**PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA**
**MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH**

**MOHAMED BOUDIAF UNIVERSITY - M'SILA**

**FACULTY OF TECHNOLOGY**
**DEPARTMENT :** Electrical engineering
**N° :**

**OPTION:** Automatic
**SPECIALTY:** Robotics

## Final Study Dissertation In The Aim Of Obtaining a Master's Degree

**Presendet by**

Roubache Bilal
Raghdi Kamel

# Title

# Design and Commissioning of a Mobile Cleaning Robot

**Supported before the jury composed of :**

| | | |
|---|---|---|
| Dr. KHATIR KHETTAB | University of M'sila | President |
| Dr. Abdelghani AIB | University of M'sila | Advisor |
| Dr. Hilal Rahali | University of M'sila | Co Advisor |
| Dr. Fayssal Ouagueni | University of M'sila | Examiner |

**Academic Year : 2023 /2024**

**Abstract**

This work presents a comprehensive study on the programming and design of an autonomous robot specialized in cleaning the environment. The research is divided into two complementary parts: theoretical and applied.

The theoretical part focuses on providing a broad background on the field of robotics, including its history, evolution, and movement mechanisms. It also details the hardware and software used in the practical project and includes a thorough explanation of the renowned YOLO (You Only Look Once) technique in object recognition.

The applied part explains the steps involved in installing and operating the robot designed for this project, detailing the programmed algorithms and techniques used. This section also presents experimental results that demonstrate the robot's effectiveness in performing its cleaning tasks.

**Keywords:** Robot, Autonomous Robot, Object Recognition, Deep Learning, YOLO.

**الملخص**

يقدم هذا العمل دراسة حول برمجة وتصميم روبوت مستقل متخص في تنظيف البيئات والمساحات المغلقة .

ينقسم البحث إلى جزئين : جزء نظري و جزء تطبيقي.

يركز الجزء النظري على تقديم خلفية عن علم الروبوتات، بما في ذلك تاريخها وتطورها وآليات حركتها.

كما يتناول بالتفصيل العتاد وبرمجيات المستخدمة في المشروع العملي، ويشمل شرحا مفصلاً لتقنية $YOLO$ ($YouOnlyLookOnce$) الشهيرة في مجال التعرف على الأشياء.

أما الجزء التطبيقي، فيعرض خطوات تثبيت وتشغيل الروبوت المصمم لهذا المشروع، مع توضيح خوارزميات مبرمجة وتقنيات مستخدمة. ويشمل هذا الجزء أيضا عرضا للنتائج التجريبية التي تبرز فعالية الروبوت في أداء مهامه.

الكلمات المفتاحية: روبوت،روبوت مستقل، التعرف على الأشياء، التعلم العميق، $YOLO$.

**Résumé**

Ce travail présente une étude complète sur la programmation et la conception d'un robot autonome spécialisé dans le nettoyage de l'environnement. La recherche est divisée en deux parties complémentaires : théorique et appliquée.

La partie théorique se concentre sur la fourniture d'un large contexte sur le domaine de la robotique, y compris son histoire, son évolution et ses mécanismes de mouvement. Elle détaille également le matériel et les logiciels utilisés dans le projet pratique et comprend une explication approfondie de la technique renommée $YOLO$($YouOnlyLookOnce$) dans la reconnaissance d'objets.

La partie appliquée explique les étapes de l'installation et de l'exploitation du robot conçu pour ce projet, en détaillant les algorithmes programmés et les techniques utilisées. Cette section présente également les résultats expérimentaux qui démontrent l'efficacité du robot dans l'accomplissement de ses tâches de nettoyage.

**Mots-clés :** Robot, Robot autonome, Reconnaissance d'objets, Apprentissage profond, YOLO.

**الإهداء**

في المقام الأول، يجب أن أعرب عن شكري اللامحدود لله ع وجل ، على تيسيره من خلال منحنا الفرصة والشجاعة والصبر والطاقة الكافية لإنجاز وإكمال هذا المشروع .

نود أن نعرب عن خالص شكرنا وامتناننا لمعلمنا الدكتور $Abdelghani\ AIB$ و $Hilal\ Rahali$ لإشرافه ودعمه.

نود أن نعرب عن خالص تقديرنا للجنة الفحص، الأستاذ، السيد $KHATIR\ KHETTAB$ ، والسيد$Fayssal\ Ouagueni$ لاهتمامهم بالعمل.

أخيراً، أود أن أعرب عن امتناني العميق لعائلتنا لدعمهم لنا دائما.

# Contents

# List of Figures



# List of Tables

# General Introduction

In today's world, society faces significant challenges in environmental and security issues, requiring innovative and sustainable solutions. With advancements in perception and sensing capabilities, robots have become vital in addressing these challenges even in unknown or unpredictable environments. Robots contribute to ensuring a safer working environment for humans and providing effective solutions to waste and environmental pollution issues.

Plastic waste, which takes decades to decompose, and toxic or acidic industrial waste pose significant risks to humans when handled. Over the past two decades, specialized robots have emerged to replace humans in these hazardous tasks. Among these robots, we find "Robofire," used in firefighting, and bomb-handling robots, which feature a tank-like mobile design with a robotic arm or a water-pumping outlet, facilitating operations in diverse environments.

Historically, robots have been introduced in factories since the 1960s, performing repetitive tasks with high precision, reducing the need for human labor in tedious and dangerous work. With technological advancements, robots have become capable of sensing and interacting with their environment, breaking free from the cages that once confined them to protect humans.[1]

In the 21st century, with advancements in computing, communication, and sensing technologies, robots have gained increased autonomy and flexibility in handling diverse environments. However, dealing with real-world data, ensuring provable correctness under uncertainty, and maintaining trust and safety remain challenges for modern robots.

Robots today operate in novel environments, requiring increased levels of resilience, reactivity, and reconfigurability at the control and planning levels. Robots must be able to handle potential disturbances, such as drastic environmental changes, and make appropriate control and planning decisions.[2]

The environments in which robots operate are often unfriendly to humans, making the use of robotic and autonomous systems highly relevant. Robots need to be designed to survive and perform with limited power, communication, and navigation capabilities, addressing challenges related to scalability and uncertainty in environmental dynamics.

In the future, humans and robots are expected to collaborate in various domains, where robots may be taught specific tasks by humans, such as assembly or cooking, or even assist humans in physical recovery or educational settings. Failure in these collaborations can be costly, underscoring the importance of safety in human-robot interaction.[1]

To realize the full potential of multi-robot systems, a fundamental understanding of the interplay between sensing, communication, and execution is necessary. In complex environments, some tasks may be conflicting or infeasible, requiring real-time replanning and adaptation by the robot team.

By providing services in hygiene and security, service robots, autonomous vehicles, UAVs, surgical robots, and field robots contribute to improving the quality of life and making the world a cleaner and safer place.

**Our project aims to program a robot that can operate autonomously, identifying and disposing of waste without human intervention. We will explain our work as follows:**

**Chapter 1: Generalities on Mobile Robots:** This chapter delves into the general concept of mobile robotics, starting with its definition, history, and various types. It covers different categories of robots, including manipulator robots, mobile robots, industrial robots, and humanoid robots. The chapter discusses the significance and applications of each type.

**locomotion:** The focus of this sub chapter is on the locomotion of mobile robots, particularly those that walk on legs and those that move on wheels. It explores different leg configurations and their stability, providing examples of legged robot movement. For wheeled robots, the chapter discusses their types, driving methods, and maneuverability.

**Mobile Robot Kinematics** This sub chapter takes a deeper look into the abstract motion of mobile robots, including its importance and basic related concepts. It covers dynamic and static constraints, geometric representation of robots, inverse motion, mapping the robot's position, and motion models.

**Hardware and Software:** Here, the discussion revolves around the hardware and software used in mobile robotics. Various components are examined, such as ultrasonic sensors, DC motors, drives, cameras, and power supply. Instructions on installing these components, along with their software and operating systems, such as Raspberry Pi, Raspbian OS, and Linux, are provided.

**Object Detection using YOLOv8:** This sub chapter introduces the object detection technique YOLO (You Only Look Once), focusing on its architectural structure and object detection process. The advantages of YOLO, the need for a confusion matrix, and different versions of YOLO are discussed. Additionally, guidance is provided on how to train YOLOv8 on a custom dataset, install it, and make predictions using a custom model.

**chapter 2 : Particle Work and Results:** This sub chapter presents the experimental work and results obtained. It discusses target determination, motion path, and the exploration algorithm. Instructions on operating system installation, robot connection, image processing, and motion towards the target are provided. Furthermore, the chapter explores the exploration algorithm, remote connection without a network, and the driver file.

# 1 Theoretical Aspect

In this chapter, we will discuss the theoretical aspect of the project. We will talk about definitions, equations, and system-specifics of the robot, as well as an introduction to computer vision that was used.

## 1.1 Generalities on Mobile Robots

### 1.1.1 Definition

The word "robot" first appeared in 1921,coined by the Czech playwright Karel Čapek , referring to a servant.[2] Technically, the term "robot" has evolved since then.Not every definition serves the purpose. For instance, the definition of the Robotics Institute of America (RIA) does not include mobile robots [3]. One common definition of a robot is:

"The robot is defined as a goal-oriented machine that can sense, plan, and act." [4]

The definition can be formulated as follows:

"A robot is an autonomous system that exists in the physical world, can sense its environment, and can act on it to achieve some goals."[5]

Figure 1: **Figure 1.1**: Karel Čapek [6]

### 1.1.2 Historical Introduction

### 1.1.3 Robots Arms

The first appearance of a mechanical arm was by George Devol Jr., who obtained a patent for it in 1954. In 1961, the Unimate robotic arm entered production lines at General Motors factories. Two years later, in 1963, another robotic arm, the RanchoArm, was introduced into therapeutic services at Rancho Los Amigos Hospital in Downey, California, to assist individuals with special needs.[4]

In the 1980s, a wave of humanoid robots emerged, beginning in Japan with robots like WABOT-1 and WABOT-2, capable of playing musical instruments.[1]


**Figure 1.2** :Unimate Industrial Robot

### 1.1.4 Robots Mobile

By the mid-twentieth century, the field of "cybernetics" emerged, focusing on understanding artificial and biological systems and controlling complex systems. Dr. William Grey Walter (1910-1977), a pioneer in this field, created the first prototype of mobile robots known as "Tortoises," named after the tortoise in "Alice in Wonderland." These robots had a tricycle-like design, utilizing a front wheel for steering and two rear wheels for propulsion. In 1970, Stanford University developed the Shakey robot, which incorporated smart sensors to study how robots interact with their surroundings. By 1979, it was enhanced with cameras, 3D vision, and autonomous navigation capabilities, enabling it to traverse rooms filled with obstacles such as chairs.[5]

In the 1990s, exploration robots were introduced for investigating hard-to-reach or hazardous environments. One notable example is the Pathfinder robot developed by NASA in 1996 for exploring the Martian surface.[2]

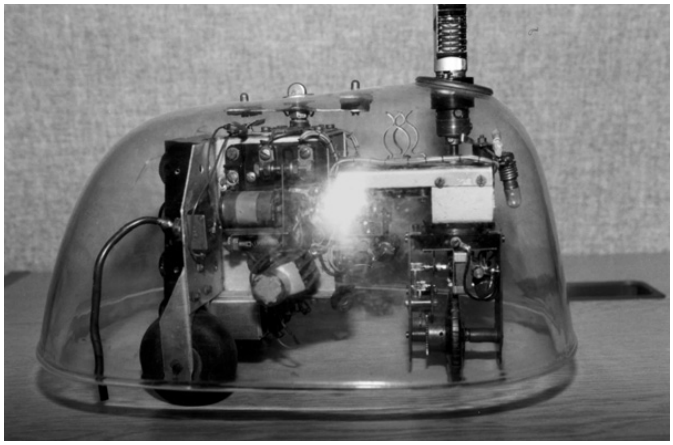


**Figure 1.3** :"Elsie the tortoise" [7]

### 1.1.5 The Types of Robots

1. **Mobile Robots** Mobile robots are designed for autonomous movement, requiring sophisticated locomotion mechanisms to navigate various environments. These robots can walk, jump, run, slide, skate, swim, fly, or roll, drawing inspiration from biological locomotion systems. Notably, actively powered wheels, a human invention, offer high efficiency on flat surfaces, a stark contrast to biological locomotion. While replicating nature's locomotion mechanisms is desirable, it's challenging due to mechanical complexity, size constraints,

and energy efficiency disparities between biological and man-made systems. [2]

2. **Industrial Robots** Industrial robots are stationary or semi-stationary machines used for manufacturing and industrial tasks. These robots excel in precision, repeatability, and endurance, often equipped with articulated arms and specialized end-effectors for handling diverse materials and performing intricate operations. Their design prioritizes robustness, safety, and high-speed performance, contributing significantly to automation across industries [1] [4]

3. **Humanoids** Humanoids represent the pinnacle of robotic design, mimicking human form and capabilities to varying degrees. These robots integrate advanced sensors, actuators, and AI algorithms to interact with the environment and humans. Humanoids have applications in healthcare, assistance, entertainment, and research, pushing boundaries in human-robot interaction and cognitive robotics. [5] [2]

#### 1.1.6 Conclusion

In the 20th century, the field of robotics witnessed significant advancements. In the 1930s, William Grey Walter invented what is believed to be the first true robot, a mechanical doll that could control its own movements. In the 1940s, robotics scientist George Devol began developing the first industrial robot, a robot capable of performing repetitive tasks in an industrial environment.

Since then, robotics has evolved tremendously, finding applications in numerous industries, including manufacturing, healthcare, and entertainment. Modern robots utilize sensors, artificial intelligence, and machine learning to interact with their environment, learn, and adapt.

Today, robots continue to evolve and become increasingly capable of performing complex tasks as researchers and inventors continue to push the boundaries of what these remarkable machines can do.

### 1.2 Locomotion

#### 1.2.1 Introduction

Locomotion, the counterpart of manipulation in robotics, involves the movement of the robot itself rather than manipulating objects within its workspace. This movement is achieved by imparting force to the environment through various mechanisms. Both locomotion and manipulation share fundamental issues such as stability, contact characteristics, and environmental adaptability. [2]

#### 1.2.2 Legged Mobile Robots

Legged locomotion is characterized by point contacts between the robot and the ground, offering adaptability and maneuverability in rough terrain. Leg configurations vary, with successful designs seen in organisms ranging from insects to mammals. The number of legs influences stability, with three-legged robots demonstrating static stability without the need for active control. Examples of legged robot locomotion include one-legged robots to six-legged designs, showcasing ongoing research in this area. [2] [5]

#### 1.2.3 Leg configurations and stability

Biologically inspired legged robots often mimic successful leg configurations seen in nature. Different leg configurations, such as four legs in mammals and six or more legs in insects, provide insights into stability and maneuverability. The number of legs directly impacts static stability, with three-legged robots demonstrating stable poses without active control mechanisms [2] [5]

#### 1.2.4 Examples of legged robot locomotion

While legged locomotion lacks high-volume industrial applications currently, ongoing research explores various designs. Examples range from one-legged robots to six-legged designs, each showcasing unique locomotion capabilities and potential applications. [1] [5]

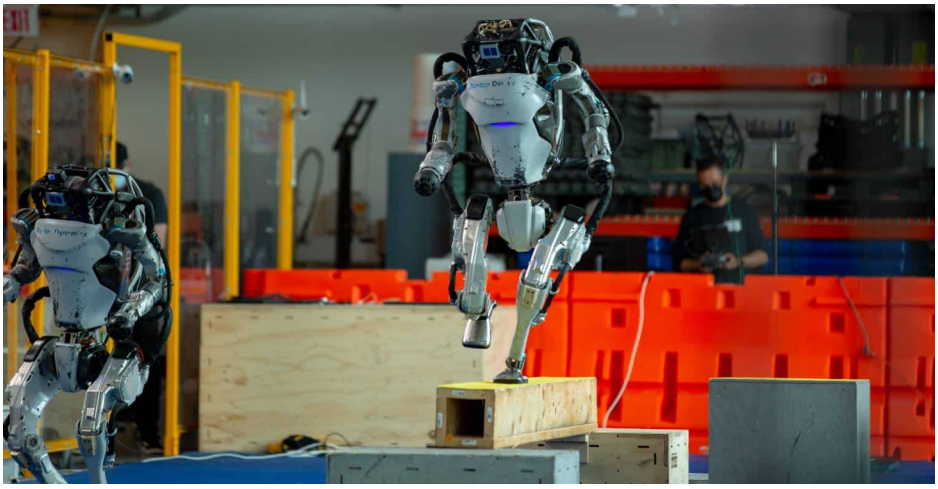


**Figure 1.4** : Atlas

### 1.2.5 Wheeled Mobile Robots

Wheeled locomotion is widely adopted in mobile robotics due to its efficiency and mechanical simplicity. Wheeled robots achieve good efficiencies and stability, with designs ranging from three-wheeled to multi-wheeled configurations. Maneuverability, traction, and control are key focus areas in wheeled robot research, with various drive types and wheel designs influencing overall performance. [2] [1] [8]

### 1.2.6 Types of Wheeled Mobile Robots

**Can the robot wheels provide The four basic wheel types:** [2]

1. Standard wheel: two degrees of freedom; rotation around the (motorized) wheel axle and the contact point.
2. Castor wheel: two degrees of freedom; rotation around an offset steering joint.
3. Swedish wheel: three degrees of freedom; rotation around the (motorized) wheel axle, around the rollers, and around the contact point.
4. Ball or spherical wheel: realization technically difficult.

### 1.2.7 Drive Types

In the realm of wheeled mobile robots (WMRs), the design choices for drive types play a critical role in determining their functionality and efficiency. From single-wheel setups to more complex articulated structures, each drive type brings unique advantages and considerations. This article delves into the world of WMRs, categorizing them based on drive types and exploring key features and applications within each category
When discussing drive types in WMRs, several key elements come into play, including wheel selection, placement, and kinematic parameters. These elements are pivotal in achieving optimal performance based on the intended environment, tasks, and cost considerations.[1]

- **Single-Wheel Robots:** While uncommon, single-wheel robots showcase innovative design concepts. Their maneuverability and compactness make them suitable for specialized applications where space is limited or unconventional terrains are encountered.
- **Two-Wheel Robots:** Two-wheel robots encompass various configurations, such as differential-drive and synchronous-drive setups. These designs offer improved stability and control, making them popular choices for navigation in structured environments. A two-wheel differential-drive robot is one of the most popular designs in robotics, consisting of two active fixed wheels and one passive caster wheel. The main advantages of this design include its simple mechanical structure, simple kinematic model, and low fabrication cost. The robot also boasts a zero-turning radius, which makes it easy to compute the obstacle-free space for a

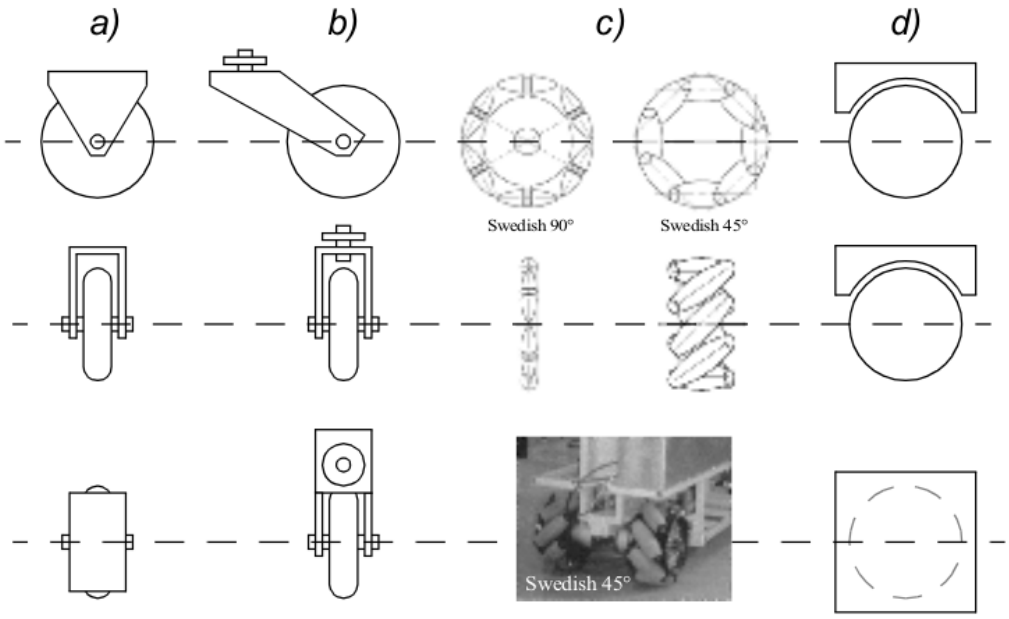



**Figure 1.5** :Wheels

cylindrical robot by expanding obstacle boundaries by the robot's radius. Systematic errors in this design are easily calibrated.

However, the two-wheel robot has some drawbacks, including the difficulty of navigating irregular surfaces. When traversing uneven terrain, the robot's orientation may change abruptly if one of the active wheels loses contact with the ground. Additionally, this design is limited to bidirectional movement only.

- **Three-Wheel Robots:** Three-wheel robots introduce additional stability compared to two-wheel counterparts. Variants like omnimobile robots with Swedish or active caster wheels provide enhanced maneuverability and versatility in navigation tasks.

- **Four-Wheel Robots:** Four-wheel robots exhibit robustness and load-bearing capabilities, making them suitable for heavy-duty applications. Their designs can range from standard configurations to specialized setups for specific tasks

- **Skid-steer robot:** This design is a special implementation of differential drive, utilized in the form of tracks on heavy machinery such as bulldozers and armored vehicles. It offers increased maneuverability in uneven terrains compared to differential drive vehicles due to higher friction provided by its tracks and multiple contact points with the surface, be it rough or even.

  For a skid-steer robot, the effective point of contact is roughly constrained on either side by a rectangular uncertainty area corresponding to the track footprint. As illustrated in the figures, a considerable slippage is required for the vehicle to turn. This design is commonly employed in robotic platforms that carry manipulators or specialized exploration equipment.

  The skid-steer robot, or tracked robot, has a much larger ground contact area compared to conventional wheeled designs, significantly improving its maneuverability in loose terrain. However, changing the orientation of the robot typically requires a skidding turn, where a large portion of the track slides against the terrain. This results in difficulties predicting the exact center of rotation and precise position and orientation changes due to variations in ground friction. As a trade-off for excellent maneuverability and traction over rough and loose terrain, dead reckoning on such robots is highly inaccurate. Furthermore, the skid-steer approach on high-friction surfaces can quickly exceed the torque capabilities of the motors. In terms of power efficiency, this method is reasonably efficient on loose terrain but extremely inefficient otherwise.. [2]

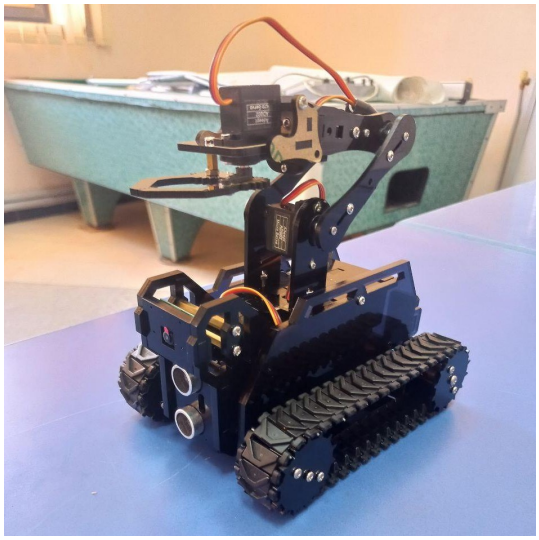


**Figure 1.6** : Our robot

### 1.2.8 WMR Maneuverability

The maneuverability of wheeled mobile robots (WMRs) depends on wheel types, arrangements, and kinematics. Different wheel classes, such as standard, castor, Swedish, and spherical wheels, offer varying degrees of directional control and stability. Design considerations for WMRs include selecting wheel types, placement, and kinematic parameters based on target environments and tasks. In conclusion, locomotion in robotics encompasses diverse mechanisms, from legged to wheeled designs, each offering unique advantages and challenges. Advances in locomotion technology continue to drive innovation and automation across various industries.[2] [1]

### 1.2.9 Conclusion

The most important aspects that locomotion mechanisms concern themselves with are stability and mobility.

The locomotion of mobile robots varies greatly, ranging from those that use wheels, a human invention, to those that employ a few legs, which is the simplest biological approach to movement. Wheels are simple and highly effective on flat surfaces, but they are less efficient in unconventional or rough environments. Legged locomotion requires a higher degree of freedom and is thus more mechanically complex.

Most locomotion systems draw inspiration from their biological counterparts, even though the actively powered wheel is a human invention. Our system of bipedal walking can be likened to a near-circle, but nature has not evolved an actively powered joint, which is necessary for wheeled movement.

Some examples of locomotion mechanisms in commercially available robots include legs with three degrees of freedom, similar to those found in insects, or wheels with different geometric configurations, such as those on bicycles or automobiles.

## 1.3 Mobile Robot Kinematics

### 1.3.1 Introduction

Robot kinematics plays a pivotal role in the realm of robotics, encompassing the intricate study of robot configurations within their operational space, the interrelations between their geometric parameters, and the constraints dictating their trajectories. This article aims to delve into the fundamental concepts and significance of various elements within robot kinematics, shedding light on its importance and core concepts.

### 1.3.2 Significance of Robot Kinematics

Understanding robot kinematics is paramount in robotic research and development for several reasons. Firstly, it provides insights into the configuration of robots and how their components move relative to each other. Additionally, it aids in analyzing and predicting the paths and motions that robots can undertake, thus optimizing task performance. Moreover, knowledge of kinematics is essential for designing effective control systems to guide robot movements accurately, enhancing efficiency and effectiveness in various applications.

### 1.3.3 Fundamental Concepts in Robot Kinematics

Within the realm of robot kinematics, several fundamental concepts merit exploration. Direct kinematics involves determining the end-effector position based on joint angles, while inverse kinematics entails calculating the joint angles required to achieve a desired end-effector position. Homogeneous transformations facilitate seamless conversion between coordinate systems, crucial for accurately mapping robot movements. Furthermore, nonholonomic constraints arising from limitations in robot motion, such as wheel slippage or directional restrictions, significantly impact the robot's maneuverability.

### 1.3.4 Rolling without slipping

The principle of "rolling without slipping" is a fundamental concept in physics and engineering, describing the motion of a rolling body, such as a wheel or a ball, without relative sliding between the body and the surface it contacts. Mathematically, this principle is expressed by the equation of zero relative velocity, implying that the point of contact between the body and the surface moves at the same speed as the body itself. This principle finds important applications in areas such as dynamics, robotics, and vehicles. It aids in the design of effective control systems and enhances the performance of mobile robots and vehicles. Although the real world may not always be ideal, understanding the principle of rolling without slipping provides a robust framework for analyzing and designing efficient mobile systems.

$$\vec{v}_Q = \vec{v}_P + \vec{\omega} \wedge \overrightarrow{PQ} = \overrightarrow{0} \tag{1.1}$$

### 1.3.5 Non-holonomic constraints

Non-holonomic constraints are restrictions on the motion of a system, such as wheeled mobile robots, that cannot be derived or expressed as a set of integrable constraints. In other words, these constraints cannot be written as derivatives of a function and thus do not follow predictable or easily integrable paths.

In the context of mobile robotics, a common example of a non-holonomic constraint is the "no-slip" condition of wheels. As a wheel rolls, it also slides laterally. However, the non-holonomic constraint here is that the wheel cannot slide sideways while rolling forward or backward. This constraint is not easily derivable and cannot be expressed as an integrable constraint.

These constraints influence the robot's ability to move and maneuver. For instance, a wheeled mobile robot can move anywhere within a given space, but its precise path depends on these non-holonomic constraints. The robot may need to move in a specific direction, but the non-holonomic constraints might cause slight deviations or require a particular sequence of motions to reach the desired target.

Other examples of non-holonomic constraints in mobile robotics include constraints on the robot's motion in certain directions, limitations on rotational speed, or even constraints arising from the robot's dynamics and interactions with the environment.

Dealing with non-holonomic constraints is an important aspect of robot path planning and control, often requiring complex control and planning methods to ensure effective and accurate robot motion.

### 1.3.6 Kinematic Parameters and Constraints

A critical aspect of robot kinematics is understanding the parameters and constraints governing robot motion. Each wheel contributes to the robot's motion and imposes specific constraints, such as preventing lateral skidding. These individual wheel constraints combine to form overarching constraints that dictate the robot's overall motion and maneuverability.

### 1.3.7 The Geometric Representation of a Robotic Manipulator

The geometric representation of a robotic manipulator can be defined by attaching reference frames to each link. While these frames can be located arbitrarily, adhering to a convention for placing the frames on the links is advantageous for consistency and computational efficiency. Denavit and Hartenberg introduced a foundational convention that has been adapted in various ways, requiring only four parameters (link length, link twist, joint offset, and joint angle) to locate one reference frame relative to another.[4]
Using this numbering system, the attachment of reference frames follows these rules:

- The $\hat{z}_i$ axis is aligned with the axis of joint $i$.
- The $\hat{x}_{i-1}$ axis is aligned along the common normal between the $\hat{z}_{i-1}$ and $\hat{z}_i$ axes.

With the reference frames in place, the four parameters that define the position of one frame relative to another are [1]:

- $a_i$: the distance from $\hat{z}_{i-1}$ to $\hat{z}_i$ along $\hat{x}_{i-1}$.
- $\alpha_i$: the angle from $\hat{z}_{i-1}$ to $\hat{z}_i$ around $\hat{x}_{i-1}$.
- $d_i$: the distance from $\hat{x}_{i-1}$ to $\hat{x}_i$ along $\hat{z}_i$.
- $\theta_i$: the angle from $\hat{x}_{i-1}$ to $\hat{x}_i$ around $\hat{z}_i$.

$$
{}^{i-1}\boldsymbol{T}_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i\cos\alpha_i & \cos\theta_i\cos\alpha_i & -\sin\alpha_i & -\sin\alpha_i d_i \\ \sin\theta_i\sin\alpha_i & \cos\theta_i\sin\alpha_i & \cos\alpha_i & \cos\alpha_i d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.2}
$$

as an example of our robot, consider a two-joint planar arm with pinned links and no joint limits. The rotational parameters $\theta_1$ and $\theta_2$ specify the configuration. Each joint angle $\theta_i$ corresponds to a point on the unit circle S1, and the C-space is defined as the product of these circles, resulting in a two-dimensional torus. If the base of the planar arm is mobile, additional translation parameters must be considered in the arm's configuration.
The Stanford Arm is another example of a robotic arm with a geometric configuration chosen to facilitate easy-to-use and time-efficient inverse kinematics programming. It has six independently driven joints, with one telescoping (prismatic) and five rotary (revolute) joints. The Stanford Arm's mechanical design is compatible with the limitations of timeshare computer control, and various end-effectors, such as a vise-grip jaw, can be attached.

| **Link** | $\theta_i$ | $d_i$ | $a_i$ | $\alpha_i$ |
|---|---|---|---|---|
| 1 | $\theta_1$ | 0 | $a_1$ | 0 |
| 2 | $\theta_2$ | 0 | $a_2$ | 0 |

**Table 1.3.1**: Table of link parameters of arm

### 1.3.8 Inverse kinematics

Inverse kinematics is the process of computing the joint angles of a robotic arm to move its end effector to a specific position. This process is the opposite of forward kinematics, which involves determining the position of the end effector based on the joint angles. Inverse kinematics is computationally intensive, making it suitable for supervised learning techniques, where correct solutions are known and errors can be easily measured.

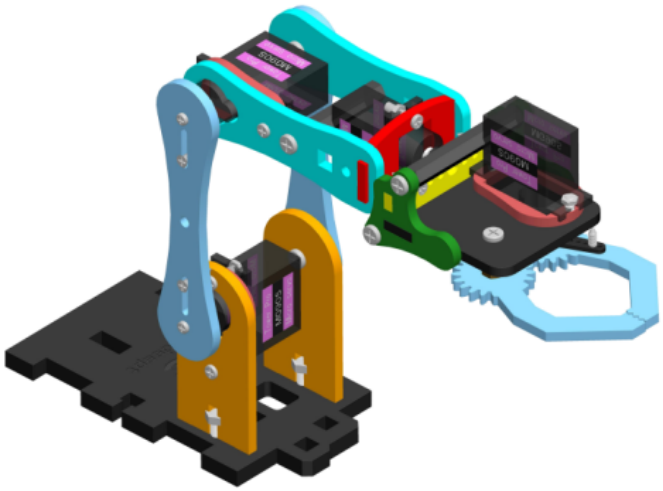


**Figure 1.7** : Arm Robot

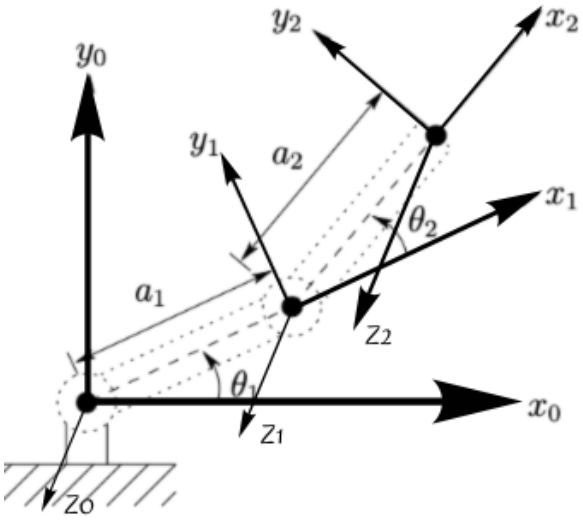



**Figure 1.8** : 2D draw of manipulator Robot arm [9]

## Change of coordinates:

In our work, we first used the angles in Figure 3.2 because they are simple and easy to explain. However, in practical applications, we will use the representation shown in Figure 3.3, which is the actual representation of the robot arm. It is noted that the definition of Theta 2 is the same, but the difference lies in the direction of rotation, and therefore: $q_2 = 2\pi - \theta_2$

Inverse kinematics of our robotic arm

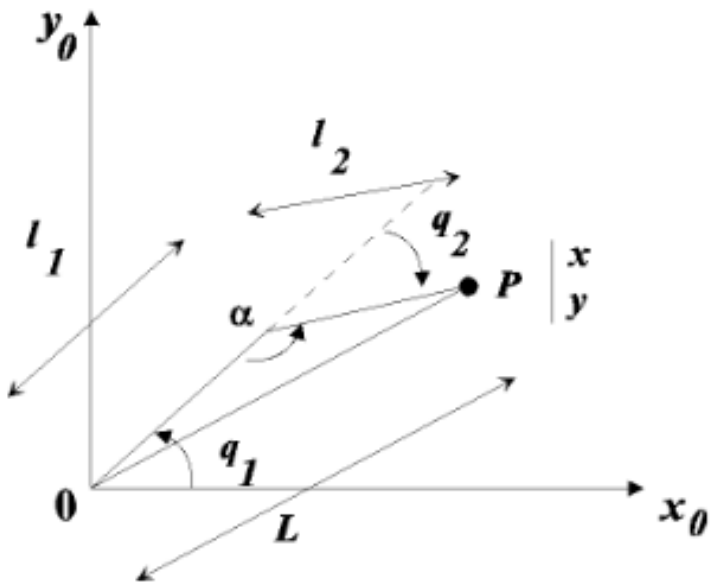



**Figure 1.9** : model mathematics of the manipulator Robot

These two equations represent the coordinates of the endpoint of a robot arm in the Cartesian plane (x, y), where:

$$x = l_1 \cos q_1 + l_2 \cos (q_2 + q_1) \tag{1.4}$$

$$y = l_1 \sin q_1 - l_2 \sin (q_2 + q_1) \tag{1.5}$$

- $l_1$ is the length of the first segment of the arm.
- $l_2$ is the length of the second segment of the arm.
- $q_1$ is the angle between the first segment of the arm and the horizontal axis.
- $q_2$ is the angle between the second segment of the arm and the horizontal axis after the first rotation.

Such that:

$$L^2 = x^2 + y^2 \tag{1.6}$$

$$L^2 = {l_1}^2 + {l_2}^2 - 2l_1l_2(\cos(q_1)\cos(q_2 + q_1) + \sin(q_1)\sin(q_2 + q_1)) \tag{1.7}$$

according to the trigonometric equations:

$$\cos(a + b) = \cos(a)\cos(b) - \sin(a)\sin(b),$$
$$\sin(a + b) = \sin(a)\cos(b) + \cos(a)\sin(b).$$

so:

$$r^2 = {l_1}^2 + {l_2}^2 + 2l_1l_2(\cos(q_2)) \tag{1.8}$$

$$x^2 + y^2 = {l_1}^2 + {l_2}^2 + 2l_1l_2(\cos(q_2)) \tag{1.9}$$

Therefore, it is:

$$q_2 = \operatorname{acos}\left(\frac{x^2 + y^2 - l_0^2 - l_1^2}{2l_1l_2}\right) \tag{1.10}$$

to find $q_1$ we obtain:

$$\begin{cases} x = (l_2\cos(q_2) + l_1)\cos(q_1) - l_2\sin(q_2)\sin(q_1) \\ y = l_2\sin(q_2)\cos(q_1) + (l_1 + l_2\cos(q_2))\sin(q_1) \end{cases} \tag{1.11}$$

as matrix form:

$$\begin{bmatrix} l_1 + l_2\cos(q_2) & -l_2\sin(q_2) \\ l_2\sin(q_2) & l_1 + l_2\cos(q_2) \end{bmatrix} * \begin{bmatrix} \cos(q_1) \\ \sin(q_1) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.12}$$

Therefore, it is:

$$det(\begin{vmatrix} l_1 + l_2\cos(q_2) & -l_2\sin(q_2) \\ l_2\sin(q_2) & l_1 + l_2\cos(q_2) \end{vmatrix}) = l_1^2 + l_2^2 + 2l_1l_2\cos(q_2) = x^2 + y^2 \tag{1.13}$$

so :

$$\begin{bmatrix} \cos(q_1) \\ \sin(q_1) \end{bmatrix} = \frac{1}{x^2 + y^2}\begin{bmatrix} l_1 + l_2\cos(q_2) & l_2\sin(q_2) \\ -l_2\sin(q_2) & l_1 + l_2\cos(q_2) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.14}$$

in th end :

$$\cos(q_1) = \frac{1}{x^2 + y^2}(x(l_1 + l_2\cos(q_2)) + yl_2\sin(q_2))$$
$$\sin(q_1) = \frac{1}{x^2 + y^2}(y(l_1 + l_2\cos(q_2)) - xl_2\sin(q_2))$$

$$q_1 = \arctan 2(y(l_1 + l_2\cos(q_2)) - xl_2\sin(q_2), x(l_1 + l_2\cos(q_2)) + yl_2\sin(q_2)) \tag{1.15}$$

### 1.3.9 Representation of Robot Position in a Map

In representing robot position, a global reference frame for the plane and a local reference frame for the robot chassis are utilized. Parameters such as position coordinates (x, y) and orientation ($\theta$) define the robot's pose relative to its environment, providing a comprehensive understanding of its spatial relationships and movements.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{1.16}$$

### 1.3.10 The Instantaneous Center of Rotation (ICR)

The Instantaneous Center of Rotation (ICR) is a virtual point used to describe the motion of robots and wheeled vehicles. It is the center of an imaginary circle on which the wheels move at any given instant. Imagine drawing a circle around each wheel of the robot. At any moment, each wheel moves along the circumference of its own circle. The ICR is the point where all these circles intersect. It is the instantaneous center of rotation of the robot at that moment.

ICR is a fundamental concept in understanding the motion of robots and wheeled vehicles. It is related to the kinematic constraints imposed on wheel motion. At any instant, the wheels must move in a way that prevents lateral slip. This kinematic constraint implies that the wheels rotate around a common point, which is the ICR.

The location of the ICR can be determined geometrically by drawing lines called "zero motion lines" through the wheel axes. These lines are perpendicular to the wheel plane, indicating that there is no lateral motion. These lines intersect at a single point, which is the ICR. the ICR is determined by the positions and axes of the wheels. If all the wheels are on a common axis, the ICR lies on that axis. If the wheels are steerable, their positions influence the location of the ICR.

One important characteristic of the ICR is that it can be used to calculate the motion of the robot. Knowing the location of the ICR and the wheel speeds, one can determine the direction and velocity of the robot. It can also be used to understand motion constraints: if the robot moves in a straight line, the ICR is at infinity; if the robot rotates in place, the ICR is at its center.

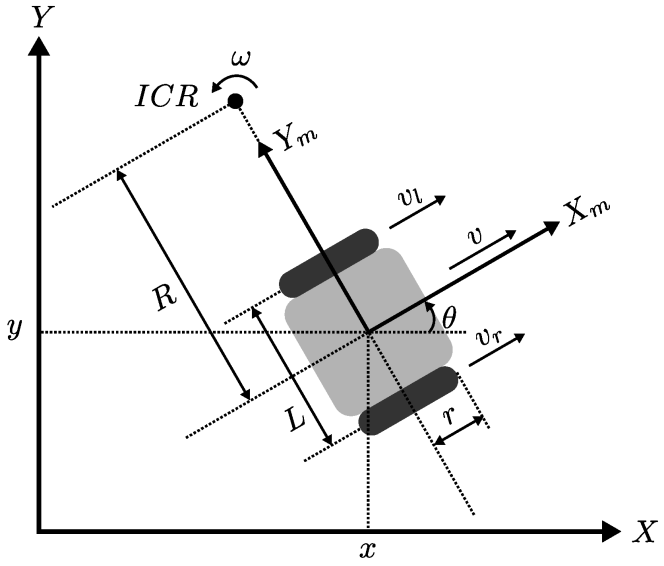



**Figure 1.10**: ICR [8]

### 1.3.11 Kinematic Model

The kinematic model of the unicycle robot relates the velocities of the robot to its configuration. The equations describing the motion of the robot are as follows [8]:

$$\begin{cases} \dot{x}_Q = v_Q \cos\phi \\ \dot{y}_Q = v_Q \sin\phi \\ \dot{\phi} = v_\phi = \omega \end{cases} \tag{1.17}$$

### 1.3.12 Wheeled Motion Analysis

With both driving wheels sharing the same axis of rotation, the robot's Instantaneous Center of Rotation (CIR) lies on this axis. Let R be the radius of curvature of the robot's path, the distance from the CIR to point $O'$ . Let $L$ be the wheelbase and $\omega$ the angular velocity of the robot around the CIR. The velocities of the right and left wheels, respectively denoted $v_r$ and $v_l$, are defined as [8]:

$$\begin{cases} v_l = -r\dot{\phi}_r = (R - L/2)\omega \\ v_r = r\dot{\phi}_l = (R + L/2)\omega \end{cases} \tag{1.17}$$

which allows $\rho$ and $\omega$ to be determined from the wheel velocities:

$$\begin{cases} v_Q = \dfrac{v_r + v_l}{L} = \dfrac{R}{L}(\dot{\phi}_r + \dot{\phi}_l) \\ R = \dfrac{L}{2}\dfrac{v_r + v_l}{v_r - v_l} \\ \omega = \dfrac{v_r - v_l}{L} = \dfrac{R}{L}(\dot{\phi}_r - \dot{\phi}_l) \end{cases} \tag{1.18}$$

Equation situates the ICR on the wheel axis. Additionally, these equations explain two particular properties of unicycle robot movement:

1. If $v_r = v_l \rightarrow$ no turn $\Rightarrow$ ICR is undefined
2. If $v_r = -v_l \rightarrow$ turn "on itself" $\Rightarrow$ ICR =Center
3. If $v_r = (v_l = 0) \rightarrow$ turn "on wheel" $\Rightarrow R = -\frac{L}{2}$ $(R = \frac{L}{2})$

Using these two modes of locomotion, although limited, decouples the movements and provides a simple solution to bring the robot from one posture to another. This simplicity is likely a reason for the success of this type of robot. However, to develop a more refined movement strategy, it is useful to understand how the robot's posture is related to the control of its wheels[8].

### 1.3.13 Conclusion

The study of kinematics is essential for understanding the mechanical behavior of robots and creating control software for mobile robot hardware. It deals with determining a robot's position in a local or global reference frame, providing a mathematical foundation for studying robot movement, control methods, and modeling. The kinematic controller's objective is to follow a trajectory described by its position or velocity profile as a function of time. This involves dividing the trajectory into motion segments of distinct shapes, such as straight lines and circular arcs, and computing a smooth path to guide the robot from its initial to its final position.

The techniques employed may vary depending on whether the robot has a mobile or stationary base. For instance, a differential-drive robot with two wheels must have its motors driven at the same velocity, which can be challenging due to variations in wheels, motors, and environmental factors. On the other hand, a robot with four Swedish wheels, like the Uranus robot, faces an even greater challenge, as all four wheels must be driven at the same speed to move in a perfectly straight line.

## 1.4 Hardware and Software

### 1.4.1 Introduction

The Raspberry Pi is a low-cost, compact computer board designed for educational and DIY purposes. It offers a platform for learning programming and electronics, empowering individuals to create digital projects. The importance of Raspberry Pi in technology lies in its ability to democratize access to computing resources, enabling students and hobbyists to innovate and create technological solutions affordably.

- **History and Evolution:**Raspberry Pi was first envisioned by Eben Upton and his team at the University of Cambridge in 2006. Their goal was to create an affordable computer to encourage students to learn programming. The initial Raspberry Pi model was released in 2012, and several improved versions have been launched since then. The latest major release, Raspberry Pi 4 Model B, introduced in 2019, offers significant advancements, including a faster processor, increased RAM options, dual-display support, Gigabit Ethernet, and USB 3.0 ports.

**Specifications and Capabilities:** The Raspberry Pi 4 Model B is a powerful yet energy-efficient board. It is powered by a Broadcom BCM2711 quad-core processor and offers 2GB, 4GB, or 8GB of LPDDR4-3200 SDRAM for improved performance and multitasking. In terms of connectivity, it features dual-band Wi-Fi, Bluetooth 5.0, Gigabit Ethernet, dual USB 3.0 ports, and dual USB 2.0 ports. It also supports 4K video output via HDMI 2.0 and includes a 40-pin GPIO header for interfacing with external devices and expansion boards. The board supports H.265 and H.264 video codecs, delivering impressive multimedia capabilities. In terms of power, it requires a USB-C supply with at least 3A for optimal performance, with power consumption ranging from 2.5W to 7.6W depending on usage. The official operating system, Raspberry Pi OS, comes with a range of pre-installed software, and the board supports various development environments and programming languages, making it a versatile platform for creators.

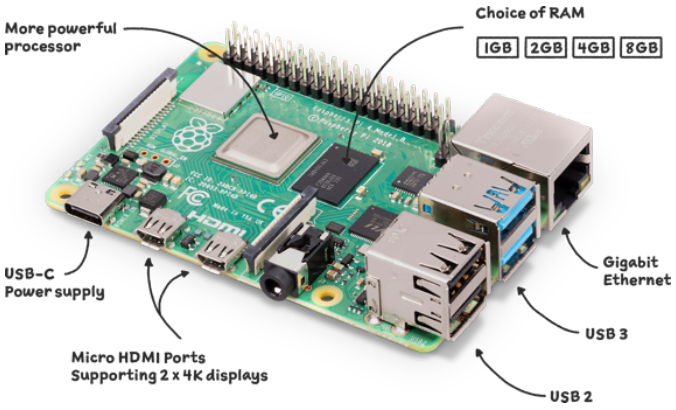



**Figure 1.11** : Raspberry Pi [10]

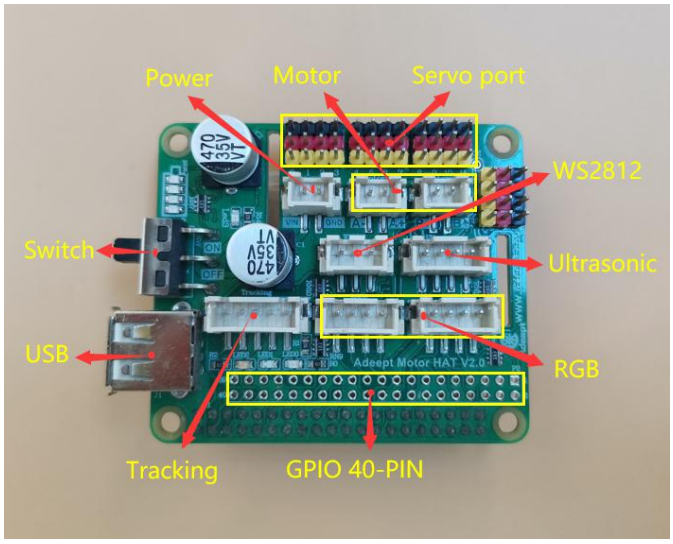



**Figure 1.12** : HATs [9]

### 1.4.2 Elements

1. **Ultrasonic**

   Ultrasonic or sonar sensing is a commonly used technique in robotics for object detection and navigation. Ultrasound refers to a range of sound frequencies beyond human hearing, hence the term "beyond sound" derived from the Latin "ultra" meaning beyond. These inaudible sound waves are transmitted and received by transducers, which convert mechanical energy into sound. While humans cannot hear the ultrasound itself, they may hear the movement of the emitter membrane.

   Ultrasonic sensors operate on the time-of-flight principle, measuring the time it takes for the sound to travel and return. This is similar to how bats and dolphins use echolocation to navigate. In robotics, ultrasound is used for depth sensing, object detection, and robot localization. It is particularly useful for underwater applications, where sonar sends an intense beam of sound to determine the distance to objects or the depth of water.

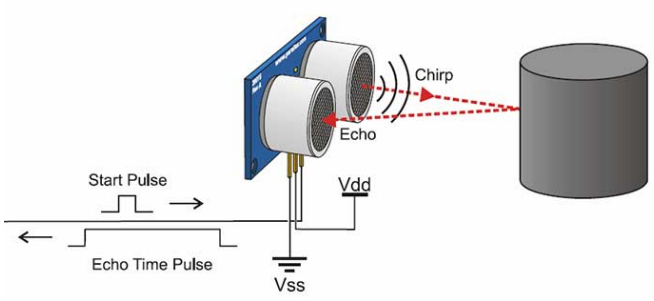



**Figure 1.13** : Ultrasonic-Sensors [11]

2. **Motors DC**

   DC motors are the most common actuators used in robotics, particularly for driving wheels. They are well-suited for this purpose due to their ability to provide rotational movement. DC motors are simple, inexpensive, and widely available in various sizes, making them versatile for different robot designs and tasks.

   The operation of DC motors is based on converting electrical energy into mechanical energy. They use magnets, loops of wire, and current to generate magnetic fields, which interact to turn the motor shaft, resulting in kinetic energy that produces movement. The speed of the motor can be controlled by adjusting the voltage within the appropriate range. A low voltage will result in reduced power, while excessive voltage can lead to increased wear and tear, shortening the motor's lifespan.

DC motors are also used as a basis for creating servo motors, which include additional components such as gear reduction, a position sensor for the motor shaft, and an electronic control circuit. This allows for precise control of the motor shaft's position and direction.

**Figure 1.14** : MotorDC

3. **Servo Motor** Servo motors, also known simply as "servos," are a type of motor that can turn their shaft to a specific position. They are commonly used in toys, such as remote-controlled cars and airplanes, to adjust steering or wing position. Servo motors are made from DC motors by adding gear reduction, a position sensor for the motor shaft, and an electronic control circuit. This allows the robot to move an effector, like an arm, to a desired position.

   Servo motors complement DC motors due to their ability to turn to a specific position, making them useful in both toys and robots. They are particularly relevant in the context of mobile robots, which require the ability to move autonomously from one place to another. This mobility enables them to perform tasks in structured and unstructured environments.


**Figure 1.15** : Motor servo

4. **Camera**

   Cameras are essential sensors for robots, providing them with visual information about their environment. They are biomimetic, meaning they imitate the function of biological eyes. However, synthetic and natural vision sensors differ significantly.

   A standard camera lens with a 512 x 512 pixel resolution can capture a wealth of information. Each pixel acts as a basic element of the image, and the entire lens provides a rich sensory experience for the robot. The combination of cameras and other sensors, such as sonar and lasers, enables robots to perceive and interact with their surroundings effectively.

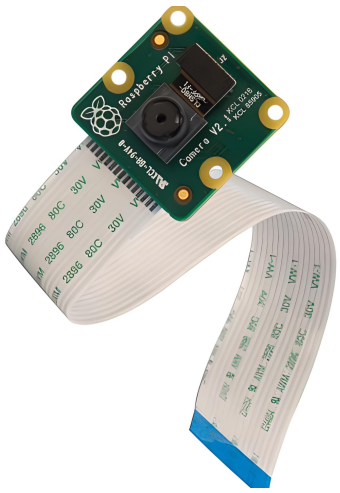



**Figure 1.16** :Pi camera [12]

5. **Power**

   The 18650 5000mah 3.7v battery is a popular choice for those seeking a rechargeable power source for a variety of devices. Offered by brands such as UltraFire, EBL, Skywolfeye, and others, these batteries are often sold in multi-packs. They can power flashlights, headlamps, doorbells, toys, remote controls, and more. The 18650 5000mah 3.7v batteries are conveniently rechargeable, capable of being recharged up to 500 times, making them a cost-effective and time-saving option over the long run.


**Figure 1.17** : 5000mah 3.7v battery

### 1.4.3 The installation of the parts

The robot sensors have been installed in the following manner, along with the installation of the robotic arm's servo motors as well as the motion chain motors.

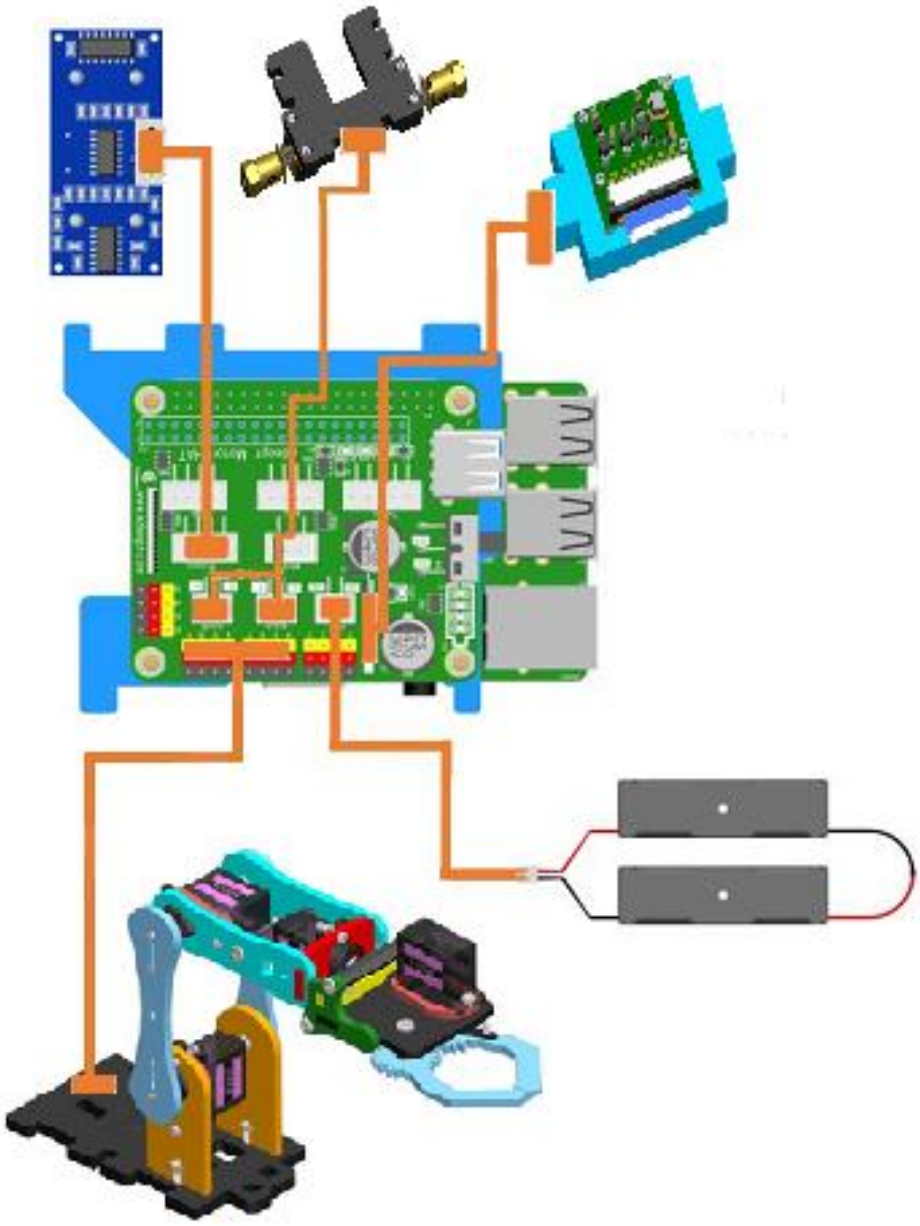


**Figure 1.18** :the parts of robot

### 1.4.4 software and OS

### 1.4.5 Linux And Raspberry Pi OS

Linux is a computer operating system, similar to Microsoft Windows or Apple macOS. Unlike other operating systems, Linux is free. Additionally, Linux users have the freedom to contribute to its development because the source code is open and accessible for study and modification. Furthermore, Linux users can share this software with others.

Linux was originally developed by Finnish programmer Linus Torvalds. It was first released in 1991, and since then, Linux has grown to include an army of developers, tens of thousands of applications and tools, and millions of users.

There are many different distributions of Linux, including Red Hat, CentOS, Fedora, Debian, Ubuntu, and Gentoo.

### 1.4.6 What is Raspbian OS?

Raspberry Pi OS is a Unix-like operating system based on the Debian GNU/Linux distribution for the Raspberry Pi family of compact single-board computers. It was first developed independently in 2012 and has been the primary operating system for these boards since 2013. Raspberry Pi OS is highly optimised for the Raspberry Pi with ARM CPUs. It runs on every Raspberry Pi except the Pico microcontroller.

### 1.4.7 Python3

Python3 is a version of the Python programming language. It is a high-level, general-purpose programming language that is rapidly being accepted and adapted by fortune organisations. Python3 is free to install, use, and distribute. It is also a portable language, meaning that if you have Python code for a platform like Windows, Linux, or Mac, you can run this code on any other platform (which has Python interpreter installed) without changing it. [13]

### 1.4.8 Opencv

OpenCV, short for Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. It was initially developed by Intel but is now maintained by a community of developers under the OpenCV Foundation. It is a free cross-platform computer vision library for real-time image processing. It has more than 2500 algorithms, including classical ones like Support Vector Machines (SVMs) and K-Nearest Neighbors (KNN), and cutting-edge deep learning techniques. These algorithms help with tasks such as object detection, image segmentation, and facial recognition. It supports C++, Python, Java, and MATLAB and works on Windows, Linux, Android, and MacOS.[14]

### 1.4.9 SSH

SSH, or Secure Shell, is a network security approach that automatically encrypts and decrypts data transmitted over a network. It is a client/server protocol that scrambles data to ensure it is only intelligible to the intended recipients. SSH also covers authentication, reliably determining someone's identity before allowing them to log into a remote account. SSH is not a product but a specification for secure communication. It offers a low-cost, software-based solution for protecting data on a network.

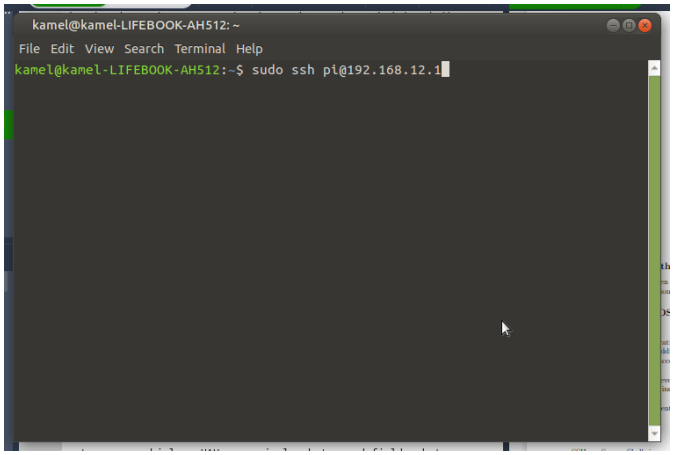



**Figure 1.19** : SSH

### 1.4.10 Counclsion

Raspberry Pi has played a significant role in the technology sphere by providing access to computer resources for students and hobbyists, empowering them to create affordable technological solutions. The prototype was launched in 2012, and since then, Raspberry Pi has undergone numerous enhancements. The latest model, Raspberry Pi 4 Model B, was introduced in 2019 with improved features and better performance.

The chapter then moves on to discuss the various components of a robot. This includes ultrasonic sensors that utilize sound wave technology for object detection and navigation. Subsequent sections explain direct current (DC) motors, commonly used in robotics to provide rotational movement, and stepper motors, which complement DC motors by enabling positional movement, essential for mobile robotics. Cameras also play a vital role in robotics by providing visual information. The chapter discusses a common power source, the 18650 5000 mAh 3.7-volt battery, which is a rechargeable and cost-effective option.

The installation of robot parts is then covered, including sensors and robot motors. The chapter transitions to the software and operating systems section, starting with Linux and Raspberry Pi OS, a dedicated operating system for Raspberry Pi devices. It explores software libraries and languages such as OpenCV for computer vision and machine learning, Python3 as a portable and adaptable programming language, and SSH for secure data transfer.

## 1.5 Object Detection With Yolov8

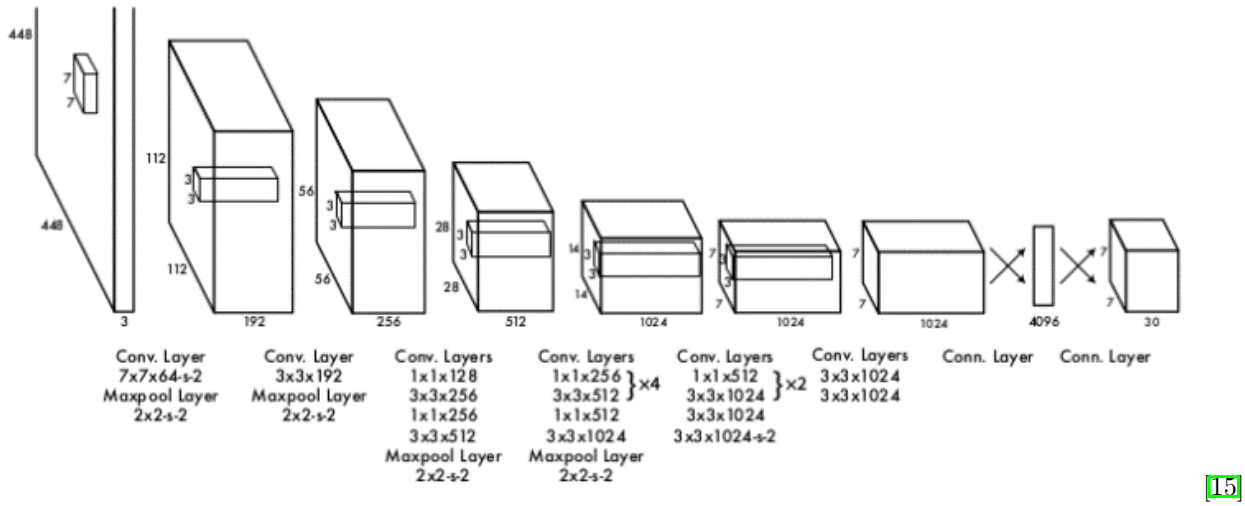
### 1.5.1 Introduction

Computer vision technology using YOLO relies on the YOLOv8 model, which is considered the latest advancement in automated object detection. YOLOv8 analyzes images and videos to detect objects in real-time, accurately identifying their locations and classifying them within the image quickly. Therefore, in this chapter, we will focus on explaining the YOLO model and how to use it for object detection tasks. We will discuss model training, image analysis using YOLOv8, and its application in real-time environments such as waste detection.

### 1.5.2 YOLO

YOLO (You Only Look Once) was developed in 2015 by Joseph Redmon et al. Proposed. The model was developed to address the issues faced by existing object detection models at the time. Fast R-CNN was one of the most advanced models at the time, and while effective, it also faced significant challenges, especially in real-time applications as it took 2-3 seconds to predict an image. In contrast, YOLO optimized this process and only required a single forward pass through the network to make the final prediction, thus enabling real-time object detection.

### 1.5.3 The Architecture



[15]

**Figure 1.20** : YOLO Architecture

The proposed architecture adopts the technique of resizing the input image to dimensions of 448 x 448 while preserving the aspect ratio and applying padding. The resized image is then passed through the Convolutional Neural Network (CNN). The architecture consists of 24 convolutional layers, followed by 4 max-pooling layers and 2 fully connected layers. To reduce the number of layers (channels), a 1 x 1 convolutional layer is applied followed by a 3 x 3 convolutional layer. It is important to note that the final layer of the YOLOv1 model predicts a cuboidal output. This is achieved by generating a tensor of size (1, 1470) from the last fully connected layer and reshaping it to dimensions (7, 7, 30). The architecture uses the Leaky ReLU activation function throughout the network, except for the final layer which uses a linear activation function. The definition of Leaky ReLU can be found in the available references. Batch normalization also helps to regularize the model. Dropout technique is also employed to prevent overfitting.[16]

### 1.5.4 Detection

This architecture divides the image into a grid of S*S size. If the centre of the bounding box of the object is in that grid, then this grid is responsible for detecting that object. Each grid predicts bounding boxes with their confidence score. Each confidence score shows how accurate it is that the bounding box predicted contains an object and how precise it predicts the bounding box coordinates wrt. ground truth prediction.

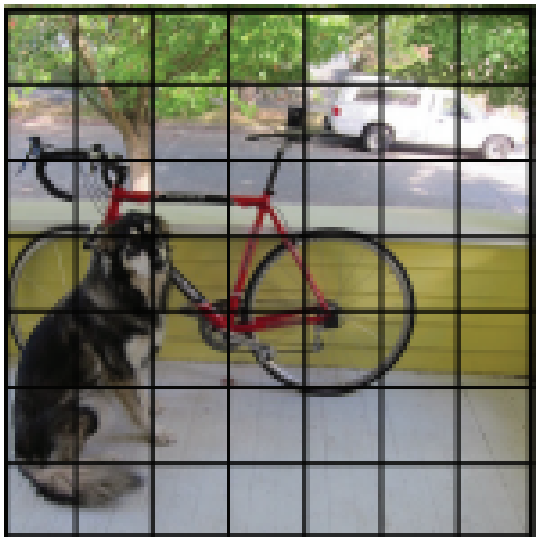


**Figure 1.21** : S × S grid on input [16]

At test time, the conditional class probabilities are multiplied by the confidence predictions of the individual bounding boxes. The confidence score is defined as follows:

$$P_r(\text{Object}) \times 100 \times \text{IOU}_{\text{truth\_pred}} \tag{2.1}$$

the confidence score should be 0 when there is no object exists in the grid. If there is an object present in the image the confidence score should be equal to IoU between ground truth and predicted boxes. Each bounding box consists of 5 predictions: (x, y, w, h) and confidence score. The (x, y) coordinates represent the centre of the box relative to the bounds of the grid cell. The h, w coordinates represents height, width of bounding box relative to (x, y). The confidence score represents the presence of an object in the bounding box. [16]

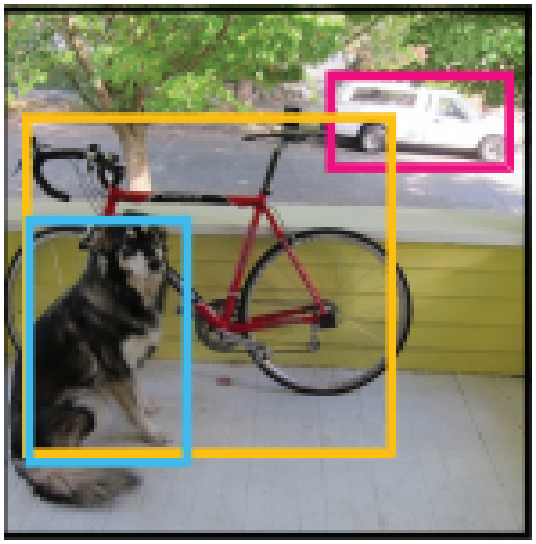


**Figure 1.22** : YOLO single Grid Bounding box-Box [16]

This results in combination of bounding boxes from each grid like this.

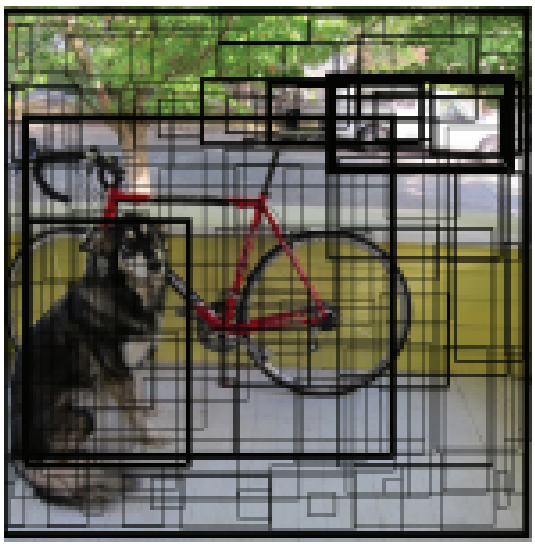


**Figure 1.23** : Bounding boxes + confdence [16]

Each grid also predicts C conditional class probability.

$$Pr(Classi|Object) \tag{2.2}$$

**Figure 1.24** : Class probability map [16]

These probabilities are conditioned on the presence of an object in the grid cell. Regardless of the number of boxes, each cell predicts only one set of class probabilities. These predictions are encoded in the three-dimensional tensor of size S * S * (5*B + C). Then, the conditional class probabilities are multiplied by the individual confidence predictions for each box. [16]

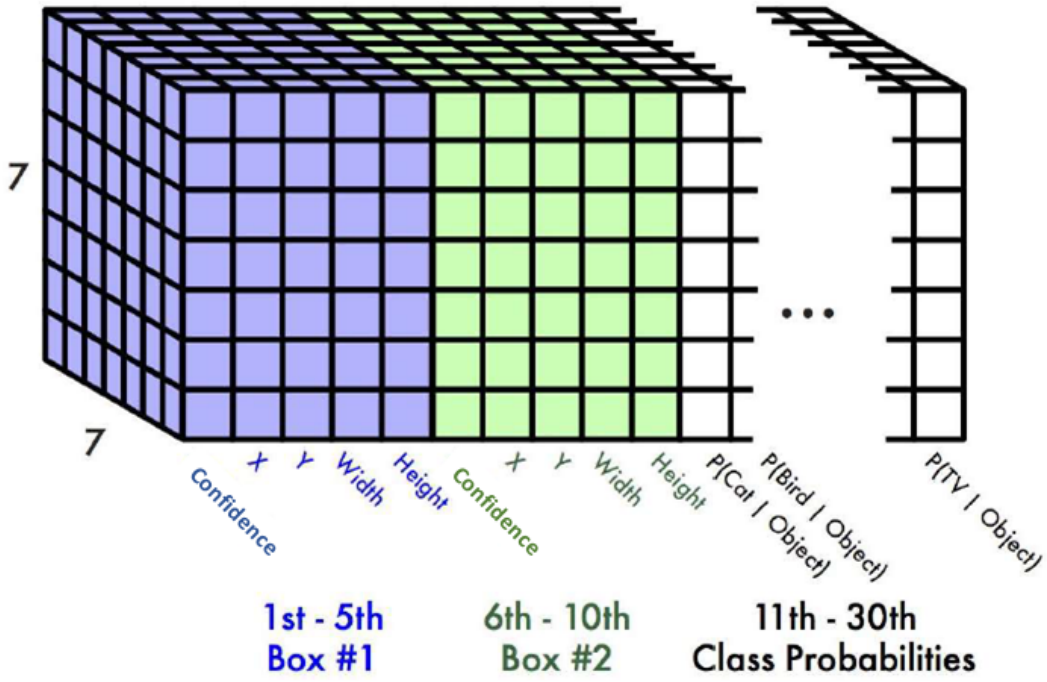



**Figure 1.25** : YOLO output feature map

$$Pr(Class|Object) \times Pr(Object) \times IOU_{truth_pred} = Pr(Class) \times IOU_{truth_pred} \tag{2.3}$$

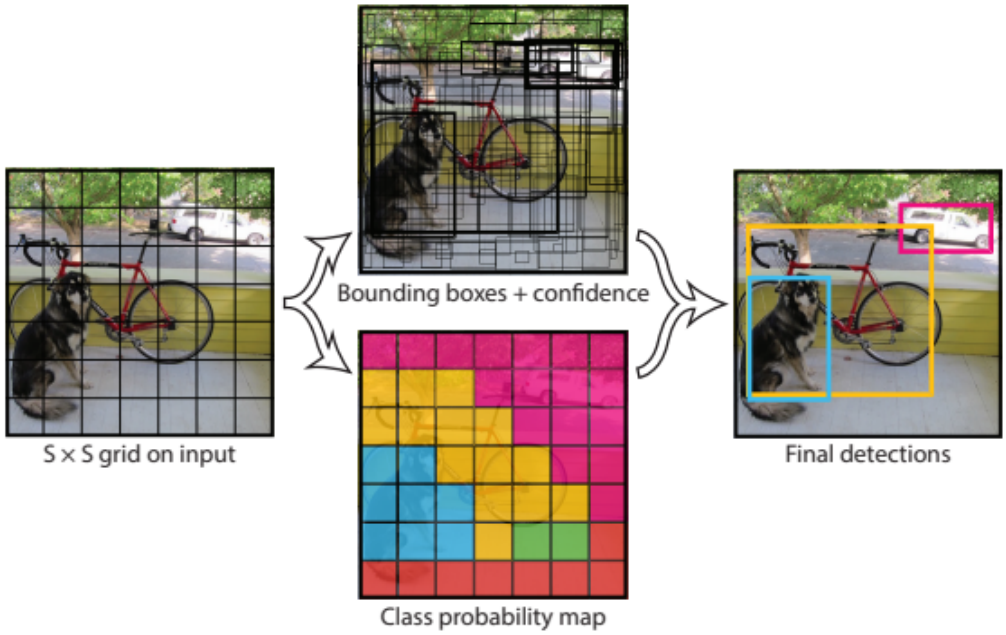



**Figure 1.26**: The YOLO Detection System as a Regression Model [16]

### 1.5.5 Why Yolo ?

The YOLO (You Only Look Once) algorithm was selected as the object detection algorithm for this study. YOLO relies on a single convolutional neural network (CNN) to quickly detect and classify objects, and it was chosen for

its real-time accuracy and the availability of its source code. YOLO's architecture allows for object classification at a rate of 45 frames per second, making it ideal for detection.

YOLO processes the entire image, which reduces background errors compared to other detection methods such as Fast R-CNN, which is considered one of the most effective object detection algorithms. Comparative analysis showed that Fast R-CNN commits nearly three times as many background errors as YOLO. [17] [16]

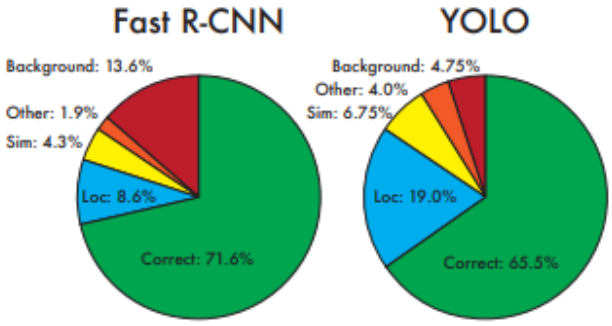



**Figure 1.27** : Comparison between Fast R-CNN and YOLO.[16]

In comparing with other real-time systems, the focus is on accelerating standard detection pipelines. Despite these efforts, previous research has yielded few systems that operate in real time. Researchers compare the YOLO system with the implementation of DPM on a graphics processing unit, where YOLO demonstrated excellent performance, being the fastest method for object detection on PASCAL. It achieved more than twice the accuracy of prior works in real-time object detection. The study also presents other models such as Fastest DPM, R-CNN, Fast R-CNN, and Faster R-CNN, showcasing the accuracy and speed of each and comparing them with YOLO. [17] [16]

| Real-Time Detectors | Train | mAP | FPS |
|---|---|---|---|
| 100Hz DPM | 2007 | 16.0 | 100 |
| 30Hz DPM | 2007 | 26.1 | 30 |
| Fast YOLO | 2007+2012 | 52.7 | **155** |
| YOLO | 2007+2012 | **63.4** | 45 |
| Less Than Real-Time | | | |
| Fastest DPM | 2007 | 30.4 | 15 |
| R-CNN Minus R | 2007 | 53.5 | 6 |
| Fast R-CNN | 2007+2012 | 70.0 | 0.5 |
| Faster R-CNN VGG-16 | 2007+2012 | 73.2 | 7 |
| Faster R-CNN ZF | 2007+2012 | 62.1 | 18 |
| YOLO VGG-16 | 2007+2012 | 66.4 | 21 |

**Figure 1.28** : Comparing the performance and speed of fast detectors [16]

### 1.5.6 Why do we need a Confusion Matrix?

A confusion matrix is an essential tool for evaluating the performance of a classification model. It provides a detailed analysis of true positive, true negative, false positive, and false negative predictions, which facilitates understanding the model's recall, precision, accuracy, and overall class discrimination. This matrix is particularly useful when there is an uneven class distribution in the dataset, as it contributes to assessing the model's performance beyond the basic accuracy metric.

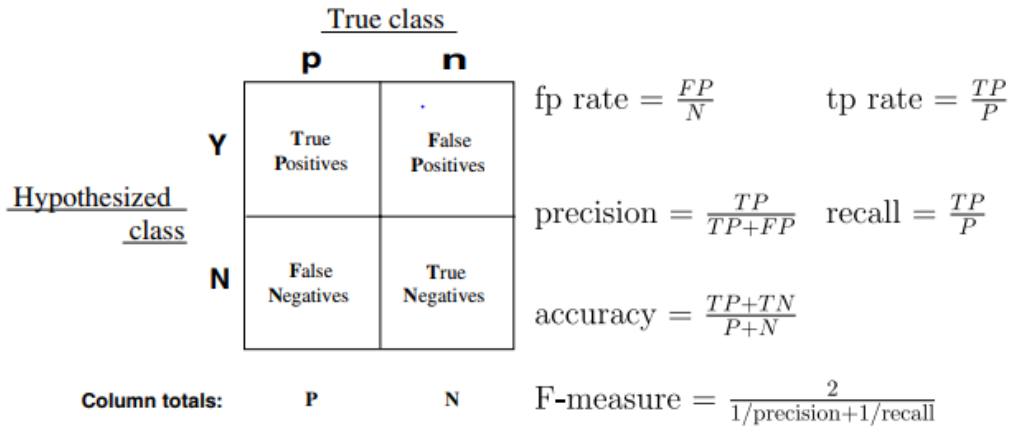



**Figure 1.29**: Confusion Matrix For binary classification citefawcett2006introduction

- **TP (True Positive):** The actual value was positive and the model also predicted a positive value.
- **FP (False Positive):** The model predicted a positive value, but the actual value was negative .
- **FN (False Negative):** The model predicted a negative value, but the actual value was positive .
- **TN (True Negative):** The actual value was negative and the model also predicted a negative value.
- **TP Rate (Sensitivity, Recall):**

$$\textbf{TP Rate} = \frac{\text{True Positives}}{\text{Total Positives}} \tag{2.4}$$

This represents the proportion of actual positive instances that are correctly classified by the model.

- **FP Rate (False Alarm Rate):**

$$\textbf{FP Rate} = \frac{\text{False Positives}}{\text{Total Negatives}} \tag{2.5}$$

This represents the proportion of actual negative instances that are incorrectly classified as positive by the model.

- **Specificity:**

$$\textbf{Specificity} = \frac{\text{True Negatives}}{\text{False Positives} + \text{True Negatives}} \tag{2.6}$$

This represents the proportion of actual negative instances that are correctly classified by the model.

- **Positive Predictive Value (Precision):**

$$\textbf{Positive Predictive Value} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2.7}$$

This represents the proportion of positive predictions that are actually true positives.

### 1.5.7 Yolo release

YOLO (You Only Look Once), a renowned model for object detection and image segmentation, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, YOLO quickly gained popularity due to its high speed and accuracy.

YOLOv2, introduced in 2016, enhanced the original model by incorporating batch normalization, anchor boxes, and dimension clusters. YOLOv3, released in 2018, further improved the model's performance using a more efficient backbone network, multiple anchors, and spatial pyramid pooling. In 2020, YOLOv4 was introduced, introducing innovations like Mosaic data augmentation, a new anchor-free detection head, and a new loss function. YOLOv5 further boosted the model's performance and added new features such as hyperparameter optimization, integrated experiment tracking, and automatic export to popular formats. YOLOv6 was made open-source by Meituan in 2022 and is utilized in many of the company's autonomous delivery robots. YOLOv7 introduced additional tasks such as pose estimation on the COCO keypoints dataset. The latest version of YOLO, YOLOv8 by Ultralytics, builds upon the success of previous versions, introducing new features and enhancements for improved performance, flexibility, and efficiency. YOLOv8 supports a comprehensive range of vision AI tasks, including detection, segmentation, pose estimation, tracking, and classification, enabling users to leverage its capabilities across diverse applications and domains.[18]

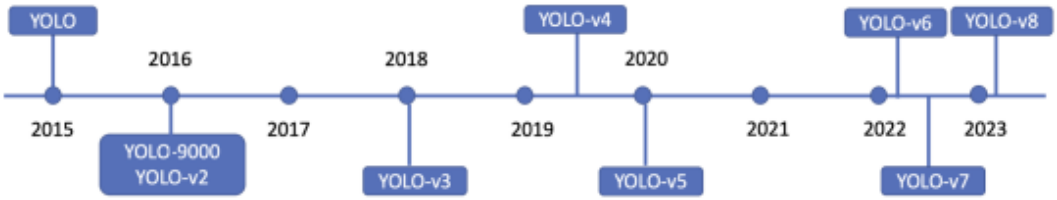



**Figure 1.30**: Timeline of You Only Look Once (YOLO) variants [19]

### 1.5.8 Conclusion

The field of robotics has experienced remarkable growth and transformation since its inception in the 20th century. From the early mechanical creations of William Grey Walter to the sophisticated industrial robots developed by George Devol, robotics has continually evolved, integrating cutting-edge technologies such as artificial intelligence and machine learning. This evolution has enabled robots to perform increasingly complex tasks and find applications in diverse industries, including manufacturing, healthcare, and entertainment.

The study of robotic movement mechanisms highlights the importance of stability and mobility. Whether utilizing wheels or legs, the design and control of these systems draw significant inspiration from biological models. Understanding the abstract principles of movement and developing precise control systems are essential for optimizing the performance of mobile robots in various environments.

Raspberry Pi has played a pivotal role in democratizing access to computing resources, empowering students, hobbyists, and professionals to innovate and build affordable robotic solutions. The continuous improvements in Raspberry Pi models, such as the Raspberry Pi 4 Model B, have further enhanced its capabilities and applications in robotics.

The integration of various components, including sensors, motors, and cameras, is crucial for the functionality of modern robots. Additionally, the role of software and operating systems, like Linux and Raspberry Pi OS, along with programming libraries and languages such as OpenCV and Python3, cannot be overstated. These tools and technologies collectively contribute to the development and operation of advanced robotic systems.

The exploration of the YOLO model for object detection demonstrates the ongoing advancements in computer vision and machine learning. By comparing YOLO with other models like Faster R-CNN and successfully training the YOLOv8n model, we have shown the potential for achieving high accuracy in object detection tasks.

the continuous innovation and interdisciplinary collaboration in the field of robotics promise a future where robots will play an even more integral role in our daily lives and industries. As researchers and developers push the boundaries of what robots can achieve, we can expect to see even more sophisticated and capable robotic systems that enhance efficiency, productivity, and quality of life

# 2 Practical aspect

## 2.1 Example with Yolov8 using Google Colab

### 2.1.1 How to Train YOLOv8 Object Detection on a Custom Dataset

YOLOv8 represents the latest and most advanced iteration in the YOLO (You Only Look Once) model series, developed by Ultralytics. This series is renowned for introducing innovative solutions in the field of object detection, starting from YOLOv3 and culminating in the latest release, YOLOv8. Building upon the foundation laid by previous versions such as YOLOv6 and YOLOv7, YOLOv8 introduces significant improvements in performance and flexibility

### 2.1.2 How to Install YOLOv8

There are two ways to install the YOLOv8 model.

- **Using pip package** You can install YOLOv8 by using the following command in the command interface:

```
!pip install ultralytics
```

  This command will automatically install YOLOv8 and all the necessary dependencies.

- **From source**

  You can install the model from the source on GitHub using the following commands:

```
git clone https://github.com/ultralytics/ultralytics
cd ultralytics
pip install -e
```

For us, we chose the first method

### 2.1.3 Export your dataset

We downloaded a pre-processed dataset from the Roboflow website and uploaded it to the Google Colab platform. Then, we imported this dataset into the Google Colab environment to train the YOLOv8 model.

```
!pip install ultralytics==8.0.196
!pip install --upgrade roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="zM3vzobJdDpMtpqxtSGX")
project = rf.workspace("kongju-university-se97i").project("it-project-recycle-4section")
version = project.version(7)
dataset = version.download("yolov8")
```

The source of the data used : [20]

### 2.1.4 Train YOLOv8 on a Custom Dataset

After downloading the dataset into our notebook in Google Colab, we will begin the training process using the following command.

```
!yolo task=detect \
mode=train \
model=yolov8n.pt \
data='/content/it-project-recycle-4section-7/data.yaml' \
```

```
epochs=20 \
imgsz=640
```

Here are the results of training the waste detection model using YOLOv8:

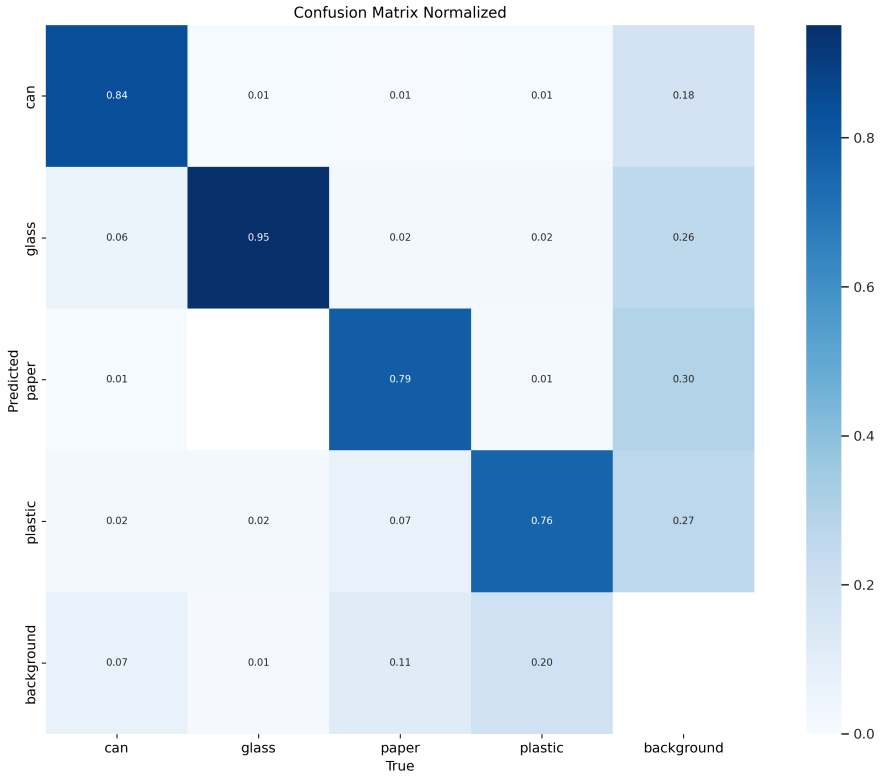



**Figure 2.1.1** : The confusion matrix returned after training

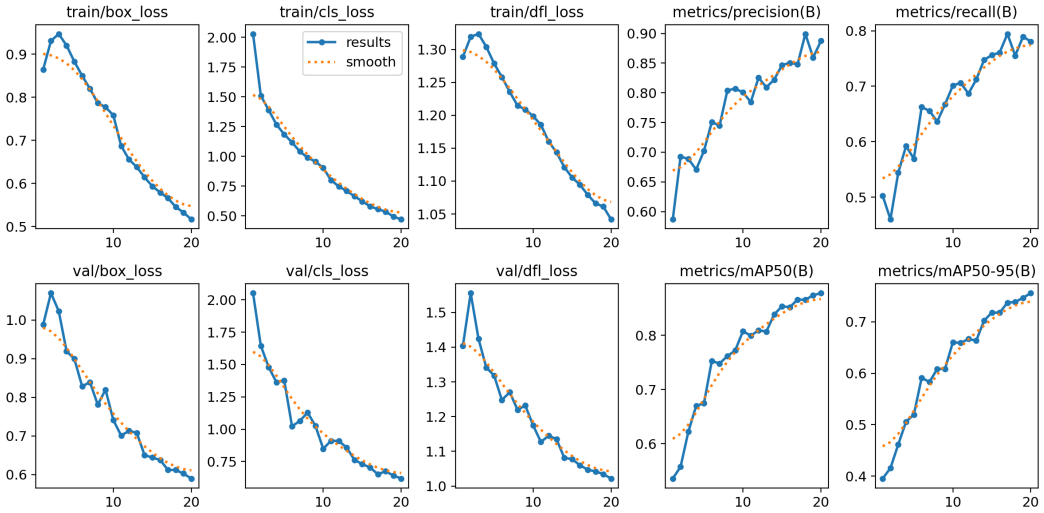



**Figure 2.1.2** : Key metrics tracked by YOLOv8

The provided graphs show the progress of training a machine learning model on a specific dataset. There are eight graphs, each displaying a different measure of progress and performance. In the first set of graphs (`train/cls_loss`,

`train/dfl_loss`, `val/cls_loss`, and `val/dfl_loss`), classification loss (`cls_loss`) and difference loss (`dfl_loss`) are shown for the training and validation sets. The actual results are represented by solid lines, while the smoothed results are shown by dashed lines. It can be observed that the loss decreases over time (the x-axis represents the number of epochs), indicating improvement of the model. In the second set of graphs (metrics/precision/recall and metrics/mAP50/mAP95(B)), performance metrics such as precision, recall, and mean average precision (mAP) at thresholds of 50% and 95% for verification are displayed. It can be noticed that these metrics improve over time, suggesting that the model becomes more accurate and has better recall in its predictions. Overall, the graphs indicate that the model improves over time and achieves better performance in terms of loss, precision, recall, and mean average precision. However, it should be noted that this analysis should be performed cautiously, and factors such as differences in the dataset, model configuration, and verification of results on an independent test set should be taken into account.

### 2.1.5 Validate with a new model

Upon completion of training, the new model's validity is checked on images it has not seen before. Therefore, when creating a dataset, we divide it into three parts, one of which we will now use as a test dataset.

```
#val
!yolo task=detect \
mode=val \
model='/content/drive/MyDrive/weights/best.pt' \
data=/content/it-project-recycle-4section-7/data.yaml
```

```
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3006428 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/it-project-recycle-4section-7/valid/labels.cache... 1535 images, 1 backgrounds, 0 corrupt: 100% 1535/1535 [00:00<?,
WARNING ⚠ Box and segment counts should be equal, but got len(segments) = 645, len(boxes) = 2004. To resolve this only boxes will be used
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded co
  self.pid = os.fork()
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 96/96 [00:19<00:00,  4.91it/s]
                 all       1535       2004      0.888      0.781      0.878      0.756
                 can       1535        474      0.945      0.801      0.925      0.826
               glass       1535        473      0.872      0.926      0.949      0.834
               paper       1535        589      0.896      0.702      0.851      0.676
             plastic       1535        468      0.838      0.694      0.787      0.687
Speed: 0.6ms preprocess, 4.0ms inference, 0.0ms loss, 1.9ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

**Figure 2.1.3** : YOLOv8 model evaluation results

### 2.1.6 Predict with a custom model

Here, we will utilize the previously trained model to predict data. Therefore, we will employ the following command.

```
!yolo task=detect \
mode=predict \
model='/content/drive/MyDrive/weights/best.pt' \
conf=0.5 \
source='/content/drive/MyDrive/file4/test_6.jpg'
```

- **Note :** The codes used were taken from [21] and modified according to our needs.

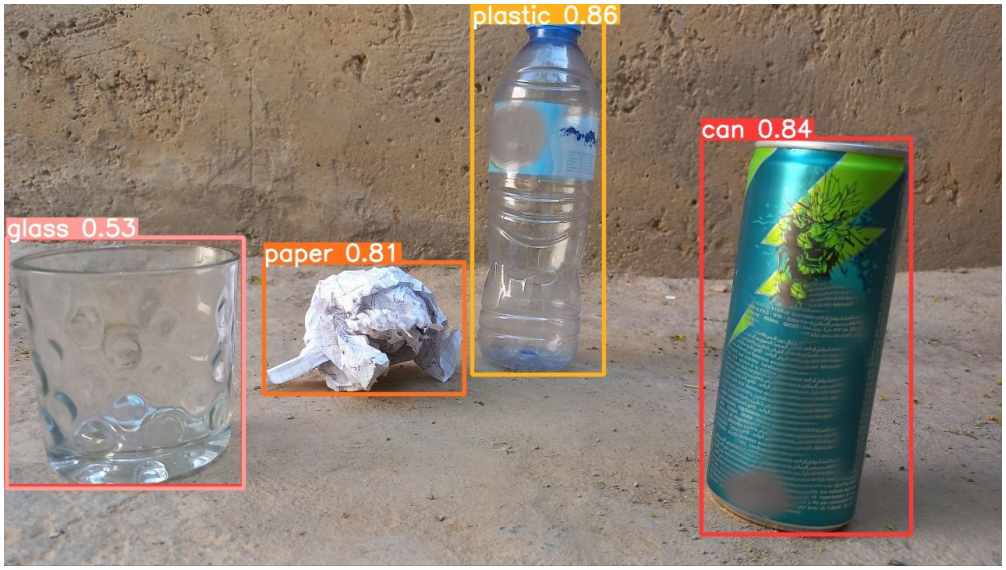



**Figure 2.1.4** : Waste Detection and classe using a Pre-trained Model

The results in the image demonstrate how the "yolov8" model can automatically identify and classify objects.

1. **Glass cup:** It was classified as "glass" with a confidence score of 0.53. This means the system believes with a 53% probability that the cup is made of glass. The score might be lower due to factors like lighting or angles that make classification less clear.

2. **Torn paper:** It was classified as "paper" with a confidence score of 0.81. This means the system believes with an 81% probability that the object is made of paper. The high score indicates that the system clearly sees the distinctive features of paper.

3. **Plastic bottle:** It was classified as "plastic" with a confidence score of 0.86. This means the system believes with an 86% probability that the bottle is made of plastic. The high score indicates that the system can recognize the properties of plastic well.

4. **Can:** It was classified as "can" with a confidence score of 0.84. This means the system believes with an 84% probability that the object is a can. The score might be slightly lower due to the complexity of the can's design and the potential presence of other factors that affect classification.

.

### 2.1.7 Conclusion

we provided an overview of the YOLO model, compared it to Faster R-CNN, and reviewed the key stages of its development. Additionally, we selected the YOLOv8n model and trained it using a pre-prepared dataset, achieving good results in object detection.

.

## 2.2 Particle Work

In mobile robotics, there are two main programming approaches: sensing-planning-action and sense-action. The first approach, sensing-planning-action, allows the robot to sense and perceive the world, plan optimal actions, and then execute those actions. The robot Shakey was an example of this approach, and most humanoid robots like Atlas or Pepper use this method. However, this approach requires significant computational power, especially when dealing with complex or unknown environments. In hostile environments, the delay caused by the decision-making process can negatively impact the robot's effectiveness and even its survival.

The second approach, sense-action, is inspired by animal behavior and neuroscience. It establishes a semi-direct link between sensor inputs and motor outputs, resembling automatic reflex responses. The first example of this approach was the robotic tortoise, which ignored the planning process to ensure faster reactions. This method is suitable for situations where speed is more important than accuracy, such as wild animals' escape responses to potential threats, whether real or perceived. The drawbacks of this approach are its reliance on environmental cues and its unpredictability in unfamiliar environments. However, it does not require sophisticated hardware.

The third approach is a hybrid of the previous two: using sensing-planning-action for complex tasks that require planning, such as path selection for navigation, and sense-action for tasks that require quick reactions, such as obstacle avoidance, heat source detection, or energy conservation. In our work, we employed the third approach as it suited the hardware capabilities we had available.

### 2.2.1 Target Setting

The robot acts as a camera and identifies the object using artificial intelligence algorithms. If the target is present, the robot determines its location in the image. If the target is not found, the robot rotates to the right angle and repeats the process

### 2.2.2 Movement Path

The robot divides the scene into three parts: right, left, and center. Based on the coordinates obtained from the initial stage, the robot determines the direction to turn and moves forward. The robot stops moving if it detects the target at a specified distance using sound waves

### 2.2.3 Groping

After the robot reaches a certain distance from the target, it lowers its arm and captures the target. The robot determines the end of the capture when the arm reaches the initial position and when the sound signal disappears.

### 2.2.4 Disposal:

In this process, the waste is picked up and placed in a cart at the back of the robot, in a programmed motion.

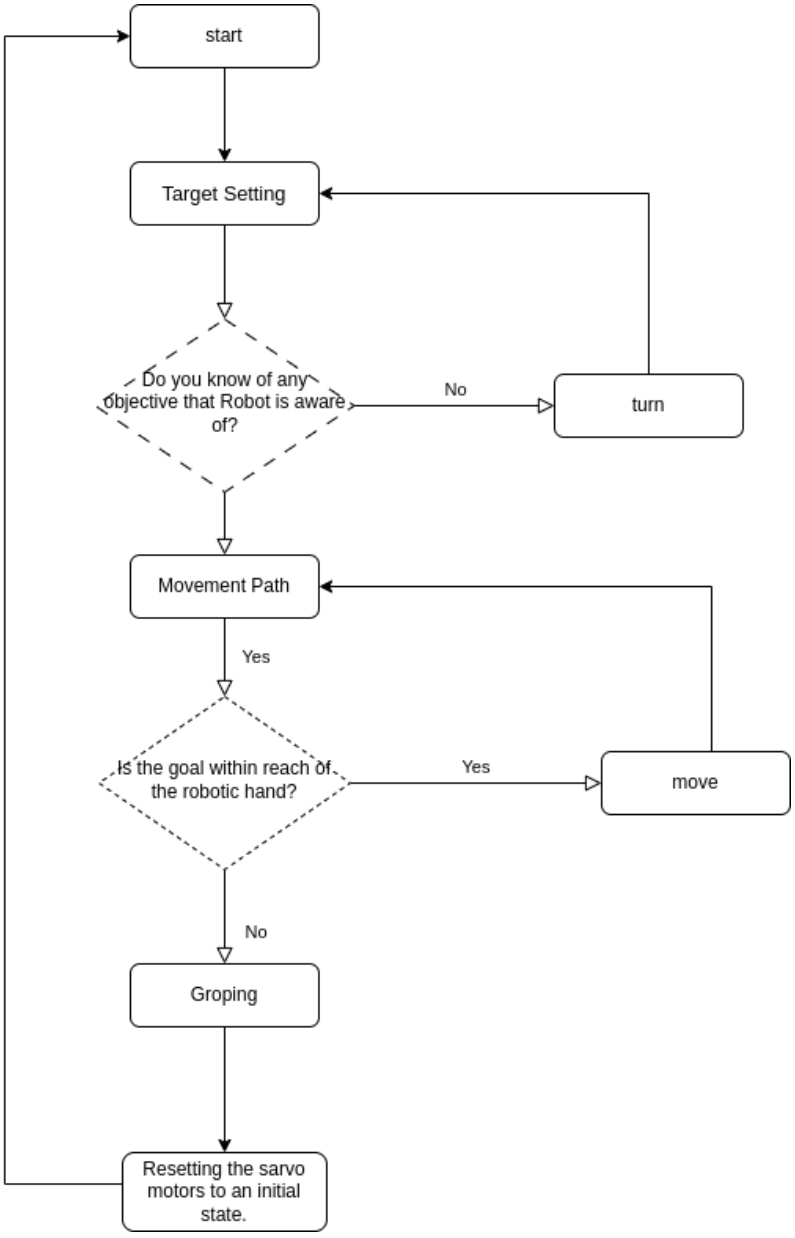



**Figure 2.2.1** :Algorthem of robot mobile

## 2.3 Initial Setup for Raspberry Pi

### 2.3.1 Operating System Installation

First, download the Raspberry Pi OS from the official website https://www.raspberrypi.com/software/operating-systems/ . We recommend downloading the 64-bit version as it is compatible with libraries such as TensorFlow and PyTorch, which may not work with the 32-bit versions Since our current work on the robot involves a standalone system, having a graphical user interface is not a major priority. Considering many unnecessary programs, we suggest downloading the server version as it is lighter, more stable, and free of extra programs. For our project, we used the server version.

Raspberry Pi OS Lite ,64−bit , Kernel version:6.6 , Debian version:12(bookworm) , Size: 414MB

### 2.3.2 Installation Method

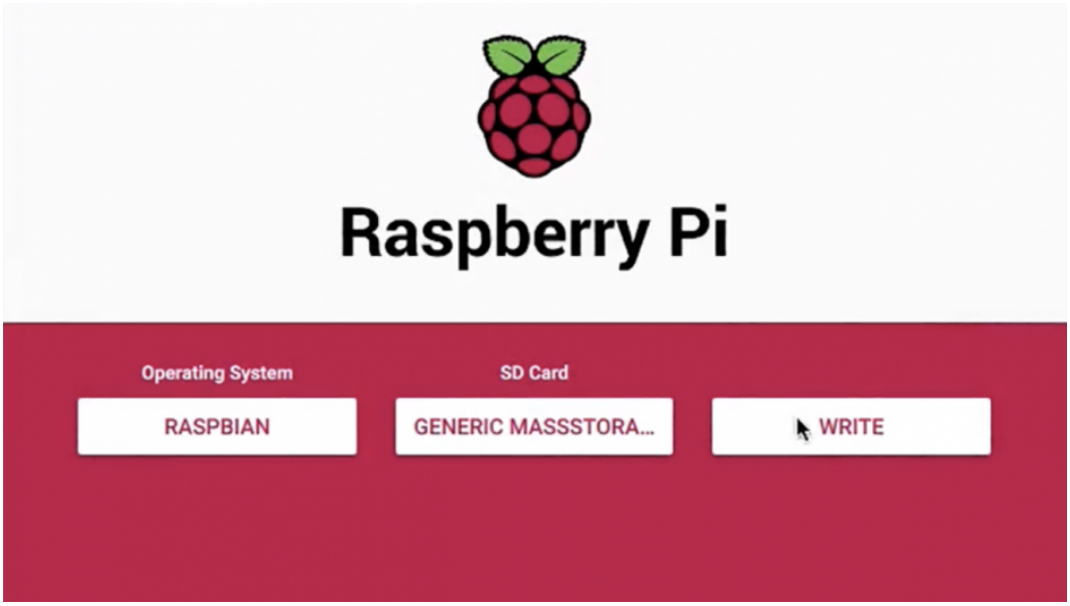
First, after downloading the version, install the image software, which can be downloaded from the following site: https://www.raspberrypi.com/software/. Open the software and select the desired version using the custom option. Next, specify the SD card. It is preferable to use an SD card with a speed of 10 for faster booting and fewer issues with the software. Additionally, it should be larger than 16 GB to accommodate the programs.

Before preparing the SD card, you must:

- Enable SSH from the settings, with a username and password.
- At the same time, set the network name and password for connection.

The goal is to connect directly to the robot after booting and secure access to the network and the secure protocol for control.

In the upcoming explanation, we will write a program so that if there is no recognized network, the robot will create an ad-hoc network, allowing for connection.



[22]

**Figure 2.3.1** :Raspberry Pi Imager

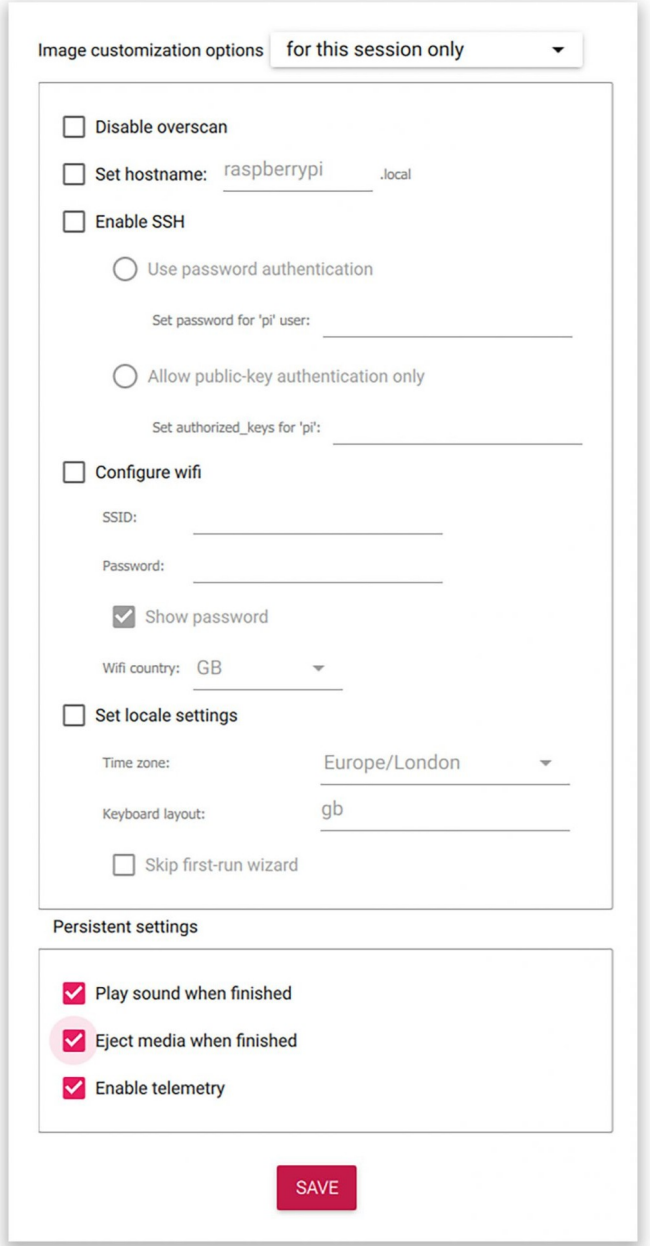



**Figure 2.3.2**:Raspberry-Pi-Imager-large-scaled

### 2.3.3 Connecting to the Robot

First, power on the robot either via batteries or the USB-C port. Initially, we recommend using the port as it provides better power stability. In some cases, the specifications for the charging port may differ, so regular computer ports are good alternatives.

When the robot's system boots up, it will automatically connect to the network configured during the installation phase and will be assigned an IP address by the router, such as:

```
192.168.x.y
```

The x and y can vary between routers, but the first two numbers indicate that you are on the local network.

To find the robot's IP address: First, connect to the network via an internal computer, then type the following command :

```
ip a
```

if your system is Linux (we prefer Linux): On an Android phone, you can find it using an app: Once you have the IP address, go to the terminal or CMD and type:

```
ssh pi@192.168.0.12
```

The SSH command is a tool for connecting via the SSH protocol. 'pi' here is the username entered during the installation phase. The number after '@' is the robot's IP address.

When the connection starts, the first thing you will see is a prompt to enter the password set during the installation process - you have 3 attempts before the connection is terminated. Once connected, the robot's terminal will appear, allowing you to control the robot. The commands here are in Linux, but this summary will explain:

```
ls      % To list files in the directory
cd      % To change directories
python3 file.py  % To run a file named file.py
clear   % To clear the screen
```

Next, install the libraries. First, update the repositories and existing programs:

```
sudo apt update
sudo apt upgrade
```

Now, install the essential libraries for the robot and HAT (this may take some time). We will focus on OpenCV2, Ultralytic, and basic Python3 essentials. To simplify, just run `setup.py`, which we programmed to install the essential libraries.

At this point, the robot is ready for programming and operation.

### 2.3.4 Librarys and Config

In this project, we utilize a Raspberry Pi and various components to create a mobile robot capable of performing tasks autonomously. We begin by enabling the camera through the command "sduo raspi-config" and setting the image format. We then navigate to the interface and enable the camera, granting read permissions. It's important to note that the operating system allows only one program or code to access the camera at a time. If a second read request is made, the system will deny access, indicating that the camera is busy. To avoid conflicts, it's advisable to be cautious about task overlap.

After installing the necessary software libraries, we proceed to write the basic code that defines the control outputs for the robot's motors, including both continuous and servo motors. This code is contained in the `"move.py"` and `"servo.py`" files, where we utilize the `Adafruit PCA9685` library to create a PWM object that controls the servo motors. We also use the `RPi.GPIO` library for direct and fast control of the PWM signals. By specifying the positive and negative outputs, we can determine the rotation direction and set the frequency to control the speed.

Our algorithms rely on rotation due to the high stability provided by the chain mechanism, despite the modeling and control inaccuracies it introduces. We choose to rotate around the robot's axis as it is simpler and more efficient. Since we're aiming for maximum speed, and the desired angle is fixed, we control the rotation by adjusting the time. The desired angle is determined through image processing as described later.

### 2.3.5 Why use TensorFlow Lite instead of TensorFlow?

TensorFlow is an open-source framework used for developing and deploying machine learning and AI models. It is fundamental to the functioning of YOLO (You Only Look Once), but it is designed for large-scale development and training. TensorFlow operates on powerful servers with support for advanced hardware like GPUs and TPUs, making it very suitable for the training and model creation process. However, it can be slow during the inference or execution phase.

To address this issue, especially on resource-constrained devices like the Raspberry Pi, the TFLite format supports techniques such as quantization to reduce model size and increase execution speed. TFLite provides an optimized runtime environment for deploying models on edge devices with limited resources.

Models trained using TensorFlow can be easily converted to the TFLite format using the following command:

```
model.export(format="tflite")
```

### 2.3.6 Image Processing and Move to Target

The first step in our algorithms involves capturing an image from the camera and processing it. We use the `cv2` library to read the image, which may require noise reduction or cropping to facilitate object recognition. The image is then converted into a `NumPy` array for easier manipulation, especially when utilizing AI libraries. To optimize memory usage, we apply resize algorithms from `cv2` to reduce the image array's size while retaining essential information.

At this point, we can process the image array using `YOLO`(You Only Look Once) object detection, which identifies objects within the image and determines the center of the bounding box for each detected object. Based on the $x, y$ coordinates of the center, we calculate the rotation angle. We divide the frame into three parts: the right, left, and middle sections. When the center of the object's bounding box is returned, we determine its position within these sections. If the center is in the right section, the robot rotates right; if it's in the left section, it rotates left; and if it's in the middle, it moves forward for a set time of 3 seconds before capturing another image.

This simple algorithm, based on rotation and forward movement, ensures the robot's ability to navigate towards a target or multiple targets. However, a problem arises when no specific target is identified. In such cases, the robot assumes the center coordinates as $0, 0$ and operates based on the same algorithm. This results in an automatic left turn at the highest possible value until a target is detected, a process we call "discovery."

Up to this point, we've been using only image information for robot control. With the addition of an ultrasonic sensor, we introduce two new capabilities: stopping at the target and avoiding collisions, as well as obstacle avoidance. The sensor's readings divide the area around the robot into three fields: the nearest field, or the avoidance/evasion field, ranging from 0 to $4cm$ centimeters; the work field, ranging from $cm$ centimeters to the maximum reachable distance by the robot's arm; and the target field, which is beyond the work field. Regardless of the presence of a target, if an object is detected in the avoidance field, the robot will back up for 1 second and then make a slight turn to the left to clear the obstacle. This allows the robot to move into the second field, the work field. In practice, measurements from ultrasonic sensors often have a certain level of noise depending on the sensor quality, resulting in illogical readings for a short moment in time. This may lead to errors in reading, causing collisions with the target or illogical behavior from the robot. We addressed this issue by taking the average of 100 measurements within a duration of less than a second, which positively impacted the robot's speed of access, making its movement more orderly and stable.

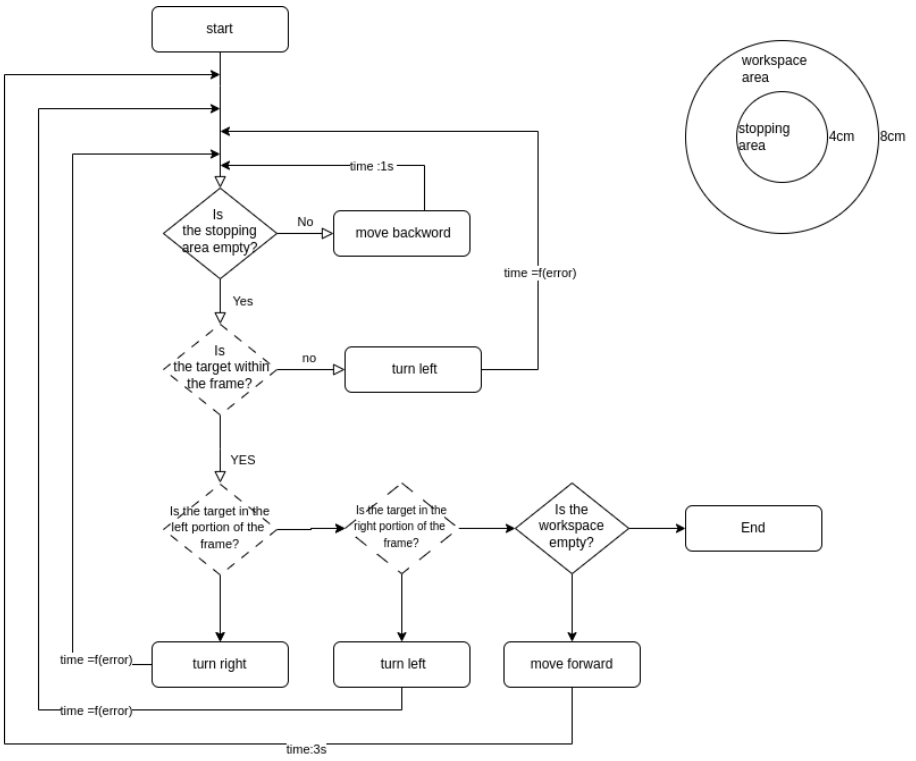



**Figure 2.3.4** : Algorthem of movemnt

**NOTE : time=f(error):**
We have a DC motor designed to operate at a constant maximum speed, and the challenge is to control the motor's rotation to stop at a specific angle. The only option we have is to precisely adjust the rotation time. We have developed a function named "ferror" to calculate the time required for the motor to reach the desired angle. The function takes two parameters: "a," representing the center of the frame on the horizontal axis, and "b," representing the current position of the center of the square on the same axis.

The function utilizes a coefficient "k," which is a small constant value (in this case, 0.005), to determine the time needed for the motor to reach the target angle. The difference in pixels between "a" and "b" is calculated, and this difference is multiplied by the coefficient "k" to obtain the required time.
In the work field, if a target is detected, the robot's arm will move towards it using the `MGI` function. The distance$x$ is measured using the ultrasonic sensor, and an approximate height can be inferred by moving the sensor slightly upward and observing the change in measurement. Knowing the distance $x$ and $\tanh th_u$, we can estimate the height of the target. If no target is present in the middle section, any object detected in the work field is considered an obstacle to be avoided, triggering the rotation and forward movement as defined in the algorithm.

While these steps enable the robot to handle a range of targets, they also present some challenges. The algorithm does not account for reflections or transparent objects like glass, which can lead to false target detection. Additionally, the current version of the algorithm does not guarantee complete coverage or rational movement within the operating area.

### 2.3.7 Groping Algorithm

Capturing objects is a challenging problem in robotics; grasping a ball is not the same as holding a cup of water. A robot must determine the topological shape of the target object and identify the grasping points to successfully groping it.

A capture attempt may fail, so the robot must know when and where to stop and decide whether to continue the movement or not. There are two main approaches to solving this problem: the first is theoretical and relies on topological shape analysis, followed by feeding this information into mathematical algorithms to determine the grasping points. This method requires significant effort and mathematical ability to analyze small shapes and generalize them to more random and diverse forms.

The second solution is machine learning, which involves creating an environment with numerous robotic arms

that attempt to lift objects. Data from each attempt is collected, and an AI model is trained based on this data to perform the capturing task. While this approach seems more intuitive than the first, it is highly time-consuming, taking approximately ten years to master, making it impractical.

Moving away from programming and algorithms, the most prevalent solutions involve changing the type and mechanism of capture, deviating from the pincer principle. Solutions such as suction or sensors, like those found in octopuses, are explored to tackle this problem.

In our robot, we work with the pincer principle, and as mentioned in the chapter, the arm operates on a two-dimensional surface. The target's coordinates are ($Xt$, $Yt$), and with an oversimplification, grasping at a single point is sufficient to hold and capture the object. The process is as follows:

The first step in the algorithm is to measure the distance $Xt$ using an appropriate sensor, such as an infrared or ultrasonic sensor. This distance represents the target or the point we want to reach. In the second step, the motor's angle, $theta_u$, is increased in increments of 5 degrees, and the distance r is measured from the new starting point. The aim here is to find the distance r that corresponds to the angle $theta_u$.

In the third step, a comparison is made to determine if the distance r is greater than the target distance $Xt$. If r is larger, we proceed to the fourth step. If r is less than or equal to $Xt$, we go back to the second step to increase the motor's angle again.

In the fourth step, Y is considered equal to r multiplied by the sine of the angle $theta_u$. Y represents the vertical distance or height in the context of this algorithm. By multiplying r by the sine of the angle, we can calculate the height corresponding to the horizontal distance $Xt$.

This algorithm aims to find the appropriate angle $theta_u$ that corresponds to the target distance $Xt$. By incrementally increasing the angle and measuring the distance at each step, we can identify when the distance r exceeds the target. At this point, we can utilize trigonometry to calculate the height or vertical distance corresponding to the desired horizontal distance.

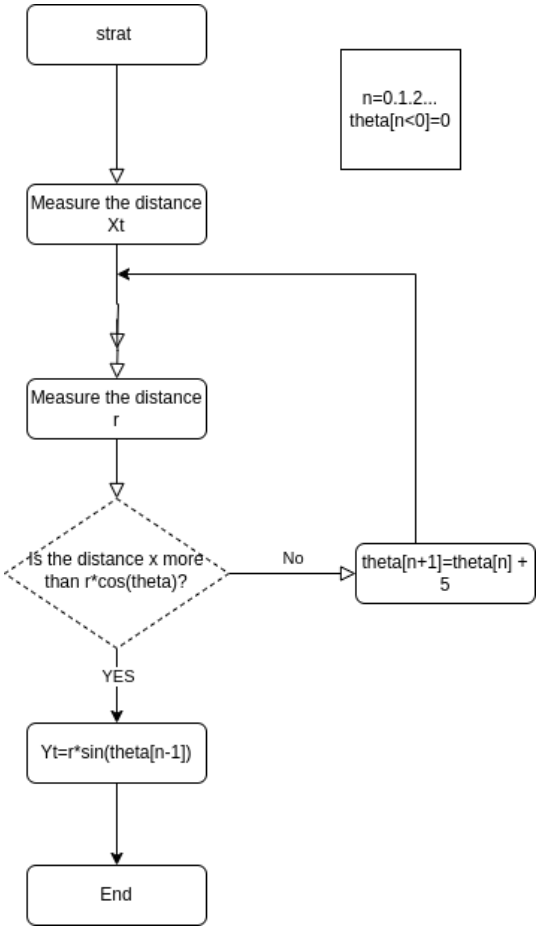



**Figure 2.3.5** : $Y_t$ Search Algorithm

### 2.3.8 Remote Connection Without a Network

We previously discussed how to connect and control the robot using the SSH protocol, but this requires a shared network. If no network is available, the robot is completely isolated. One solution is to create a dedicated robot network that enables remote access. In Linux, you can use the following command:

```
sudo create_ap wlan0 eth0 wifi-name wifi-pass
```

Replace wifi-name and wifi-pass with the desired network name and password, respectively. This command will launch a network with the specified name and password.

However, to write this command on the robot, you need to be connected to it via SSH, which brings us back to the initial point. The solution is to configure the robot to execute this command automatically upon boot without user intervention.

### 2.3.9 Boot Program File

This file will serve several purposes: executing the main program and algorithms, initializing the robot to its initial state, enabling remote connection, and launching a network for SSH access. First, create the file, either as file.sh or file.py, depending on your preference. In our custom robot, we used the Python language.

To write commands from a Python script to the terminal, we'll utilize the os library, specifically the os.system() function, which executes the command within its argument. After writing the file, open the terminal and run the following command:

```
sudo crontab -e
```

This command will open a file in the terminal's text editor. Navigate to the end of the file using the arrow keys and add the following line:

```
@reboot python3 \path\file.py &
```

The @*reboot* indicates that the command should be executed upon boot. The path specifies the location of the file, and the & symbol at the end ensures that the robot doesn't wait for the task to complete before continuing with other tasks.

## 2.4 Experimental Results

To analyze the results and identify issues, the tests for the YOLO model were separated from the hardware tests. A valid reason for this approach is to pinpoint the exact source of any errors.

### 2.4.1 Speed Analysis

Initially, an image of a water bottle was captured to be recognized by the system. According to the observed results, the YOLOv8n model took 1.5 seconds, while the tflite32 version took approximately 1.1 seconds, and the tflite16 version took around 1.0 second.

**Figure 2.4.1** : Testing conditions.

**Figure 2.4.2** : Image captured during the experiment.

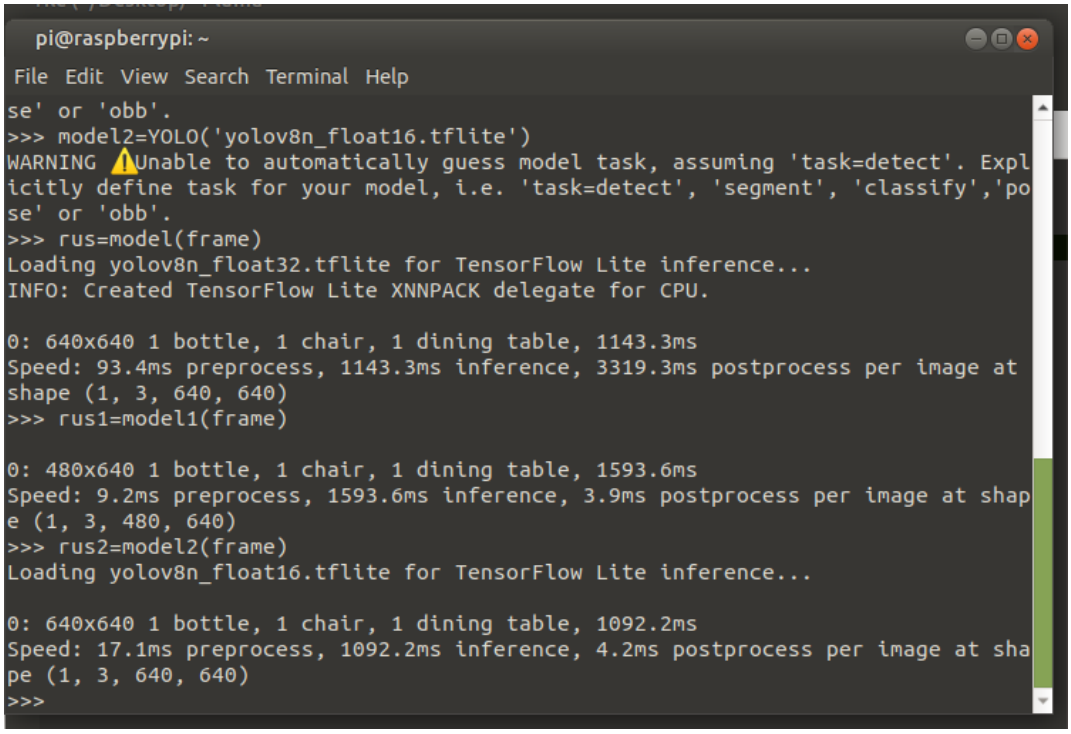



**Figure 2.4.3**: Robot performance with different models.

It is important to note that the TFLite format modifies the dimensions of the image, making it square. This is achieved by increasing the image size from 480 to 640 pixels.

### 2.4.2 Image Quality

It was also noted that the image appeared slightly dark, despite the appropriate lighting conditions during the capture, as described. The reason for this issue, as mentioned earlier in this chapter, is that the camera was operating at 32-bit while the processor and operating system were 64-bit, resulting in empty upper values.

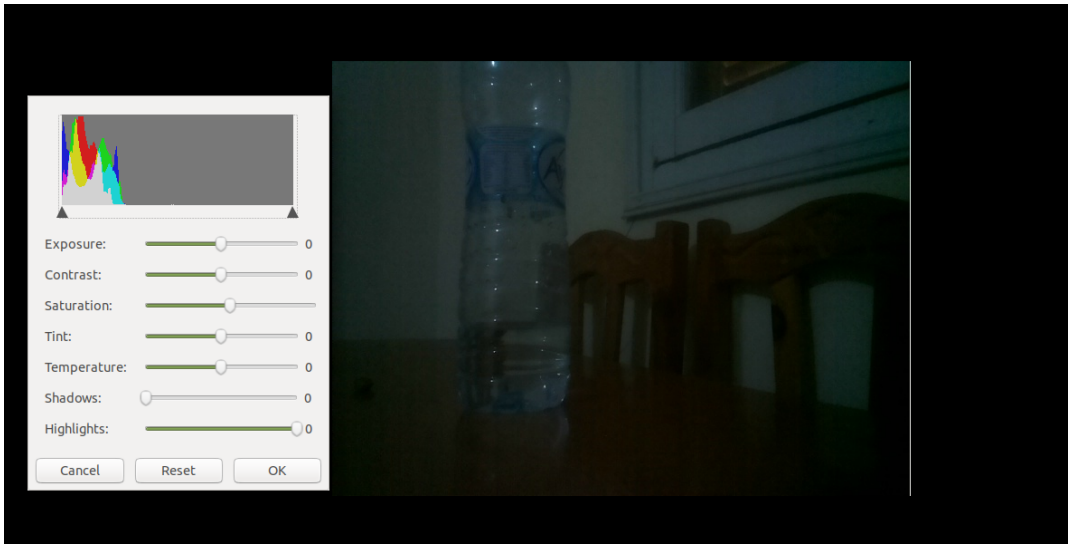



**Figure 2.4.4** : The color spectrum of an image.

To address this problem, we applied a filter to adjust the image and enhance its quality

```
v_equalized = cv2.equalizeHist(v)
```

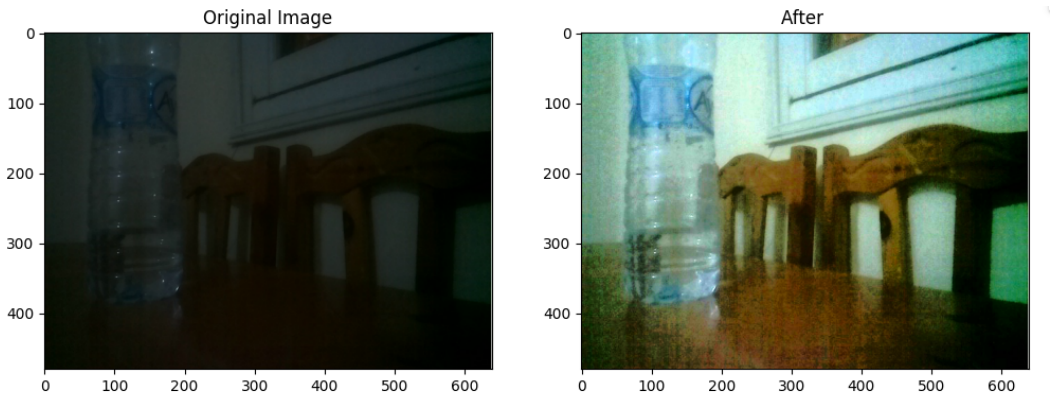



**Figure 2.4.5** : Image processing with a filter.

### 2.4.3 Testing the Robot for Distance Determination.

In this exciting experiment, we will test the capabilities of an ultrasonic sensor in measuring distance and range. We have designed a simple yet effective setup, placing a robot stationary with an obstacle exactly 30 centimeters in front of it. The objective is to gauge the sensor's accuracy in determining the distance to the obstacle without any movement from the robot.

Once the experiment commences, the ultrasonic sensor emits sound waves and measures the time it takes for them to bounce back from the obstacle. Since the speed of sound is constant, the distance can be easily calculated based on the time it takes for the signals to return to the sensor.

**Figure 2.4.6** : Image of Testing the Robot for Distance.

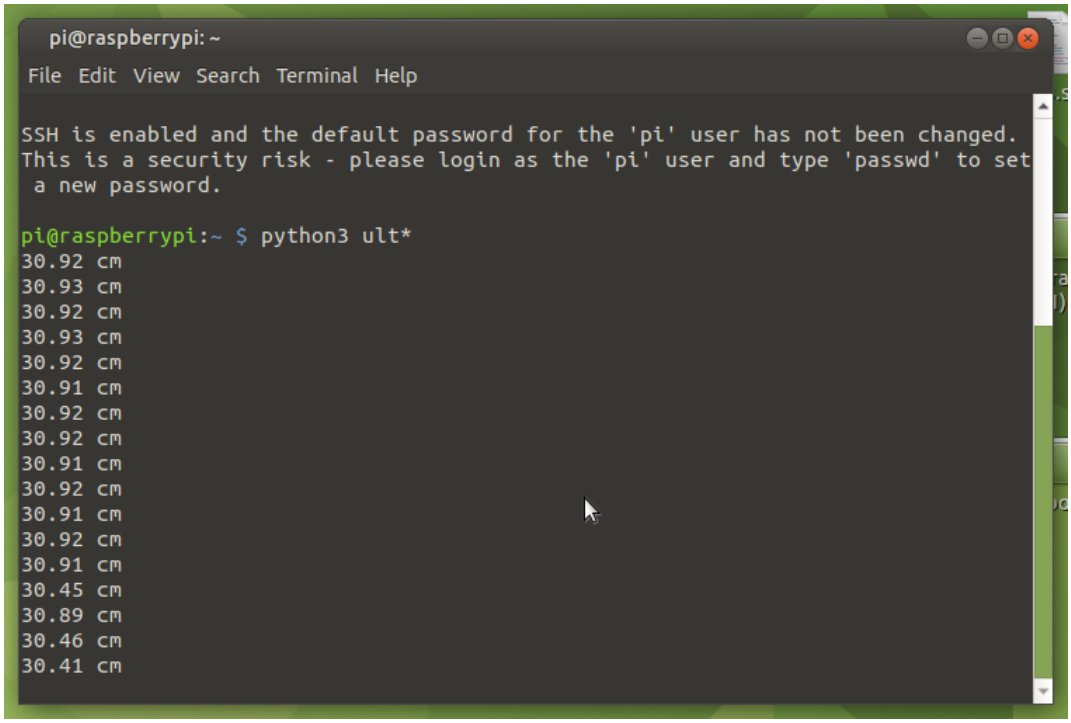



**Figure 2.4.7**:Results of Robot Testing for Distance Determination..

In this segment of the experiment, we focused on testing the robot's ability to recognize and estimate height. We designed a scenario where an obstacle was placed near the robot, primarily focusing on measuring its height. Our objective was to understand how the robot handles nearby obstacles and estimates their height accurately.

To execute this, we adjusted the height of the obstacle and positioned it close to the robot. The obstacle was set at a specific height that we wanted the robot to detect. After setting up the scene, we ran the program utilizing the Yt Search Algorithm, an advanced algorithm that enables the robot to search for specific features and estimate measurements based on sensory data. In the grasping experiment, a water bottle was placed in front of the robot, and the grasping program was activated. It is important to highlight that the center of the robot's hand base is not the same as the center of distance measurement from the ultrasonic sensor. The distance measurement needs to be taken from a specific point on the robot's body, and then the offset is added to the equation.

Secondly, the grasping operation requires precise measurements of the arm lengths to be accurate. In this case, the robot was able to reach the target point, but the grasping action was not successful. If the bottle was empty, it would fall upon collision with the robot's hand, indicating that the grip was not secure.

To improve the grasping capability, it is essential to calibrate the arm lengths accurately and ensure that the gripper has sufficient force to hold objects securely. By fine-tuning these parameters and conducting further tests, the robot can enhance its grasping precision and successfully hold objects, regardless of their weight or size.

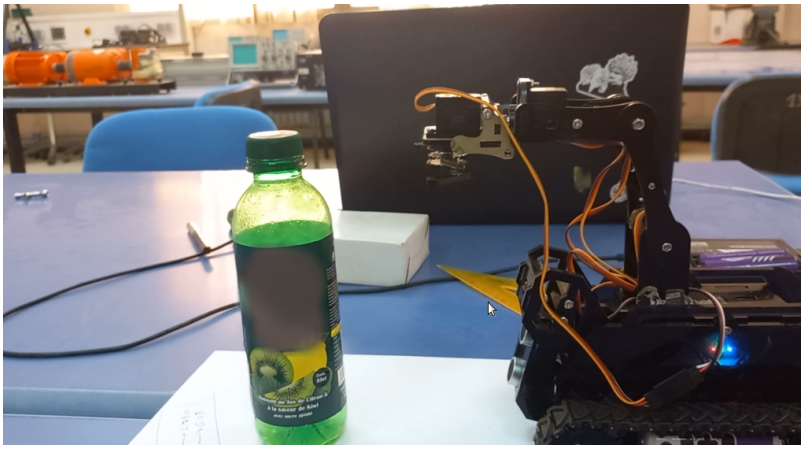


**Figure 2.4.8** :the grasping operation

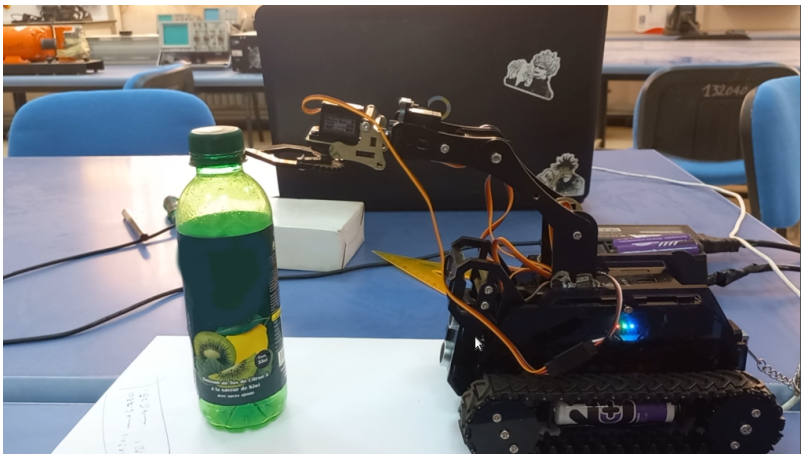


**Figure 2.4.9**:the grasping operation end

### 2.4.4 The problem of "Closest First"

The problem of "Closest First" involves determining which object is nearest among a set of objects by measuring the size of their bounding boxes and their proximity to the center of the frame. However, the information provided may not be sufficient to conclusively identify the closest object, but a selection algorithm can be designed for specific targets.

One challenge with prioritizing proximity to the center of the frame is that the method is not static and may not consistently lead to the same target as the frame itself changes. Additionally, it can sometimes result in pursuing distant objects, making the process chaotic. On the other hand, prioritizing the size of the bounding box may cause the robot to focus on larger objects, such as bottles, while ignoring smaller ones like papers.

To address these issues, a hybrid approach can be considered. By combining proximity to the center of the frame and the size of the bounding box, a more robust selection algorithm can be developed. For example, objects can be assigned a score based on their distance from the center and their size, with closer and larger objects receiving higher scores. This way, the robot can prioritize targets that are both near and substantial, reducing the chaos and ensuring a more efficient selection process.

Additionally, incorporating machine learning techniques can enhance the system's adaptability. By training a model on various scenarios and objects, the robot can learn to make more informed decisions based on context. This enables it to dynamically adjust its priorities and improve its ability to select the most relevant targets, regardless of their size or distance from the center of the frame.

In conclusion, the "Closest First" problem presents challenges in selecting the nearest object due to incomplete information and changing frames. By employing a hybrid selection algorithm that considers both proximity and

size, the robot can make more effective choices. Furthermore, integrating machine learning can improve the system's flexibility and enable it to adapt to different environments and objects efficiently.

### 2.4.5 Performance

After testing and evaluating the actual performance, we obtained the following results:

- The robot takes a significant amount of time to invoke libraries, especially those related to object recognition. Additionally, in the first image, the transfer of data from the processor to the graphics card is a lengthy process. However, this transfer is much faster in the subsequent image.
- The robot's motors are incapable of lifting certain objects, even if it can recognize them. This limitation arises because the robot lacks an algorithm to determine appropriate grasping points, and it does not possess information about the objects' mass or slipperiness.
- Occasionally, the robot exhibits a hallucination behavior, where it identifies a target even when none exists. Fortunately, this issue only persists for a few seconds as the robot moves and changes its perspective.
- Optimizing the division of time between processing and movement is crucial. Increasing processing time enhances accuracy but slows down the robot's movements. On the other hand, increasing movement time makes the robot faster but more susceptible to collisions or missing targets.

### 2.4.6 conclusion

while the robot has shown promising results, there is still room for improvement. By addressing the identified constraints and refining aspects such as library invocation, data transfer, grasping algorithms, object detection, and time management, we can significantly enhance the robot's performance, making it more efficient, reliable, and adaptable to a wide range of tasks and diverse environments.

. Firstly, we find that the robot takes a significant amount of time to invoke libraries, especially those related to object recognition. Optimizing this process can enhance the overall responsiveness of the robot. Additionally, streamlining the data transfer between the processor and graphics card will improve the system's general efficiency.

We have also observed that the robot's motors have limitations in lifting certain objects, even when they are successfully recognized. This constraint highlights the need for advanced algorithms to determine optimal grasping points, taking into account the mass and surface characteristics of objects.

Furthermore, on certain occasions, the robot exhibits unexpected behavior, identifying targets that do not exist. Although this issue resolves as the robot moves and changes its perspective, it emphasizes the importance of robust object detection and recognition algorithms to minimize false positives.

Lastly, balancing processing time and movement speed is critical. Adjusting processing time and movement speed is essential for achieving both accuracy and agility. Increasing processing time improves accuracy but slows down movements, whereas increasing movement time makes the robot faster but more susceptible to collisions or missing targets. Calibrating these parameters according to specific task requirements is necessary for optimal performance.

# General Counclsion

Our project aims to assemble and develop a mobile robot software for environmental cleaning purposes.

The hardware consists of a Raspberry Pi 4, servo motors, DC motors, an ultrasonic sensor, and a camera. We used Python for software development and employed the YOLO (You Only Look Once) AI model for object detection. The standard YOLO model is quite large, so we opted for the smaller Nano version. To enhance speed and accuracy, we first trained the model on four classifications. Subsequently, we converted the model from the PT format to the TFLite format, which is more suitable for Raspberry Pi.

The second issue pertained to the inaccuracy of sensors, both the camera and the ultrasonic sensor. To address this, we adjusted the image histogram and selected the appropriate filter to achieve optimal results. For the ultrasonic sensor, we calculated the average of 100 values to minimize uncertainty during measurements.

The third challenge was determining the robot's path. We utilized the outputs from the image, the AI model, and the ultrasonic sensor to pinpoint the center of the square bounding box surrounding the target. Additionally, we employed an algorithm based on the target's position within the image frame to decide between four actions: move forward, backward, turn right or left, or attempt to grasp.

A partial problem within this step was determining the appropriate rotation value. To tackle this, we drew inspiration from fuzzy logic and calculated the difference between the center value of the frame and the center value of the square on the horizontal axis. This difference was then multiplied by a constant K, which would be determined through experimentation to obtain the optimal rotation value.

In the presence of certain conditions, such as the target being at the center of the image frame and the detection of an object within the workspace, the robotic arm is activated for grasping. However, determining the grasping mechanism itself was a challenge. While many complexities were overlooked in this step, the simplified process involves first identifying the distance and then calculating the height using ultrasonic sensor measurements while manipulating the dedicated servo motor. Once the distance and height values are obtained, the arm is directed towards the next point, and if the grasping is successful, the waste is deposited into a cart behind the robot through a pre-programmed movement.

Upon completion of this cycle, the robot returns to the initial step, and the process is repeated.

Performance measurements revealed that the robot's initial stages are time-consuming due to library invocations, especially for object recognition tasks, and data transfer from the processor to the graphics card. However, subsequent image processing is faster. The robot's motors struggle to lift certain objects, even when recognized, due to the lack of an algorithm for determining grasping points and considering object properties like mass and slipperiness.

The hallucination phenomenon, or the background problem, occasionally affects the model's performance in unpredictable ways, but this only lasts for a few seconds as the robot's movement changes its perspective. Optimizing performance requires a delicate balance between processing time and movement time. Increasing processing time improves accuracy but slows the robot down, while reducing it makes the robot faster but more prone to collisions or target overshoots.

Suggested future solutions:

- First: Convert the software from Python to C or C++ to enhance speed and efficiency.
- Second: Integrate additional algorithms:

  VSlam algorithm for visual mapping and localization. An algorithm to determine the optimal path on the map, along with an IME sensor.
- Third: Upgrade the robotic arm to have a higher degree of freedom:

  To improve access to grasping points and avoid unnecessary complexity. Alternatively, replace it with another machine, such as a backhoe, for more efficient waste transfer to the cart.
- Fourth: Implement the robot operating system (ROS): ROS provides a suitable environment for modification, addition, and stability assurance, aligning with the second suggestion.

# References


[1] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*, vol. 200. Springer, 2008.

[2] S. G. Tzafestas, *Introduction to mobile robot control*. Elsevier, 2013.

[3] R. Illah, "Autonomous mobile robots," 2004.

[4] P. I. Corke, W. Jachimczyk, and R. Pillat, *Robotics, vision and control: fundamental algorithms in MATLAB*, vol. 73. Springer, 2011.

[5] M. J. Mataric, *The robotics primer*. MIT press, 2007.

[6] 2024. SITE : https://www.nndb.com/people/951/000113612/.

[7] 2024. SITE : https://robotnews.wordpress.com/2005/10/10/early-robots-1940s-50s-2/.

[8] B. Bayle, "Robotique mobile," *Ecole Nationale Superieure de Physique de StrasbourgUniversite Louis Pasteur*, vol. 2007, 2008.

[9] Adeept, "Assembly tutorial," 2024. SITE : https://www.adeept.com/learn/tutorial-390.html.

[10] 2024. SITE : https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[11] 2024. SITE : https://circuitdigest.com/comment/30079.

[12] 2024. SITE : https://i7y.org/en/jetson-nano-yolov5-with-csi-2-camera/.

[13] 2024. SITE : https://www.geeksforgeeks.org/introduction-to-python3/.

[14] 2024. SITE : https://opencv.org/about/.

[15] Y. Wai, Z. Yussof, and S. Salim, "A scalable fpga based accelerator for tiny-yolo-v2 using opencl," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 8, p. 206, 11 2019.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[17] E. A. ISSAME, MAHDJOUBI, "Object detection for quadrotor using deep learning," vol. 82, 2022-07-20.

[18] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023.

[19] Y. Zhang, "Stall number detection of cow teats key frames," 03 2023.

[20] "It project recycle 4section dataset: https://universe.roboflow.com/kongju-university-se97i/it-project-recycle-4section/dataset/7," 2024.

[21] 2024. SITE: https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/.

[22] "https://www.raspberrypi.com/news/raspberry-pi-imager-imaging-utility/," 2024.