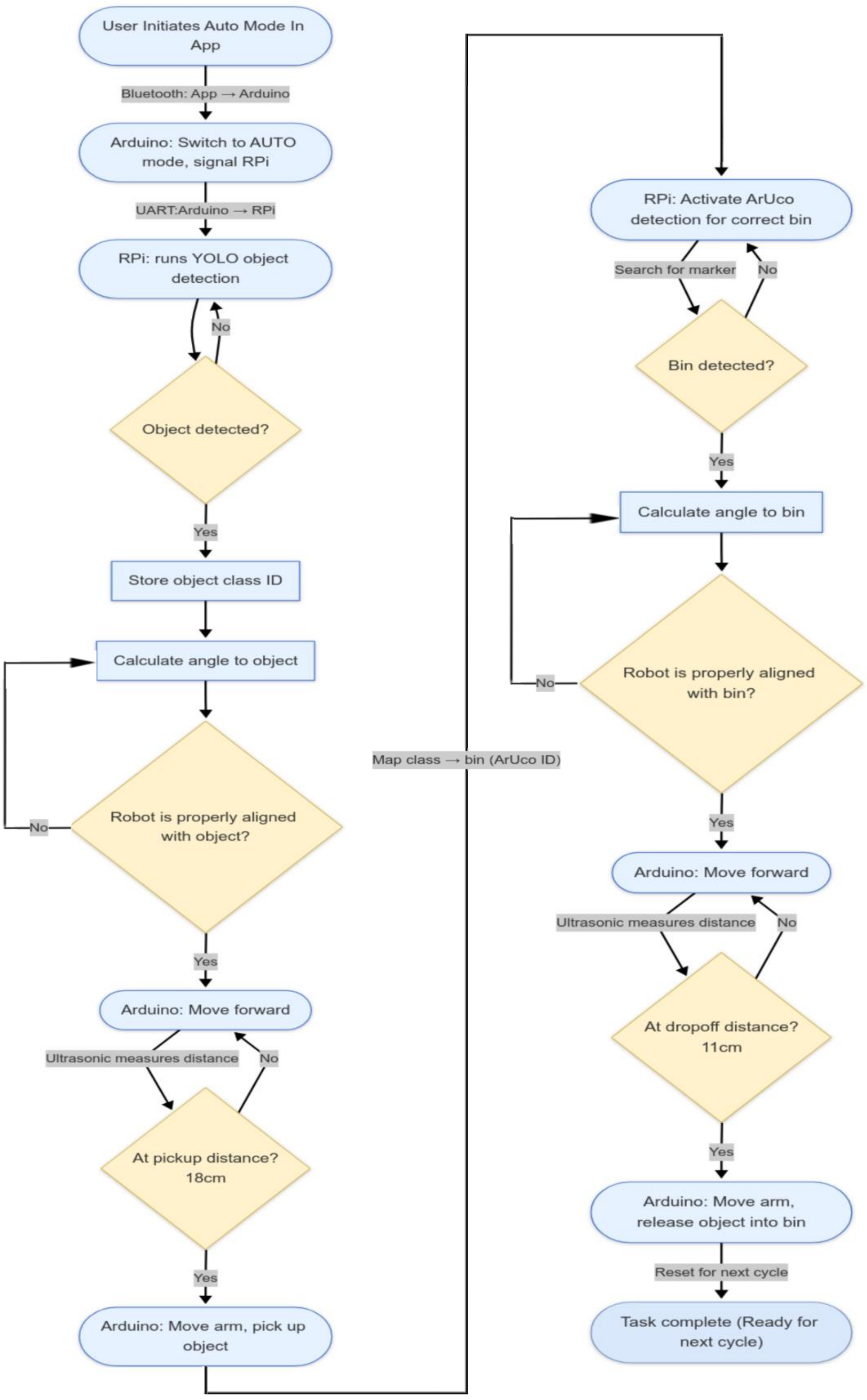

User Initiates Auto Mode In App
Bluetooth: App → Arduino
Arduino: Switch to AUTO mode, signal RPi
UART:Arduino → RPi
RPi: runs YOLO object detection
No
Object detected?
Yes
Store object class ID
Calculate angle to object
No
Robot is properly aligned with object?
Yes
Arduino: Move forward
Ultrasonic measures distance
No
At pickup distance? 18cm
Yes
Arduino: Move arm, pick up object
Map class → bin (ArUco ID)
RPi: Activate ArUco detection for correct bin
Search for marker
No
Bin detected?
Yes
Calculate angle to bin
No
Robot is properly aligned with bin?
Yes
Arduino: Move forward
Ultrasonic measures distance
No
At dropoff distance? 11cm
Yes
Arduino: Move arm, release object into bin
Reset for next cycle
Task complete (Ready for next cycle)

***Figure** III.3 Complete System Operation Flow for Object Classification and Sorting*

The robot operates via a distributed software architecture, which we designed with three primary, interconnected components (see Figure 1 for a conceptual overview):

- **Vision Processing Layer (Raspberry Pi 4):** This layer serves as the "brain" of the robot. The software, primarily written in Python, is responsible for computer vision tasks (including object detection and ArUco marker identification), complex calculations, and high-level decision-making processes.
- **Actuation Layer (Arduino Mega 2560):** This layer acts as the "muscle." The firmware, developed in C++/Arduino language, directly interfaces with and controls all low-level hardware components, including DC motors for chassis movement and servo motors for the robotic arm.
- **User Interface Layer (Android App):** This component provides an interface for human-robot interaction, enabling remote control, switching between operational modes (manual and autonomous), and monitoring the robot's status.

The following sections will explore each layer in detail, illustrating how they collaborate to achieve the robot's objectives.

III.4 Vision Processing Layer (Raspberry Pi 4)

The Raspberry Pi 4 manages all vision processing tasks and executing high-level decision-making logic. The software for this layer was developed in Python, leveraging several powerful open-source libraries.

III.4.1 Initial Setup and Development Environment on Raspberry Pi

Before diving into the core application code, a foundational setup was established on the Raspberry Pi.

III.4.2 Raspberry Pi OS

We selected Raspberry Pi OS as the operating system for our Raspberry Pi 4. This Debian-based OS is optimized for Raspberry Pi hardware, offering excellent compatibility and performance. Its native support for GPIO pins and camera modules [54] was crucial for our project. Raspberry Pi OS provided a stable and developer-friendly platform, well-suited for Python development, running our object detection model, and managing serial communication with the Arduino microcontroller.

III.4.2.1 Enabling VNC for Remote Access

During development and testing, especially when the robot was assembled and direct physical connection of the monitor, keyboard, mouse was impractical, remote access was essential. We used a tool called RealVNC [55], to remotely view and control the raspberry pi over a network, This tool consists of two main components: The **VNC Server** (installed on the machine to be controlled, i.e., the Raspberry Pi) and a **VNC Viewer** (used on a separate computer, e.g., a laptop, to access the remote machine). Both devices must be on the same local network. It transmits screen pixel data and relays keyboard/mouse inputs [55].

To enable VNC on the Raspberry Pi, we performed the following steps via the terminal:

1. Opened the Raspberry Pi configuration tool: `sudo raspi-config`
2. Navigated to "Interface Options."
3. Selected "VNC" and enabled it.
4. Finished and exited the configuration tool.

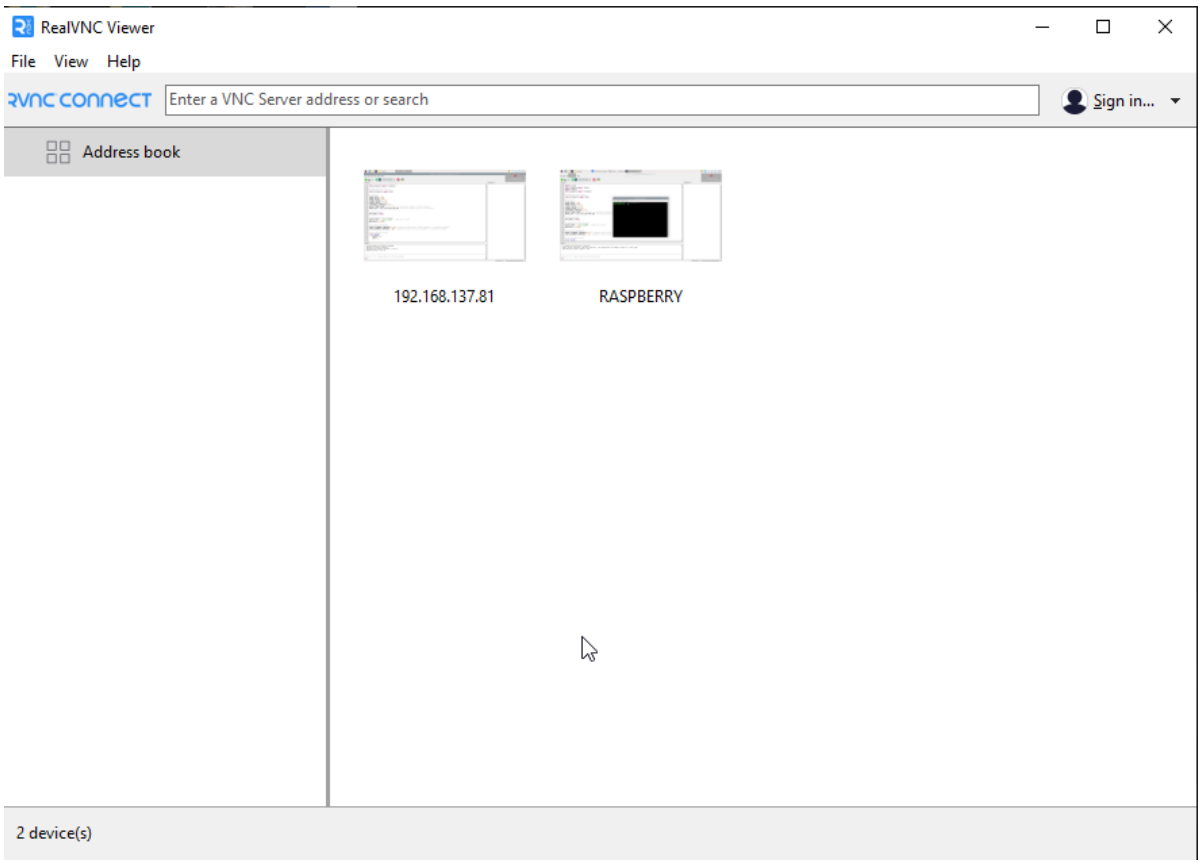


Figure III.4 RealVNC remote desktop interface

III.4.2.2 Setting Up UART Communication with Arduino

To Enable communication between the Raspberry Pi and the Arduino Mega, UART (Universal Asynchronous Receiver/Transmitter) serial communication was configured by doing the following[68]:

1. **Disabling Serial Console:** Using `sudo raspi-config`, we navigated to Interface Options > Serial. The login shell over serial was disabled, and the serial hardware interface was enabled. This frees up the serial port for our application.
2. **Enabling UART:** We ensured the line `enable_uart=1` was present in the `/boot/config.txt` file. Reboot the system to apply the changes.
3. **Verification:** Running `/dev/serial*` in the terminal was used to verify UART device availability (e.g., `/dev/serial0` or `/dev/ttyAMA0`).

III.4.2.3 Creating a Python Virtual Environment

To maintain a clean and manageable development environment and avoid conflicts between project-specific dependencies and the system-wide Python installation, a Python virtual environment was created:

1. **Install the venv package:**
   ```
   sudo apt install python3-venv
   ```
2. **Create the virtual environment (e.g., named recycle):**
   ```
   python3 -m venv recycle
   ```
3. **Activate the virtual environment:**
   ```
   source recycle/bin/activate
   ```

Once activated, all Python packages installations (using pip) were kept to this isolated environment.

III.4.3 Core Libraries and Frameworks for Vision Processing

III.4.3.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an extensive open-source library for computer vision and machine learning applications [56]. It offers a vast array of over 2,500 optimized algorithms for tasks such as object detection, motion tracking, and image manipulation. OpenCV supports multiple programming languages, including Python, which we used.

In our project, OpenCV was indispensable for:

- **Frame Acquisition and Preprocessing:** Capturing images from the PiCamera and resizing them to the required dimensions for our model input.
- **ArUco Marker Detection:** This involved converting images to grayscale, a crucial step for robust ArUco marker identification.
  - ❖ **What are ArUco Markers?**
    ArUco markers are square patterns with a black border and an inner binary matrix encoding a unique identifier (ID). They are widely used in computer vision for tasks like pose estimation and object identification due to their ease of detection and robustness to lighting variations [57].

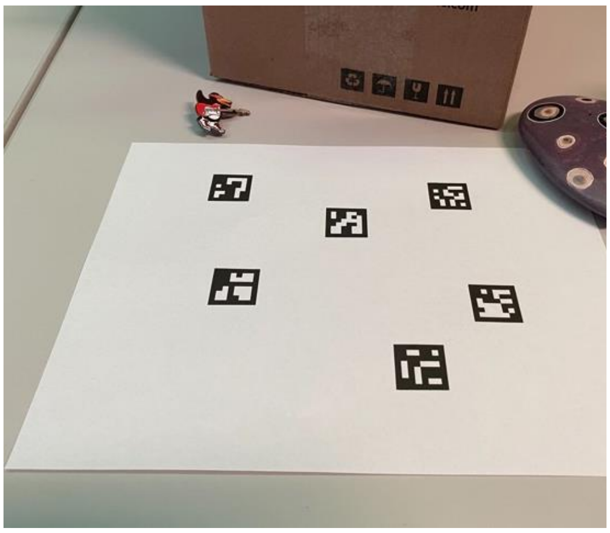


Figure III.5 image with aruco markers

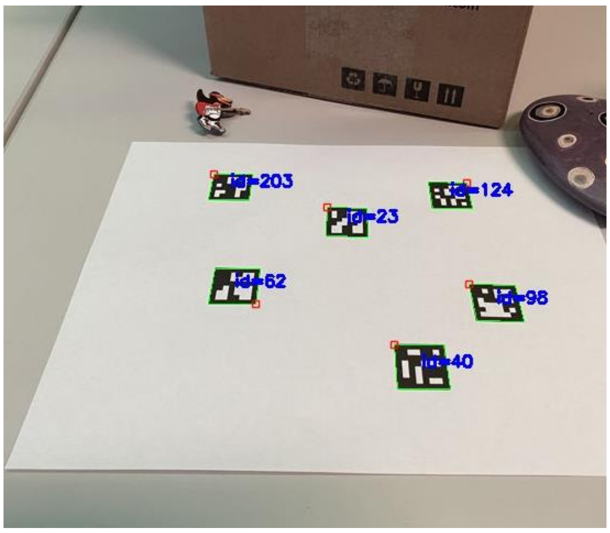
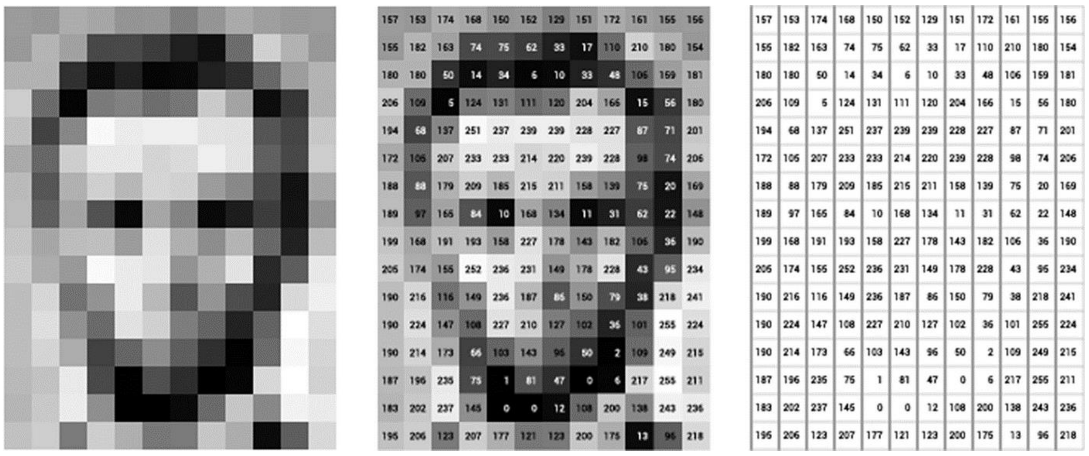


Figure III.6 Image with detected markers

- **Why is Grayscaling Important for ArUco Detection?**
  Converting images to grayscale simplifies ArUco marker detection by removing color information. This allows the detection algorithm to focus on contrast and edges, which are key features of the black-and-white patterns of ArUco markers. Grayscaling enhances their visibility, improves detection accuracy, and reduces computational load, leading to faster and more efficient marker recognition [58].



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Figure III.7 Grayscale Conversion in Image Processing

- **Drawing Utilities**: Overlaying visual information (bounding boxes, lines, labels) onto the video feed for debugging, real-time monitoring, and visual feedback of the detection process.

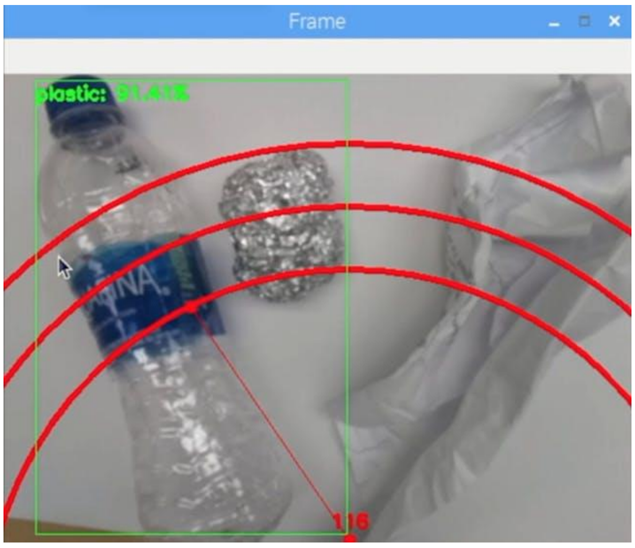


Figure III.8 an example showing open cv visual elements in the video feed

III.4.3.2 YOLOv8n (You Only Look Once version 8 nano)

YOLO (You Only Look Once) is a family of highly efficient real-time object detection models. YOLO processes an entire image in a single pass through a neural network, enabling simultaneous detection of multiple objects. YOLOv8, released by Ultralytics in January 2023, includes several improvements compared to earlier versions, including improved backbone and neck architectures for better feature extraction, and an anchor-free split Ultralytics head for enhanced accuracy and efficiency [59]. YOLOv8 supports various tasks like object detection, segmentation, and classification, The figure below shows the YOLOv8 network structure, which is divided into three parts: the Backbone extracts important features from the input image, the Neck processes and combines these features, and the Head performs object detection at multiple scales

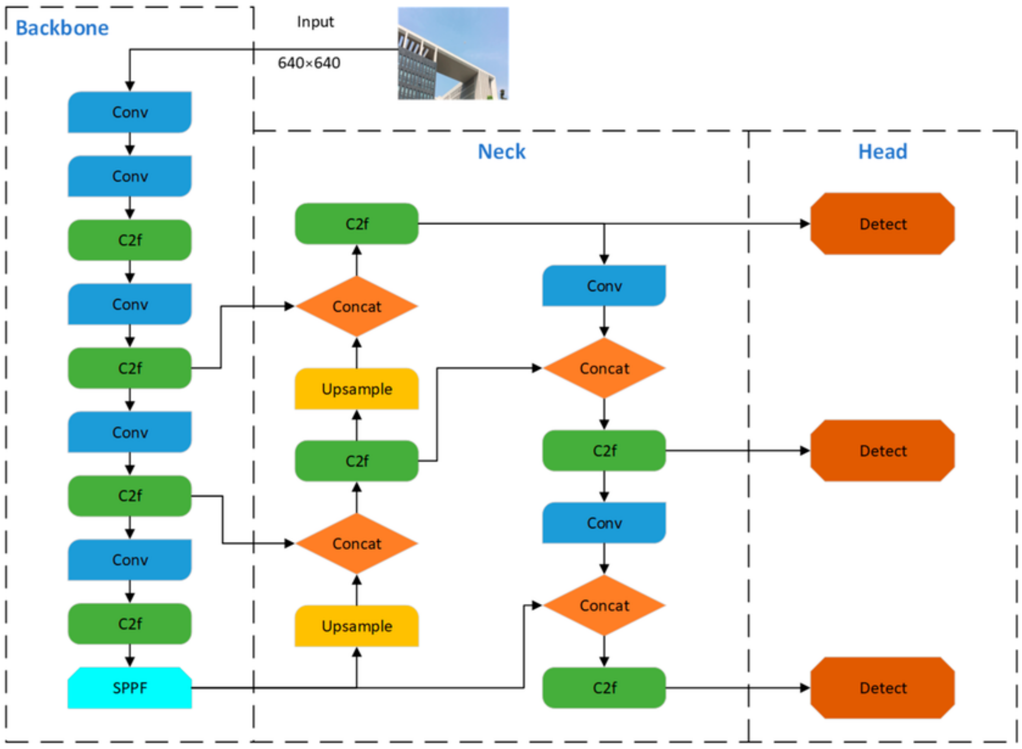


Figure III.9 YOLOv8 network structure diagram.

For our waste detection system, we selected **YOLOv8n (nano)**, the smallest variant of YOLOv8. This choice was driven by the need to balance detection accuracy with inference speed, a critical consideration for real-time applications on resource-constrained platforms like the Raspberry Pi 4 [59]. While newer YOLO versions might offer higher accuracy, they often come with increased computational demands. Older versions, on the other hand, lack some of the key optimizations present in YOLOv8. YOLOv8n achieves a mean Average

Precision (mAP) of 37.3 on the COCO dataset while maintaining high-speed performance [59].

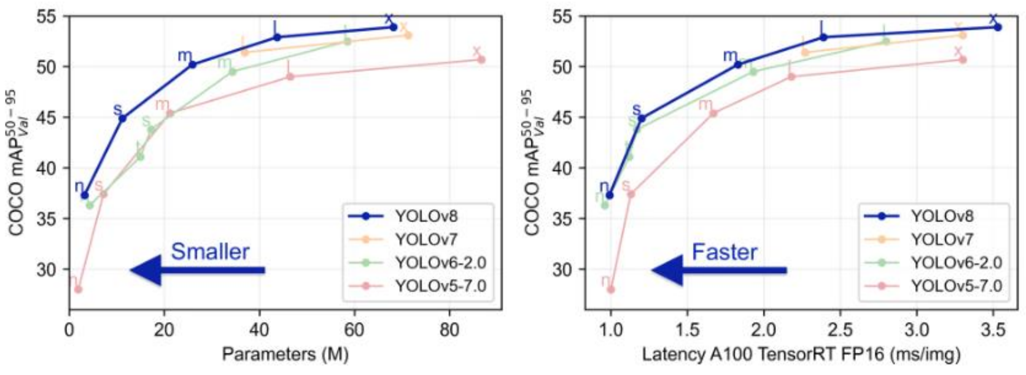


Figure III.10 YOLO Performance Comparison

III.4.3.2.1 Converting YOLOv8n to NCNN for Raspberry Pi Optimization

Running deep learning models like YOLOv8 directly on a Raspberry Pi 4 can be computationally demanding, leading to slow inference times. To make the model a little bit faster, we converted our trained YOLOv8n model to the NCNN format. NCNN (High-performance Neural Network Inference Computing Framework) is a framework optimized for ARM-based architectures, such as our processor in the Raspberry Pi 4. This conversion significantly improves inference speed and reduces memory usage.

The conversion process using Ultralytics' tools is straightforward:

1. **Load** the YOLO **Model (e.g., in a Python script):**
   ```
   from ultralytics import YOLO
   model = YOLO("yolov8n.pt")# Load the pre-trained or custom-trained .pt model
   ```
2. **Export to NCNN Format:**
   ```
   model.export(format="ncnn", imgsz=640) # Or a lower resolution like 320 for more speed
   ```
   This command creates a directory (e.g., yolov8n_ncnn_model) containing the NCNN model files. The image size (**imgsz**) should ideally be a multiple of 32.
3. **Use the Converted Model in the Application:**
   ```
   model = YOLO("yolov8n_ncnn_model") # Load the converted NCNN model
   ```

This optimization was crucial for achieving a better detection performance on the Raspberry Pi 4. As shown in comparative benchmarks, NCNN offers significantly better inference times on Raspberry Pi hardware compared to other formats like PyTorch or

TensorFlow Lite for YOLOv8n [66].

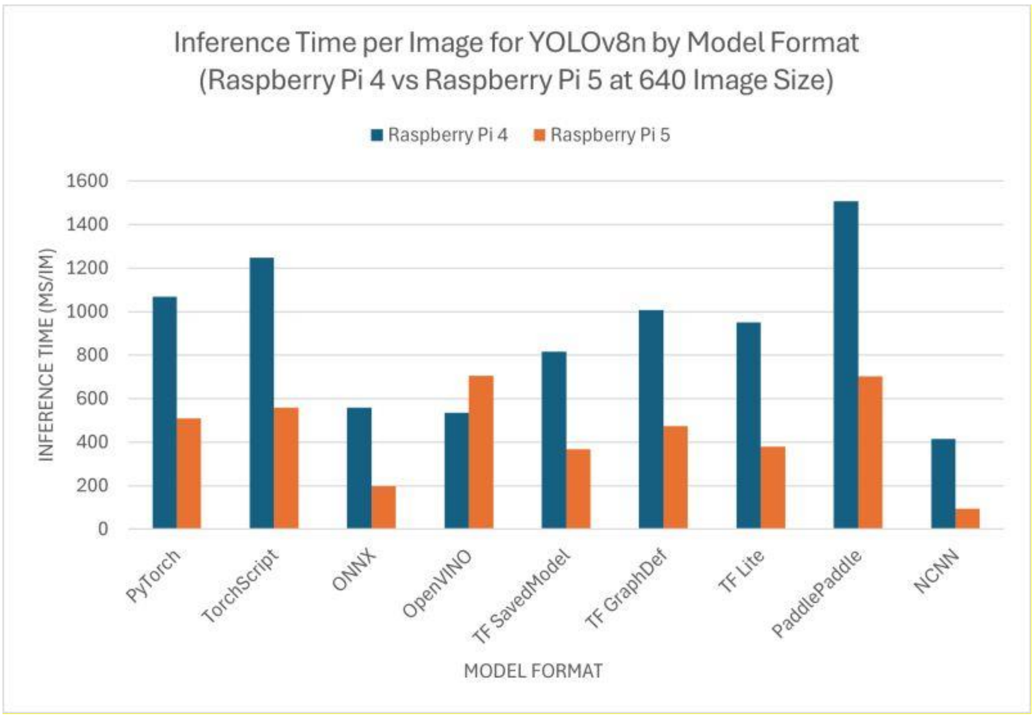


Figure III.11 internce time per image for yolov8n by model format

III.4.3.2.2 Dataset Acquisition and Preprocessing using Roboflow

The training of our custom YOLOv8n model relied on a dataset managed and processed using Roboflow which is a web-based platform designed to streamline computer vision model development, offering tools for dataset annotation, preprocessing, augmentation, and management [60].

We utilized a publicly available "waste-detection Dataset" [61] from Roboflow Universe, which originally contained approximately 15,962 images across 22 distinct waste classes. It was split into:

- **Training set:** 13,568 images (85%)
- **Validation set:** 1,607 images (10%)
- **Test set:** 787 images (5%)

To match our robot's arm capabilities, we excluded classes that were impractical for the gripper to grasp, such as caps and straws etc, we retained only the classes that matched the types of materials our robot could effectively grasp and sort:

- **Plastic:** retained the plastic_bottle class.

- **Paper:** retained the scrap_paper class.
- **Metal**: Retained the can class.

Roboflow was also used to apply essential preprocessing and augmentation techniques to enhance the model's robustness and ability to generalize to real-world scenarios:

- **Resolution Standardization:** Images were resized to 640×640 pixels using Roboflow's "fit (letterbox)" method, which preserves aspect ratio by padding with black edges if necessary.
- **Augmentation Strategies (applied to the training set):**
  - **Flipping:** Horizontal and vertical flips.
  - **Rotation:** 90° clockwise/counter-clockwise rotations, and random rotations between -15° and +15°.
  - **Grayscale Conversion:** Applied to 25% of images.
  - **Color Adjustments:** Random modifications to hue, saturation, brightness, and exposure (±25% range).
  - **Blur and Noise:** Gaussian blur (up to 2.5 pixels) and random noise (up to 5% of pixels).
  - **Cutout Augmentation (Mosaic):** Introduction of seven random occlusions (each 3% of image size) to improve detection of partially obscured objects.

III.4.3.2.3 Model Training in Google Colab

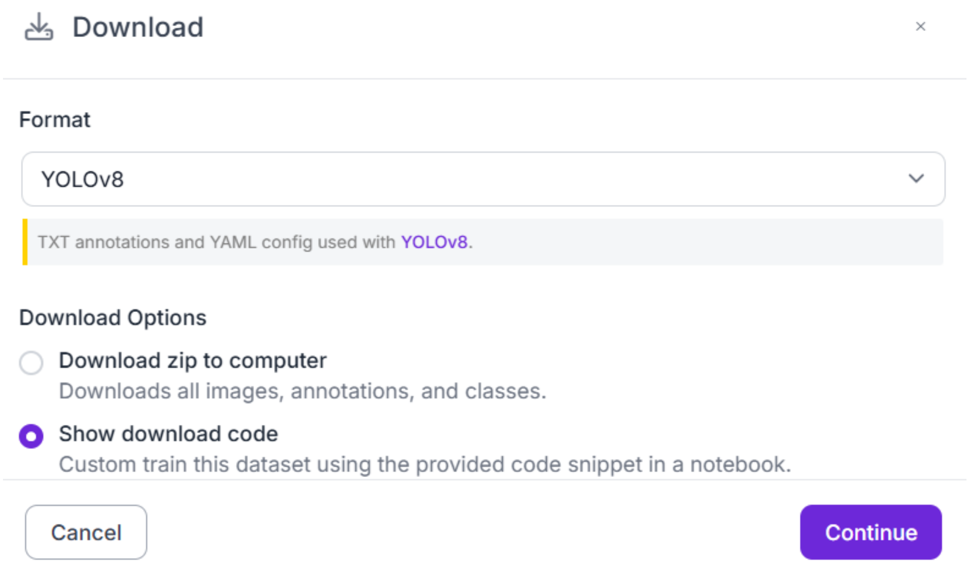
The YOLOv8n model was trained using Google Colaboratory (Colab) which is a free, cloud-hosted Jupyter notebook environment that provides access to Google Cloud resources, including GPUs and TPUs, without requiring local setup [62]. This was particularly beneficial for accelerating the computationally intensive model training process.

We adapted an open-source training notebook provided by Roboflow [63]. The training process involved the following key steps:

1. **Environment Setup in Colab:**
   First, the necessary Python libraries were installed in the Colab environment. We recommend pinning versions for reproducibility:

```
!pip install ultralytics==8.0.227
!pip install roboflow==1.1.10
```

2. **Dataset Integration via Roboflow SDK:**
   The dataset was downloaded directly into the Colab environment using the Roboflow Python SDK. This required an API key from our Roboflow account. Roboflow's interface provides a code snippet to simplify this.



Format: YOLOv8

TXT annotations and YAML config used with YOLOv8.

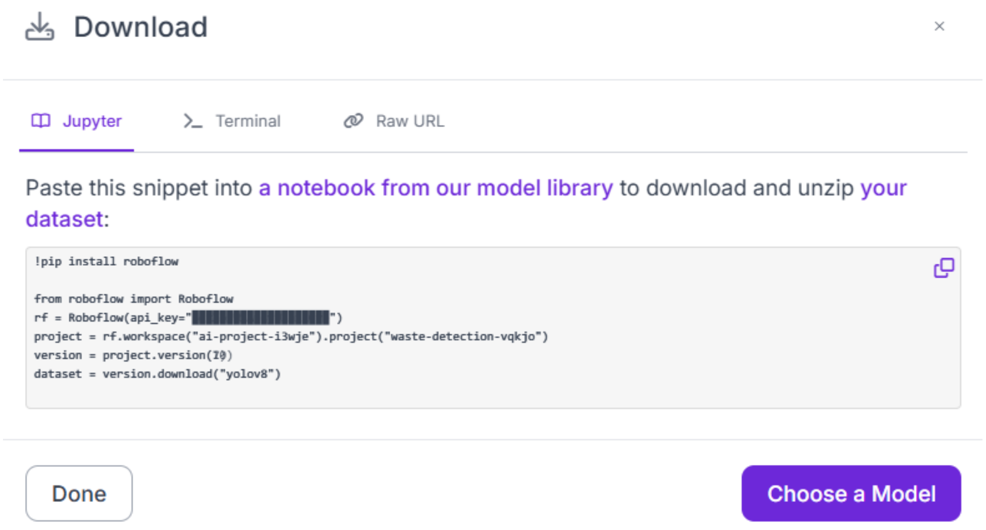
Download Options

Download zip to computer
Downloads all images, annotations, and classes.

[✓ Show download code]
Custom train this dataset using the provided code snippet in a notebook.

Figure III.12 the interface for downloading datasets in roboflow

The generated snippet, once pasted and run in a Colab cell, downloads and prepares the dataset in the YOLOv8 compatible format.



Paste this snippet into a notebook from our model library to download and unzip your dataset:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("ai-project-i3wje").project("waste-detection-vqkjo")
version = project.version(10)
dataset = version.download("yolov8")
```

Figure III.13 the dataset's code snippet for notebooks

3. **Model Training Execution:**

   Training was initiated using the YOLOv8 command-line interface within the Colab notebook.

```
!yolo task=detect mode=train model=yolov8n.pt
data=/content/path/to/your/dataset/data.yaml epochs=50 imgsz=640 plots=True
```

   Training Parameters Overview:

   - **Model:** yolov8n.pt (utilizing pretrained weights from the COCO dataset for transfer learning)
   - **Task:** detect (object detection)
   - **Epochs:** 50 (number of passes through the entire training dataset, we avoided a bigger number to avoid overfitting)
   - **Image size (imgsz):** 640 (input image resolution 640x640 pixels)
   - **Data:** Path to the data.yaml file generated by Roboflow, which describes the dataset paths and class names.
   - **Plots:** True (to generate performance visualization plots like confusion matrices and PR curves).

   Upon completion, the trained model weights (e.g., best.pt and last.pt in a runs/detect/train directory) and performance plots are saved within the Colab environment, from where they can be downloaded.

III.4.3.2.4 Understanding Performance Metrics in Object Detection

To evaluate the trained model, several standard object detection metrics are used to quantify its ability to correctly identify and localize objects while minimizing errors. These metrics can be divided into numerical evaluations and visual outputs, both of which help assess the strengths and weaknesses of the model. [64]:

- **Key Numerical Metrics:**
  - **True Positives (TP):** The model correctly predicts a bounding box for an object that is actually present, with sufficient overlap and correct class.

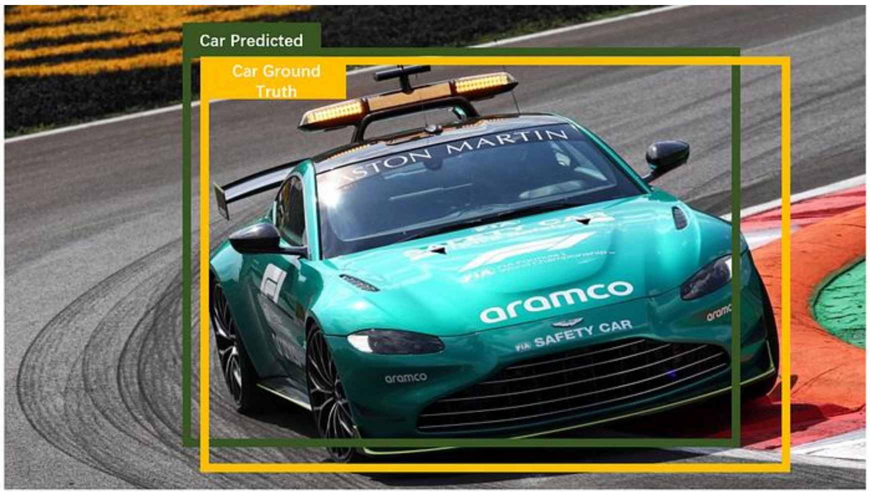


Figure III.14 Example of A True Positive Prediction[69]

- **False Positives (FP):** The model predicts a bounding box where no object exists, or predicts the wrong class for an object, or the bounding box has insufficient overlap with the ground truth.

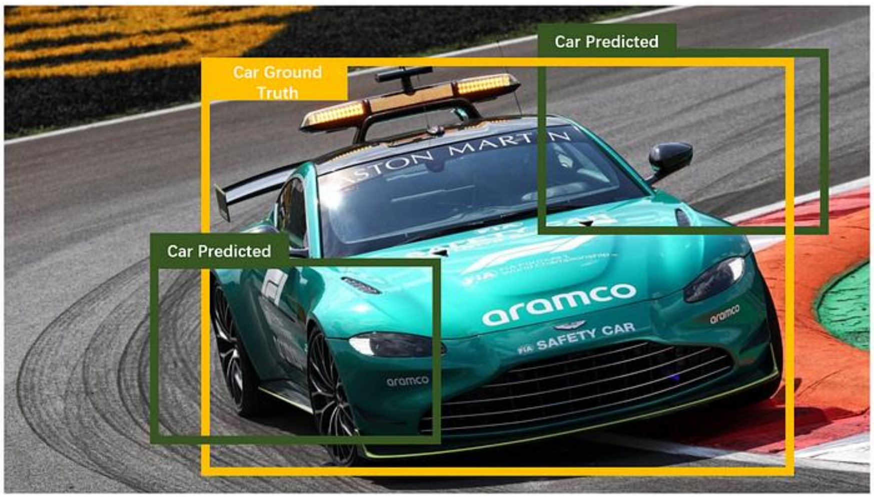


Figure III.15 Example of Two false positive predictions because of not enough overlapping region[69]

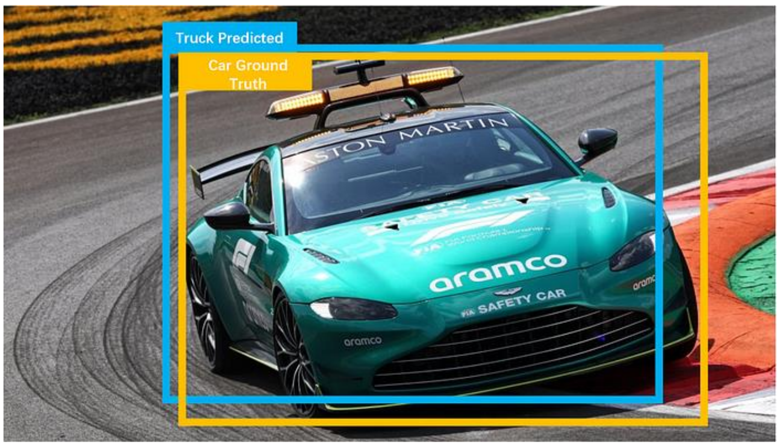


Figure III.16 Example of Misclassifying a car as a truck. One False Positive in the image[69]

- **False Negatives (FN):** The model fails to detect an object that is actually present.

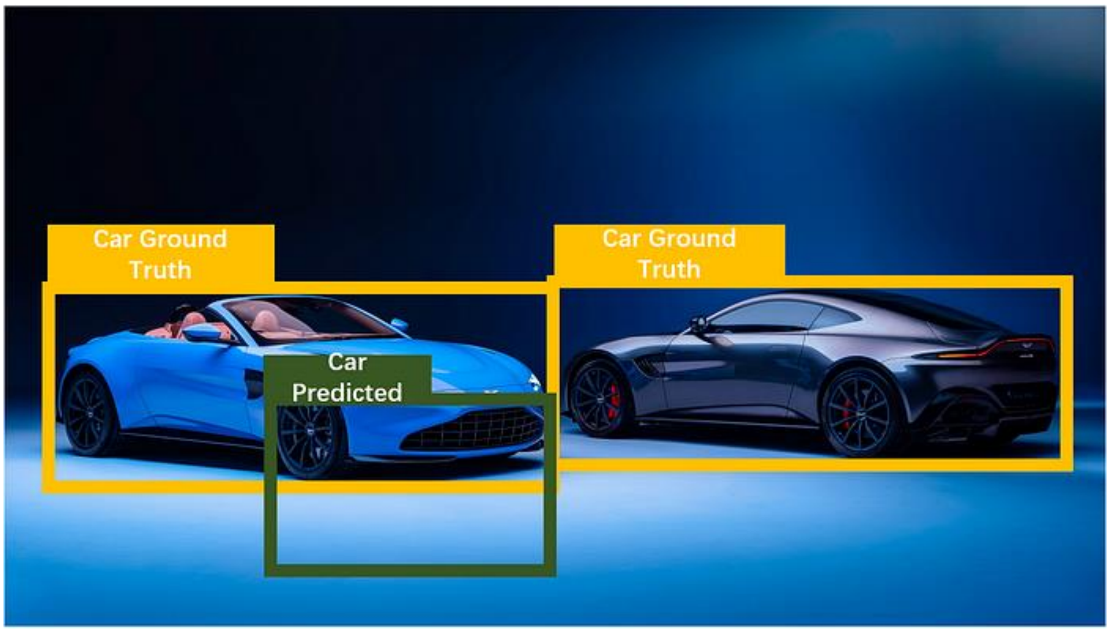


Figure III.17 Example of Two False Negatives and One False Positive[69]

- **True Negatives (TN):** While less commonly emphasized in object detection (which focuses on finding objects), this would refer to correctly not predicting an object in areas where none exist.
- **Intersection over Union (IoU):** Measures the overlap between the predicted bounding box and the ground truth bounding box. It's calculated as: IoU = Area of Overlap / Area of Union. A higher IoU indicates a better localization of the object. An IoU threshold is often used to classify a detection as a TP or FP.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

- **Precision:** Measures the accuracy of the positive predictions.

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Example: If the model predicts 10 objects as "plastic," and 7 of them are indeed plastic, the precision for "plastic" is 70%.

- **Recall (Sensitivity):** Measures the model's ability to find all relevant instances.

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Example: If there are 10 plastic items in an image, and the model correctly identifies 7 of them, the recall for "plastic" is 70%.

- **Mean Average Precision (mAP):** The primary metric for evaluating object detection models like YOLO. It is the average of Average Precision (AP) values calculated for each class. AP itself is an area under the Precision-Recall curve. mAP is often reported at a specific IoU threshold (e.g., mAP@0.5) or averaged over a range of IoU thresholds (e.g., mAP@0.5:0.95). A higher mAP indicates better overall model performance.

- **Visual Outputs for Performance Evaluation:**
  Beyond numbers, visual outputs provide intuitive insights into model behavior:
  - **Confusion Matrix (confusion_matrix_normalized.png):** Visualizes how often each class was correctly predicted versus misclassified as another. It shows percentages, making it easier to spot confusion between classes..
  - **Precision-Recall Curve (PR_curve.png):** Displays precision-recall trade-offs at different thresholds, helping fine-tune detection sensitivity.
  - **F1 Score Curve (F1_curve.png):** Illustrates changes in F1 score, showing the balance between precision and recall.
  - **Validation Batch Labels (val_batchX_labels.jpg):** Images displaying ground truth labels to compare with detections.
  - **Validation Batch Predictions (val_batchX_pred.jpg):** Visual representations of the model's detected objects, revealing potential errors.

By combining numerical evaluations and visual analysis, object detection models can be fine-tuned for better performance. High precision and recall, combined with strong IoU and mAP scores, indicate a model that effectively detects objects with minimal errors. Visual outputs further aid in diagnosing weaknesses and optimizing results.

III.4.4 Raspberry Pi Software System Architecture

The Python code running on the Raspberry Pi serves as the main intelligence of the robot. Its core responsibilities include:

- Capturing and processing video frames from the camera in real-time.
- Detecting recyclable waste objects using the custom-trained and NCNN-converted YOLOv8n model.
- Identifying waste bins using ArUco markers.

- Calculating precise alignment angles for navigation and manipulation.
- Communicating commands and receiving status updates from the Arduino Mega via UART.

To achieve these functionalities, we utilized several key Python libraries in our custom code:

- **OpenCV (cv2):** For image acquisition, preprocessing, ArUco marker detection, and drawing utility functions.
- **Ultralytics YOLO:** For loading the NCNN-converted YOLOv8n model and performing object detection inference.
- **Picamera2:** For interfacing directly with the Raspberry Pi camera module, offering optimized performance.
- **PySerial:** For handling serial communication over UART with the Arduino.
- **NumPy:** For efficient numerical operations, particularly on image data and geometric calculations.
- **Threading:** To let vision processing and control run at the same time, so the heavy vision tasks don't slow down the robot..

III.4.4.1 Vision Systems Implementation

Two distinct vision systems were implemented on the Raspberry Pi:

III.4.4.1.1 Object Detection System (YOLOv8n)

The YOLOv8n model was trained to recognize three classes of recyclable waste:

- **Class 0:** Metal items
- **Class 1:** Plastic items
- **Class 2:** Paper items

Here is a snippet code of the YOLO detection function from our implementation::

```
# Run YOLO detection on the current frame
def run_yolo_detection(model, frame, confidence):
    try:
        # Ensure frame is properly formatted
        if frame is None:
            return []

        # Run YOLOv8 inference on the frame
        results = model(frame, conf=confidence)[0]

        detections = []
        # Process the results
        for detection in results.boxes.data.tolist():
      # Extract detection data [x1, y1, x2, y2, confidence, class_id]
            x1, y1, x2, y2, conf, class_id = detection

            if conf >= confidence:
                # Calculate center point and store detection data
                center_x = (x1 + x2) / 2
                center_y = (y1 + y2) / 2

                detections.append({
                    'bbox': (int(x1), int(y1), int(x2), int(y2)),
                    'confidence': conf,
                    'center': (center_x, center_y),
                    'class_id': int(class_id)
                })

        class_names = {0: "metal", 1: "plastic", 2: "paper"}
        # ... additional logging and processing ...

        return detections
    except Exception as e:
        print(f"Error in YOLO detection: {e}")
        return []
```

III.4.4.1.2 Bin Identification System (ArUco Markers)

For reliable bin identification, especially under varying lighting conditions where color detection can be unreliable, ArUco markers were employed. Each recycling bin was equipped with a unique ArUco marker.

Here is a snippet code of the ArUco marker detection function from our implementation:

```
# Initialize ArUco detector
def initialize_aruco():
    aruco_params = cv2.aruco.DetectorParameters()
    aruco_dict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT)
    return aruco_dict, aruco_params

# Detect ArUco markers in the frame
def detect_aruco_markers(frame, aruco_dict, aruco_params):
    try:
        # Convert to grayscale for better detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect markers
        detector = cv2.aruco.ArucoDetector(aruco_dict, aruco_params)
        corners, ids, rejected = detector.detectMarkers(gray)

        markers = []
        if ids is not None:
            for i, corner in enumerate(corners):
                # Get marker ID and calculate center
                marker_id = ids[i][0]
                corners_array = corner[0]
                center_x = np.mean([c[0] for c in corners_array])
                center_y = np.mean([c[1] for c in corners_array])

                markers.append({
                    'id': marker_id,
                    'corners': corner,
                    'center': (center_x, center_y)
                })
            # ... logging and additional processing ...

        return markers
    except Exception as e:
        print(f"Error in ArUco detection: {e}")
        return []
```

The ArUco marker IDs were mapped to specific bin types:

- **ArUco ID 0:** Plastic bin
- **ArUco ID 1:** Paper bin
- **ArUco ID 2:** Metal bin

The mapping links the detected object class (from YOLO) to the corresponding target bin ArUco ID:

```
# Object-to-bin mapping
OBJECT_TO_BIN_MAP = {
    0: 2,  # Metal -> ArUco ID 2
    1: 0,  # Plastic -> ArUco ID 0
    2: 1,  # Paper -> ArUco ID 1
}
```

III.4.4.2 Alignment Algorithm

Precise alignment of the robot with detected objects (for pickup) and bins (for drop-off) was critical. An angle calculation function was developed to determine the necessary turn adjustment, Here is a snippet code of the angle calculation function from our implementation:

```
# Calculate angle between center of frame and object
def calculate_angle(object_x, object_y):
    # Calculate vector from camera center to object
    vector_x = object_x - CAMERA_CENTER_X
    vector_y = CAMERA_CENTER_Y - object_y  # Y is inverted in image coordinates

    # Calculate angle in degrees (0 degrees is straight ahead)
    angle = math.degrees(math.atan2(vector_x, vector_y))

    return angle
```

Key aspects of this alignment approach:

- **Camera Center Calibration:**
  - ➢ Center point established at (275, 370) pixels
  - ➢ Calibrated by testing until camera view aligned with ultrasonic sensor direction
  - ➢ Serves as the critical reference point for all navigation calculations

- **Angle Calculation Method:**
  - Vector created from camera center to object/ArUco marker center
  - Vector components:
    (object_x - CAMERA_CENTER_X, CAMERA_CENTER_Y - object_y)
  - Angle calculated using arctangent:
    math.degrees(math.atan2(vector_x, vector_y))
  - 0 degrees represents object/bin is straight ahead
- **Angle Interpretation:** A positive angle typically indicates the target is to the robot's right, requiring a right turn, while a negative angle indicates the target is to the left, requiring a left turn.

III.4.4.3 Multi-threaded Design for Simultaneous Tasks

To manage simultaneous vision processing (which can be computationally intensive) and control logic (which needs to be responsive), a multi-threaded architecture was implemented:

- **Main Thread:** Initializes all components (camera, serial communication, models) and manages the overall program lifecycle.
- **Video Processing Thread:** Dedicated to continuously capturing frames from the camera, performing object detection (YOLO) and ArUco marker detection, and updating shared data structures with the latest vision information.
- **Control Thread:** Runs the main state machine of the robot, makes decisions based on the vision data (from the video processing thread) and sensor inputs (e.g., from Arduino), and sends commands to the Arduino.

This simultaneous design ensures that the robot remains responsive to commands and environmental changes even while complex image processing is underway.
Here is a snippet code of the multi-threaded design from our implementation:

```python
# Main function
def main():
    # Initialize global variables and components
    target_aruco_id = 0
    AUTO_MODE = False

    # Initialize hardware and models
    picam2 = initialize_camera()
```

```
model = initialize_yolo()
aruco_dict, aruco_params = initialize_aruco()
ser = initialize_serial()

# Start threads
video_thread = Thread(target=video_processing_thread,
                args=(picam2, model, aruco_dict, aruco_params))
video_thread.daemon = True
video_thread.start()

control_thread_instance = Thread(target=control_thread, args=(ser,))
control_thread_instance.daemon = True
control_thread_instance.start()

print("System initialized and running. Press Ctrl+C to exit.")

# Main loop - keep program running and handle keyboard interrupt
try:
    while True:
        time.sleep(1.0)
except KeyboardInterrupt:
    print("Exiting...")
    cv2.destroyAllWindows()
    picam2.close()
```

III.4.4.4 Visual Debugging Interface

To support development, testing, and real-time monitoring, a visual debugging interface was created using OpenCV's drawing capabilities. This interface, displayed in a window showing the camera feed, showing crucial information such as:

- Bounding boxes around detected objects, along with their class labels and confidence scores.
- Detected ArUco markers with their IDs.
- Alignment line drawn from the robot's center to the target object/bin.
- Calculated angles for alignment.
- The current state of the Raspberry Pi's state machine.

- The current target (e.g., "Target: Plastic Bottle" or "Target: Bin ID 0").
- Distance readings received from the Arduino's ultrasonic sensor.
- Keyboard control options for manually triggering actions or modes during testing.

Here is a snippet code of the visual debugging interface implementation from our code:

```
# Draw debugging visuals on display_frame
if detection_data:
    if current_target == Target.OBJECT:
        # Draw bounding box for YOLO
        x1, y1, x2, y2 = detection_data['bbox']
        cv2.rectangle(display_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        # Draw center point and line to frame center
        center_x, center_y = detection_data['center']
        cv2.circle(display_frame, (int(center_x), int(center_y)), 5, (0, 0, 255), -1)
        cv2.line(display_frame, (int(center_x), int(center_y)),
                (CAMERA_CENTER_X, CAMERA_CENTER_Y), (255, 0, 0), 2)

        # Draw class and confidence information
        class_names = {0: "metal", 1: "plastic", 2: "paper"}
        class_id = detection_data.get('class_id', -1)
        class_name = class_names.get(class_id, "unknown")
        conf_text = f"{class_name}: {detection_data['confidence']:.2f}"
        cv2.putText(display_frame, conf_text, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        # ... additional visualization for target bin ...
    elif current_target == Target.BIN:
        # Draw ArUco marker visualization
        # ... marker visualization code ...
# Draw frame center reference
cv2.circle(display_frame, (CAMERA_CENTER_X, CAMERA_CENTER_Y), 5, (0, 255, 255), -1)
cv2.circle(display_frame, (CAMERA_CENTER_X, CAMERA_CENTER_Y), 50, (0, 255, 255), 1)

# Add state and target information to frame
cv2.putText(display_frame, f"Target: {target_names[current_target]}", (10, 30),
           cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
# ... additional status information ...
```

This real-time visual feedback was invaluable for understanding the robot's "perception" and decision-making process, significantly aiding in debugging and refining the algorithms.

III.4.4.5 Raspberry Pi State Machine

The Raspberry Pi software is built around a state machine that guides the robot’s autonomous behavior. It handles vision processing, manages decision-making, and works with the Arduino to carry out physical actions.

Here is a snippet code of the state machine definition:

```python
# State machine states
class Target:
    OBJECT = 0  # Target is a waste object
    BIN = 1     # Target is a recycling bin

class State:
    MANUAL = -1      # Manual control mode
    SEARCH = 0       # Searching for target
    ALIGN=1
    APPROACH = 2     # Moving toward target
    ACTION = 3       # Performing pickup/dropoff
    # ... additional states ...
```

The state transitions are managed in the control thread, which continuously evaluates the current state and decides on the next appropriate action:

```python
# Main control thread (simplified)
def control_thread(ser):
    while True:
        # State machine logic
        if current_state == State.SEARCH:
            # Wait for object/bin detection
            if detection_data and consecutive_detections >= DETECTION_THRESHOLD:
                current_state = State.ALIGN_ROUGH
        elif current_state == State.ALIGN_ROUGH:
            # Calculate angle to target and send turn command
            if detection_data:
                angle = calculate_angle(detection_data['center'][0], detection_data['center'][1])
                send_command(ser, f"TURN:{angle}")
                current_state = State.ALIGN_FINE
    # ...
additional states ...
```

This state machine approach provides several benefits:

- Each part of the robot's operation is clearly separated
- Behavior is predictable, with clear transitions between tasks
- Easier debugging as the robot’s current state is always known
- Each state can be adjusted or improved independently

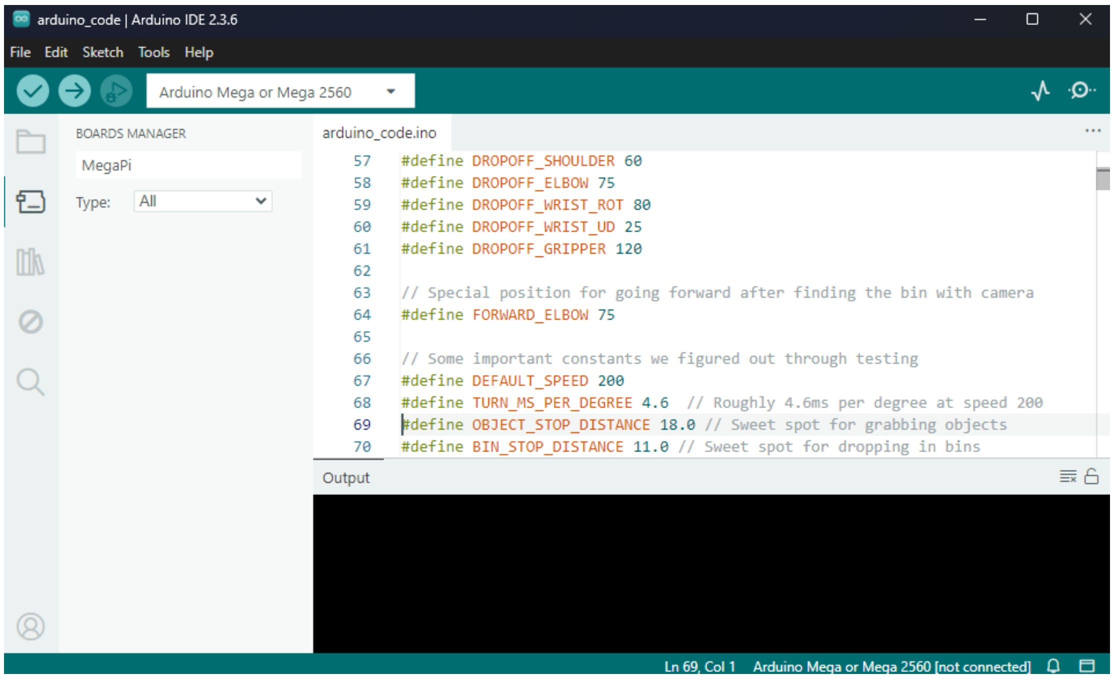
The Raspberry Pi handles perception and decision-making through its own state machine, while the Arduino runs a separate one that controls physical actions (explained in Section 4.7).

III.5 Actuation Layer (Arduino Mega 2560)

The Arduino Mega 2560 serves as the low-level hardware control of the robot. Its firmware, written in C++ using the Arduino programming language, is responsible for direct interaction with motors, servos, and sensors, executing commands received from the Raspberry Pi.

III.5.1 Development Environment: Arduino IDE

The Arduino Integrated Development Environment (IDE) was used for developing, compiling, and uploading firmware to the Arduino Mega 2560,The Arduino IDE provides a simple text editor for writing "sketches" (Arduino programs), a message area for compiler feedback, a serial monitor for debugging and communication[65]., and tools for managing libraries and boards. Its ease of use, extensive library support, and large community make it a popular choice for microcontroller projects.


arduino_code | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Mega or Mega 2560
BOARDS MANAGER
MegaPi
Type: All
arduino_code.ino
Output
Ln 69, Col 1 Arduino Mega or Mega 2560 [not connected]

```
#define DROPOFF_SHOULDER 60
#define DROPOFF_ELBOW 75
#define DROPOFF_WRIST_ROT 80
#define DROPOFF_WRIST_UD 25
#define DROPOFF_GRIPPER 120

// Special position for going forward after finding the bin with camera
#define FORWARD_ELBOW 75

// Some important constants we figured out through testing
#define DEFAULT_SPEED 200
#define TURN_MS_PER_DEGREE 4.6  // Roughly 4.6ms per degree at speed 200
#define OBJECT_STOP_DISTANCE 18.0 // Sweet spot for grabbing objects
#define BIN_STOP_DISTANCE 11.0 // Sweet spot for dropping in bins
```

Figure III.18 Arduino code in Arduino IDE

III.5.2 Arduino System Architecture and Responsibilities

The core responsibilities of the Arduino firmware include:

- Controlling the four DC motors for robot chassis movement (forward, backward, turning).
- Actuating the robotic arm's six servo motors for precise positioning during scanning, pickup, carrying, and drop-off sequences.
- Reading and processing data from the HC-SR04 ultrasonic sensor to measure distances to objects and bins.
- Managing bi-directional serial communication:
  - ➢ With the Raspberry Pi (via UART on Serial3) for receiving high-level commands and sending status updates/sensor data.
  - ➢ With the Android application (via Bluetooth module connected to Serial1) for remote control and status display.

Key libraries utilized in the Arduino firmware:

- **Wire.h:** For I2C communication, used by the servo driver library.
- **Adafruit_PWMServoDriver.h:** Essential for controlling the multiple servo motors via the PCA9685 16-channel PWM/Servo Driver board. This library simplifies the

generation of precise PWM signals required for servo positioning.

- **AFMotor**.h (Adafruit Motor **Shield Library):** Used for controlling the DC motors connected to an L293D-based motor driver shield. It provides straightforward functions for setting motor speed and direction.

III.5.3 Chassis Control System (DC Motors)

A set of functions was implemented to control the robot's four TT DC motors, enabling basic movements:

Here is a snippet code of the motor control functions from our implementation:

```
// Motor control functions
void moveForward(int speed) {
  setMotorSpeeds(speed);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD); }
void moveBackward(int speed) {
  setMotorSpeeds(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD); }
void turnLeft(int speed) {
  setMotorSpeeds(speed);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(FORWARD);
  motor4.run(FORWARD); }
void turnRight(int speed) {
  setMotorSpeeds(speed);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD); }
void stopMotors() {
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
```

```
 motor4.run(RELEASE); }
void setMotorSpeeds(int speed) {
 motor1.setSpeed(speed);
 motor2.setSpeed(speed);
 motor3.setSpeed(speed);
 motor4.setSpeed(speed);
}
```

Function	What It Does	Why we Implemented It
moveForward(speed)	Drives all motors forward at specified speed for forward movement	Provides simple control with variable speed for approach control (gradual speed while approaching objects/bins to slow down the robot)
moveBackward(speed)	Reverses all motors at specified speed for background movement	
turnLeft(speed)	Runs left motors backward and right motors forward for left turns	
turnRight(speed)	Runs right motors backward and left motors forward for right turns	
stopMotors()	Halts all motors immediately to stop the robot	To stop the robot after approaching object/bin

Table III.1 Robot Movement Control Functions

Turning System Implementation: When the Raspberry Pi calculates an angular offset to a target, it sends a **TURN:<angle>** command (e.g., TURN:30 for 30 degrees right, TURN:-45 for 45 degrees left) to the Arduino.

Here is a snippet code of the turn execution function from our implementation:

```
void executeTurn(int angle) {
 // Positive angle = turn right, negative angle = turn left
 unsigned long turnTime = abs(angle) * TURN_MS_PER_DEGREE;
 // Execute the appropriate turn direction
 if (angle > 0) {
  turnRight(DEFAULT_SPEED);
 } else {
  turnLeft(DEFAULT_SPEED);
 }

 delay(turnTime); // Wait for turn to complete
 stopMotors();

 // ... additional stabilization and reporting ...
}
```

Through repeated experimentation, we found that at our **DEFAULT_SPEED of 200**, it takes approximately 4.6ms of motor operation to turn 1 degree. This calibration was crucial for precise alignment with objects and bins otherwise the robot would keep trying to align and fail or not move at all.

III.5.4 Robotic Arm Control (Servo Motors)

The 5-DOF (Degrees of Freedom) robotic arm utilizes six servo motors, controlled via a PCA9685 servo driver board:

- **Three MG996R servos:** For base rotation, shoulder joint, and elbow joint (requiring higher torque).
- **Three SG90 micro servos:** For wrist rotation, wrist up/down tilt, and gripper actuation (requiring less torque but good precision).

The arm's movements were based on pre-defined, calibrated servo positions for various actions (scan, pickup, carry, drop-off). These positions are typically stored as angles or raw PWM pulse width values.

Here is a snippet code of the servo position definitions from our implementation:

```
// Servo definitions
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz
#define BASE_CHANNEL 0
#define SHOULDER_CHANNEL 4
#define ELBOW_CHANNEL 8
#define WRIST_ROT_CHANNEL 10
```

```
#define WRIST_UD_CHANNEL 15
#define GRIPPER_CHANNEL 14


#define SCAN_BASE 55
#define SCAN_SHOULDER 60
#define SCAN_ELBOW 100
#define SCAN_WRIST_ROT 80
#define SCAN_WRIST_UD 25
#define SCAN_GRIPPER 120

#define PICKUP_BASE 55
#define PICKUP_SHOULDER 23
#define PICKUP_ELBOW 90
#define PICKUP_WRIST_ROT 80
#define PICKUP_WRIST_UD 25
#define PICKUP_GRIPPER 10
#define PICKUP_BASE 55
#define PICKUP_SHOULDER 23
#define PICKUP_ELBOW 90
#define PICKUP_WRIST_ROT 80
#define PICKUP_WRIST_UD 25
#define PICKUP_GRIPPER 10

#define CARRY_BASE 55
#define CARRY_SHOULDER 60
#define CARRY_ELBOW 100
#define CARRY_WRIST_ROT 80
#define CARRY_WRIST_UD 10
#define CARRY_GRIPPER 10
// ... more position definitions ...
```

These calibrated positions were used to create sequences for arm movements, encapsulated in functions:

Function	**What It Does**	**Why It's Important**
moveArmToScanPosition()	Positions arm upright for environmental scanning	Gives the camera a clear view while the robot is searching
moveArmToPickupPosition()	Executes a careful pickup sequence	Uses a two-stage approach for gentler, more reliable grasping
moveArmToCarryPosition()	Secures the object in a stable carrying position	Prevents dropping while navigating to the target bin
moveArmToDropoffPosition()	Extends arm over the bin and releases the object	Ensures accurate disposal into the correct bin

Table III.2 Robot arm movement sequences Functions

Here is a snippet code of one of the arm movement functions from our implementation:

```
void moveArmToPickupPosition() {
  // First open the gripper to prepare
  moveServo(GRIPPER_CHANNEL, SCAN_GRIPPER);
  delay(500);

  // Move base servo
  moveServo(BASE_CHANNEL, PICKUP_BASE);

  // Use a two-stage approach for smoother motion:
  // First move to a pre-pickup position slightly higher
  int prePickupShoulder = PICKUP_SHOULDER + 15;
  int prePickupElbow = PICKUP_ELBOW - 10;

  moveServo(SHOULDER_CHANNEL, prePickupShoulder);
  moveServo(ELBOW_CHANNEL, prePickupElbow);
  moveServo(WRIST_ROT_CHANNEL, PICKUP_WRIST_ROT);
  moveServo(WRIST_UD_CHANNEL, PICKUP_WRIST_UD);

  // Wait for arm to reach the pre-pickup position
  delay(1200);

  // Now gradually lower the arm to the final position
```

```
 for (int pos = prePickupShoulder; pos >= PICKUP_SHOULDER; pos -= 3) {
   moveServo(SHOULDER_CHANNEL, pos);
   delay(100);
 }

 // ... additional positioning and gripper control code ...

 // Keep the gripper powered but detach other servos
 detachServo(WRIST_ROT_CHANNEL);
 detachServo(WRIST_UD_CHANNEL);
}
```

While those movement sequences were successful, they had some problems, Some of servo motors kept jittering, this occurs when the servos try to maintain their exact position but receive slightly fluctuating signals or experience varying loads, causing them to constantly make small adjustments. This jittering wastes power, creates noise, and puts unnecessary stress on the servo gears

To solve this, We used a servo detachment function that stops sending PWM signals to servos when they don't need to actively hold position, We kept the gripper servo powered when holding objects, as detaching it would cause the gripper to lose its gripping force and potentially drop the object.

III.5.5 Sensing Systems (Ultrasonic Sensor)

Distance measurement is performed by an HC-SR04 ultrasonic sensor, crucial for stopping the robot at appropriate distances from objects for pickup and bins for drop-off. To improve the reliability of distance readings and reduce the impact of spurious measurements or noise, multiple readings are taken, filtered, and then averaged.

Here is a snippet code of the distance measurement function from our implementation:

```
float getDistance() {
 // Take multiple measurements for better reliability
 const int numReadings = 5;
 float readings[numReadings];
 int validReadings = 0;
 float sum = 0;

 // Perform multiple readings
 for (int i = 0; i < numReadings; i++) {
   // Trigger ultrasonic pulse
   digitalWrite(TRIG_PIN, LOW);
   delayMicroseconds(2);
   digitalWrite(TRIG_PIN, HIGH);
   delayMicroseconds(10);
   digitalWrite(TRIG_PIN, LOW);

   // Measure time for echo
   long duration = pulseIn(ECHO_PIN, HIGH, 20000); // 20ms timeout

   // Calculate distance if reading is valid
   if (duration > 0) {
     float distance = duration * 0.034 / 2;  // Speed of sound wave divided by 2

     // Filter out unreasonable values
     if (distance <= 400.0 && distance >= 2.0) {
        readings[validReadings] = distance;
        sum += distance;
        validReadings++;
     }
   }
   delay(20);  // Small delay between readings
 }
 // ... average calculation and calibration code ...
}
```

Key features of this implementation:

- Takes 5 readings instead of just 1 to reduce noise and outliers
- Filters out impossible readings (too close or too far)
- Applies a small calibration offset when approaching bins (we found the robot was stopping a bit too far from bins)

We defined two critical distance thresholds:

- `#define OBJECT_STOP_DISTANCE 18.0 // cm, stop distance for object pickup`
- `#define BIN_STOP_DISTANCE 11.0 // cm, stop distance for bin dropoff`

III.5.6 Communication Interfaces and Protocol

The Arduino Mega manages serial communication on two primary hardware serial ports:

- **Serial1 (Pins 19 RX, 18 TX on Mega):** Connected to an HC-05 or HC-06 Bluetooth module. This channel is used for communication with the Android application, typically at a baud rate of 9600.
- **Serial3 (Pins 15 RX, 14 TX on Mega):** Used for UART communication with the Raspberry Pi. This connection operates at a higher baud rate 115200 to facilitate faster exchange of commands, status updates, and sensor data.

Here is a snippet code of the serial communication initialization and command processing from our implementation:

```
void setup() {
 // Initialize serial communications
 Serial.begin(115200);   // Debug serial
 Serial1.begin(9600);    // Bluetooth serial
 Serial3.begin(115200);  // UART to Raspberry Pi
 // ... other initialization code ...
}
// Process commands from the Raspberry Pi
void processRaspberryPiCommand(String command) {
 if (command.startsWith("TURN:")) {
   // Format: TURN:angle (positive = right, negative = left)
   int angle = command.substring(5).toInt();
   executeTurn(angle);
   Serial3.println("TURN_COMPLETE");
   // Update state after turn completion if needed
   if (currentState == ALIGN_ROUGH || currentState == ALIGN_FINE) {
    // ... state transition code ...
    currentState = APPROACH;
    moveForward(DEFAULT_SPEED);
    Serial3.println("APPROACH_STARTED");
   }
 }
 else if (command == "APPROACH") {
   currentState = APPROACH;
   moveForward(DEFAULT_SPEED);
   Serial3.println("APPROACH_STARTED");
 }
 else if (command == "PICKUP") {
   performPickup();
 }
 else if (command == "DROPOFF") {
   performDropoff();
 }
 // ... additional commands ...
}
```

The command protocol between the Raspberry Pi (master) and Arduino (slave) is text-based for simplicity and ease of debugging.

Commands from Raspberry Pi to Arduino (Examples):

Command Type	**Examples**	**Purpose**
Movement commands	"TURN:45", "APPROACH", "STOP"	Control robot position and movement
Arm action commands	"PICKUP", "DROPOFF", "SCAN"	Trigger complex arm movement sequences
Status requests	"DISTANCE"	Request sensor data

***Table** III.3 Commands from Raspberry Pi to Arduino*

Responses/Acknowledgements from Arduino to Raspberry Pi (Examples):

Response Type	**Example Response String**	**Purpose**
Action Completion	TURN_COMPLETE, PICKUP_COMPLETE, APPROACH_TARGET_REACHED	Confirm successful completion of a commanded physical action.
Data Responses	DISTANCE:18	Provide requested sensor data
Ready/Acknowledgement	READY	Arduino signals it's ready for a new command.

***Table** III.4 Responses/Acknowledgements from Arduino to Raspberry Pi*

This protocol lets the Raspberry Pi send high-level commands, while the Arduino out the physical actions and sends back sensor data or status updates. It's important that both sides handle these command messages consistently.

III.5.7 Arduino State Machine

Just as the Raspberry Pi implements a high-level state machine for perception and decision-making, the Arduino runs a complementary state machine focused on physical actuation and hardware control. This state machine ensures that motor movements, sensor readings, and arm actions are executed in the correct sequence.

Here is a snippet code of the Arduino state machine definition:

```
// Operating modes
enum Mode {
  MANUAL,  // Manual control via app
  AUTO     // Autonomous operation
};

// Current target type
enum Target {
  OBJECT,      // Waste object
  BIN_TARGET   // Recycling bin
};

// State machine states
enum RobotState {
  IDLE,         // Initial state
  SEARCH,       // Look for target
  ALIGN_ROUGH,  // Rough alignment
  ALIGN_FINE,   // Fine alignment
  APPROACH,     // Move toward target
  ACTION,       // Pickup or dropoff action
  // ... additional states ...
};
```

The state machine execution is managed in the *executeAutoMode()* function, which is called repeatedly in the Arduino’s main loop when in autonomous mode:

```
// Main state machine logic (simplified)
void executeAutoMode() {
 switch (currentState) {
  case SEARCH:
   // Wait for commands from RPi
   waitingForRPiResponse = true;
   break;

  case ALIGN:
   // Wait for turn commands from RPi
   waitingForRPiResponse = true;
   break;

  case APPROACH:
   // Actively approach until proper distance
   executeApproach();
   break;

  case ACTION:
   // Execute pickup or dropoff actions
   if (currentTarget == OBJECT && !actionCompleted) {
    performPickup();
    actionCompleted = true;
   }
   // ... additional action code ...
   break;

  // ... additional states ...
 }
}
```

During the approach state, the Arduino actively monitors the distance to the target and adjusts its speed accordingly:

```
void executeApproach() {
 float distance = getDistance();
 float stopDistance = (currentTarget == OBJECT) ?
           OBJECT_STOP_DISTANCE : BIN_STOP_DISTANCE;
 // Check if target reached
 if (distance <= stopDistance) {
   stopMotors();
   Serial3.println("TARGET_DISTANCE_REACHED");
   currentState = ACTION;
 } else {
   // Continue approach
   // ... speed adjustment code ...
 }
}
```

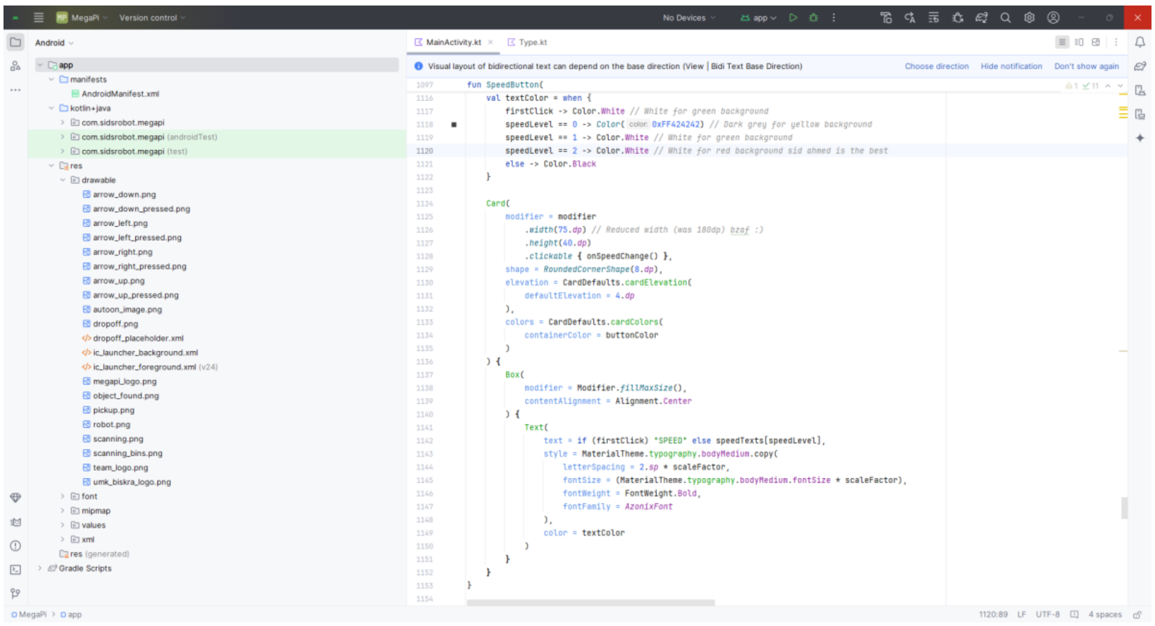
The Arduino state machine is designed to work alongside the Raspberry Pi state machine, with each handling its specific responsibilities. The coordination between these two state machines is what enables the robot's autonomous behavior to work.

III.6 User Interface Layer (Android app)

We built a simple but effective Android app using Kotlin and Jetpack Compose to control the robot remotely.

III.6.1 Development Environment: Android Studio

Android Studio, the official Integrated Development Environment (IDE) for Android app development provided by Google [67], was the chosen platform for this task. We utilized Kotlin as the primary programming language.



```
fun SpeedButton(
    val textColor = when {
        firstClick -> Color.White // White for green background
        speedLevel == 0 -> Color( color: 0xFF424242) // Dark grey for yellow background
        speedLevel == 1 -> Color.White // White for green background
        speedLevel == 2 -> Color.White // White for red background sid ahmed is the best
        else -> Color.Black
    }

    Card(
        modifier = modifier
            .width(75.dp) // Reduced width (was 180dp) bzaf :)
            .height(40.dp)
            .clickable { onSpeedChange() },
        shape = RoundedCornerShape(8.dp),
        elevation = CardDefaults.cardElevation(
            defaultElevation = 4.dp
        ),
        colors = CardDefaults.cardColors(
            containerColor = buttonColor
        )
    ) {
        Box(
            modifier = Modifier.fillMaxSize(),
            contentAlignment = Alignment.Center
        ) {
            Text(
                text = if (firstClick) "SPEED" else speedTexts[speedLevel],
                style = MaterialTheme.typography.bodyMedium.copy(
                    letterSpacing = 2.sp * scaleFactor,
                    fontSize = (MaterialTheme.typography.bodyMedium.fontSize * scaleFactor),
                    fontWeight = FontWeight.Bold,
                    fontFamily = AzonixFont
                ),
                color = textColor
            )
        }
    }
}
```

Figure III.19 Android app development in Android Studio

III.6.1.1 Application Architecture

The app establishes a Bluetooth Serial Port Profile (SPP) connection with the Arduino and provides two main modes:

1. **AUTO Mode**: Start/stop autonomous operation.
2. **MANUAL Mode**: Direct control of robot chassis movement.

The control interface includes:

- feedback of connection status
- Mode selection buttons (Auto/Manual)
- In Manual mode: directional control buttons and speed selection
- Status display showing current robot state

III.6.2 Communication Protocol

The app communicates with the Arduino using a simple character-based protocol:

Commands sent to Arduino:

Command	**Action**
‘A’	Activate Auto Mode
‘O’	Enter Manual Mode
‘G’	Move Forward
‘H’	Move Backward
‘I’	Turn Left
‘J’	Turn Right
‘K’	Stop Movement
‘X/Y/Z’	Set Speed (Low/Med/High)

Table III.5 Commands sent from the android app to Arduino

Status messages received from Arduino:

Message String from Arduino	Meaning Displayed in App
“SEARCHING?”	Looking for objects
“OBJECT_FOUND?”	Object detected and aligning
“APPROACHING?”	Moving toward object
“GRABBING?”	Picking up object
“BIN_SEARCH?”	Looking for correct bin
“BIN_FOUND?”	Bin detected and aligning
“DROPPING?”	Depositing object in bin
“COMPLETED?”	Task complete

Table III.6 Status messages received from Arduino to the app

The question mark at the end of each status message is a simple acknowledgment protocol - the app removes it when displaying to the user.

III.7 System Integration

The successful operation of the autonomous waste management robot hinges on the seamless integration and coordinated communication between its three distinct software components: the Raspberry Pi's vision and decision-making software, the Arduino's low-level hardware control firmware, and the Android application's user interface.

A typical autonomous waste sorting cycle unfolds as follows:

1. **Initialization & Connection:**
   - ✓ The robot is powered on. The Raspberry Pi and Arduino boot up their programs.
   - ✓ The user launches the Android application and establishes a Bluetooth connection with the Arduino.
2. **Mode Selection (User Initiated):**
   - ✓ The user selects "**AUTO Mode**" on the Android app. The app sends the 'A' command to the Arduino.

- ✓ The Arduino receives 'A', switches its internal currentMode to AUTO_MODE, stops any manual movements

3. **Autonomous Operation (Pi and Arduino Coordination):**

- ✓ **Object Search:** enters its SEARCHING_OBJECT state. Its video processing thread captures frames, and the YOLO model attempts to detect waste objects.
- ✓ **Object Detection & Alignment:** Upon detecting an object, the Pi calculates the object's center and the alignment angle needed. It transitions to ALIGNING_TO_OBJECT and sends a TURN:<angle> command to the Arduino.
- ✓ The Arduino receives TURN:<angle>, executes the turn using its DC motor control, and upon completion, sends TURN_COMPLETE back to the Pi.
- ✓ **Approach:** The Pi, confirming alignment (angle is near zero or turn is complete), transitions to APPROACHING_OBJECT and sends an APPROACH command to the Arduino.
- ✓ The Arduino moves the robot forward (moveForward()) while continuously checking the distance using getDistance(). When the distance is less than or equal to OBJECT_STOP_DISTANCE, it stops the motors (stopMotors()) and sends APPROACH_TARGET_REACHED to the Pi.
- ✓ **Pickup:** The Pi receives APPROACH_TARGET_REACHED, transitions to WAITING_FOR_PICKUP_COMPLETION, and sends a PICKUP command to the Arduino.
- ✓ The Arduino executes its pre-defined moveArmToPickupSequence(), manipulating servos to grasp the object. Upon successful completion, it sends PICKUP_COMPLETE to the Pi.
- ✓ **Bin Search & Navigation:** The Pi receives PICKUP_COMPLETE. Based on the detected object's class (e.g., plastic, class_id 1), it uses the OBJECT_TO_BIN_MAP to determine the target bin's ArUco ID (e.g., ID 0 for plastic). It transitions to SEARCHING_BIN. To search for the targeted aruco marker.

- ✓ **Bin Detection & Alignment:** When the Pi's vision system detects the correct ArUco marker for the target bin, it calculates the alignment angle and commands the Arduino to turn, similar to object alignment.
- ✓ **Approach Bin:** Once aligned with the bin, the Pi commands the Arduino to approach. The Arduino moves forward, stopping at BIN_STOP_DISTANCE and reporting APPROACH_TARGET_REACHED.
- ✓ **Drop-off:** The Pi sends a DROPOFF command. The Arduino executes its moveArmToDropoffSequence() to release the object into the bin and sends DROPOFF_COMPLETE to the Pi.

4. **User Monitoring & Intervention:**

- ✓ Throughout this autonomous cycle, the Arduino sends status updates (e.g., "Status: Looking for trash," "Status: Searching for bins") to the Android app, allowing the user to monitor progress.
- ✓ The user can typically interrupt the autonomous mode at any time by selecting "MANUAL Mode" on the app, which would send an 'O' command to the Arduino, causing it to stop autonomous actions and revert to manual control.

III.8 Conclusion

The software architecture of our robot integrates carefully selected tools, platforms, and custom code to work together smoothly. We began with design work in SolidWorks and brought those designs to life through Creality Slicer. For the brain of our system, we used a Raspberry Pi running Python, OpenCV, and our trained YOLOv8n model to enable smart decision-making.

For controlling the hardware, we developed firmware using Arduino IDE, which allowed precise control of motors and sensors in real-time. We also created a user-friendly mobile application with Android Studio so operators could control the robot remotely and check its status.

We divided responsibilities across the system: the Arduino handles basic control tasks, the Raspberry Pi manages complex vision and decision-making, and the Android app provides the user interface. These components communicate primarily through serial connections and Bluetooth, ensuring all parts work together effectively.

A key part of our development process was repeatedly testing and refining the state machines on both the Arduino and Raspberry Pi. This was essential for creating the robot's autonomous behaviors, especially our new targeting system that helps the robot align perpendicular to waste objects.

By bringing together these different software elements, we built a system that can identify, collect, and sort waste materials.

Chapter IV

Results, General Conclusion and Future Perspectives:

IV.1 Introduction

In this chapter, we present the results obtained during the development of our waste management robot. We outline the construction process, highlight key features and advantages of the final prototype, and offer insights for future improvements and enhancements.

IV.2 Hardware Prototype

IV.2.1 3D Printed Parts

We successfully completed the implementation of our waste management robot, achieving a fully operational system. The following section showcases the construction process and the practical realization of the design.

We began by individually 3D-printing all the required components for both the mobile base and the robotic arm. The prints were produced using a Creality Ender 3 Pro 3D printer and PLA filament.

IV.2.1.1 Chassis Components:

The images presented in this section document the printing process and verify the quality of the output

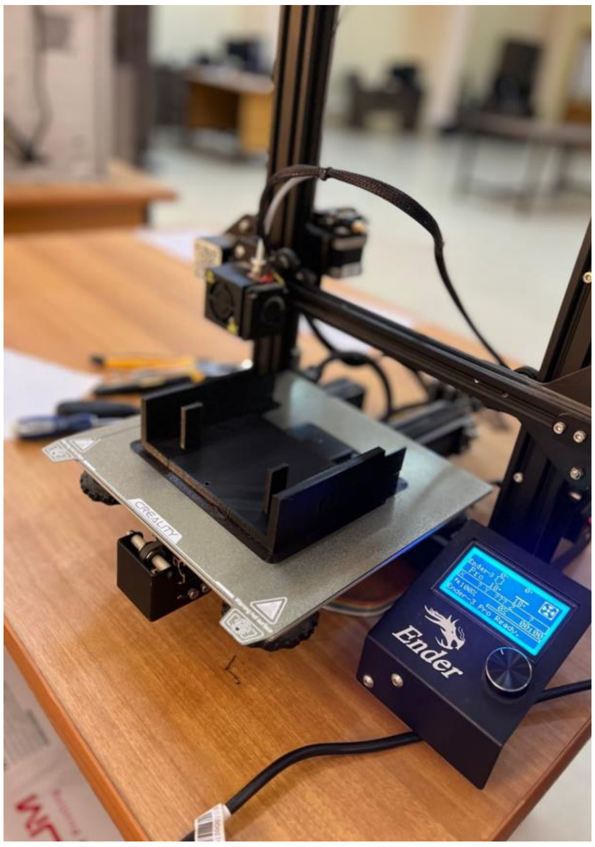


Figure IV.1 Ongoing 3D Printing of the Robot’s Chassis

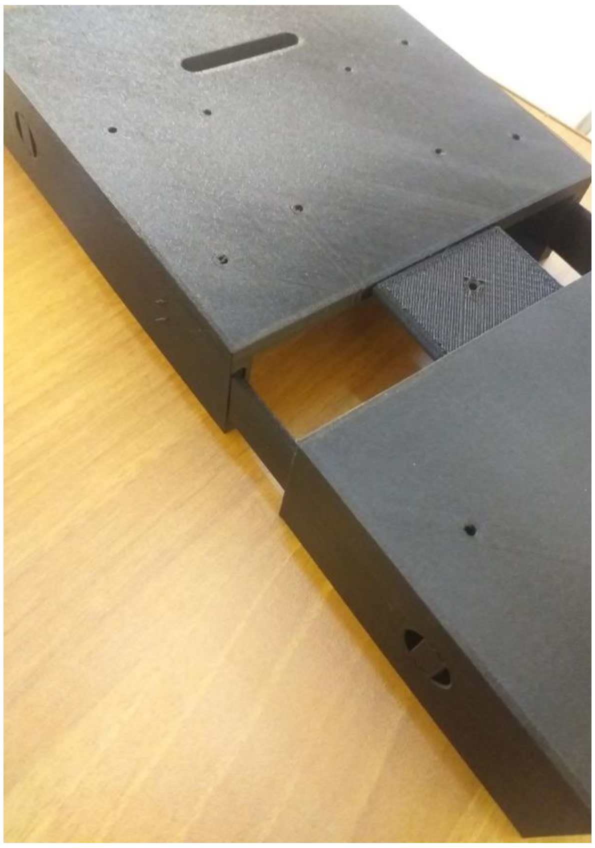


Figure IV.2 Completed 3D-Printed Robot Chassis

Following the printing phase, we assembled the mobile base by mounting all four wheels and TT DC motors onto the chassis. The mechanical integration was executed with precision, ensuring proper alignment and robust support.

The wheels were securely affixed to the motor shafts, enabling smooth and stable movement. Motor brackets were employed to ensure tight fitment and reduce vibration during operation. This stage marked a key milestone in the build process, transforming the static structure into a fully mobile platform.

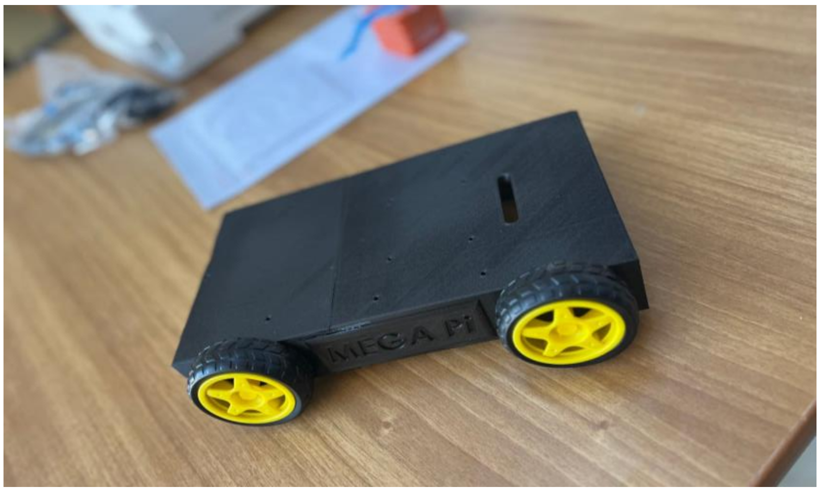


Figure IV.3 Assembled Robot Chassis with Wheels

IV.2.1.2 Arm Links and gripper

The images below illustrate the 3D-printing process of the robotic arm's individual components, which were printed to a high standard of quality, as anticipated.

***Figure** IV.4 Ongoing 3D Printing of the Arm's Base*

***Figure** IV.5 Ongoing 3D Printing of the Arm's gripper*

After printing, we proceeded with the sub-assembly of each component by attaching the corresponding servo motors. Careful attention was paid to the orientation and mechanical fit of each servo, as correct alignment at this stage is critical to achieving smooth articulation and functional integrity.

Once all parts were correctly assembled and mounted in their designated positions, the result was a well-integrated, fully articulated robotic arm. The final product delivered robust mechanical stability and met the functional requirements set out for this phase of the project.

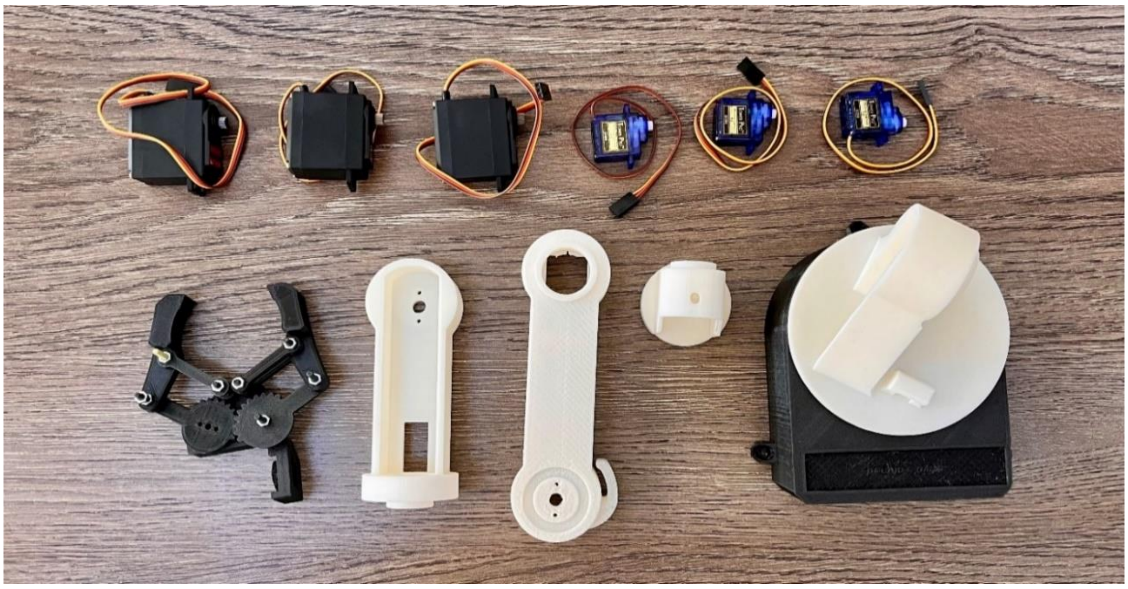


***Figure** IV.6 Completed 3D-Printed robotic arm parts*

After assembling the parts and placing each component in its correct position, we obtained a good result: a fully integrated robotic arm with solid quality that allows it to perform the required function effectively**[19].**

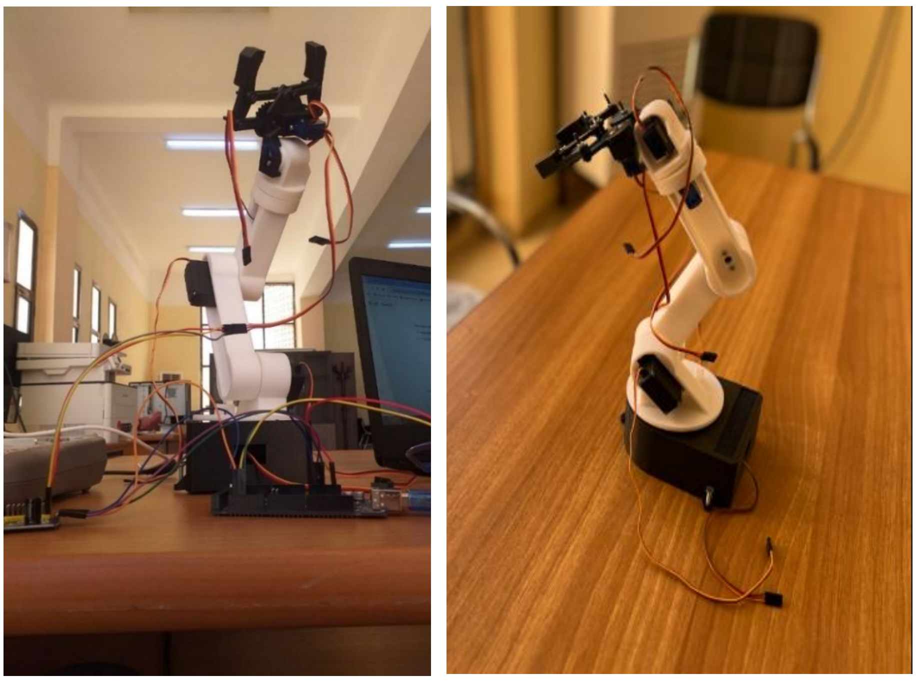


Figure IV.7 Assembled Robot Arm

IV.2.2 Electronics & Wiring

This section outlines the wiring methodology and power management strategy employed to integrate the system's electronic components.

Two power sources were used. The first is a 7.4V 3000mAh LiPo battery dedicated exclusively to powering the Raspberry Pi. Prior to connection, a voltage regulator was added to step the voltage down to 5V, the safe input level for the Raspberry Pi. A power switch was also installed to allow manual activation and deactivation of this circuit, The Raspberry Pi Camera Module is directly connected to the Raspberry Pi via the CSI (Camera Serial Interface) port.

The second power source consists of three 3.7V 2800mAh lithium cells connected in series, producing a total of 11.1V. A power switch was similarly placed along the positive terminal to control power delivery.

From this 11.1V source, both the positive and negative lines were connected to a voltage regulator that reduced the output to 6.1V. This regulated power was then used to supply the PCA9685 module, which controls and powers the servo motors.

In parallel, two lines were branched off before the 6.1V regulator to deliver the full 11.1V directly to the L293D motor driver shield. This module, in turn, powers the Arduino Mega

2560 and drives the DC motors. The ultrasonic sensor and Bluetooth module are also directly connected to and powered by the Arduino.

To ensure electrical stability and reliable operation, all GND (ground) lines were unified across the entire system.

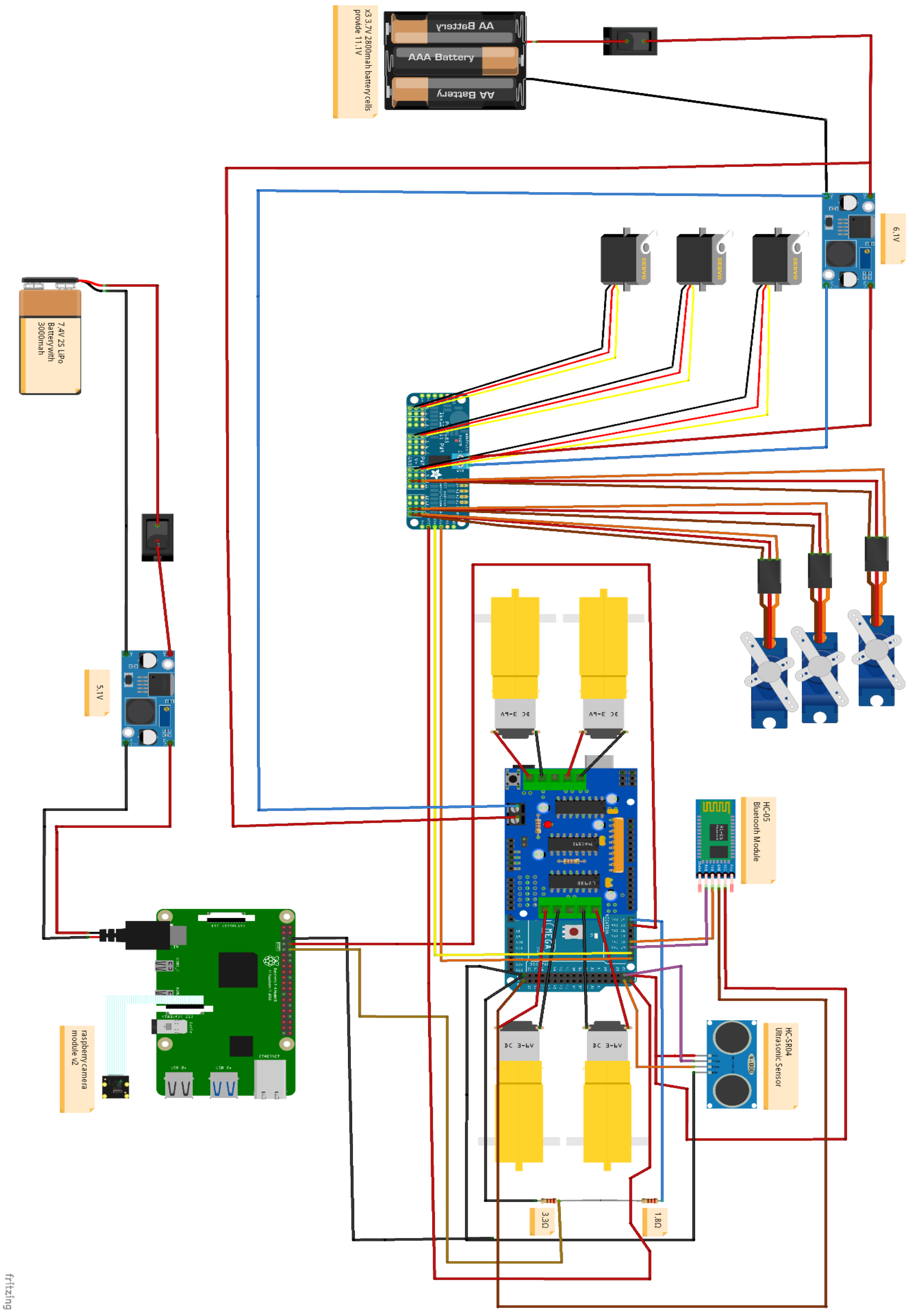

x3 3.7V 2800mah battery cells provide 11.1V
AA Battery
AAA Battery
AA Battery
6.1V
7.4V 2S LiPo Battery with 3000mah
5.1V
HC-05 Bluetooth Module
HC-SR04 Ultrasonic Sensor
raspberry camera module v2
3.3Ω
1.8Ω
fritzing

Figure IV.8 Circuit Diagram of the Robot's Complete Electronics System

The image below shows the actual wiring setup implemented on the physical robot. It reflects the schematic previously presented

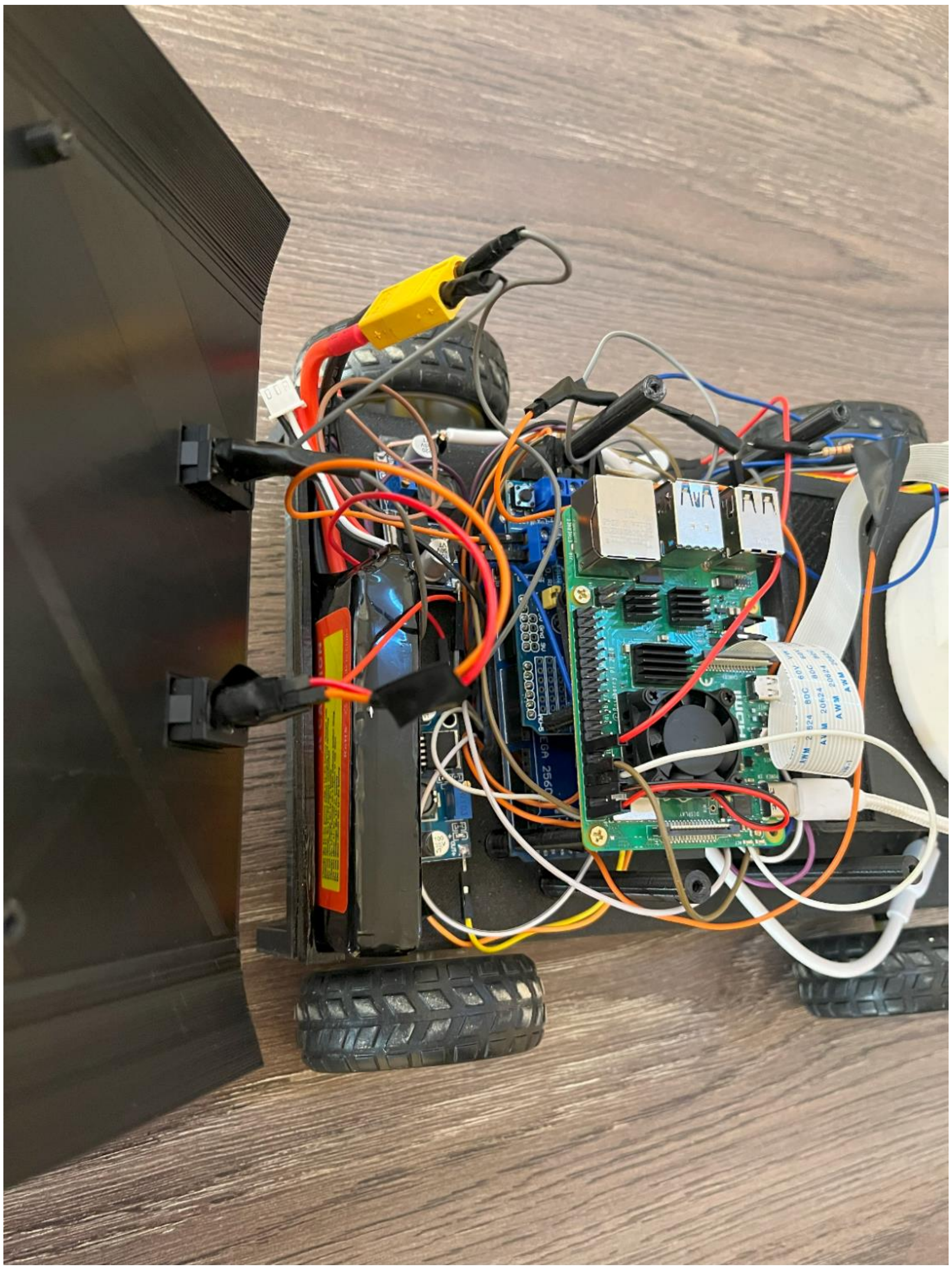
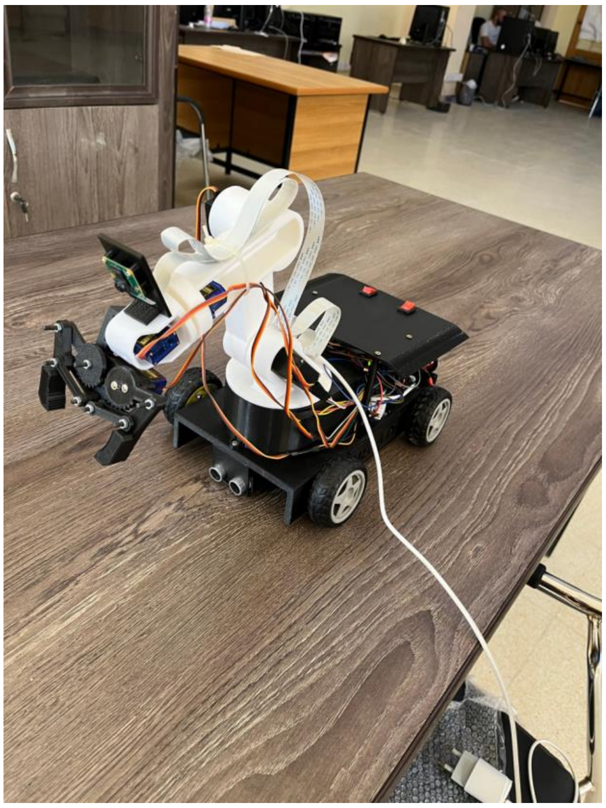


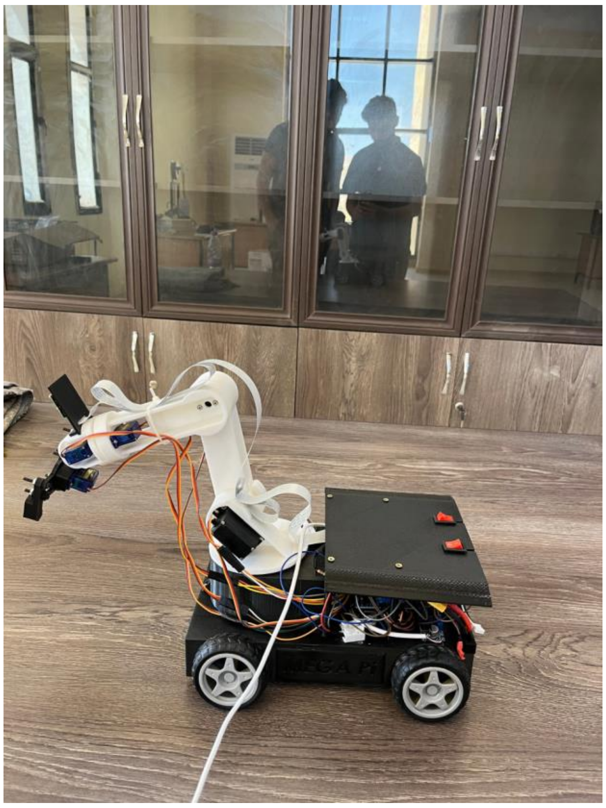
Figure *IV.9 Assembled Robot with Fully Connected Electronics*

The successful integration of the robotic arm with the mobile base resulted in a fully functional robotic system. The custom 3D-printed chassis provides both durability and balance, and the four DC motors with mounted wheels offer smooth mobility. The six-degree-of-freedom robotic arm is solidly attached and each servo motor has been installed with accuracy to allow precise movement.

The electronics have been carefully wired and organized to ensure consistent power distribution and clear communication among the modules. The resulting system is compact and efficient, capable of navigating autonomously while manipulating objects in its environment.

The following images display the final robot from various angles, showcasing the complete structure, the seamless integration between the chassis and the robotic arm, and the overall quality of assembly. These perspectives provide a comprehensive view of how the components fit together and demonstrate the effectiveness of the design.





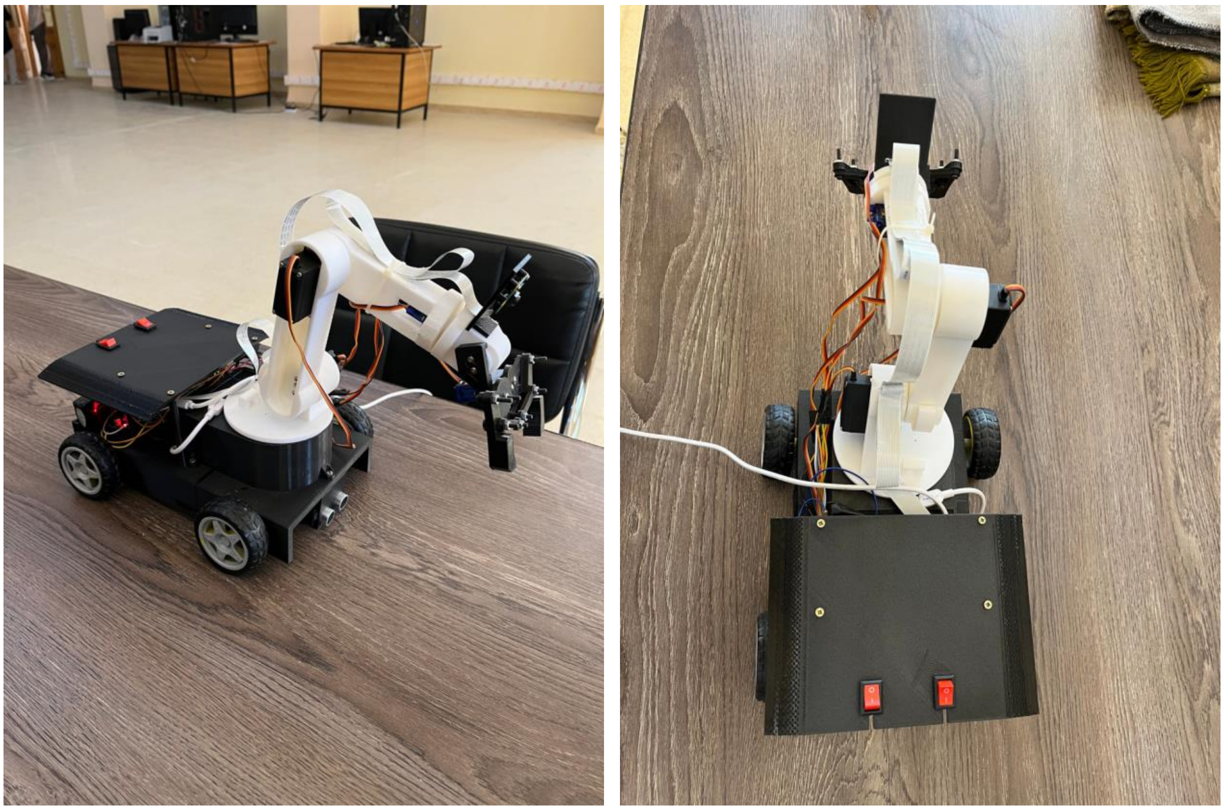


***Figure** IV.10 Completed Robot Prototype*

IV.3 Object Detection (YOLOv8n)

The YOLOv8n model was trained over 50 epochs to detect and classify three primary waste categories **metal, plastic, and paper**. This simplified classification scheme reflects common sorting practices in recycling facilities.

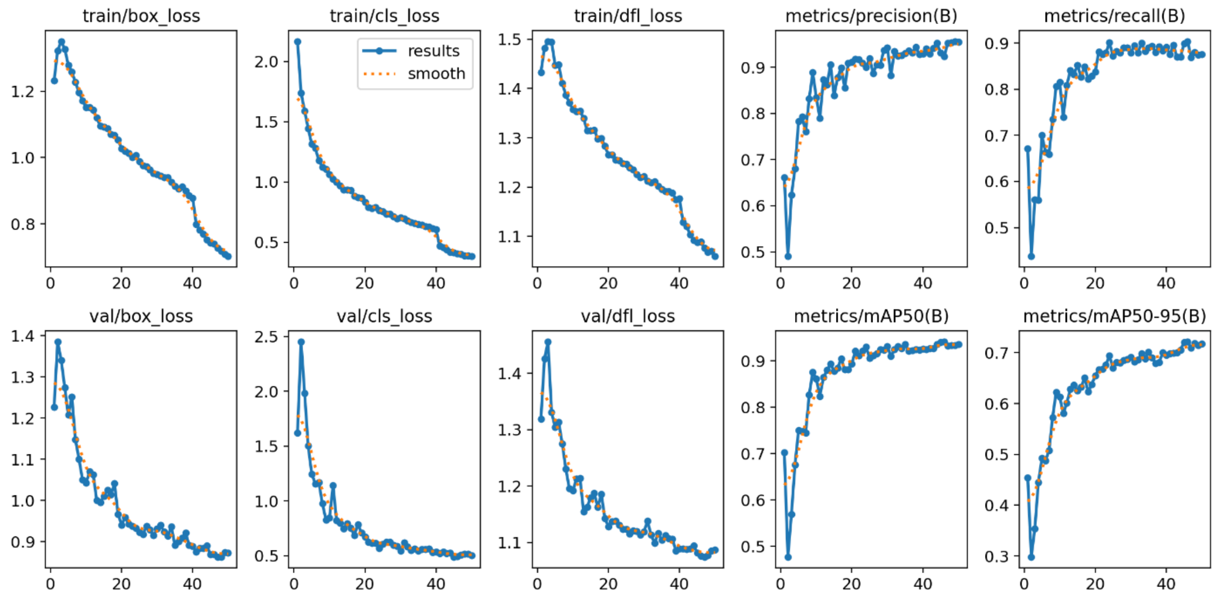


***Figure** IV.11: key performance indicators for the YOLOv8n waste classification model.*

Training metrics showed steadily decreasing losses, confirming model convergence with no signs of overfitting. Validation loss trends closely mirrored the training curves, supporting strong generalization to unseen data.

By the end of training, the model achieved:

- **Precision**: 0.95
- **Recall**: 0.90
- **mAP@50**: 0.941
- **mAP@50-95**: 0.72

These results reflect a robust detection capability across all waste categories, especially given the lightweight architecture of YOLOv8n.

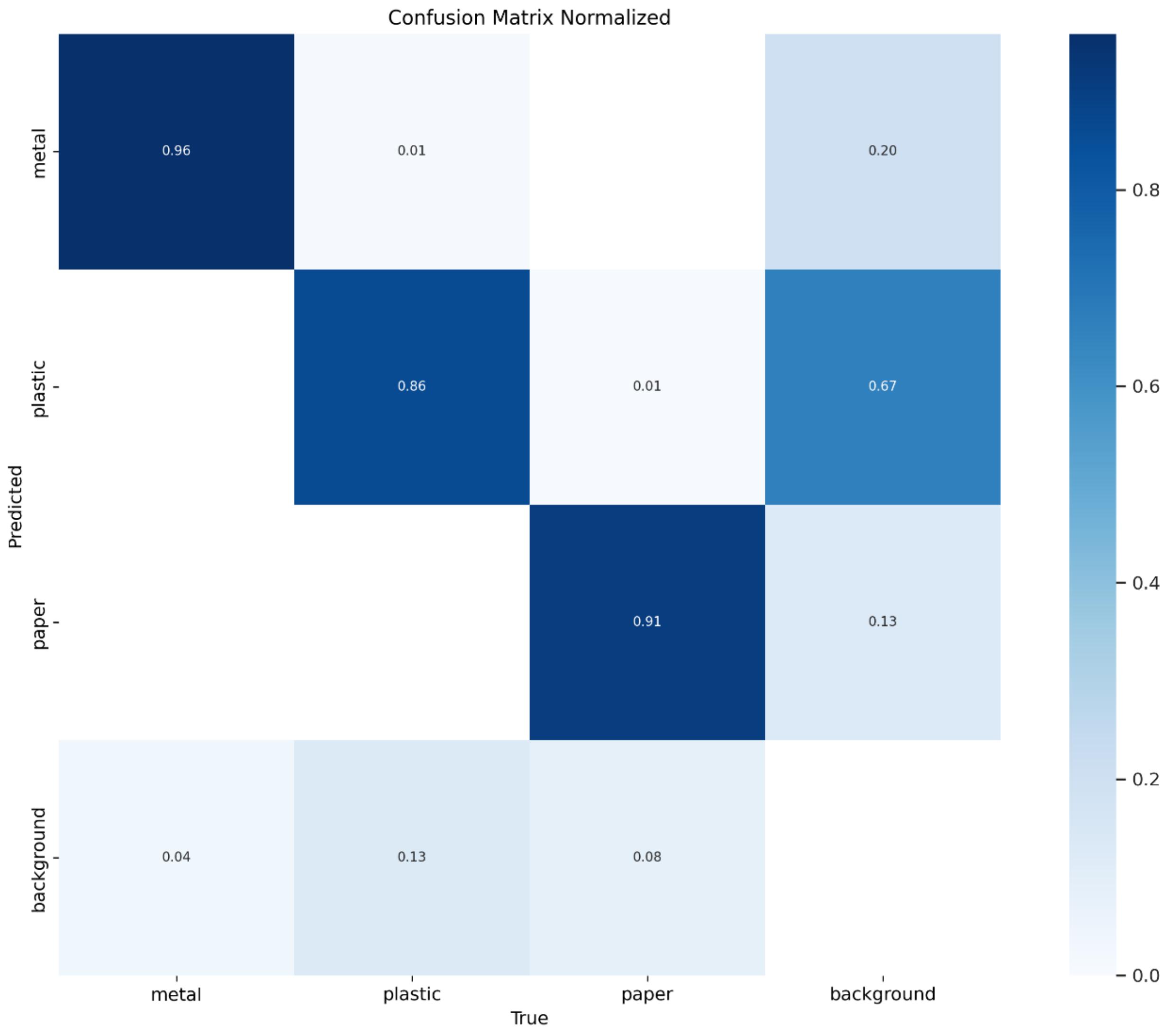


***Figure** IV.12: the classification performance across waste categories*

The normalized confusion matrix in Figure 5.2 provides deeper insights into the model's classification accuracy for each waste category:

- **Metal**: Exhibits the highest classification accuracy at 0.96, with minimal confusion with other categories. This suggests that metal waste items have distinctive visual characteristics that the model can reliably identify.
- **Plastic**: Shows strong performance with an accuracy of 0.86, though there is some confusion with the background class (0.13). This is likely due to the transparent or semi-transparent nature of some plastic items, which can visually blend with background elements.
- **Paper**: Achieves an accuracy of 0.91, with minimal confusion with other waste types. However, there is some misclassification (0.08) with the background class, possibly due to flat paper items that may not always present distinctive three-dimensional features.
- **Background**: The background class shows some interesting patterns of confusion, with the most significant being its misclassification as plastic (0.67) and metal (0.20). This suggests that some background elements contain visual features like these waste materials, presenting a challenge for the model.

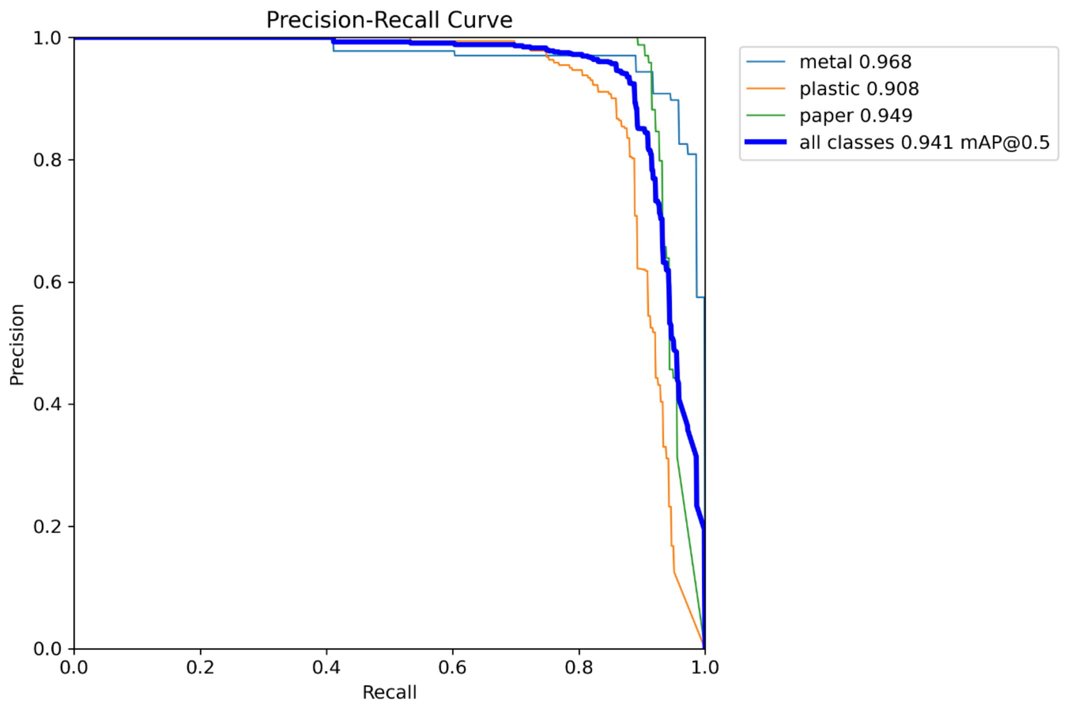


Figure IV.13: Precision-Recall curve for the waste classification model

The Precision-Recall (PR) curve (Figure 5.3) further illustrates model behavior at different confidence thresholds. Metal achieved the strongest precision-recall balance (0.968), followed by paper (0.949) and plastic (0.908). All classes maintained precision >0.9 until recall exceeded ~0.8, demonstrating that the model reliably detects waste items with minimal false positives.

Overall, the YOLOv8n model delivered excellent detection and classification performance for waste sorting, achieving an mAP@50 of 0.941.

IV.3.1 Visual Output and Detection Samples

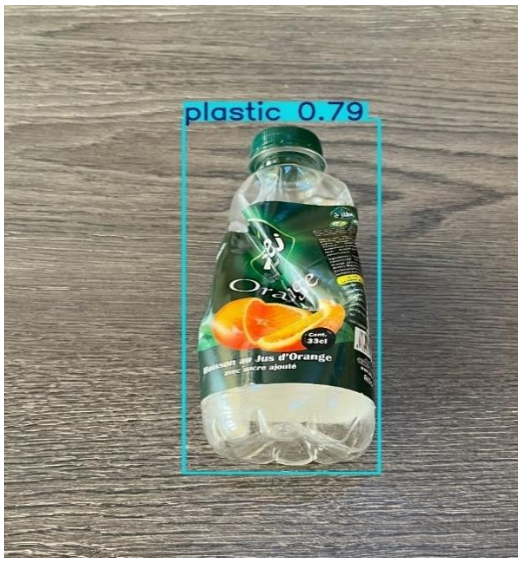


Figure IV.14 Detected Plastic Bottle – Confidence: 0.79

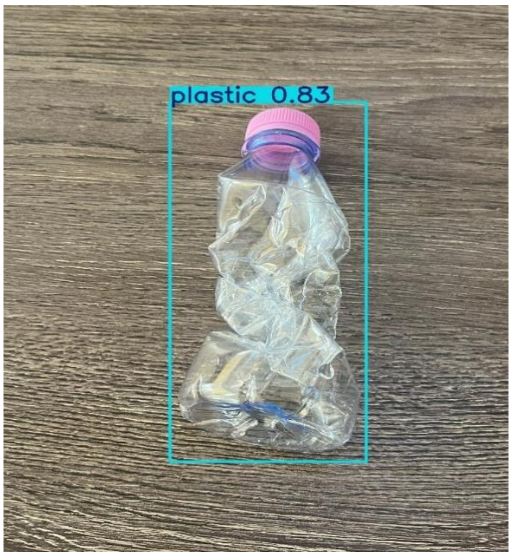


Figure IV.15 Detected Crushed Plastic Bottle – Confidence: 0.83

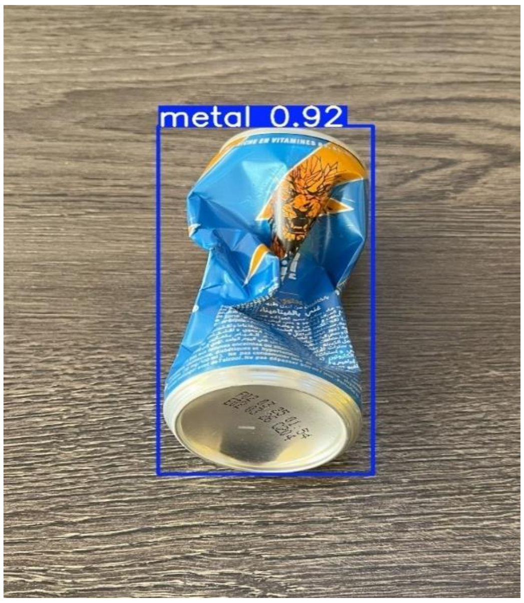


Figure IV.16 Detected Crushed Metal Can – Confidence: 0.92

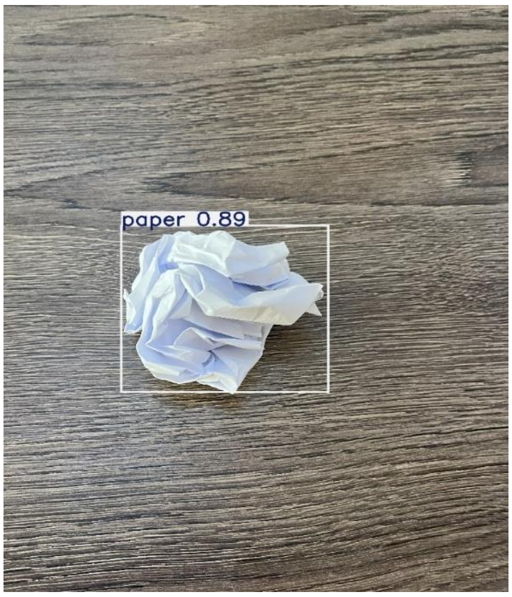
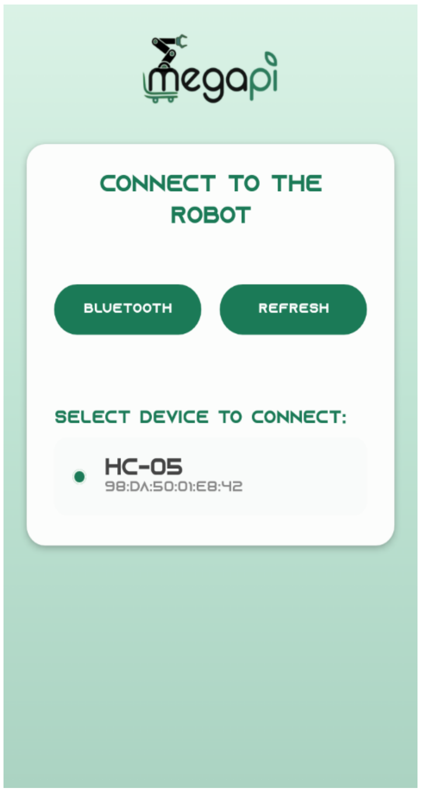
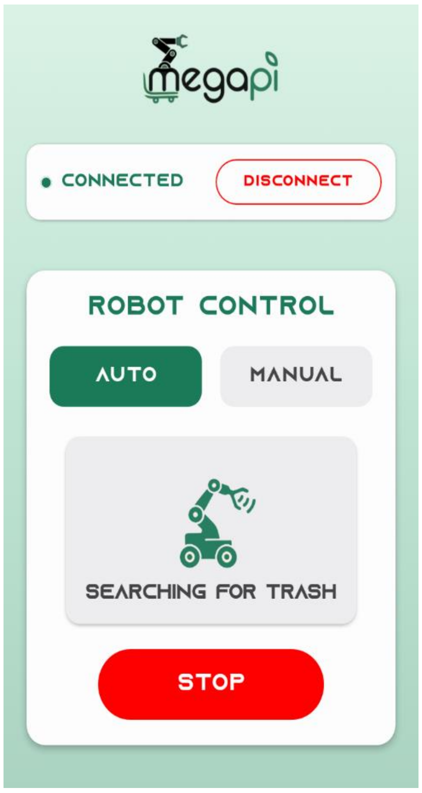


Figure IV.17 Detected Crumpled Paper – Confidence: 0.89

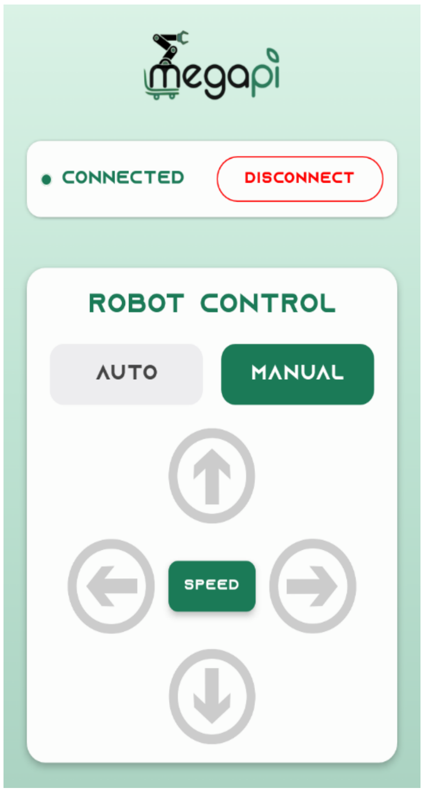
IV.4 Android Application



(a) Bluetooth Connection Interface



(b) Auto Mode Interface



(c) Manual Mode Interface

***Figure** IV.18 User Interface of the Robot Control Application*

The Figure above shows the user interface of the robot control android application:

a. **Connection Interface:** This is the initial screen where users can connect the application to the robot via Bluetooth. The user selects the available device (e.g., HC-05 module), Once connected, the Mode Selection interface appears.

b. **Auto Mode interface:** When **Auto** mode is selected and started, the robot begins operating autonomously. The interface shows feedback from the robot's operation in real time.
A **STOP** button is available to halt the operation.

c. **Manual Mode interface:** When **Manual** mode is selected, directional control buttons appear, allowing the user to move the robot manually.
A **SPEED** switch button provides additional control options for adjusting movement speed (LOW,MEDIUM,HIGH).

IV.5 Robot Operation Demonstration

The following sequence of images illustrates the robot successfully performing its core recycling task. The robot is shown detecting, picking up a plastic bottle, and correctly placing it into the designated plastic recycling bin. This demonstrates the effective integration of object detection, path planning, and actuation subsystems developed during the project.

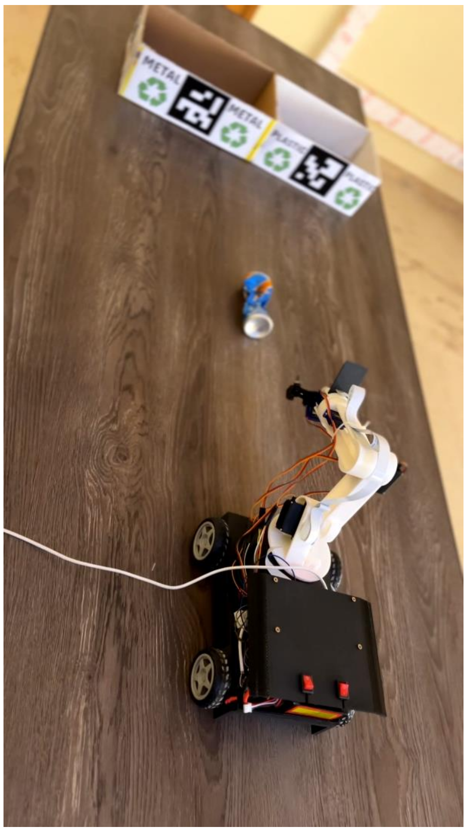
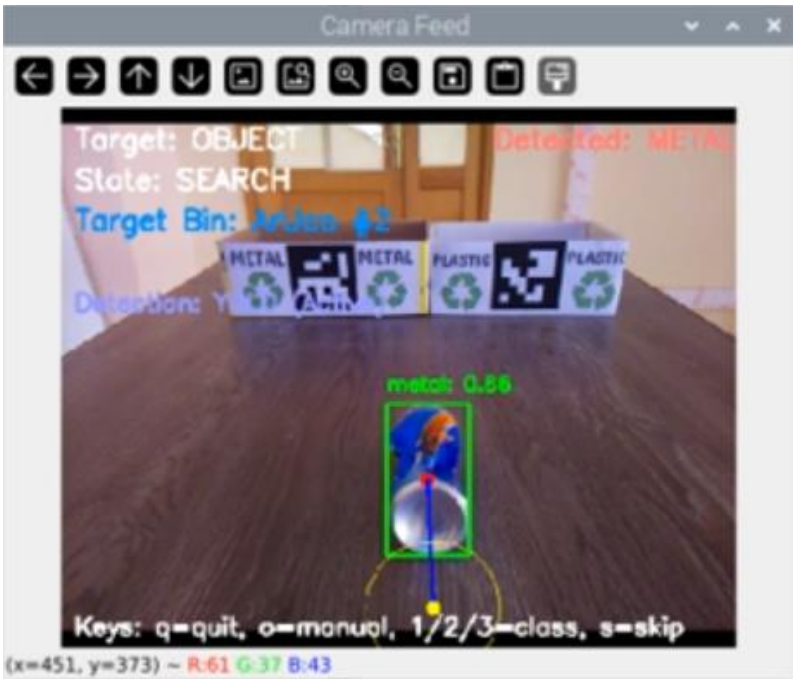


Figure IV.19 Robot Detecting a Metal Can



Target: OBJECT
State: SEARCH
Target Bin: ArUco #2
Detected: METAL
Detection: Y
metal: 0.86
Keys: q=quit, o=manual, 1/2/3=class, s=skip
(x=451, y=373) ~ R:61 G:37 B:43

Figure IV.20 Camera Feed Showing Detection of Metal Can

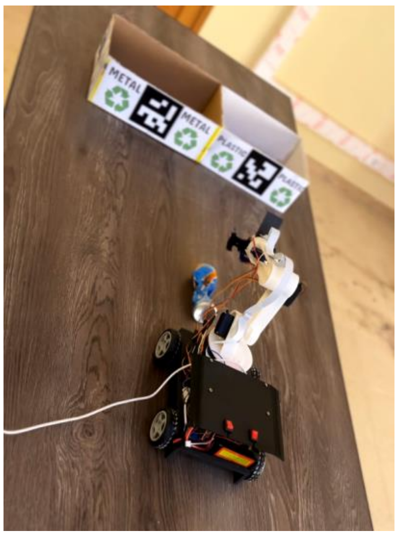


Figure IV.21 Robot Stopped After Approaching the Metal Can

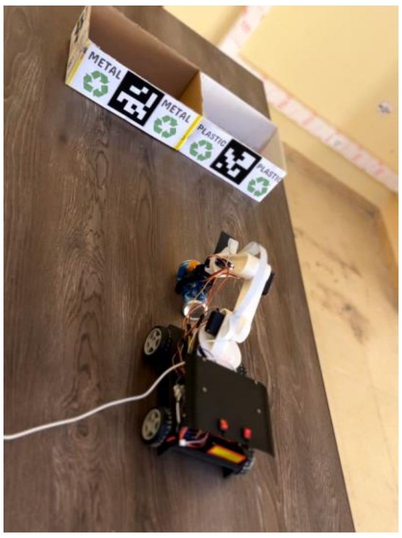


Figure IV.22 Robot performing Pick-Up Position

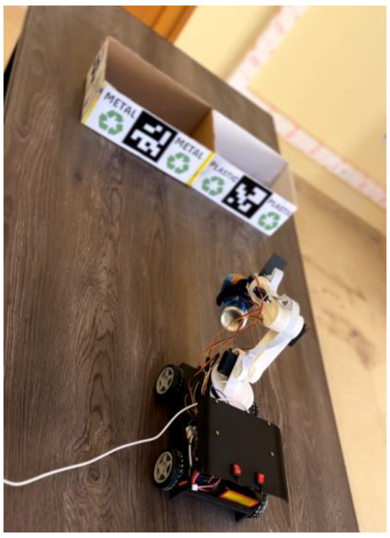


Figure IV.23 Robot Picking Up the Metal Can

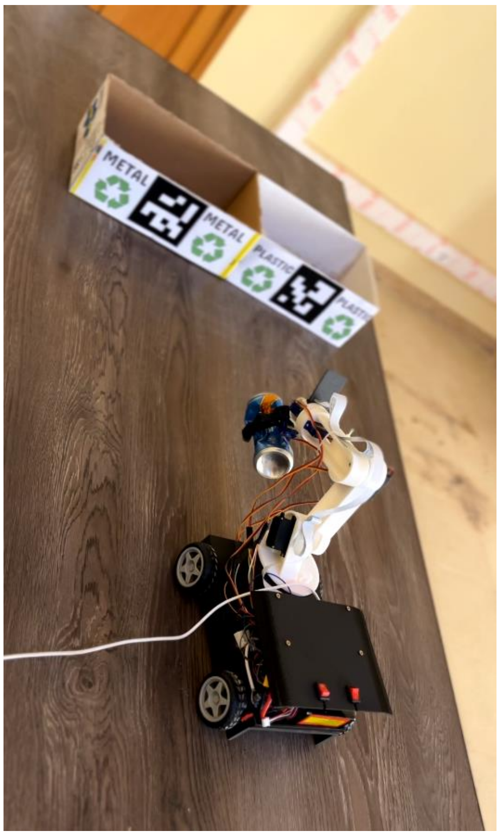
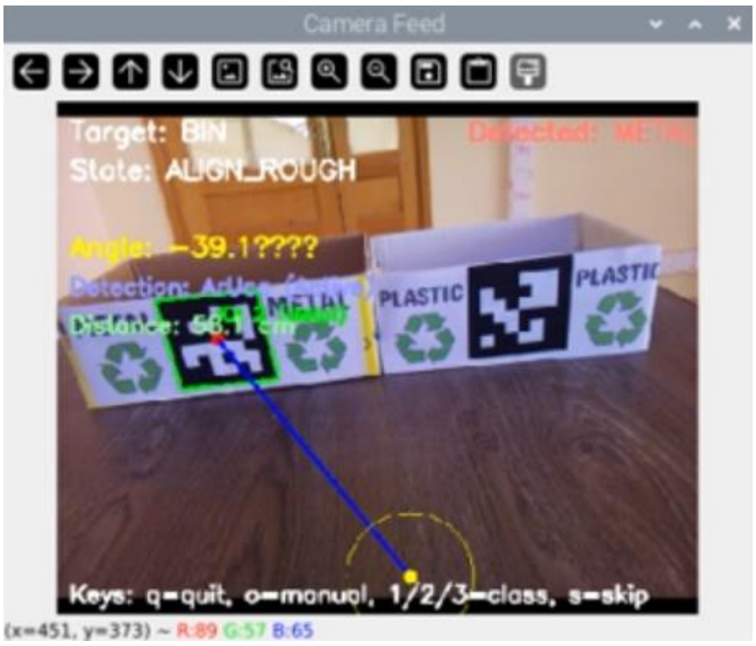


Figure IV.24 Robot Detecting Bin Using ArUco Marker



```
Target: BIN                    Detected: METAL
State: ALIGN_ROUGH

Angle: -39.1????
Detection: ArUco
Distance: 68.1 cm

Keys: q=quit, o=manual, 1/2/3=class, s=skip
(x=451, y=373) ~ R:89 G:57 B:65
```

Figure IV.25 Camera Feed Showing Detection of Bin via ArUco Marker

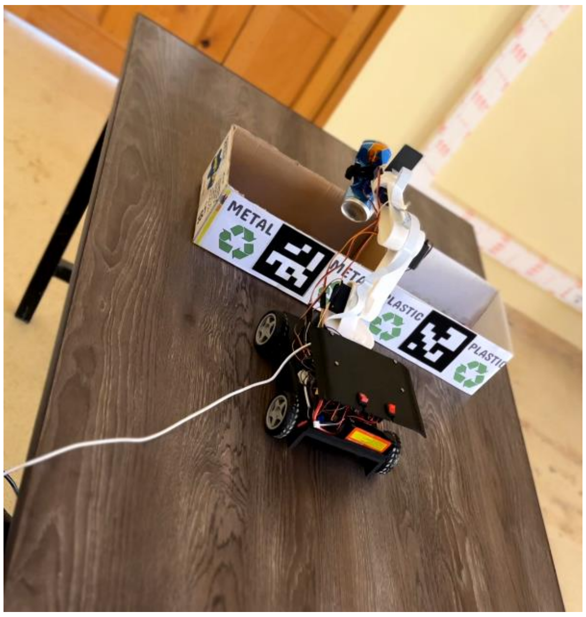


Figure IV.26 Robot Stopped After Approaching the Bin

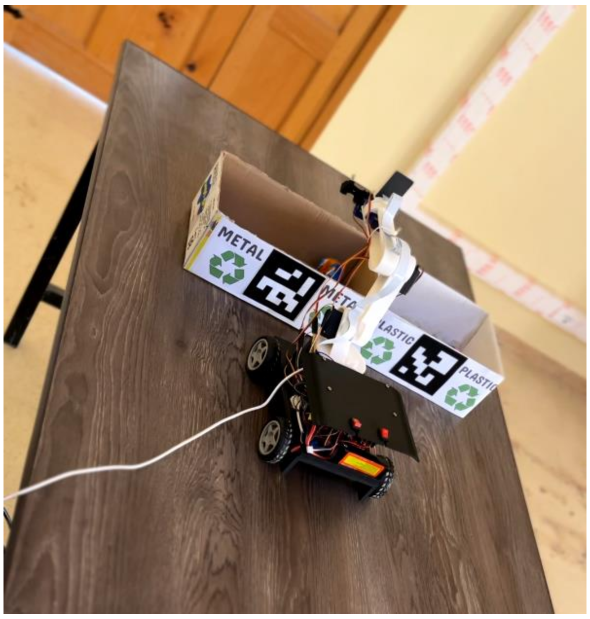


Figure IV.27 Robot Dropping the Metal Can into the Bin

Video Demonstration:

A video showcasing the working of the robot is available here:

https://youtu.be/VdJLIz-9X8A

Or by scanning the QR code:

IV.6 Limitations and Future Development

While the developed prototype demonstrates the core functionality, it presents several areas for enhancement to improve its efficiency and intelligence:

Limitations:

1. Robotic Arm and Manipulation

- The arm follows fixed movements using pre-set servo positions (e.g., scan, pick, drop), so it can't adjust if objects are in slightly different places or shapes.

2. Locomotion and Chassis Control

- The robot moves using TT DC motors without any feedback system. It turns using timers, which can be inaccurate if the floor is uneven or the battery is low.
- The wheels and their couplers and motors are not strong enough to support the full weight of the robot.

3. Environmental Perception and Sensing

- The ultrasonic sensor (HC-SR04) gives wrong distance readings sometimes, and is unreliable in complex or noisy environments.
- The camera (Raspberry Pi Camera V2) only sees in 2D, which limits depth understanding.
- The robot only finds bins if ArUco markers are clearly visible.
- The robot doesn't scan its surroundings, it only sees what's in front while moving.

4. Object Recognition Performance

- The object detector (YOLOv8n) is slow on the Raspberry Pi (about 1 frame per second).
- It sometimes mistakes objects, especially similar looking trash.

5. Software Architecture and Communication

- The system uses state machines split between the Raspberry Pi and Arduino, communicating through simple text messages. This setup can cause timing or sync issues.
- It uses an older Bluetooth module (HC-05), which has limited speed and range.

6. Power Management

- The robot uses two separate power sources that must be turned on and charged separately.
- There is no battery monitoring system to report power status to the software, so the robot doesn't know when it's running low on power.

7. User Interface

- The Android app only offers basic control with simple feedback.

<u>Future Improvements:</u>

1. Robotic Arm and Manipulation

- Add a 3D depth camera and use inverse kinematics to calculate arm movements in real time, so the robot can adjust its actions based on where the object really is.

2. Locomotion and Chassis Control

- Add encoders to the motors to track how far the robot moves and how much it turns. This will make movement more accurate.
- Upgrade to stronger motors and tougher wheels/couplers to better support the robot's weight.

3. Environnemental Perception and Sensing

- Replace the ultrasonic sensor with a more reliable sensor like LiDAR for accurate distance detection.

- Train the robot to find bins without needing visible markers.
- Make the robot actively scan its environment by rotating or moving a camera to build a simple map before making decisions.

4. Object Recognition and Performance

- To enhance inference speed, the object detector can be accelerated using a dedicated AI co-processor such as the Google Coral USB Accelerator. This would offload the model inference to an Edge TPU, potentially increasing performance from ~1 FPS to 10–30 FPS, enabling near real-time object detection on the Raspberry Pi. Alternatively, lighter models natively supported by the Edge TPU, such as MobileNet SSD, could be explored for faster deployment.
- Retrain the object detection model using more images of different objects to improve accuracy.

5. Software Architecture and Communication

- Move to a standard robotics software platform like ROS (Robot Operating System) for better communication, state management, and easier development.
- Upgrade to modern Bluetooth (like BLE 4.0 or 5.0) for faster and longer-range communication.

6. Power Management

- Create a single power system with one large battery.
- Add sensors (like INA219) to monitor battery level and send updates to the app so the robot can warn the user or shut down safely when power is low.

7. User Interface

- Improve the Android app with a better interface, including features like live video streaming and more detailed robot status

General conclusion

This work has presented the thorough design, development, and testing of an autonomous mobile robot equipped with a 5-DOF robotic arm and an advanced deep learning-based vision system. The project effectively integrated key disciplines including mechanical engineering, embedded control, and intelligent software algorithms to create a versatile robotic platform capable of navigating complex environments, recognizing objects with high accuracy, and performing precise manipulation tasks.

Using a systematic methodology, we tackled the key components of the autonomous mobile robot system. The initial step involved constructing the mechanical platform, which serves as the basis for the robot's movement and manipulation functions. Subsequently, control and navigation functionalities utilizing visual data were developed. The navigation system depends on visual input processed by a specially trained YOLOv8n model alongside ArUco markers, all running on a Raspberry Pi 4. A Pi Camera captures video streams, while OpenCV is employed for real-time image processing, feature detection, and delivering visual feedback to facilitate accurate object localization and manipulation. The robot's motors and servos are precisely controlled by an Arduino Mega, ensuring smooth motion and accurate positioning. Both manual and automatic control modes are supported through a custom-built Android application.

To validate the robot's performance, an automated waste sorting task was implemented, showcasing its effectiveness in real-world, unstructured settings. This application accomplished the robot's ability to accurately detect objects, execute precise grasping, and reliably place items. The execution of this task highlights the system's adaptability and robustness. These results emphasize the robot's potential for practical use in industrial automation and environmental management. Overall, the project confirms the feasibility of deploying autonomous mobile manipulators in complex, dynamic environments.

Several limitations were identified during the practical implementation. The robotic arm operates with fixed preset movements, restricting its ability to adapt to variations in object positioning. The locomotion system uses motors without feedback, resulting in navigation inaccuracies, and the chassis lacks sufficient strength to support the robot's full weight. Environmental sensing is constrained by a 2D camera, an unreliable ultrasonic sensor, and a reliance on visible ArUco markers without active environmental scanning. Object recognition is slow and prone to occasional misclassification. Communication between the Raspberry Pi and Arduino is rudimentary, which may lead to synchronization problems, further affected by the use of an outdated Bluetooth module. Power management relies on separate sources without battery monitoring, and the Android application provides only basic control functionality.

Moving forward with this project, planned enhancements include integrating a 3D depth camera and inverse kinematics to enable adaptive manipulation, incorporating motor encoders for improved navigation accuracy, and upgrading sensing capabilities with LiDAR technology. Additionally, the system will be trained for markerless bin detection to reduce reliance on visual markers. Further development efforts will focus on enhancing AI processing hardware, adopting advanced software frameworks such as ROS, implementing modern Bluetooth standards, streamlining power management into a unified system, and improving the user interface to boost overall robot performance and usability.

Bibliography

[1] Niku, S.B., 2020. Introduction to robotics: analysis, control, applications. John Wiley & Sons.

[2] « ROBOTICS Introduction to Robotics Robot Definition and Classification » TEMPUS IV Project: 158644 – JPCR Development of Regional Interdisciplinary Mechatronic Studies - DRIMS ROBOTICS.

[3] Mayor, Adrienne. (2018). Gods and Robots: Myths, Machines, and Ancient Dreams of Technology.

[4] Hill, Donald R. (1984). A History of Engineering in Classical and Medieval Times.

[5] Hill, Donald R. (1974). Studies in Medieval Islamic Technology: From Philo to Al-Jazari.

[6] Bedini, Silvio A. (1964). "The Role of Automata in the History of Technology." Technology and Culture.

[7] Gleick, James. (2011). The Information: A History, a Theory, a Flood.

[8] Riskin, Jessica. (2003). The Restless Clock: A History of the Centuries-Long Argument Over What Makes Living Things Tick.

[9] Berg, Maxine. (1980). The Machinery Question and the Making of Political Economy.

[10] Čapek, Karel. (1920). R.U.R. (Rossum's Universal Robots).

[11] Devol, George C. (1954). Patent: Programmed Article Transfer.

[12] McCarthy, John. (1955). Dartmouth Proposal for Artificial Intelligence.

[13] NASA. (1976). Viking Project Final Report.

[14] Brooks, Rodney A. (2002). Flesh and Machines: How Robots Will Change Us.

[15] Colgate, J. Edward et al. (1996). "The Cobotics Concept." Proceedings of the 1996 ASME International Congress and Exposition.

[16] Hanson Robotics. (2016). Sophia's Development Report.

[17] Intuitive Surgical Inc. (2020). Da Vinci System User Manual.

[18] Robotic automation systems (2022)

[19] Book « Robot Manipulators Position, Orientation and Coordinate Transformations »

[20] Website « https://www.researchgate.net/ »

[21] Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control (3rd ed.). Pearson

[22] Merlet, J.-P. (2006). Parallel Robots (2nd ed.). Springer.

[23] Gosselin, C., & Anglart, H. (2007). Design and Control of Hybrid Robots. Springer.

[24] Siciliano, B., & Khatib, O. (2016). Springer Handbook of Robotics (2nd ed.). Springer.

[25] Book « Industrial Automation and Robotics », by A.K. Gupta, Jean Riescher Westcott, Satish Kumar Arora, 2007.

[26] Msitec, image source: https://msitec.com/

[27] Science News, image source: https://www.sciencenews.org/

[28] ProLift, image source: https://www.prolift.ie/

[29] ABB, image source: https://new.abb.com/

[30] ISS National Lab, image source: https://issnationallab.org/

[31] Robotics and Automation News, image source: https://roboticsandautomationnews.com/

[32] Carnegie Mellon University, image source: http://www.cs.cmu.edu/

[33] ROS Mobile Robots, image source: https://ros-mobile-robots.com/

[34] RobotShop, image source: https://www.robotshop.com/

[35] CAD Crowd, image source: https://www.cadcrowd.com/

[36] Parvalux, image source: https://www.parvalux.com/

[37] Invia Robotics, image source: https://inviarobotics.com/

[38] Tower Pro, “SG90 9g Micro Servo,” DataSheet4U, [Online]. Available: https://datasheet4u.com/datasheet/Tower-Pro/SG90-791970 (Accessed Apr. 27, 2025).

[39] Tower Pro, “MG996R,” DataSheet4U, [Online]. Available: https://datasheet4u.com/datasheet/ETC/MG996R-942981 (Accessed Apr. 27, 2025).

[40] Adafruit Industries, “DC Gearbox Motor – TT Motor – 200RPM (3–6VDC),” Adafruit.com, [Online]. Available: https://www.adafruit.com/product/3777 (Accessed Apr. 27, 2025).

[41] SkyRC, “iMAX B6mini Charger,” skyrc.com, [Online]. Available: https://www.skyrc.com/iMAX_B6mini_Charger (Accessed Apr. 27, 2025).

[42] Instructables, “How to use DC to DC Buck Converter LM2596,” Instructables, Dec. 01, 2020. https://www.instructables.com/How-to-Use-DC-to-DC-Buck-Converter-LM2596/

[43] Raspberry Pi Foundation, "Raspberry Pi 4 Model B Datasheet," [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf. [Accessed: Apr. 27, 2025].

[44] Arduino.cc, “Mega 2560 Rev3 – Arduino Documentation,” [Online]. Available: https://docs.arduino.cc/hardware/mega-2560 (Accessed Apr. 27, 2025).

[45] “L293D Motor Driver Shield - Wiki.” http://wiki.sunfounder.cc/index.php?title=L293D_Motor_Driver_Shield

[46] Instructables, “Mastering servo control with PCA9685 and Arduino,” Instructables, Feb. 15, 2024. https://www.instructables.com/Mastering-Servo-Control-With-PCA9685-and-Arduino/

[47] SparkFun Electronics, "HC-SR04 Ultrasonic Sensor Datasheet," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed: Apr. 27, 2025].

[48] Raspberry Pi Foundation, "Raspberry Pi Camera Module v2 Datasheet," [Online]. Available: https://www.farnell.com/datasheets/2056179.pdf. [Accessed: Apr. 27, 2025].

[49] ElectronicWings, “HC-05 Bluetooth Module Specifications,” [Online]. Available: https://www.electronicwings.com/sensors-modules/bluetooth-module-hc-05- (Accessed Apr. 27, 2025).

[50] A. S. Sadun, J. Jalani, J. A. Sukor, and B. Pahat, "A comparative study on the position control method of DC servo motor with position feedback by using Arduino," in Proc. Eng. Technol. Int. Conf. (ETIC), pp. 10–11, Aug. 2015.

[51] Omartronics, “6-DOF Robotic Arm for Arduino - Full 3D Printed Model STL,” Cults 3D, Sep. 02, 2024. https://cults3d.com/en/3d-model/gadget/6-dof-robotic-arm-for-arduino-full-3d-printed-model

References

[52] Dassault Systèmes, *Introducing SOLIDWORKS*, SolidWorks Official Documentation, 2023. [Online]. Available: https://my.solidworks.com/solidworks/guide/SOLIDWORKS_Introduction_EN.pdf

[53] L. Fuentes, “Creality Slicer: All you need to know,” *All3DP*, Jan. 11, 2024. https://all3dp.com/2/creality-slicer-overview/

[54] “Raspberry Pi OS - Raspberry Pi Documentation.” https://www.raspberrypi.com/documentation/computers/os.html

[55] “How does VNC technology work?,” *RealVNC Help Center*. https://help.realvnc.com/hc/en-us/articles/360002320638-How-does-VNC-technology-work

[56] OpenCV, “About - OpenCV,” *OpenCV*, Nov. 04, 2020. https://opencv.org/about/

[57] GeeksforGeeks, “Detecting ArUco markers with OpenCV and Python,” *GeeksforGeeks*, Jul. 23, 2024. https://www.geeksforgeeks.org/detecting-aruco-markers-with-opencv-and-python-1/

[58] “OpenCV: Detection of ARUCO markers.” https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[59] Ultralytics, “YOLOV8,” Mar. 31, 2025. https://docs.ultralytics.com/models/yolov8

[60] B. Dwyer, “Getting Started with Roboflow,” *Roboflow Blog*, Feb. 03, 2025. https://blog.roboflow.com/getting-started-with-roboflow/

[61] AI project, *waste-detection Dataset*. Roboflow Universe, Roboflow, Dec. 2023. [Online]. Available: https://universe.roboflow.com/ai-project-i3wje/waste-detection-vqkjo [Accessed: May 16, 2025].

[62] Google, "Google Colaboratory FAQ," Available: https://research.google.com/colaboratory/faq.html. Accessed: May 15, 2025.

[63] Roboflow, *Roboflow Notebooks*. GitHub. [Online]. Available: https://github.com/roboflow/notebooks

[64] Ultralytics, “YOLO Performance Metrics,” Mar. 30, 2025. https://docs.ultralytics.com/guides/yolo-performance-metrics

[65] “Overview of the Arduino IDE 1 | Arduino Documentation.” https://docs.arduino.cc/software/ide-v1/tutorials/Environment/

[66] “Getting Started with YOLO Object and Animal Recognition on the Raspberry Pi,” *Core Electronics*. https://core-electronics.com.au/guides/getting-started-with-yolo-object-and-animal-recognition-on-the-raspberry-pi/

[67] “Meet Android Studio,” *Android Developers*. https://developer.android.com/studio/intro

[68] “Configuration - Raspberry Pi documentation.” https://www.raspberrypi.com/documentation/computers/configuration.html

[69] J. Geng, “How to Evaluate an Object Detection Model: Explain IoU, Precision, Recall and mAP with examples,” *Medium*, Dec. 13, 2022. [Online]. Available: https://zihaogeng.medium.com/how-to-evaluate-an-object-detection-model-iou-precision-recall-and-map-f7cc12e0dcf6

[70] “Choosing a transistor - Other Hardware / Motors, Mechanics, Power and CNC - Arduino Forum,” *Arduino Forum*, Jun. 13, 2020. https://forum.arduino.cc/t/choosing-a-transistor/660484/9

[71] Nitin, “Getting Started with Raspberry Pi + Installation Guide for Tensorflow 2.3.1 and OpenCV 4.5.1,” *Medium*, Dec. 31, 2021. [Online]. Available: https://nitin9809.medium.com/getting-started-with-raspberry-pi-installation-guide-for-tensorflow-2-3-1-and-opencv-4-5-1-3a5bd0b6de2d

[72] “Arduino Mega2560 R3 pinouts photo - Other Hardware / 3rd Party Boards - Arduino Forum,” *Arduino Forum*, Oct. 06, 2012. https://forum.arduino.cc/t/arduino-mega2560-r3-pinouts-

photo/123330

[73] “How to connect a tx and rx pins on l293d - Other Hardware / 3rd Party Boards - Arduino Forum,” *Arduino Forum*, Nov. 26, 2024. https://forum.arduino.cc/t/how-to-connect-a-tx-and-rx-pins-on-l293d/1326048/3

[74] D. Mittal, “PCA9685 - Multiple servo control using arduino,” Jan. 16, 2025. https://circuitdigest.com/microcontroller-projects/pca9685-multiple-servo-control-using-arduino

[75] Smart Dustbin Utility System using IoT - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Ultrasonic-Sensor-HC-SR0_fig2_354250727 [accessed 25 May 2025]