PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

University 8 Mai 1945 – Guelma

Faculty of Mathematics, Computer Science and Material Sciences

Department of Computer Science

Thesis
Presented for the Diploma of Academic Master

Branch**: Computer Science**

Option**: Informatique System**

Design and Implementation of an Autonomous Mobile Robot Based on ESP-32

Presented by: HAMICI NASSIMA

Number	Full name	Quality
1	BRAHIMI SAID	Chairman
2	BERREHOUMA NABIL	Supervisor
3	ZEDADRA OUARDA	Examiner

JUNE 2023

The Three Laws of Robotics

1. *A robot may not injure a human being or, through inaction, allow a human being to come to harm.*
2. *A robot must obey the orders given it by human beings, except where such orders would conflict with the First Law.*
3. *A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.*

Isaac Asimov 1942

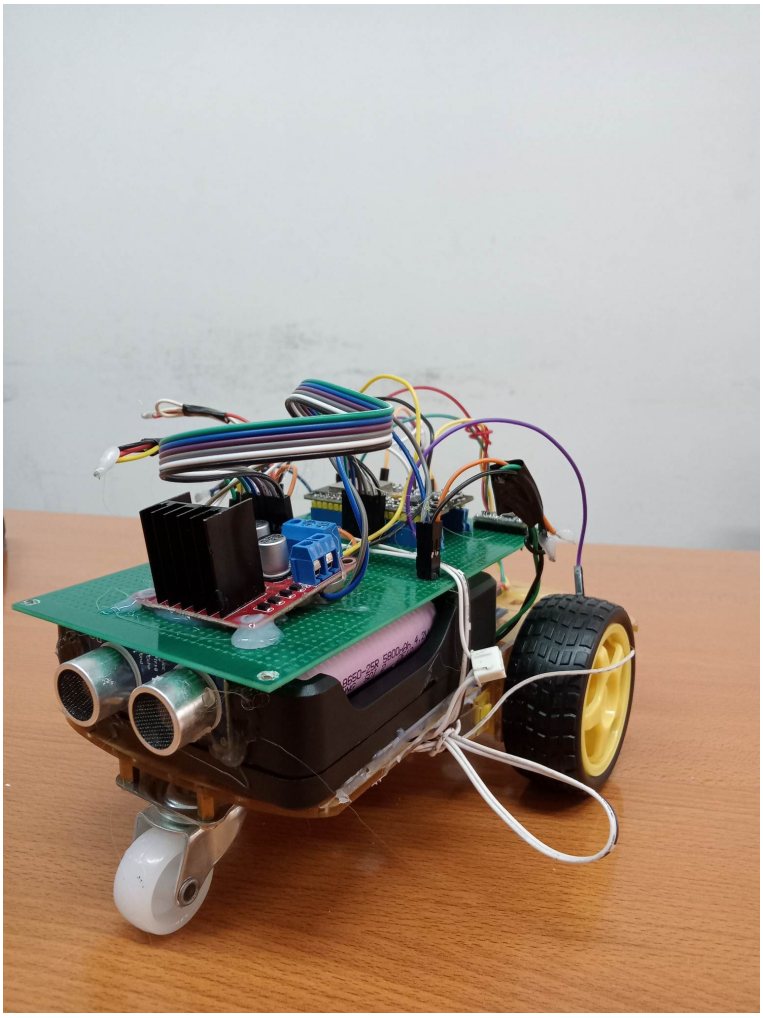


Figure 1: Our mobile robot

ABSTRACT

This thesis presents a comprehensive solution for designing and implementing a real autonomous mobile robot based on ESP-32 micro-controller. The proposed solution is based on a mathematical motion model for differential wheeled mobile robots. The goal of the robot is to predict its position and perform trajectory planning using only its on board sensors and actuators.

The thesis consists of four chapters, including an introduction to mobile robots, motion modeling for differential wheeled mobile robots, the study of ESP-32 based systems, and the design and implementation of the mobile robot prototype. The solution's advantages include the use of micro-controller enhanced by RTOS real time operating system, a graphical interface to control and visualize robot motion. Furthermore, a simulation model is provided for testing and validation purposes.

DEDICATION

I dedicate this thesis to my loving family who have been my unwavering support system throughout my academic journey. To my mother Djamila and father Hocine, whose love, guidance, and sacrifices have been the foundation that made it possible for me to pursue my dreams. To my sister Fatima, brother Mohamed, and his wife Houda, and their adorable daughter Anais, all those people have been a constant source of love, inspiration, and encouragement. Without your unwavering support, I wouldn't be standing here today.

To my supervisor Mr. Berrehouma Nabil, whose expertise, guidance, and mentorship have been instrumental in shaping this thesis. Your dedication, patience, and unwavering support have been invaluable, and I am truly grateful for the opportunity to have worked with you. You pushed me to my limits, believed in my abilities, and inspired me to achieve more than I thought was possible.

To my dear friends, Seif and Hanane, thank you for being there for me during the most challenging times. Your unwavering support and encouragement have been integral to my success, and I cannot thank you enough for your friendship.

And lastly, a special dedication to my sister, my soulmate. Your unwavering support, love, and encouragement have been a constant source of strength for me. I dedicate this thesis to you with all my love, admiration, and gratitude for everything you have done for me.

ACKNOWLEDGMENT

I begin by expressing my utmost gratitude to Allah for His endless blessings, guidance, and strength throughout my academic journey.
I am also deeply indebted to my supervisor, Mr. Berrehouma Nabil, for his valuable support, encouragement, and constructive feedback throughout my research work. His expertise, guidance, and mentorship have been instrumental in shaping my research and academic career.
I extend my sincere appreciation to the members of the jury for taking the time to evaluate my thesis and for their valuable comments and suggestions. I also thank all my teachers for imparting their knowledge and expertise and for inspiring me to pursue academic excellence.
I am deeply grateful to my family for their unconditional love, support, and motivation throughout my studies. Finally, I thank my friends for their continuous encouragement, understanding, and unwavering support.
Without the contribution of each and every one of these individuals, this achievement would not have been possible. Thank you all from the bottom of my heart.

TABLE OF CONTENTs

List of Figures

List of Tables

GENERAL INTRODUCTION

In recent years, mobile robots have gained significant attention due to their wide range of potential applications, from logistics and manufacturing to agriculture and healthcare. Autonomous mobile robots are particularly promising, as they can operate without human intervention, improving efficiency and reducing costs.

However, designing and implementing an autonomous mobile robot is a complex task that involves several challenges, such as motion modeling, trajectory planning, and localization. Moreover, to achieve true autonomy, the robot must be able to make decisions based on the environment and its objectives, requiring the use of intelligent algorithms such as artificial intelligence (AI). In this thesis we limit ourselves to the search for an answer to the localization issues where we try to exploit only the embedded sensors data.

In contrast to those theses[16], [10] and [33], our thesis aims to design and implement an autonomous mobile robot based on ESP-32, a micro-controller widely used in IoT and robotics applications. The proposed solution includes the development of a motion model for differential wheeled mobile robots, the study of ESP-32 based systems, and the implementation of a mobile robot prototype that can operate autonomously by its capacity to self localization. The rest of this thesis is organized as follow :
Chapter 1 provides an introduction to mobile robots, including their history, types, and applications. Chapter 2 focuses on the motion modeling of differential wheeled mobile robots, explaining the kinematics of their movement. Chapter 3 focuses on the study of ESP-32 based systems, including their architecture, programming, and communication protocols. Finally, chapter 4 describes the design and implementation of the mobile robot prototype, including the hardware and software components and some experimentation scenarios.

Overall, this thesis aims to contribute to the development of autonomous mobile robots, by proposing a solution that combines motion modeling, ESP-32 based systems, the results of this work could have important implications for various industries and fields that could benefit from the use of autonomous mobile robots.

CHAPTER I

INTRODUCTION TO MOBILE ROBOTS

I.1 Introduction

During the last years there has been an increasing interest in the area of autonomus mobile robots. The goal is to enhance our ability to do work, increase our quality of life, or perform tasks that are either beyond our ability. The design and control of autonomous mobile robotic systems working in unstructured environments includes many difficulties. There are several studies about the ways in which, robots precepts their enviromenement and behave autonomsly to realize their goals.

In this chapter we will see a brief history of robotics, the definition of a robot as well as the definition of a mobile robot, we will also mention the architecture, components and classification of mobile robots. We will also talk about some of its applications

I.2 Definition of a Robot

In general, robotics plays a very important role in industry and is becoming more and more important in service areas such as maintenance, exploration and medicine. Etymologically, the word robot comes from the Czech word robota, which means work and refers to an automatic system controlled by a computer control unit. In the master thesis [36] a robot is defined as a group of associated mobile devices whose movements are digitally controlled and synchronised digitally. The structure, form and function of these robots must be adapted to the environment with which they interact. Each basic movement is a computer numerically controlled (CNC). However, a robot is just a programmable machine that does nothing more than execute what humans teach it.

I.3 Chronological History of Mobile Robots

Figure I.1 shows the important periods of mobile robotics history which can be divided into six (06) periods.

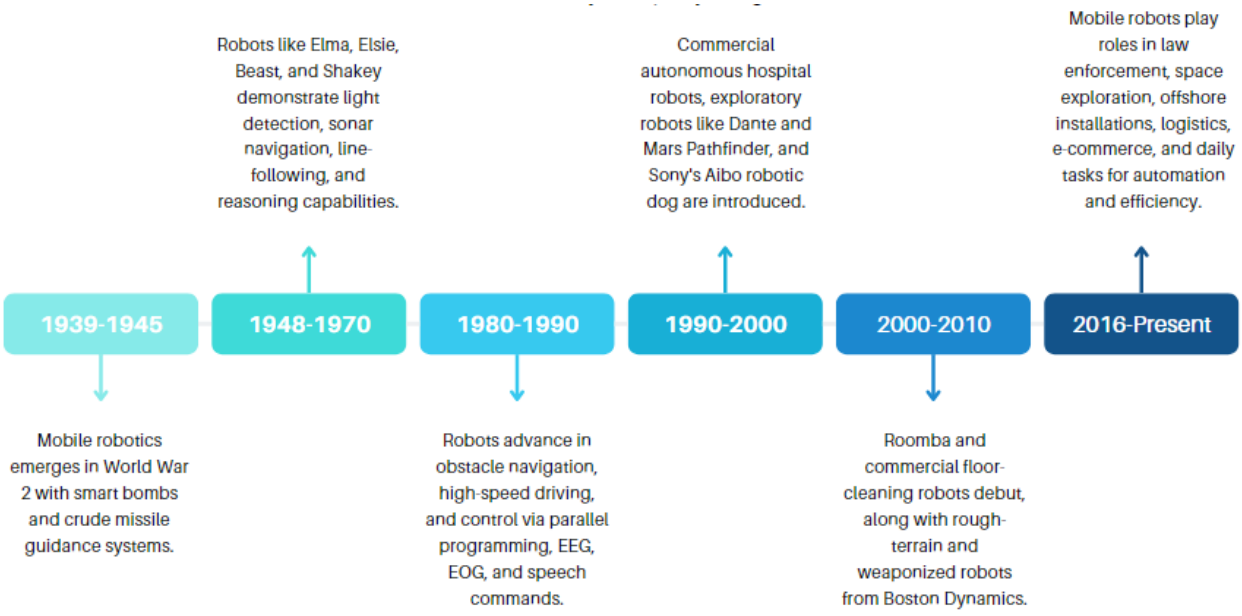


Figure I.1: Timeline of mobile robots

I.3.1 Categories of Robots

Robots can be divided into two categories:

- **Fixed-based robots or manipulators** are designed to perform tasks within a fixed area or workspace. They typically consist of an arm with several joints, which can move in different directions to manipulate objects. These robots are often used in manufacturing, assembly lines, and other industrial applications where precision and repetitive tasks are required.
- **Mobile robots** are designed to move around and perform tasks in various environments. They can have wheels, legs, or other means of locomotion, and are often equipped with sensors and cameras to navigate their surroundings. Mobile robots can be used for tasks such as exploration, surveillance, delivery, and transportation. We will focus on this particular category of robots in our thesis, exploring its various sub-types, capabilities.

I.4 Mobile Robot

According to [20], a mobile robot is a sophisticated mechanical system that autonomously moves within its environment. It consists of sensors to gather information about its sur-

roundings and determine its location, actuators to facilitate movement, and an intelligent algorithm to process sensor data and generate commands for performing tasks. This combination of components empowers the mobile robot to operate independently and accomplish its objectives effectively.

I.4.1 Architecture and Components of a Mobile Robot

The concept of a robot includes the capacity to sense the environment, analyze gathered information, make decisions, and take action. In this regard, perceiving the environment is an essential function and inherent characteristic of every robotic system. In a mobile robot, a distinction is made between: the control part, which processes information, the actuators that carry out the actions and the sensors that inform the robot [9].

- **Sensors:** Their role is to gather data from the surroundings. They provide information to the robots about the external environment and their own actions by checking the state of their actuators. They are crucial for autonomous robots to understand their actions, assess the situation, and make appropriate decisions. The information provided and its accuracy vary across different sensors.
- **Actuators:** Robots are equipped with actuators to navigate and interact with their environment. For instance, a mobile robot has one or more motors to control its wheels. Most mobile robots utilize direct current motors as a power source.
- **Modules (software):** Multiple software modules are employed to operate a mobile robot. These modules enable the interpretation of sensor data, extraction of information, execution of commands, and generation of new commands.

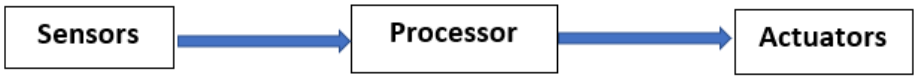


Figure I.2: Components of a mobile robot

I.5 Classification of Mobile Robots

Mobile robots are classified according to several criteria, but we will focus only on three main ones:

1. The degree of autonomy.
2. The locomotion system.
3. Motor skills and energy.

I.5.1 Degree of Autonomy

According to this criteria sited in [29], mobile robots can be classified as follows:

1. **Remote-controlled robot:** They are robots controlled by operators (machines or people), who dictate to them every basic task to be done (moving forward, backward, turn right etc...).

2. **Semi-autonomous robot:** They are robots that perform many tasks themselves completely autonomously, but can be interrupted by an operator to receive control commands.

3. **Autonomous Robot:** A robot is considered to be fully autonomous if it is able to adapt its behaviour to the environment in which it operates.

I.5.2 Locomotion System

According to the locomotion system, we can distinguish four main types:

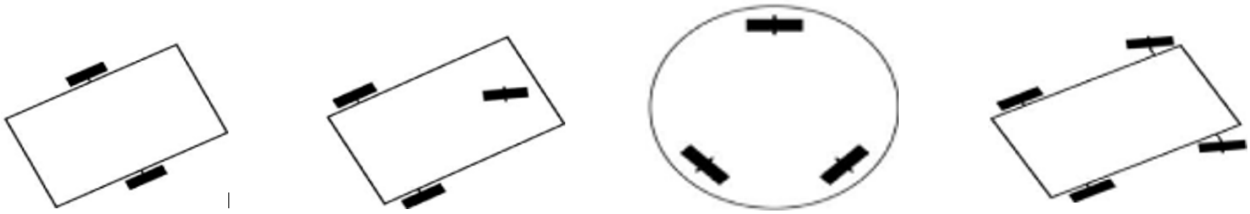
1. **Wheeled mobile robots:** There are several classes of wheeled robots determined by the position and number of wheels used. We will cite here the four main classes of wheeled robots:

   (a) **Differential robots**: This kind of robot is powered by two wheels. The center of rotation is located on the axis connecting the two driving wheels. It moves in a 2D plane with a certain speed of advancement, but without instantaneous lateral movement because it is impossible to move it in a direction that is perpendicular to its wheels, as depicted in figure I.3.A.

   (b) **Tricycle robots**: As illustrated in figure I.3.B this type of robot consists of two fixed wheels placed on the same axis and a centered steerable wheel placed on the longitudinal axis . The movement of the robot is determined by the speed of the two fixed wheels and the direction or orientation of the steerable wheel which plays the role of a steering wheel.

   (c) **Omnidirectional robots**: Omnidirectional mobile robot can move freely in all directions. But this is only done at the cost of much greater mechanical complexity compared to other types of mobile robots. Figure I.3.C shows an example of this type of robots.

   (d) **Car (four-wheel) robots**: A robot car consists of two fixed wheels placed on the same rear axle and two steering wheels placed on the same frontal axle, as presented in figureI.3.D.



(a) Differential (b) Tricycle (c) Omnidirectional (d) Car

Figure I.3: Wheeled mobile robots

I.5.2.1 Comparison of Different Types

In the table I.1 we can see a summary of the advantages and drawbacks of the different types of wheel robots:

Differential robot	– Can not moves in all directions. + Stable. + Rotation on itself. + Low mechanical complexity.
Tricycle robot	– Can not moves in all directions. – Not very stable. – No rotation on itself. + Moderate mechanical complexity.
Car robot	– Can not moves in all directions. + Stable. – No rotation on itself. + Moderate mechanical complexity.
Omnidirectional robot	+ Can move in all directions. + Stable. + Self-rotation. – Significant mechanical complexity.

Table I.1: Comparison of the different types[24]

2. **Walking robots** or humanoid robots represent an important class of mobile robots where the motion is made by the articulation of two or more legs. Their flexibility makes them uniquely capable of serving in many areas such as maintenance and security, disaster response, personal companion, and so on. At present, it appears that defense-related applications are the most likely to experience practical use of this type of robots. As shown in figure I.4 walking robots come in many forms:

   - With two legs (humanoids).
   - With four legs (Quadrupedal).
   - With six legs (Hexapod robot).

(a) Humanoid (b) Quadrupedal (c) Hexapod

Figure I.4: Types of walking robots

3. **Crawling robots:** Shown in Figure I.5, crawling robots utilize techniques similar to the locomotion methods of crawling animals, such as snakes. They are specifically designed for tunnel-like environments or restricted spaces. The system of crawling robots consists of multiple modules with various degrees of mobility [37].

Figure I.5: Crawling robot[37]

4. **Tracked robots** are robots that use tracks instead of wheels (figure I.6). This locomotion method offers the advantages to have more stability compared to wheeled robots since there is a large contact area between the tracks and the ground. For that reason tracked robots are usually used for the locomotion of high load engines.

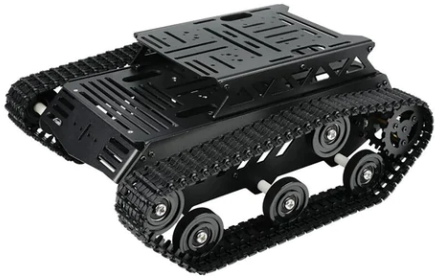


Figure I.6: Tracked robot

I.5.3 Motor Skills and Energy

Robots can be powered by different types of motors, including electric, thermal, and hydraulic motors [30]. Electric motors are the most commonly used motors in robotics due to their ease of control. There are several methods for powering electric motors, including using batteries that can be recharged, or by exchanging them. Alternatively, a cord can be used to power the robot, but this can limit its mobility. There are essentially three types of electrical motors used by wheeled mobile robots:

1. **Direct current (DC) motors:** These motors are commonly used in wheeled mobile robots due to their simplicity and controllability. DC motors operate using direct current and typically have two terminals, one for the positive voltage and one for the negative voltage. By varying the voltage and polarity applied to the motor, its speed and direction can be controlled.

2. **Stepper motors:** Are another type of motor used in wheeled mobile robots. They are designed to move in discrete steps or increments, making them ideal for precise positioning tasks. Stepper motors have multiple electromagnetic coils, or phases, that are energized in a sequence to rotate the motor shaft. The rotation angle is determined by the number of steps and the sequence in which the coils are activated.

3. **Alternative current (AC) motors:** While less commonly used than DC and stepper motors in wheeled mobile robots, AC motors can still be found in certain applications. AC motors rely on alternating current to generate a rotating magnetic field that drives the motor shaft. They are typically more complex in design and require additional components such as capacitors or motor controllers to operate effectively in robotic systems.

I.6 Applications of Mobile Robots

In the literature, many studies have detailed how wheeled mobile robots have been used by different domains. The following table extracted from [30] provides a non-exhaustive summary of various applications:

Nuclear industry	- Site monitoring. - Handling of radio-active materials. - Plant decommissioning.
Civil security	- Laying of explosives. - Mine clearance. - Neutralization of terrorist activity. - Ammunition surveillance.
Military	- Surveillance, patrol. - Explosives laying. - Ammunition handling.
Chemical	- Site surveillance. - Handling of toxic materials.
Medical	- Emergency assistance. - Assistance to the physically disabled, the blind.
Fire fighting	- locating a fire source. - Smoke detection. - Flame suppression.
Submarine	- Cable laying. - Module search. - Search for submerged vessels. - Seabed inspection.
Agricultural	- Fruit picking. - Milking, harvesting, vineyard treatment...
Construction	- Mortar spraying. - Concrete smoothing.
Cleaning	- Ship's hull. - Industrial cleaning.
Space	- Exploration.
Industrial	- Conveying. - Surveillance.

Table I.2: Applications of mobile robots

I.7 Conclusion

In conclusion, this chapter provided a comprehensive overview of mobile robotics, encompassing definitions, components, classification, and applications. Mobile robots offer several advantages, such as increased productivity, task automation, and improved safety in hazardous environments. However, there are also potential drawbacks, including job

displacement and ethical concerns. In the next chapter, we will delve into the topic of motion modeling for differential wheeled mobile robots, exploring advanced aspects that contribute for a best design and implementation of our prototype.

CHAPTER II

MOTION MODELING

II.1 Introduction

Studying robots motion is a main step to understand their behavior. To do so, it is essential to establish a mathematical foundation for many of aspect such kinematics, dynamics, velocity control, path planning and so on. In the rest of this chapter, we start by studying kinematics of differential wheeled mobile robot then we discuses how to control its motion in order to achieve its goal. We conclude by introducing a new motion control algorithm which will be implemented in next chapters.

II.2 Kinematics of Wheeled Mobile Robots

Motion models can describe robot kinematics, and we are interested in mathematics of robot motion without considering its causes, such as forces or torques. Kinematic model describes geometric relationships that are present in the system. It describes the relationship between input (control) parameters and behavior of a system given by state-space representation. A kinematic model describes system velocities and is presented by a set of differential first-order equations. According to this informative book [21] several types of kinematic models exist:

- Internal kinematics explains the relation between system internal variables. (e.g., wheel rotation and robot motion).
- External kinematics describes robot position and orientation according to some reference coordinate frame.
- Forward kinematics and inverse kinematics. A forward or direct kinematics describes robot states as a function of its inputs (wheel speeds, joints motion, wheel steering,

etc.). From inverse kinematics one can design a motion planning, which means that the robot inputs can be calculated for a desired robot state sequence.

- Motion constraints appear when a system has less input variables than degrees of freedom (DOFs). Holonomic constraints prohibit certain robot poses while a non-holonomic constraint prohibits certain robot velocities (the robot can drive only in the direction of the wheels rotation).

II.2.1 Robot Position Representation

All through this thesis, a wheeled mobile robot is considered as a rigid body mounted on wheels and moves on horizontal plane. With respect to this specification, We can describe its position only by three coordinates (figure II.1):

1. (x, y) for local position in the cartesian plane.
2. θ for orientation along the z-axis.

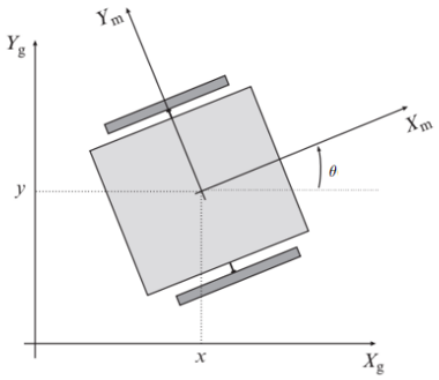


Figure II.1: The global reference frame and the robot local reference frame

In figure II.1 (O, X_g, Y_g) defines an arbitrary inertial global reference basis on the plane. To specify the WMR [1] position, we choose a reference point P on its chassis -usually its gravitation center-. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by θ. The position of the WMR in the global frame is formalized by the vector ξ_g.

$$\xi_g = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{II.1}$$

[1] Wheeled Mobile Robot

The basis (P, X_m, Y_m) defines the WMR's local reference frame. To describe robot motion in terms of component motions, it will be necessary to map motion along the axes of the global reference frame to motion along the axes of the WMR's local reference frame using the Jacobian matrix J.

$$J(\theta) = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{II.2}$$

This matrix II.2 can be used to map motion in the global reference frame to motion in terms of the local reference frame using the equation II.3.

$$\dot{\xi_g} = J(\theta)\dot{\xi_m} \tag{II.3}$$

II.3 Differential Wheeled Mobile Robots

Differential drive is a very simple driving mechanism that is quite often used in practice, especially for smaller mobile robots. Robots with this drive usually have one or more castor wheels to support the vehicle and prevent tilting. Both main wheels are placed on a common axis. The velocity of each wheel is controlled by a separate motor. According to Figure II.2 the input (control) variables are the velocity of the right wheel $v_R(t)$ and the velocity of the left wheel $v_L(t)$. The meanings of other variables in Figure II.2 are as follows:

- r is the wheel radius.
- L is the distance between the wheels.
- $R(t)$ is the instantaneous radius of the vehicle driving trajectory (the distance between the vehicle center (middle point between the wheels) and ICR point).
- In each instance of time both wheels have the same angular velocity $\omega(t)$ around the ICR.

II.3.1 Forward kinematic Models for Differential Drive MWR

Forward kinematic models consists in determining the global WMR motion given its geometry and its wheels speed and orientation. In figure II.2 we have a differential WMR with two wheels each with diameter r. Given a point P centered between the two drive wheels, each wheel is a distance l from P. Given r, l, θ, and the spinning speed of each

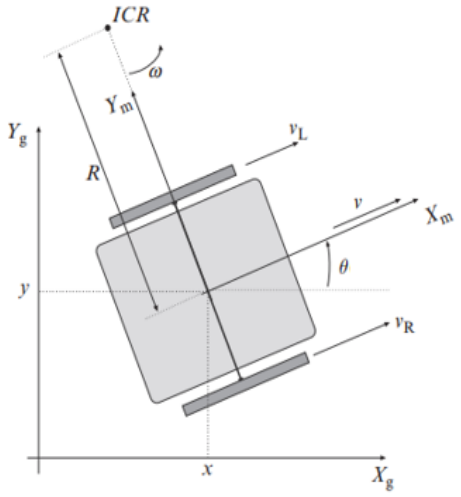


Figure II.2: Differential drive kinematics

wheel, ω_L and ω_R, a forward kinematic model would predict the WMR's overall speed in the global reference frame:

$$\dot{\xi}_g = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \omega_R, \omega_L) \tag{II.4}$$

II.3.2 Angular and Tangential Velocity of the Robot

In each instance of time both wheels have the same angular velocity $\omega(t)$ around the ICR, so:

$$\omega(t) = \frac{v_L(t)}{R(t) - \frac{L}{2}} \tag{II.5}$$

and:

$$\omega(t) = \frac{v_R(t)}{R(t) + \frac{L}{2}} \tag{II.6}$$

From II.5 ,II.6 we know that:

$$\frac{v_L(t)}{R(t) - \frac{L}{2}} = \frac{v_R(t)}{R(t) + \frac{L}{2}}$$

Using cross multiply rule we find:

$$v_R(t) * (R(t) - \frac{L}{2}) = v_L(t) * (R(t) + \frac{L}{2})$$

Which gives the formula of $R(t)$ as follow:

$$R(t) = \frac{L}{2} * \frac{v_L(t) + v_R(t)}{v_R(t) - v_L(t)}$$

Then, by replacing $R(t)$ in equation II.5 we find:

$$\omega(t) = \frac{v_L(t)}{\frac{v_R(t)+v_L(t)}{v_R(t)-v_L(t)} * \frac{L}{2} - \frac{L}{2}}$$

After reduction, we obtain the angular velocity $\omega(t)$ as follow:

$$\boxed{\omega(t) = \frac{v_R(t) - v_L(t)}{L}} \tag{II.7}$$

The robot tangential velocity is the product of the angular velocity by the instantaneous rotation radius, thus:

$$\boxed{v(t) = w(t)R(t) = \frac{v_R(t) + v_L(t)}{2}} \tag{II.8}$$

II.3.3 Internal Robot Kinematics

Wheel tangential velocities are $v_L(t) = r\omega_L(t)$ and $v_R(t) = r\omega_R(t)$, where $\omega_L(t)$ and $\omega_R(t)$ are left and right angular velocities of the wheels around their axes, respectively. Considering the above relations and using the equation II.7 and II.8, the internal robot kinematics (in local coordinates) can be expressed as:

$$\boxed{\begin{bmatrix} \dot{x}_m \\ \dot{y}_m \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V_{x_m}(t) \\ V_{y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{L} & \frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix}} \tag{II.9}$$

II.3.4 External Robot Kinematics

Robot external kinematics (in global coordinates) is obtained using the Jacobian matrix II.2 and given by:

$$\boxed{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta(t)) & 0 \\ sin(\theta(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}} \tag{II.10}$$

where $v(t)$ and $\omega(t)$ are the control variables.

Global coordinates given by II.10 can be calculated in discrete form using Euler in-

tegration[2] and evaluated at discrete time instants $t = kT_s$ $k = 0, 1, 2, ...$ where T_s is the following sampling interval:

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_s cos(\theta(k)) \\ y(k+1) &= y(k) + v(k)T_s sin(\theta(k)) \\ \theta(k+1) &= \theta(k) + \omega(k)T_s \end{aligned} \tag{II.11}$$

II.4 Forward and Inverse Kinematics

II.4.1 Forward Kinematics

A robot pose at some time t is obtained by integration of the kinematic model. Determination of the robot pose for given control variables is called direct (also forward) kinematics:

$$\begin{aligned} x(t) &= \int v(t)cos(\theta(t))dt \\ y(t) &= \int v(t)sin(\theta(t))dt \\ \theta(t) &= \int \omega(k)dt \end{aligned} \tag{II.12}$$

II.4.2 Reverse Kinematics

Inverse kinematics allows to determine control variables to drive the robot to the desired robot pose or to follow a path trajectory. Robots are usually subjected to many constraints, which means that not all driving directions are possible. There are also many possible solutions to arrive to the desired pose. One simple solution to the inverse kinematics problem would be if we allow a differential robot to:

1. Drive only in straight motion by giving the same velocities in the same direction for the two wheels:
$$v_R(t) = v_L(t) = v_R \implies \omega(t) = 0, v(t) = v_R$$

2. Only rotate around itself:
$$v_R(t) = -v_L(t) \implies \omega(t) = \frac{2v_R}{L}, v(t) = 0$$

For straight motion equation II.12 simplifies to:

$$\begin{aligned} x(t) &= x(0) + v_R cos(\theta(0)) \\ y(t) &= y(0) + v_R sin(\theta(0)) \\ \theta(t) &= \theta(0) \end{aligned} \tag{II.13}$$

[2] A numerical method used to evaluate the derivate of a function in a given point in its definition domain

For rotation motion equation II.12 simplifies to:

$$\begin{aligned} x(t) &= x(0) \\ y(t) &= y(0) \\ \theta(t) &= \theta(0) + \frac{2v_R}{L} \end{aligned} \tag{II.14}$$

Motion strategy could then be to:

1. Orient the robot to the target position by rotation.
2. Drive the robot to the target position by a straight motion.
3. Align (with rotation) the robot orientation with the desired orientation in the desired robot pose.

The required control variables for each phase (rotation, straight motion, rotation) can easily be calculated from equations II.14, II.13.

II.5 Control of Wheeled Mobile Robot

Motion control of wheeled mobile robots in the environment without obstacles can be performed by two ways:

1. Controlling motion from some start pose to some goal pose (classic control, where intermediate state trajectory is not prescribed).
2. By reference trajectory tracking.

In our thesis, we limit to the first way for WMR motion control.

II.5.1 Control to Reference Pose

In this case the path or trajectory to the reference pose is not prescribed, and the robot can drive on any feasible path to arrive to the goal. The control to the reference pose will be split into two separate tasks:

1. Orientation control.
2. Forward-motion control.

These two tasks cannot be used individually, but by combining them, several control schemes for achieving the reference pose are obtained.

II.5.1.1 Orientation Control

Let us assume that the orientation of the wheeled robot at some time t is $\theta(t)$, and the reference or desired orientation is $\theta_{ref}(t)$. The control error can be defined as:

$$e\theta(t) = \theta_{ref}(t) - \theta(t)$$

We start with the model of the system to be controlled. using the equation II.10. The orientation equation of the kinematic model is given by:

$$\dot{\theta} = w(t)$$

The control law is then given by:

$$\omega(t) = K_{rotation}(\theta_{ref}(t) - \theta(t))$$

where the control gain $K_{rotation}$ is an arbitrary positive constant. The interpretation of the control law is that the angular velocity $\omega(t)$ of the robot is set proportionally to the robot orientation error. Combining precedent two equations, the dynamics of the orientation control loop can be rewritten as:

$$\boxed{\dot{\theta} = K_{rotation}(\theta_{ref}(t) - \theta(t))}$$

II.5.1.2 Forward-Motion Control

By forward-motion control we refer to the algorithms that controls the instantaneous velocity of the mobile robot $v(t)$ to achieve some control goal. It is obvious that forward-motion control alone cannot drive the robot to a desired position unless the robot is directed to its goal initially. This means that forward-motion control is inevitably interconnected with orientation control.

In the case of trajectory tracking, the velocities of each wheel must be adapted continuously to follow the trajectory while in the reference pose control wheels velocities should decrease when the robot approaches the final goal. A reasonable idea is to apply the control that is proportional to the distance to the reference point $(x_{ref}(t), y_{ref}(t))$:

$$\boxed{v(t) = K_{forward} * \sqrt{(x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2}} \quad \text{(II.15)}$$

The reference position can be constant or it can change according to some reference trajectory[3]. The control expressed by II.15 equation has the next limitations:

[3]In our thesis, we are limited to constant reference point

- If the distance to the reference point is large, the control command given by II.15 also becomes large. We can correct this situation by limiting the maximum velocity command.
- If the distance to the reference point is very small, the robot can bypass the reference point. The problem is that after bypassing the reference pose the velocity command will start increasing because the distance between the robot and the reference starts increasing. And the robot is accelerating and will diverge from its goal. In chapter 4, we have proposed some correction to this situation in the cost to lose in precision.

II.6 Control to Reference Position Algorithm

To reach its goal from its original pose, the robot is required first to arrive at a reference (final) position and then to rotate to the final orientation. In order to arrive to the reference point, the robot's orientation is controlled continuously in the direction of the reference point. This direction is denoted by θ_r (figure II.3), which can be obtained easily using geometrical relations:

$$\theta_r(t) = arctan(\frac{y_{ref} - y(t)}{x_{ref} - x(t)})$$

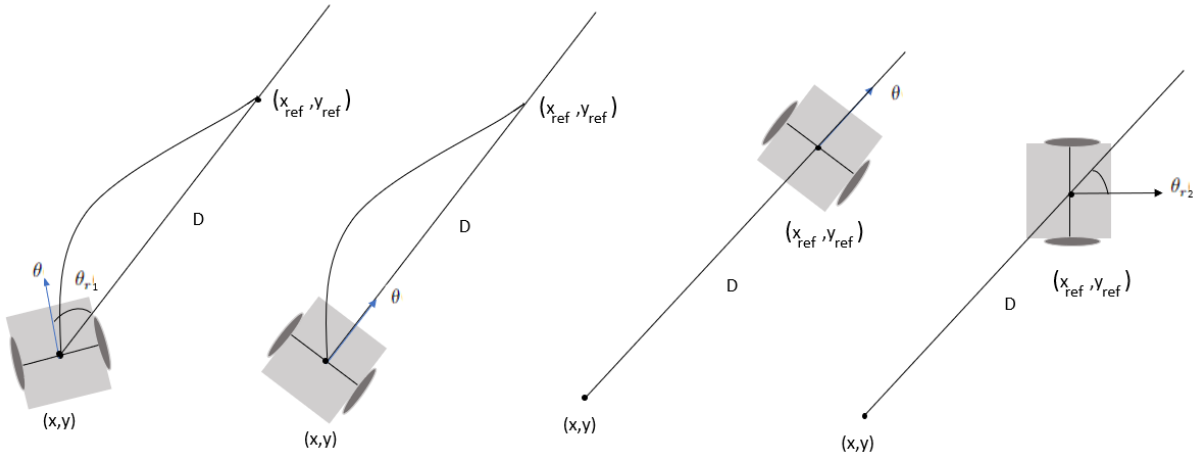


Figure II.3: Control to reference position

Angular velocity control $\omega(t)$ is therefore commanded as:

$$\omega(t) = K_{rotation_1}(\theta_r(t) - \theta(t))$$

When the robot gets ahead to the goal position, forward motion control is engaged following the equation II.15.

When the robot reaches a certain neighborhood of the reference point, the approach phase is finished and zero velocity commands are sent. This mechanism to fully stop the

vehicle needs to be implemented even in the case of the modified control law, especially in case of noisy measurements. Finally, the robot must turn to its final orientation, so we control again the angular velocity $\omega(t)$ according to the next equation:

$$\omega(t) = K_{rotation_2}(\theta_{ref}(t) - \theta(t))$$

II.7 Conclusion

Throughout this chapter, we have acquired a deep comprehension of differential wheeled mobile robots motion modeling and control. Many works in the literature have been devoted to this subject and provided a variety of dynamics equations for both angular and tangential velocities to permit the robot achieving its goal in less time and in more precision. Based on these works, we have proposed our approach which will be developed in chapter 4 to be implemented both by a simulated and real robot.

CHAPTER III

STUDY OF ESP-32 BASED SYSTEMS

III.1 Introduction

The last chapter focused on motion modeling for differential wheeled mobile robots, the resulting models will be implemented by an embedded micro-controller of type ESP-32. Per consequence, a good comprehension of this micro-controller will be indispensable for the rest of this work.

The next sections discuss many aspects of ESP-32 system. Starting by a review on its evolution history, the architecture and main components are presented also. A particular interest will be devoted to some of its functionalities that we will use in the next chapter such as communication protocols and real time operating system process synchronization, scheduling and timers.

III.2 What is ESP-32 ?

Different definitions are given in the literature, we are inspired by [7] in which an ESP-32 is a low-cost, little power micro-controller that replaces its predecessor version ESP-8266. It is at the heart of a system on chip architecture [1] along with integrated Wi-Fi and dual mode Bluetooth connectivity. ESP-32 offers a variety of functionalities compatible with a wide range of sensors for a different use, including IoT (Internet of Things). It as well offers programmers a powerful toolkit with dual cores microprocessor cadenced at 240 MHz at max, 520 Kbyte SRAM (Static Random Access Memory) and peripherals including: I2C[2],

[1] SOC, or system on chip, is a compact integrated circuit that combines numerous computer or electronic system components.

[2] Inter-Integrated Circuit, is a serial communication protocol designed to connect low-speed devices.

DAC[3], ADC[4], I2S[5], SPI[6], UART[7], 34 physical GPIO (General Purpose Input/Output) pins. The power management unit and the lower power controller enable the ESP-32 to run lower than 1mA in deep sleep mode. These advantages make the ESP-32 a better device for low power applications. For more details about our micro-controller pass to this book [19].

(a) SoC (b) Module (c) Board

Figure III.1: ESP-32 is provided in different forms: as a Soc Ship or as a Module with Wi-Fi antenna or embedded on a DevKit Board

III.3 ESP-32 Timeline

ESP-32 is a micro-controller developed by Espressif systems [8], a Chinese company specializing in Wi-Fi and Bluetooth chips for IoT and other connected applications. Figure III.7 resumes the historical evolution and main timeline stones of the ESP-32 micro-controller:

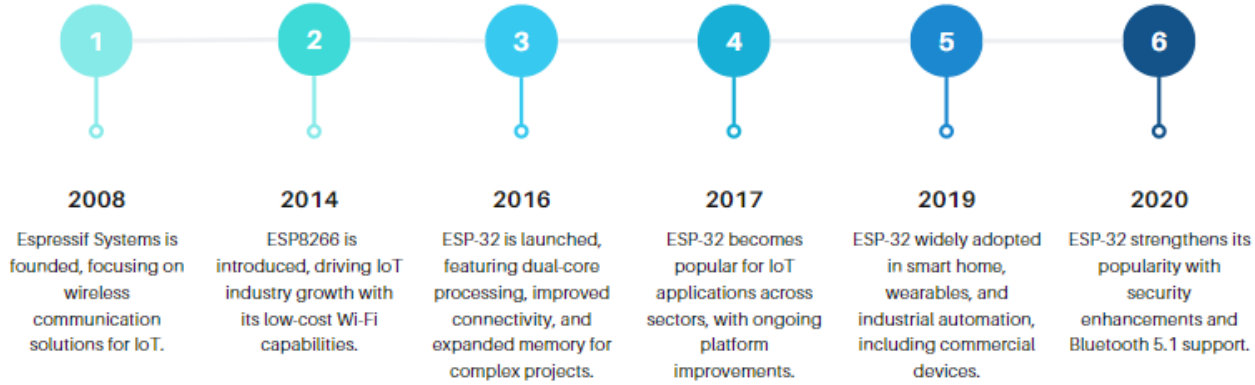


Figure III.2: ESP-32 timeline

The history of the ESP-32 is ongoing, and Espressif is likely continue to develop and improve the platform in the years to come.

[3]Digital to Analog Converter.
[4]Analog to Digital Converter.
[5]Inter-IC Sound is a serial bus for digital audio communication between integrated circuits.
[6]Serial Peripheral Interface can enable fast and synchronized communication between micro-controllers and peripheral devices, facilitating data transfer and control.
[7]Can enable asynchronous serial communication between devices without the need for a clock signal, facilitating data transmission and reception.
[8]https://www.espressif.com

III.4 Description of ESP-32

The ESP-32 is a small, rectangular micro-controller that can be easily integrated into a wide range of electronic devices and products. From the outside, it has the following features:

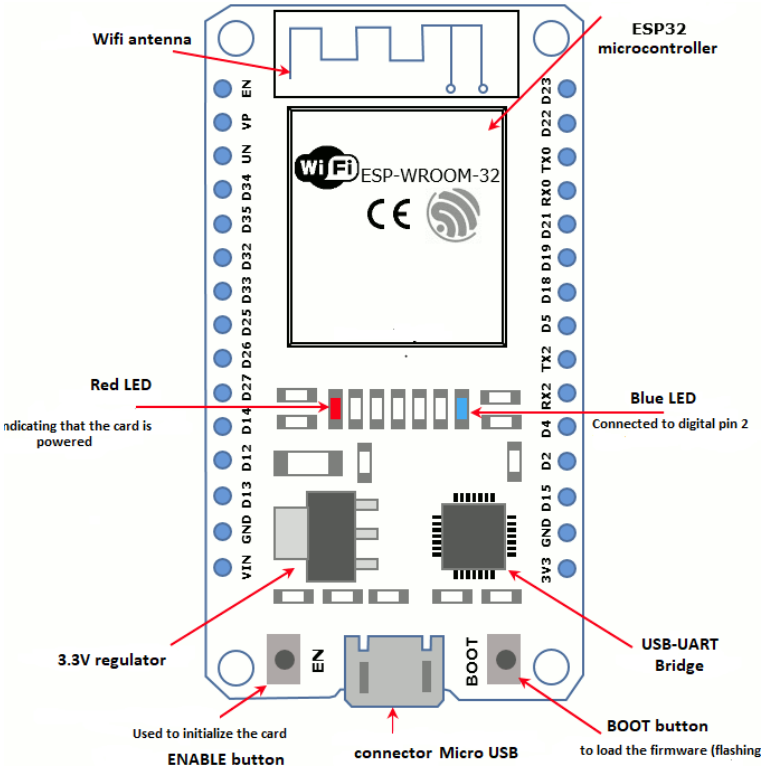


Figure III.3: ESP-32 description

- **Dimensions:** The ESP-32 typically comes in a compact form factor, with dimensions of around 25mm x 15mm.
- **Connectors:** It comes with a micro-USB interface that you can use to connect the board to your computer to upload code or apply power.
- **Enable and Boot button:** For reset and restarting options.
- **3.3V regulator:** Is a voltage regulating component. It provides a stable 3.3V output voltage to the micro-controller and other components on the board. This ensures that the device operates within its specified voltage range, protecting it from over-voltage or under-voltage conditions.
- **USB-UART bridge:** Is a device that allows communication between the USB port and the UART interface of the micro-controller. It provides a convenient way to program and debug the ESP-32 by converting USB signals to UART signals, and vice

versa. The USB-UART bridge makes it easy to connect the ESP-32 to a computer or other USB device for programming, debugging, and data transfer.

- **Some LEDs:** To indicate various states or conditions of the device.
- **WIFI / Bluetooth Antenna:** Is a component that facilitates wireless communication. It sends and receives signals, enabling the micro-controller to connect to Wi-Fi and Bluetooth networks and transfer data. The antenna is crucial for the device's wireless capabilities.

III.5 ESP-32 Pins and Peripherals

The ESP-32 micro-controller has a total of 34 physical GPIO (General Purpose Input/Output) pins. These GPIO pins are grouped into several ranges of pin numbers: (GPIO0 to GPIO19, GPIO21 to GPIO23, GPIO25 to GPIO27 and GPIO32 to GPIO39) [4]. ESP-32 boards come in various designs with different pin arrangements, and the diagram shown in the figure III.4 provides a detailed pin layout specifically for the ESP32 WROOM Generic DevKit boards:

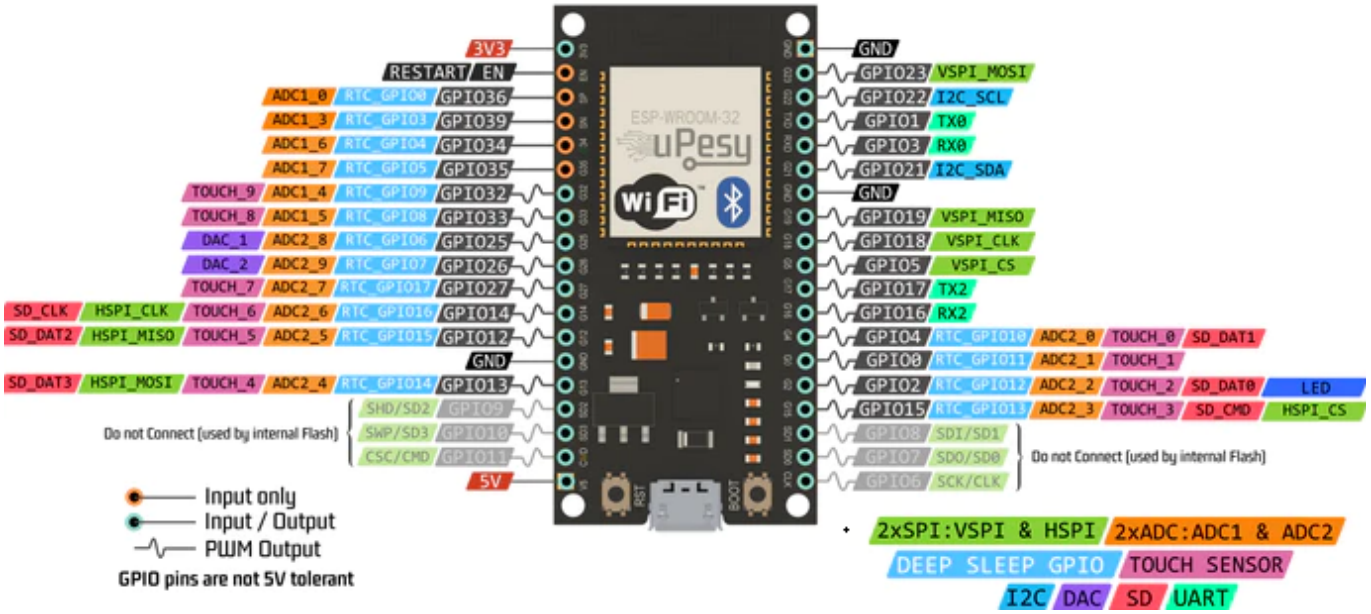


Figure III.4: The pins of the ESP-32 board

Each of these pins can be used as a general-purpose I/O pin, meaning that they can be programmed to perform various functions based on the needs of the application. For example, they can be used to read input signals from sensors, control output signals to other devices, or communicate with other devices using various communication protocols such as SPI, I2C or UART.

In addition to their general-purpose use, certain pins on the ESP32 can also be connected to internal peripheral signals, allowing them to control various internal components on the chip. Figure III.5 capture the ESP-32 internal peripherals. For more detail about technical specification, we advice the reader to visit the ESP-32 data-sheet [1].

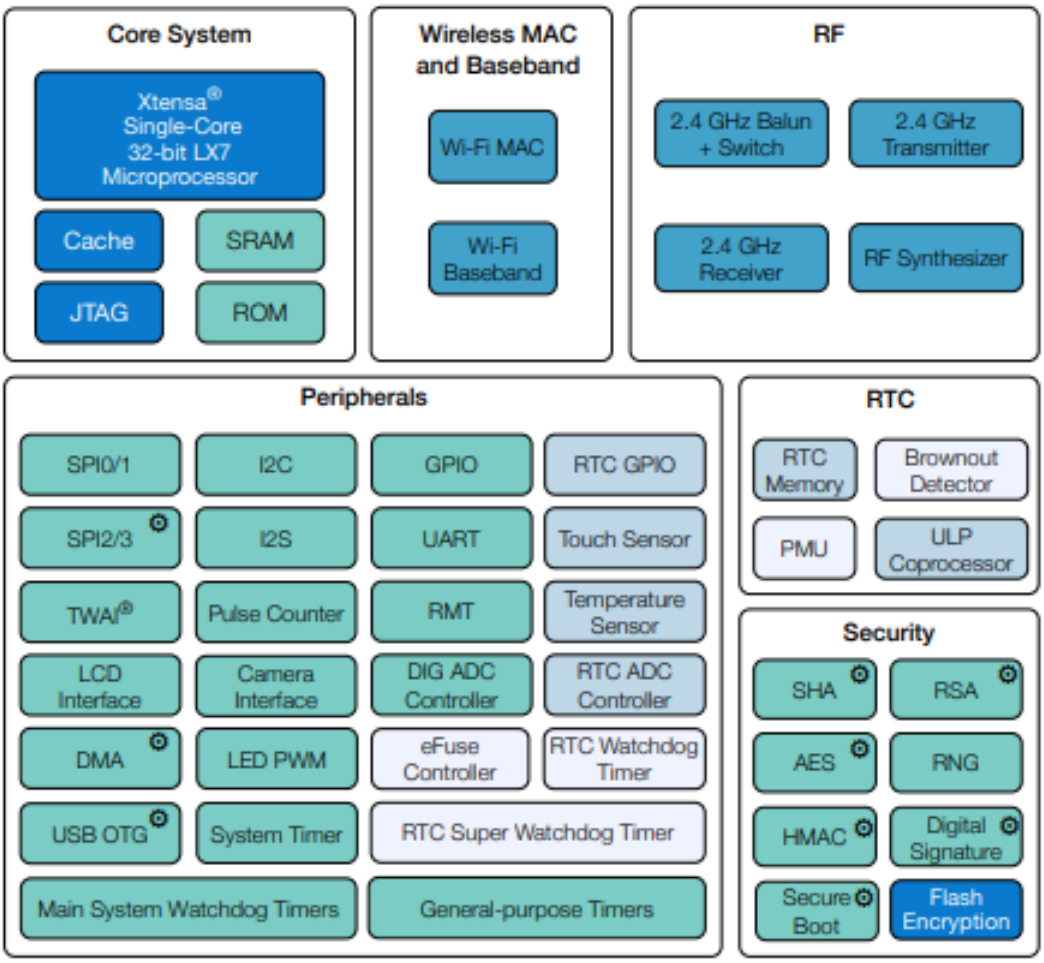


Figure III.5: All peripherals of ESP-32[1]

III.5.1 ESP-32 Programming

ESP-32 micro-controller is a versatile device which can be programmed with different programming languages essentially C++ [27], Python [25] and so on. Next is a set of the popular used frameworks:

1. **ESP-IDF:** (Espressif IoT Development Framework) [9] this is the official software development framework for the ESP-32 and ESP-32-S series of micro-controllers. It is based on the C programming language and provides a set of libraries, APIs, and tools to help developers build applications for these micro-controllers. The ESP-IDF

[9]https://www.espressif.com/en

framework includes support for various communication protocols and interfaces such as Wi-Fi, Bluetooth, SPI, I2C, and UART.

2. **Arduino IDE:** The Arduino IDE [10] is a popular open-source development environment for programming micro-controllers, including the ESP-32. It uses the C++ programming language and provides a simplified and easy-to -use interface for beginners. The Arduino IDE also includes a large community of developers who share their code and libraries, making it easy to find and use existing code.

3. **Micropython:** Micropython [11] is a version of the Python programming language that is optimized for micro-controllers, including the ESP-32. It provides a simple and easy-to-use interface for beginners and supports a wide range of programming features. With Micropython, you can write and execute Python code directly on the ESP-32.

4. **PlatformIO:** PlatformIO [12] is an open-source platform for developing IoT applications. It supports different micro-controllers, including the ESP-32, and provides a unified development environment that supports multiple programming languages such as C++, Python, and JavaScript. PlatformIO also includes a large library of code and provides easy integration with popular development tools such as Visual Studio Code and Atom.

Figure III.6: ESP-32 programming

After careful consideration, we have decided to use the ESP-IDF as a blugin under VScode[13] framework for our project development as it is the most suitable and optimal choice.

III.6 ESP-32 Main Functions

ESP-32 comes with a set of a built-in functions such communication protocols, digital/Analog conversion, cryptography arithmetic, etc. But also, a set of API [14] helping programmers

[10] https://www.arduino.cc/en/software
[11] https://micropython.org/
[12] https://platformio.org/
[13] https://code.visualstudio.com
[14] Application Programming Interface : programming libraries of recompiled code offering a service to best exploiting the hardware.

to easily exploiting these functions. In the following subsection, we focus on the main ESP-32 function used in our project.

III.6.1 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a simple messaging protocol, designed for IoT devices. MQTT communication uses Publish/Subscribe paradigm as shown in figure III.7. More detail about MQTT can be found in [31], [17] and in there official web-site [5]. MQTT Protocols is build on five basic concepts:

1. **Publisher:** Creates a message with a topic and payload, and then sends it to the broker. The publisher may also include a quality of service (QoS) level to indicate the level of reliability required for message delivery.

2. **Subscriber:** It subscribes to one or more topics on the broker and receives messages from the topics it is interested in. The subscriber may also specify a QoS level for message delivery.

3. **Messages:** Messages are the payload exchangeb between devices. It can be a message like a command or data like sensor readings. In our project, we have used JSON [15] messages.

4. **Topics:** Is a string-based identifier that specifies the subject of a message. It serves as an addressing mechanism, allowing publishers to send messages to subscribers who have expressed interest in receiving messages on that topic.

5. **Broker:** The broker is a server that receives messages from publishers and delivers them to subscribers. It maintains a list of subscribed topics and their associated clients, and it routes messages based on the topic. The broker may also apply security policies, QoS levels, and other rules to the messages it handles.

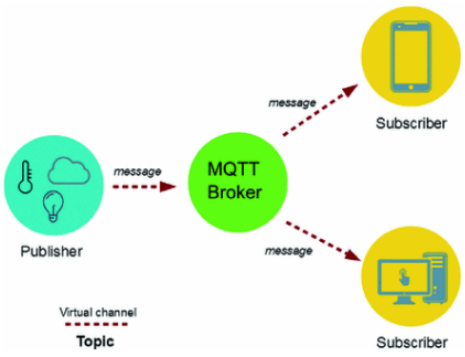
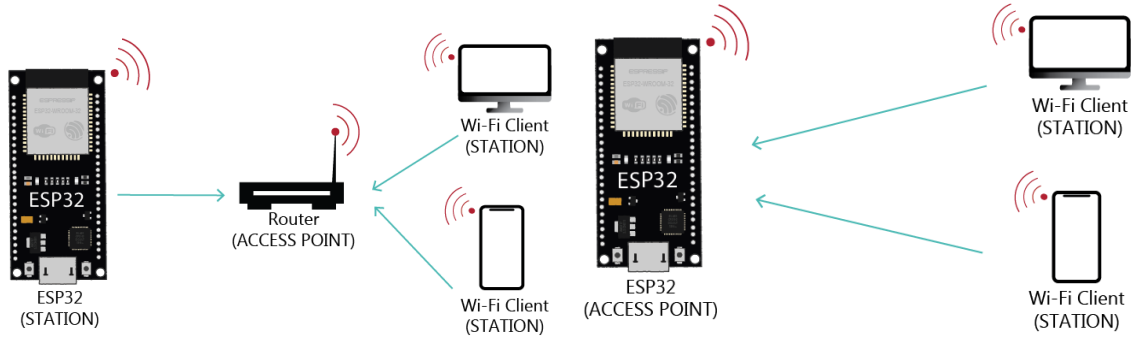


Figure III.7: MQTT architecture

[15] JSON (JavaScript Object Notation) is an open-standard, text-based data format for semi-structured data exchange. It is composed of a set of key:value pairs.

III.6.2 ESP-32 Wi-Fi Support

The ESP-32 is a popular micro-controller that supports Wi-Fi connectivity. The ESP-32's Wi-Fi subsystem includes a Wi-Fi stack, hardware drivers, and APIs for Wi-Fi programming. The ESP-32 supports both 2.4 GHz and 5 GHz Wi-Fi bands, and it supports various Wi-Fi security protocols and modes, including station mode, access point mode, and hybrid station access point mode. For more technical detail about Wi-FI support, The reader can visit next references: [22], [6] and [14].



(a) Station mode (b) Access point mode

Figure III.8: Wi-Fi modes

III.6.3 Pulse Width Modulation (PWM)

In simple terms, Pulse Width Modulation (PWM) is a technique used by devices like the ESP-32 boards to create an analog signal using digital pins. It works by using a square waveform that alternates between high and low states. In the case of a 3.3V power source, the PWM signal can be either 3.3V (high) or 0V (low). For a comprehensive understanding of this topic, I recommend that the reader consults those books for further details [18] and [23]. To understand PWM on the ESP32 board, it's important to know two related terms:

1. **Duty Cycle:** This represents the percentage of time the PWM signal stays in the high state. A signal that is always off has a duty cycle of 0%, while a signal that is always on has a duty cycle of 100%. By adjusting the duty cycle, users can control the "on time" of the signal. The duty cycle is calculated using the formula:

$$\text{Duty Cycle} = \frac{\text{ON Time}}{\text{Period}}$$

2. **Frequency:** This refers to the number of cycles the PWM signal completes per second. It indicates how quickly the PWM signal completes one cycle, which is also known as the time period. The time period is determined by adding the on time and off time of the signal. For example, if a signal has a time period of 20ms, its

frequency is 50Hz. The relationship between frequency and time period is expressed by the formulas:

$$\text{Frequency} = \frac{1}{\text{Time Period}}$$

$$\text{Time Period} = \text{ON time} + \text{OFF time}$$

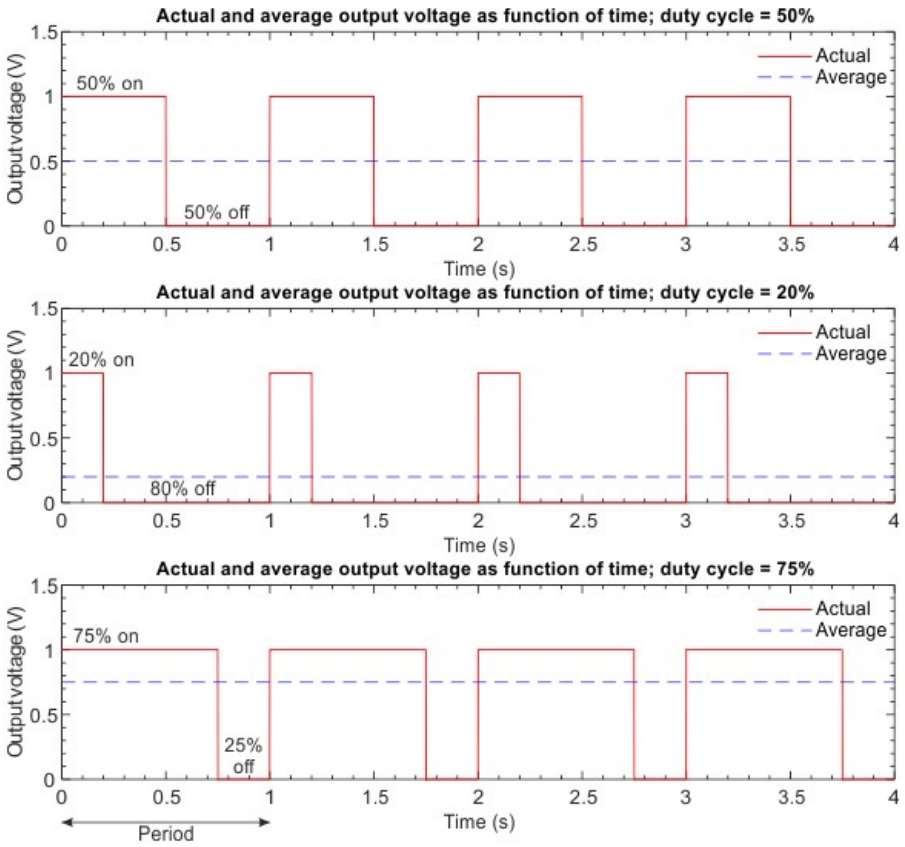


Figure III.9: PWM signal

PWM programming mainly involves adjusting the duty cycle, which allows for effective control of the average power or voltage delivered to the load. PWM is widely used in various applications such as motor control, power converters, audio amplification, and LED dimming. Its versatility and efficiency make it an ideal choice for precise and flexible control of motor speeds in robots.

III.6.3.1 PWM usage under ESP-32

The ESP-32 micro-controller provides PWM support on its output pins through the two next peripherals:

1. **MCPWM (Motor Control Pulse Width Modulator):** The MCPWM peripheral is a versatile PWM generator which can be used to generate analog signal. Which

contains various sub-modules (figure III.10) to make it a key element in power electronic applications like motor control, digital power, and so on.

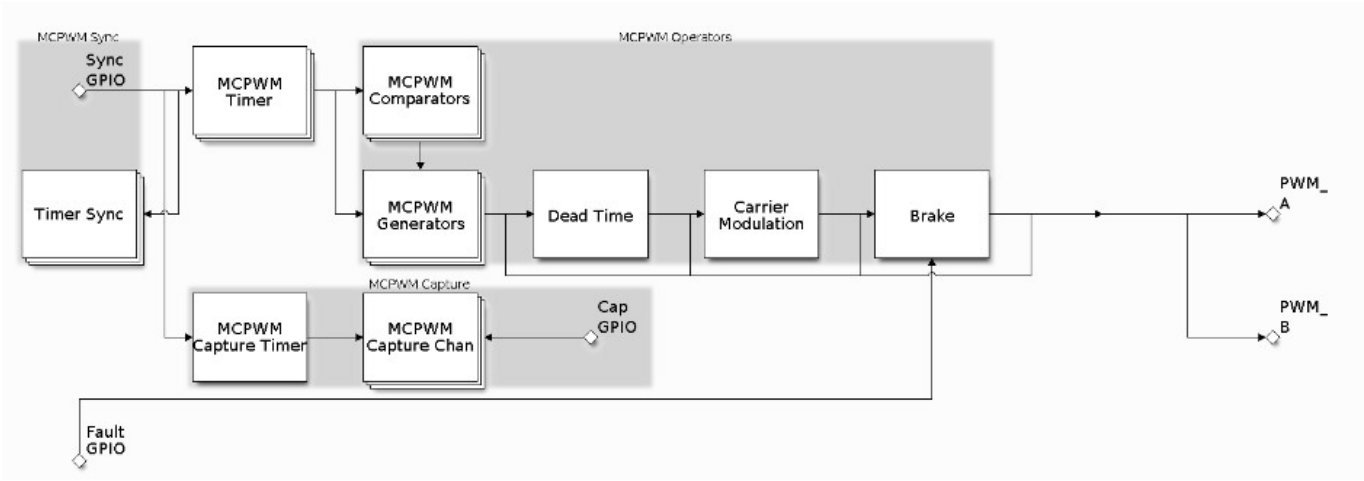


Figure III.10: MCPWM Overview

- **MCPWM Timer:** Determines the PWM signal's time base and event timing.
- **MCPWM Operator:** Generates PWM waveforms and includes submodules like comparator, PWM generator, dead time, and carrier modulator.
- **MCPWM Comparator:** Compares time-base count with threshold values, generating compare events.
- **MCPWM Generator:** Produces complementary or independent PWM waves based on triggered events.
- **MCPWM Fault:** Detects external faults and safeguards the system.
- **MCPWM Sync:** Synchronizes MCPWM timers to achieve a fixed phase difference in PWM signals.
- **Dead Time:** Inserts additional delay to PWM edges.
- **Carrier Modulation:** Modulates a high-frequency carrier signal into PWM waveforms.
- **Brake:** Controls how generators respond to detected faults, either shutting down or regulating PWM output.
- **MCPWM Capture:** Standalone submodule for precise pulse width measurement using dedicated timer and GPIO channels, optionally synchronized with MCPWM Sync.

2. **LEDC (LED Control):** The primer role of this peripheral is to control the intensity of LED dimming. Nevertheless, it can also be used to generate PWM signals for other purposes. LEDC has 8 channels which can generate independent wave-forms that can

be used, for example, to drive the speed of direct current motor. LEDC channels are divided into two groups.

(a) Group of LEDC channels operates in high speed mode. This mode is implemented in hardware and offers automatic and glitch-free changing of the PWM duty cycle.

(b) Group of channels operate in low speed mode, the PWM duty cycle must be changed by the driver in software.

Each group of channels is also able to use different clock sources. The PWM controller can automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference. To configure a LEDC channel 3 steps must be done:

(a) Timer configuration by specifying the PWM signal's frequency and duty cycle resolution.

esp_err_t ledc_timer_config(const ledc_timer_config_t *timer_conf)
Parameters: •**timer_conf:** Pointer of LEDC timer configure struct
Returns: •**ESP_OK:** Success •**ESP_ERR_INVALID_ARG**: Parameter error •**ESP_FAIL:** Can not find a proper pre-divider number base on the given frequency and the current duty_resolution. •**ESP_ERR_INVALID_STATE**: Timer cannot be de-configured because timer is not configured or is not paused.

(b) Channal configuration by associating it with the timer and GPIO to output the PWM signal.

esp_err_t ledc_channel_config(const ledc_channel_config_t *ledc_conf)
Parameters: •**ledc_conf:** Pointer to the LEDC channel configure struct.
Returns: •**ESP_OK:** Success •**ESP_ERR_INVALID_ARG**: Parameter error

(c) Modify PWM signal that drives the output in order to change motor speed. This can be done under the full control of software or with hardware fading functions.

esp_err_t ledc_update_duty(ledc_mode_t speed_mode, ledc_channel_t channel)
Parameters: •**speed_mode:** Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode. •**channel:** LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
Returns: •**ESP_OK:** Success •**ESP_ERR_INVALID_ARG**: Parameter error

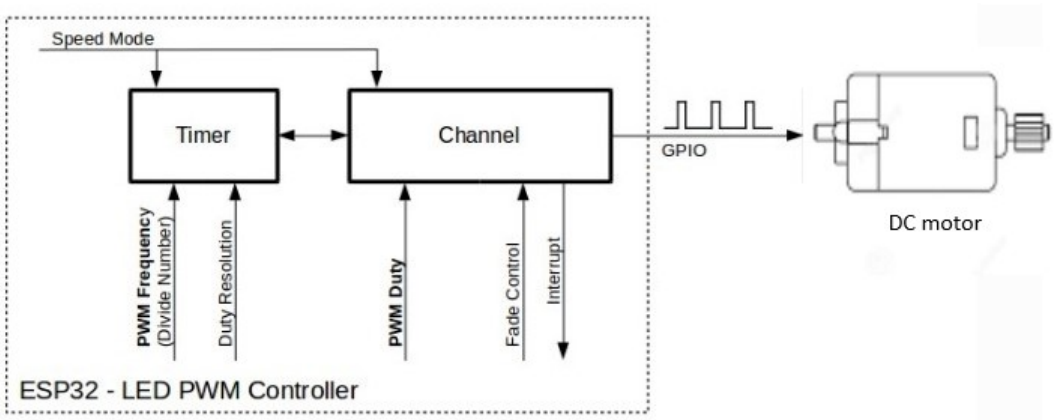


Figure III.11: LED PWM controller

III.7 RTOS Operating System

RTOS or Real-Time Operating System is a type of operating system designed for system requiring prompt and predictable responses to events within a specified amount of time [11]. In such systems, it is critical that each process provides predictable and deterministic behavior. In recent years, RTOSs have gained a larger market share (figure III.12) due to their use in a wide range of fields, including automotive systems, industrial control systems, medical devices, and consumer electronics. Examples of popular RTOS systems include FreeRTOS, Zephyr, VxWorks and QNX.

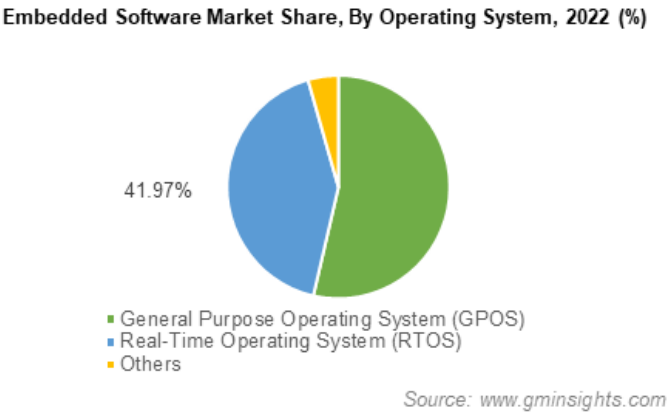


Figure III.12: RTOS market

RTOSs differ from general-purpose operating systems, such as Windows or Linux, in several ways:

- **Preemptive multitasking:** RTOSs allow tasks to be interrupted and resumed at any time, based on priority levels assigned to each task. This enables the system to respond promptly to high-priority events, even if lower-priority tasks are currently executing.
- **Deterministic behavior:** RTOSs provide predictable and consistent response times for tasks, even in the presence of hardware interruptions or other system events.
- **Small memory footprint:** RTOSs typically have a small memory footprint, which is important for resource-constrained devices such as micro-controllers.
- **Interrupt management:** RTOSs provide advanced interrupt management capabilities, which enable them to respond promptly to external events and signals.

III.7.1 FreeRTOS

FreeRTOS is one of the most popular open source RTOS implementations for micro-controllers and small microprocessors. Distributed freely under the MIT [16] open source license as mentioned in [15] , FreeRTOS includes a kernel and a set of libraries suitable for IoT applications.

III.7.2 ESP-IDF FreeRTOS

ESP-32 uses ESP-IDF FreeRTOS which is a FreeRTOS implementation based on Vanilla FreeRTOS v10.4.3, but contains significant modifications to support SMP[17]. ESP-IDF FreeRTOS supports two cores at most (i.e., dual core SMP) and offers a range of features and tools that allow developers to create complex applications with ease. Some of the key concepts used include Timers, Tasks, Queues, Interrupts, WiFi, GPIO, etc.

1. **Tasks:** In ESP-IDF with FreeRTOS, a task represents an independent unit of execution that performs a specific function within the multitasking environment. It is a sequential flow of code that runs concurrently with other tasks and is managed by the scheduler based on priority and scheduling algorithms. Depending on [2] tasks are created using the **xTaskCreate()** function and can be deleted using **vTaskDelete()**, tasks can be suspended and resumed using **vTaskSuspend()** and **vTaskResume()**, respectively, the **vTaskDelay()** function allows tasks to introduce delays, and task priorities can be set with **uxTaskPrioritySet()**. Tasks enable

[16] Massachusetts Institute of Technology.

[17] SMP (Symmetric Multiprocessing) is a microprocessor structure consisting of two or more identical CPUs (cores) connected to a single shared main memory and controlled by a single operating system.

concurrent and coordinated execution in ESP-IDF applications, facilitating responsive and efficient systems.

2. **Timers:** A timer in ESP-IDF with FreeRTOS is a software component that facilitates scheduling and execution of tasks or events at specific intervals. It relies on the FreeRTOS software timer functionality, which is part of the FreeRTOS real-time operating system. According to [3], timers can be created using the **xTimerCreate()** function, specifying parameters such as the timer period, type, and callback function, they can be started with **xTimerStart()**, stopped with **xTimerStop()**, and reset with **xTimerReset()**, a timer can be deleted using **xTimerDelete()**, the period of a running timer can be modified using **xTimerChangePeriod()**. When a timer expires, the associated callback function is automatically executed to perform the desired action or trigger a task. Timers in ESP-IDF with FreeRTOS provide a reliable and efficient mechanism for timed event management in IoT applications.

3. **Queue:** In ESP-IDF with FreeRTOS, a queue is a synchronized and thread-safe data structure that facilitates inter-task communication and synchronization. It operates on a first-in, first-out (FIFO) basis, ensuring that data items are retrieved in the same order they were added. Queues are created using **xQueueCreate()** and support functions such as **xQueueSend()** and **xQueueReceive()** to send and receive data between tasks. These functions can block tasks if the queue is full or empty, and they can include timeout values for controlled waiting periods. Queues offer a reliable mechanism for tasks to exchange data and control the flow of execution, enabling the development of responsive and coordinated systems in ESP-IDF with FreeRTOS.

III.8 Conclusion

During this chapter , we have tried to introduce ESP-32 micro-controller and discussing its powerful functionalities when it is used with ESP-IDF FreeRTOS operating system. Compared to other micro-controller with the same cost range such as Arduino, STM32 or Raspberry Pico. ESP-32 has a more powerful processor and integrated Wi-Fi and Bluetooth connectivity, making it more suitable for Internet of Things (IoT) applications. The ESP-32 also has a higher pin count, allowing for more peripherals to be connected. Moreover, it is endorsed by a very large community and very rich documentation.

CHAPTER IV

AUTONOMOUS MOBILE ROBOT PROTOTYPE

IV.1 Introduction

In this chapter, we will exploit the theoretical results obtained from former chapters to build a differential wheeled robot which will be controlled by an embedded ESP-32 micro-controller. We seek through this project to arm this robot with self decision capabilities allowing it to explore its environments and to define paths toward a given referenced final goal. However, to achieve this level of decision autonomy, many steps must be done first. Starting by the absolute need to define fine-grained mechanism for self localization. Otherwise, the robot must know at each instant its pose in a global coordinate system putting in mind the next assumptions: (1) The robot works in indoor environment which make out of use of GPS sensor. (2) The distance precision is in the order of centimeters which make very hard using accelerometer sensors. And finally (3) The only used sensors are DC motors speed encoder and ultra-sonic obstacle detector. Based on these limitations, we will try to response through the next sections to the question "How to calculate the instantiate pose of the robot with a minimal error ?".

IV.2 Background Works

Due to the noisy nature of used sensors, and the fact of imperfect robot dimension relative to wheel deformation and slippage. it is impossible to get a precise pose for the robot based only on acquired sensor information. So, the use of approximate methods is essential. In this direction, many works have been employed in the literature to improve the accuracy and reliability of the predictions. From these

approaches, we can distinguish:

(a) **Kalman Filtering**: Kalman filters are widely used for sensor fusion and state estimation in robotics. They combine measurements from multiple sensors with knowledge about the system dynamics to estimate the true state of the robot. Kalman filters are effective in handling sensor noise and can provide accurate pose estimation. In this mentioned article [8] there is an example of using Kalman filtering with HC-SR04 ultrasonic sensor.

(b) **Particle Filtering** (Monte Carlo Localization): Particle filtering is a probabilistic method that represents the robot's pose using a set of particles [13]. Each particle carries a hypothesis about the robot's pose, and they are updated based on sensor measurements. Particle filtering is robust to sensor noise and can handle non-linearities in the robot's motion model.

(c) **Sensor Fusion:** Sensor fusion involves combining measurements from different sensors to obtain a more accurate estimate of the robot's pose. This can be done using techniques such as weighted averaging, Bayesian filtering, or information fusion methods like the Dempster-Shafer theory or Fuzzy logic [35]. By fusing data from multiple sensors, the overall pose estimation can be improved, reducing the impact of sensor noise.

(d) **Filtering Techniques:** There are various filtering techniques that can be applied to sensor data to reduce noise and improve accuracy. These include low-pass filters, median filters, or adaptive filters like the Kalman filter mentioned earlier. Applying appropriate filters to the sensor measurements can help in reducing the impact of noise on pose estimation.

(e) **Sensor Calibration:** Calibrating the sensors is crucial to reduce systematic errors and improve accuracy. Calibration involves determining the relationship between the sensor measurements and the actual physical quantities they represent. By accurately calibrating the sensors, the noise and biases can be minimized, leading to more reliable pose estimation.

In practice, a combination of these techniques may be employed to achieve the best possible pose estimation results in the presence of noisy sensors.

IV.3 Project Road Map

To realize our project, we have adopted the following approach (figure IV.1):

- Firstly, we have focused on realizing a real mobile wheeled robot. This involves designing and assembling the physical components, such as the chassis, wheels,

motors and sensors, to create a functional robot. We tried to select appropriate sensors capable of accurately capturing the robot's pose and motion. Unfortunately, the available sensors in our computer science department does not meet the needed requirement.

- Once the real robot is built, the next step is to program a simulation model for the robot. This involves developing a virtual representation of the robot, including its geometry, kinematics and sensor characteristics. The simulation model should accurately mimic the behavior of the real robot, allowing for the testing and evaluation of various calculation and control strategies without the need for physical experimentation.
- With both the real robot and simulation model in place, the acquired data from both sources can be collected and compared. This entails performing experiments and running the robot in different scenarios, while simultaneously running the simulation model with equivalent inputs. The data collected from the real robot and the simulated robot can then be analyzed and compared to assess the performance and accuracy of the simulation model.

By comparing the acquired data from both the real and simulated robot, it becomes possible to evaluate the simulation model's fidelity and its ability to accurately represent the behavior of the real robot. Errors between the two datasets can help identify areas where the simulation model may need refinement or calibration. Conversely, if the acquired data from the real and simulated robot align closely, it provides confidence in the simulation model's accuracy and its suitability for further experimentation and development.

Through this roadmap, the integration of a real mobile wheeled robot, the development of a simulation model, and the comparison of acquired data from both sources enable a comprehensive understanding of the robot's behavior. This iterative process facilitates improvements in control algorithms, sensor fusion techniques, and overall system performance, contributing to the advancement of robotics research and development.

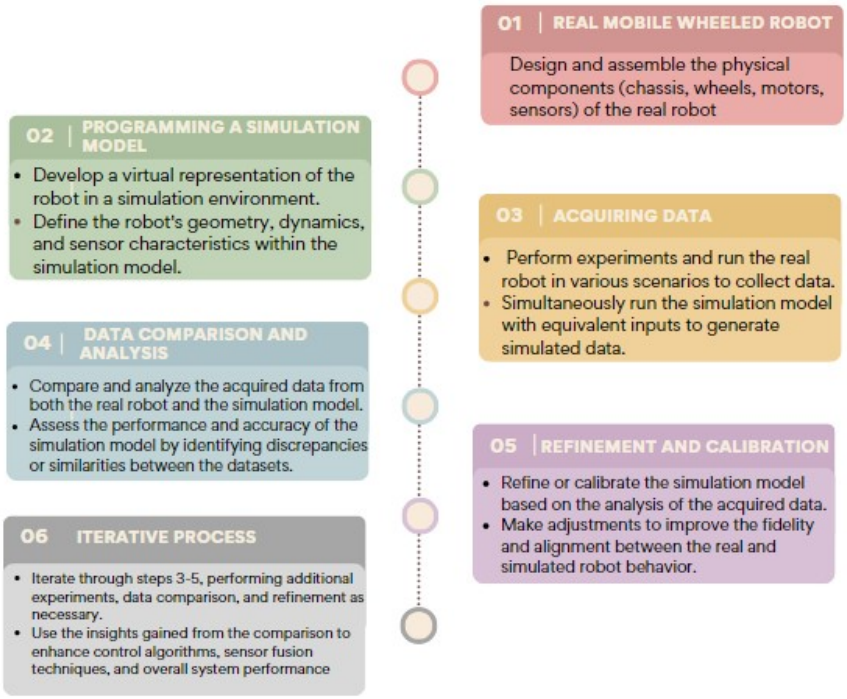


Figure IV.1: Project road map

IV.4 Differential Wheeled Robot Construction

We have used an acrylic robot chassis made with laser-CNC[1] machine. The chassis is equipped with two dual shaft motors for either side. The mechanical structure is simple, 2 DC gear motors, flexible turning, eco-friendly chassis that is stable, and very easy to expand. The robot features a universal castor to avoid slippage.

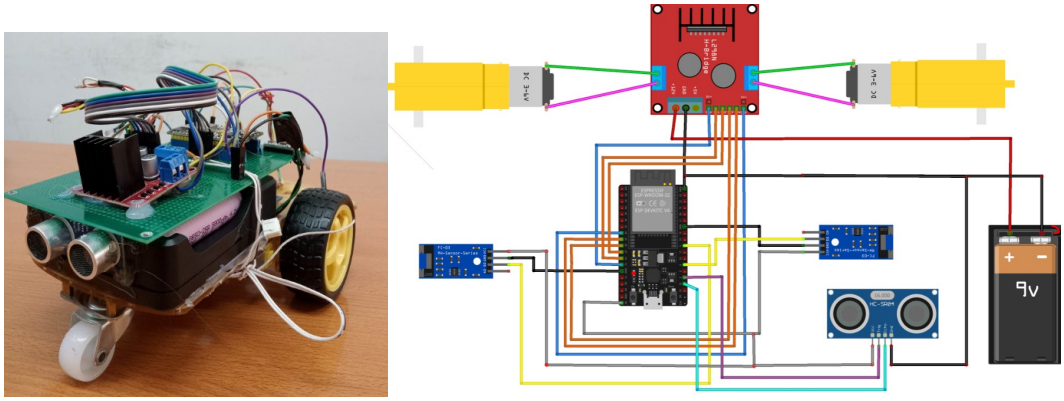


Figure IV.2: The robot chassis and electronic connection schema

[1]CNC stands for Computerized Numerical Control. It is a computerized manufacturing process in which pre-programmed software controls the movement of laser head to cut or draw on a plane gadgets.

IV.4.1 DC Gearbox Motor

A DC gearbox motor is an electric motor that integrates a gearbox with a direct current (DC) motor, as it is showen in the figure IV.3. The gearbox, consisting of multiple gears arranged in stages, works in conjunction with the motor to reduce the rotational speed and increase the torque output. This combination allows for precise control of speed and provides higher torque at lower speeds, making DC gearbox motors well-suited for applications requiring precise speed control and high torque, such as robotics, automation systems, industrial machinery, electric vehicles, and appliances. For more details about our motor navigate into [12].

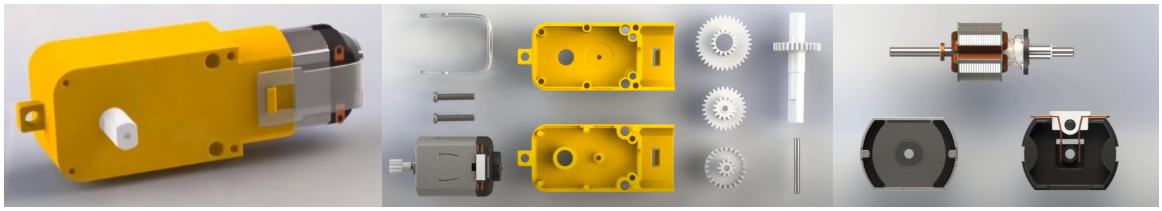


Figure IV.3: The DC gearbox motor

IV.4.2 L298N DC Motor Driver

The L298N DC motor driver presented in figure IV.4 is an integrated circuit commonly used to control and drive DC motors with a wide range of voltages and currents. It serves as a bridge between a micro-controller and the DC motor, enabling precise control over its speed and direction. The L298N includes built-in H-bridge circuits [2], which allow for bidirectional control of the motor by switching the direction of current flow. Additionally, the L298N provides protection features such as over-current and over-temperature protection, enhancing the safety and reliability of the motor system. Its versatility and ease of use make the L298N DC motor driver a popular choice in various applications, including robotics, automation, and hobby projects.

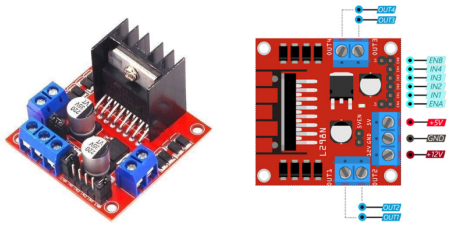


Figure IV.4: L298N DC Motor Driver

[2]H-bridge circuits are like traffic junctions for electricity. They have four switches that can make the electricity go forward or backward through a motor, helping us control its direction easily.

We can organize the figure IV.3 into a table extracted from [26]. This table presents the pin-out of L298N DC motor driver with more details :

Pin name	Description
IN1	Motor A input pin. To control the spinning direction of Motor A.
IN2	Motor A input pin. To control the spinning direction of Motor A.
IN3	Motor B input pin. To control the spinning direction of Motor B.
IN4	Motor B input pin. To control the spinning direction of Motor B.
ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1	Output pin 1 of Motor A
OUT2	Output pin 2 of motor A
OUT3	The output pin of Motor B
OUT4	Output pin 2 of motor B
12V	12V input from a DC power source
5V	Supplies power for the switching logic circuitry inside L298N
GND	Ground pin

Table IV.1: pin-out of L298N motor driver

The L298N Motor Driver is a popular integrated circuit controlling DC motors with the L298N is a great way to make our robot move. The L298N motor driver acts as a translator between our robot's brain (ESP-32 micro-controller) and the motors. By sending signals (HIGH '1' or LOW '0') to the input pins (IN1, IN2, IN3, IN4) of the L298N, we can tell the motors which way to spin. The L298N also allows you to generate pulse width modulation (PWM) signals for precise speed control using the ENA and ENB pins. By connecting the output pins (OUT1, OUT2, OUT3, OUT4) of the L298N to the motors, we provide them with the power they need to turn. With the L298N motor driver, we can easily control the movement and speed of the motors in your robot, helping it to navigate and perform tasks. The input pin states for different robot directions are as follows:

Direction	IN1	IN2	IN3	IN4
Forward	LOW	HIGH	LOW	HIGH
Backward	HIGH	LOW	HIGH	LOW
Right	LOW	HIGH	LOW	LOW
Left	LOW	LOW	LOW	HIGH
Stop	LOW	LOW	LOW	LOW

Table IV.2: The input pin states for different robot directions

IV.4.3 HC-SR04 Ultra-Sonic Detector

The HC-SR04 ultrasonic sensor utilizes sonar technology to measure the distance between itself and an object. It is capable of accurately detecting distances ranging from 2cm to 400cm, with a precision of 0.3cm. This range and accuracy make it suitable for various projects. Moreover, the HC-SR04 sensor module includes both ultrasonic transmitter and receiver components, enabling it to emit and capture ultrasonic signals for distance measurement. For more specifications about this sensor check [28]. The pin-out of the ultrasonic sensor can be customized according to the information presented in the table IV.3:

Pin Number	Pin Name	Description
1	Vcc	The Vcc pin powers the sensor, typically with +5V
2	Trigger	Trigger pin is an Input pin. This pin has to be kept high for 10us to initialize measurement by sending US wave.
3	Echo	Echo pin is an Output pin. This pin goes high for a period of time which will be equal to the time taken for the US wave to return back to the sensor.
4	Ground	This pin is connected to the Ground of the system.

Table IV.3: Ultrasonic sensor pin-out

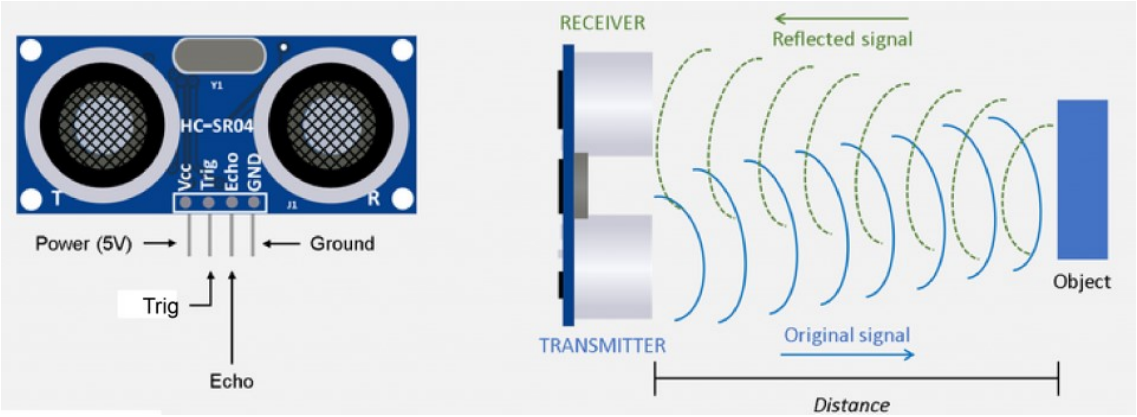


Figure IV.5: HC-SR04 Ultra-Sonic detector

According to [34], the HC-SR04 ultrasonic sensor operates in the following steps:

(a) Emitting sound wave: The trig pin of the ultrasonic sensor emits a high-frequency sound wave with a frequency of 40 kHz.

(b) Wave propagation: The emitted sound wave travels through the air and may encounter objects in its path. If an object is present, the sound wave bounces back towards the sensor module. As it is presented in figure IV.5.

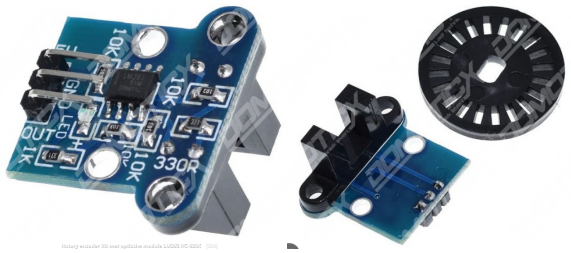
(c) Echo reception: The ultrasonic sensor's echo pin receives the reflected sound wave, also known as an echo.

(d) Distance calculation: The time between the transmission and reception of the signal allows us to calculate the distance to an object. This is possible because we know the speed of sound in the air, which is approximately equal to 343 m/s at a temperature of 20°C. The distance to the object can be calculated using the following formula:

$$\text{DistanceToAnObject} = \frac{\text{SpeedOfSoundInTheAir} \times \text{Time}}{2}$$

IV.4.4 HC-020K Encoder

The HC-020K encoder is a sensor designed to measure the rotational speed of our DC motor. It is based on a black encoder disk attached to the motor shaft, which rotates and triggers the sensor to generate electrical signals. These signals are then processed to calculate the motor speed based on the number of rotations per unit time. For more features about this encoder check [32].



(a) Its front facade (b) Its back facade

Figure IV.6: HC-020k encoder

It pins are typically labeled as follows:

(a) VCC: This pin is used to supply power to the encoder. It typically requires a voltage input between 3.3V to 5V.

(b) GND: This pin is the ground connection for the encoder.

(c) OUT: This pin is the output pin for the encoder signals. It generates a square wave signal that corresponds to the rotation of the encoder disk.

IV.4.5 ESP-32 Micro-controller

ESP32 micro-controller (see chapter III.1) serves as the pivotal component at the core of our robot. It collects data from various sensors (encoders and ultra-sonic

sensors). This data is then analyzed and processed within to derive meaningful insights about the robot's surroundings and its position. Leveraging the power of the MQTT protocol and based on a WiFi connection, the ESP32 efficiently transmits this analyzed information to the command and visualization station as JSON message. This seamless communication enables the command and visualization station to have real-time access to vital data, allowing for informed decision-making and control. Additionally, the ESP32 micro-controller plays a key role in executing commands received from the command station also as JSON messages, directly interfacing with the motors. By intelligently determining the appropriate speed and direction of rotation, the micro-controller ensures precise and accurate movements (forward, back word, turn to left or right and stop) and eventually to determine motion steps to reach a final pose. The next algorithm resumes the logic done by micro-controller:

Algorithm 1 Robot Control Algorithm

```
Configure WiFi Connection
Configure MQTT Protocol
while Robot is active do
    Collect sensor data
    Analyze sensor data
    Send analyzed data to command station using MQTT protocol
    Receive commands from command station
    Determine motor speed and direction based on received commands
    Actuate motors accordingly
end while
```

In the following we detail each of the instruction of the last algorithm:

(a) **WiFi Connection Configuration:** Under ESP-32 , WiFi usage go through 3 steps : (1) Station configuration (2) Event handlers definition and (03) Service starting. Figure IV.7 shows a snapshot of main instruction to do that .

```
static void event_handler(void* arg, esp_event_base_t event_base,int32_t event_id, void* event_data)
                                    {

                                    }
          wifi_config_t wifi_config = {
        .sta = {
            .ssid = WIFI_SSID,
            .password = WIFI_PASS,
            .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
            .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,
        },
    };
    esp_wifi_set_mode(WIFI_MODE_STA);
    esp_event_handler_instance_register(WIFI_EVENT,ESP_EVENT_ANY_ID,&event_handler,NULL,&instance_any_id));
    esp_wifi_start();
```

Figure IV.7: WiFi connection configuration

(b) **MQTT Protocol Configuration:** ESP-MQTT is an implementation of MQTT protocol client for ESP-32 micro-controller. The minimal configuration for its usage is given by the code given in Figure IV.8:

```
const esp_mqtt_client_config_t mqtt_cfg = {
   .broker.address.uri = "mqtt://mqtt.eclipseprojects.io",
};
esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, client);
esp_mqtt_client_start(client);
```

Figure IV.8: MQTT protocol configuration

(c) **Sensor Data Collection, Analyze and Sending:** ESP-32 micro-controller receives pulse signals from the encoders attached to the left and right motors, enabling it to measure the number of rotations performed by each wheel. By converting these pulses into rotational speed in radians per second, the ESP32 micro-controller calculates the linear speed and rotational speed of the robot. These derived values are instrumental in approximating the robot's position. Once all the necessary data is collected, the ESP32 constructs a JSON message formatted according to the specifications illustrated in the provided figure IV.9using appropriate libraries [3], and transmits it to the command station via an MQTT message.

```
 // json_message:
{
 "rwv": "...",                // right wheel velocity
 "lwv": "...",                // left wheel velocity
 "teta": "...",               // oriontation
 "x": "...",                  // x coordinate of position
 "y": "...",                  // y coordinate of position
 "v": "...",                  // liniere speed
 "omega": "...",              // rotation speed
}

esp_mqtt_client_publish(client, "/topic/robotMotionData", json_message, strlen(json_message), 0, 0);
```

Figure IV.9: Data collection, analyze and sending

To realize these actions, we have instrumented many RTOS primitives. Starting by configuring interrupts ISR (Interrupt Service Routines) on the positive eage of input pins attached to encoders. Next, we have used real-time and threading safe queues for inter tasks communication. Moreover, many tasks have been created and scheduled via the RTOS tasks manager to obtain a coherent behavior of the overall system deployed on the ESP32- micro-controller. Finally, an RTOS timer is configured to send periodically collected data to the visualization and command station. This logic is resumed in Figure IV.10

[3]json.h from https://github.com/nkolban

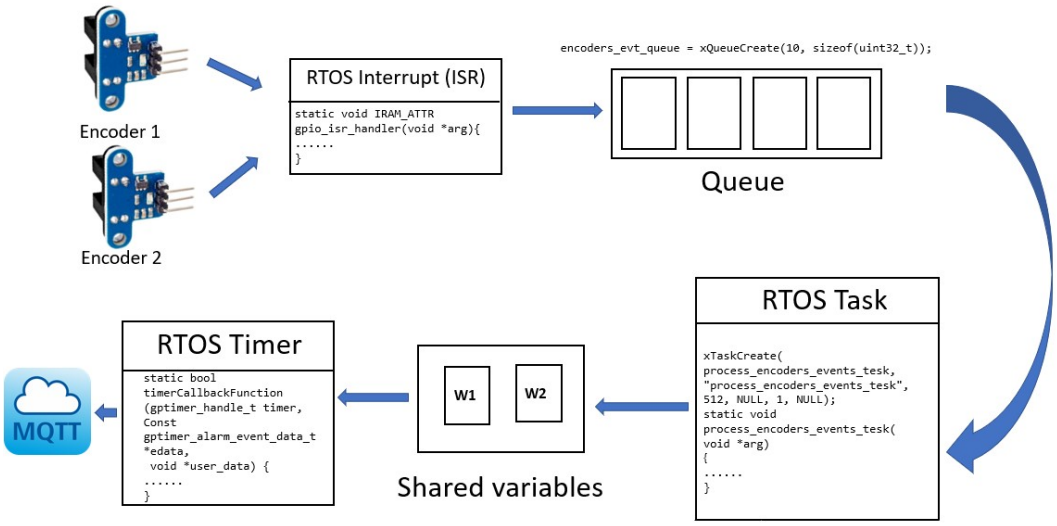


Figure IV.10: Sensor data collection, analyze and sending using RTOS primitives

(d) **Receiving Commands and Controlling Actuators:** After subscribing to receive JSON messages containing commands from the command station, the micro-controller processes the received commands. Based on the analysis, the micro-controller directs the two motors of the robot accordingly. The available commands for direct manipulation include moving forward, moving backward, turning left, turning right, and stopping. For reverse kinematics, we have implemented only one command consisting to navigate towards a specific goal using the algorithm given in section II.6. By interpreting the received commands, the micro-controller orchestrates the appropriate motor actions, enabling the robot to execute the desired movements with precision and accuracy.

```
// json_message:

        obj.put("x",... );
        obj.put("y", ...);
        obj.put("teta", ...);
        obj.put("msgid", ...);
        obj.put("source", ...);
        obj.put("rwv", ... );
        obj.put("lwv", ... );
        obj.put("v", ... );
        obj.put("omega", ... );


void robot_control_task(char *  command)
{
          if (strcmp(command, "forward") == 0) {}

          else if (strcmp(command, "back") == 0) {}
          else if (strcmp(command, "left") == 0) {}
          else if (strcmp(command, "right") == 0) {}
          else if (strcmp(command, "stop") == 0) {}
          else if (strcmp(command, "adjust") == 0) {}
          else if (strcmp(command, "goto") == 0) {}
}
```

Figure IV.11: Receiving commands and controlling actuators

IV.5 Differential Robot Simulation Model

As stated in the road map presented earlier, in addition to programming the control code for the real robot, we have also developed a simulation model as a part of our project. The primary objective is to continuously evaluate the accuracy of the pose predictions made by the real robot, based on sensor data, by comparing them with the abstract values calculated by the simulation using the mathematical model. At each instance, we measure the distance between the predicted pose and the absolute calculation performed by the simulation. To minimize the error between the real and simulated results, various strategies and techniques can be applied. These include refining sensor calibration, enhancing sensor fusion algorithms, fine-tuning control parameters, and exploring advanced localization and mapping approaches (see IV.2 section). Unfortunately, our work is limited to highlighting the error. But we did not come to apply the different techniques for its minimization due to the lack of time.

Technically specking, the simulation model is composed of multiple threads to handle various aspects of the robot's behavior:

(a) **Thread 1: Position Calculation and Time Advancement** this thread is responsible for calculating the robot's next position based on its current position and advancing time by an infinitesimal value. It utilizes the robot's local motion vector (angular velocity and tangential velocity) defined in II.7, II.8 to determine the displacement and orientation change of the robot. By continuously updating the robot's pose, this thread simulates the robot's motion in the simulated environment.

```
public void run() {
        timeStamp = 5e-3;
        time = 0;
        double pose[] = {x,y,teta};
        while(true)
        {
            synchronized (this) {
            time=time+timeStamp;
            pose[0]= pose[0]+getLocalTangentialVelocity(time)*timeStamp*Math.cos(pose[2]);
                  pose[1]= pose[1]+getLocalTangentialVelocity(time)*timeStamp*Math.sin(pose[2]);
                pose[2]=(pose[2]+getLocalAngularVe'ocity(time)*timeStamp)%(2*Math.PI);
                                    }
            setPose(pose[0], pose[1], pose[2]);

            Thread.sleep((long) (timeStamp *1000));
        }
     }
```

Figure IV.12: Position calculation and time advancement

(b) **Thread 2: Wheel Speed Adjustment and Control** the second thread is dedicated to adapting the speeds of the left and right wheels based on the instructions received from the command station or internal calculations for controlling the robot's direction towards a given objective. It takes into account the control strategies implemented in the model, which are:

i. **Basic Controller** given in figure IV.13: This strategy consists to control the robot to a final pose with a tangential velocity proportional to the distance between the current position and the final position.

ii. **Distance Based Controller** given in figure IV.14: This Strategy consists to calculate the distance to the goal position and then to move that distance at max speed (for real robot we can decrease the speed when the robot approaches to the final position to avoid going over the goal).

```
public void run() {
 // step 1 calculate phi_r(t)= arctg(y_ref-y)/x_ref-x)
      phi_r = Math.atan2(yRef-y, xRef-x);
  // step 2 make the robot ahead with the final pose.
 do {
    omega= k1*(phi_r- teta);
    setAnglarVelocityRightWheel(distanceBetweenWheels*omega/2);
    setAngularVelocityLeftWheel(-distanceBetweenWheels*omega/2);
    phi_r = Math.atan2(yRef-y, xRef-x);
  }while(Math.abs(teta-phi_r)>accepted_error_1);

 // step 2 go to the final position

 do {
     v_error = Math.sqrt(Math.pow(xRef-x,2)+Math.pow(yRef-y, 2));
     v =k* v_error;

     setAnglarVelocityRightWheel(Math.min(v, max_angular_velocity));
     setAngularVelocityLeftWheel(Math.min(v, max_angular_velocity));

 }while(v_error>accepted_error2);


 // step 3 turn to the final orientation

 phi_r = Math.abs(teta-tetaRef);

 do {
        omega= k3*phi_r;
        setAnglarVelocityRightWheel(distanceBetweenWheels*omega/2);
        setAngularVelocityLeftWheel(-distanceBetweenWheels*omega/2);
        phi_r = teta-tetaRef;
 }while(Math.abs(teta-tetaRef)>accepted_error3);

setAnglarVelocityRightWheel(0);
   setAngularVelocityLeftWheel(0);
      }
```

Figure IV.13: Wheel speed adjustment and control for reverse kinematics (to reach to a reference final pose) with the basic controller strategy

```
   public void run() {
// step 1 calculate phi_r(t)= arctg(y_ref-y)/x_ref-x)  similar to Basic Controller strategy

// step 2 go to the final position

double distanceToWalk= Math.sqrt(Math.pow(xRef-x,2)+Math.pow(yRef-y, 2));
double walkedDistance=0;
double t0 = time;
double v=  max_angular_velocity;

while(walkedDistance < distanceToWalk)
{
    if((distanceToWalk-walkedDistance)==DistenceBeforGoal )
    {
        setAnglarVelocityRightWheel(v/3);
        setAngularVelocityLeftWheel(v/3);
        double t = time-t0;
           walkedDistance=getLocalTangentialVelocity(0)*t;
     }
    else {
    setAnglarVelocityRightWheel(v);
    setAngularVelocityLeftWheel(v);
   double t = time-t0;
   walkedDistance=getLocalTangentialVelocity(0)*t;
   System.out.println(distanceToWalk+","+walkedDistance);
    }

}

// step 3 turn to the final orientation   similar to Basic Controller strate

setAnglarVelocityRightWheel(0);
setAngularVelocityLeftWheel(0);
   }
```

Figure IV.14: Wheel speed adjustment and control for reverse kinematics (to reach to a reference final pose) with the distance based controller strategy.

By adjusting the wheel speeds, this thread governs the robot's movement and ensures it aligns with the desired direction. Figure IV.15 demonstrates the angular and tangential velocities for basic controller strategy. It is clear that the tangential velocity start to increase when the robot reach it goal. It was very difficult to find the best values to avoid this situation and force the robot to stop. However, with the distance based strategy, the robot behavior is more stable, as shown in figure IV.16.

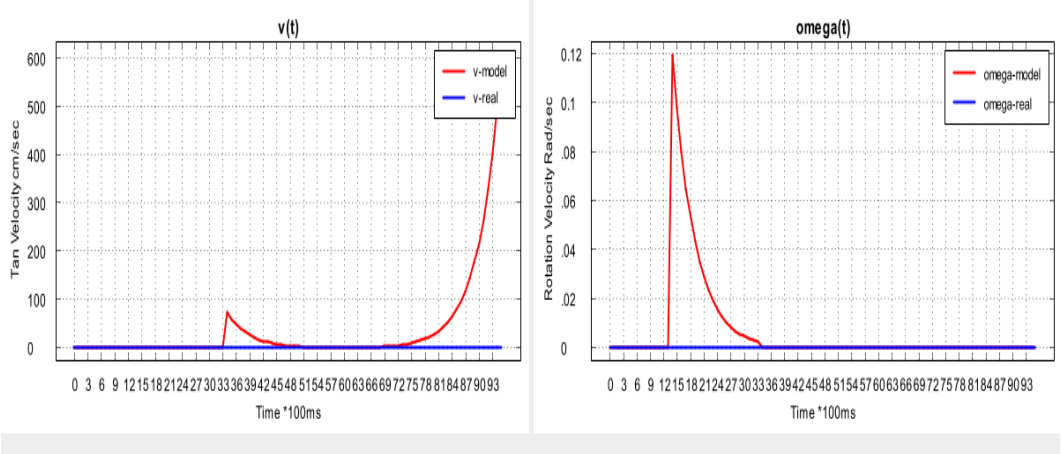


Figure IV.15: Angular and tangential velocities of the robot in basic controller strategy

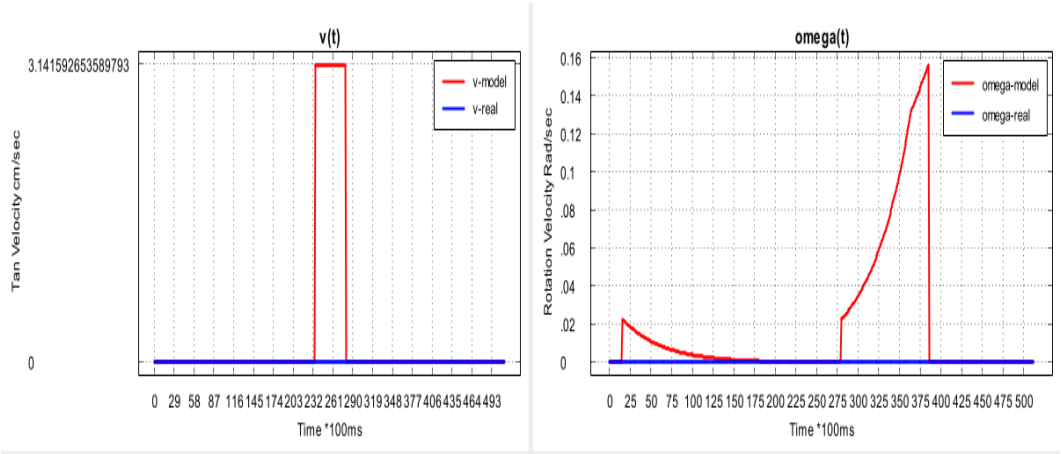


Figure IV.16: Angular and tangential velocities of the robot in distance based strategy

(c) **Thread 3: Information Collection and Transmission** this thread handles the collection and transmission of information from the robot to the command station. At predefined time intervals, it gathers relevant data such as the robot's pose (position and orientation), rotational speed, and linear velocity. This information is then formatted into JSON messages and sent to the command station, enabling real-time monitoring and control of the robot's state.

By utilizing these multiple threads, the simulation model achieves concurrency and

effectively manages different aspects of the robot's behavior. It separates the tasks of position calculation, control adjustment, and information transmission, ensuring efficient execution and accurate simulation of the wheeled differential robot.

IV.6 Data Acquisition and Comparison

Data collection and analysis play a crucial role in our project road map, enabling us to gather valuable insights and make informed decisions. The functionalities of this step can be summarized as follows:

(a) **Information and Command Communication Through MQTT Infrastructure:** In our project, we have implemented an MQTT architecture (depicted in figure IV.17) to facilitate information and command exchange[4] among all components: (1) The visualization and command station, (2) The robot simulation model, and (3) the physical or real robot. Each component is equipped with an MQTT publisher for sending messages and an MQTT subscriber that is subscribed to relevant topics to receive only the messages of interest. At the core of this architecture lies the MQTT Broker, which serves as the central hub for communication. All publishers and subscribers interact with the broker, leveraging its capabilities. The MQTT broker not only enables communication but also provides additional services, including security, quality of service, and scalability adaptation. To administer the MQTT broker, we utilize a versatile Dashboard that offers various statistical views (figure IV.18 illustrates one of these views).

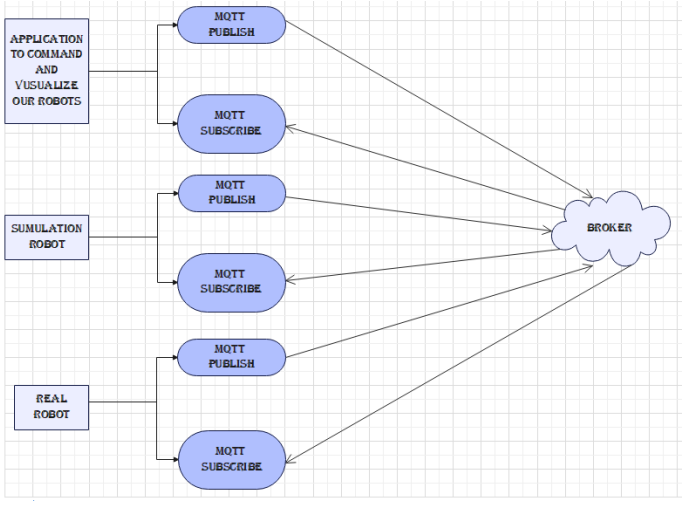


Figure IV.17: MQTT Communication architecture

[4]All information and commands are formatted as JSON messages

This communication architecture ensures efficient and reliable data transfer between the different system components, facilitating effective coordination and synchronization. By leveraging MQTT and its associated services, we create a robust framework for seamless information exchange, enhancing the overall functionality and performance of our project.

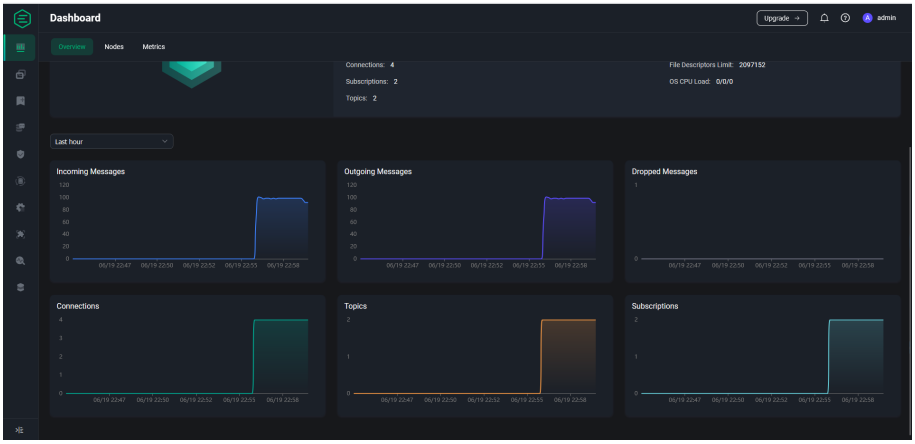


Figure IV.18: MQTT dashBoard

(b) **Visualization of Robot Parameters (Real and Simulation):** A key aspect of the data acquisition and analysis step is the visualization of essential robot parameters and behavior. This includes real-time monitoring and display of the robot's pose (position and orientation) as well as its angular and tangential speed. By visualizing these parameters, we gain valuable insights into the robot's motion and performance, both in real-world scenarios and simulation environments. The figure IV.19 captures the main window of the visualization and command station This window contains four (04) regions:

i. **Command Region:** In which the user can control each robot (simulated and real) by selecting its name in the left tree then apply one of the following command described in the point (c).

ii. **Available Robots Tree**: In which the list of all available robots are displayed with their motion information. The user can select one of many robot to apply different commands.

iii. **The Main Region:** This region represents the global position reference. It is equipped with an orthonormal plan whose unit of measurement is the centimeter. The plane is sensitive to the resizing of the main window. All robots positions are updated and visualized in real-time.

iv. **Logging Region:** This region contains two (02) taps one for MQTT logging (all the sent or received messages are logged) and the second to show graphs of robots motions.

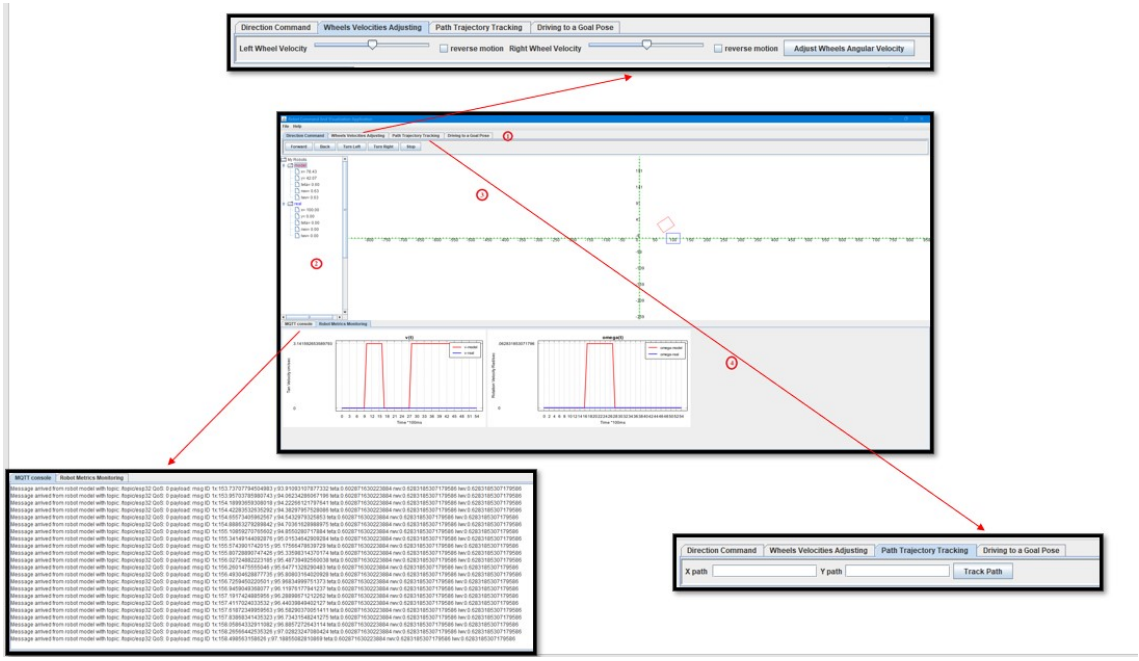


Figure IV.19: Visualization and command station main window

(c) **Remote Control of Real and Simulation Robots:** Another important functionality of this step is the ability to remotely control both real and simulated robots. By leveraging the collected data, we can manipulate the robot's actions, adjust its trajectory, and interact with its behavior. This remote control capability empowers us to conduct experiments, test different control strategies, and evaluate the performance of the robots under various conditions. We can categorize the different commands in four families:

i. **Direct Command:** Go forward, go back, turn left , turn right or stop.

ii. **Wheels Velocity Adjustment:** Where the speed of each wheel can be controlled independently.

iii. **Path Tracking Commands:** For a reverse kinematics control where a mathematical defined path is should be tracked by the robot, (this type of command will be implemented in our further Work).

iv. **Driving to a Goal Pose:** A goal pose is defined, (position and orientation) and the robot try to reach it using one of two strategies presented earlier.

Visualization and command application is a powerful tool for various robots command and motion control. The main purpose of its usage is to control the evolution of the error between the estimated and the real robot pose. But it cans be used for other purposes such as the control of the velocities of right and left wheels, the logging of MQTT communication messages.

IV.7 Conclusion

In conclusion, this chapter has materialized our vision for the implementation of a versatile platform that not only facilitates the realization of an autonomous mobile differential wheeled robot but also provides a comprehensive suite of tools for simulation, experimentation, and data analysis in the realm of understanding differential mobile robots behavior. While we have achieved significant advancements thus far, it is worth to acknowledge that there remains a substantial work to fully operational our platform in all its dimensions. The challenges primarily due to some details, which may seem obvious but often give rise to complex technical issues. Resolving these challenges allows to reach the true potential of our platform and realizing its intended capabilities. Further research and development efforts are required to overcome these obstacles and propel the platform towards its ultimate fruition.

CONCLUSION AND FURTHER WORK

In this thesis, we presented a comprehensive solution for designing and implementing an autonomous mobile robot based on ESP-32. We started by providing an introduction to mobile robots, followed by a detailed explanation of the motion modeling for differential wheeled mobile robots. Then, we studied ESP-32 based systems, including their architecture, programming, and communication protocols. Finally, we designed and implemented a mobile robot prototype that can operate autonomously, perform trajectory planning and localization,

While our solution has several advantages, such as the use of ESP-32 microcontroller that is widely used in IoT and robotics applications, there are also some limitations. For example, our prototype is a differential wheeled mobile robot, and thus our motion model is specific to this type of robot. Also, we did not explore the use of other AI methods that could improve the robot's decision-making process, such as deep learning or reinforcement learning.

Nonetheless, the application we developed has several benefits. Firstly, we developed an interface to control the robot and see its trajectory in real-time, which can be useful for monitoring and debugging purposes. Secondly, we implemented a simulation model for the robot, allowing us to test and validate our solution in a virtual environment. Finally, we built a real robot model that can operate autonomously, which can be useful for various applications that require autonomous robots.

In conclusion, this thesis presented a solution for designing and implementing an autonomous mobile robot based on ESP-32, which includes a motion model, ESP-32 based systems, While our solution has some limitations, it has several advantages, and the results of this work could have important implications for various industries and fields that could benefit from the use of autonomous mobile robots. Future work could focus on improving our solution by exploring other AI methods or implementing it on different types of robots.

REFERENCES

[1] Esp32 series datasheet. https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf. Accessed: 2023-14-06.

[2] Freertos (esp-idf). https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos_idf.html#tasks. Accessed: 2023-03-27.

[3] General purpose timer. https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/peripherals/timer.html. Accessed: 2023-03-27.

[4] Gpio summary. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html#api-reference-normal-gpio/. Accessed: 2023-02-16.

[5] Mqtt: The standard for iot messaging. https://mqtt.org/. Accessed: 2023-06-15.

[6] Wi-fi. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html. Accessed: 2023-06-15.

[7] N. ABEKIRI, A. RACHDY, M. AJAAMOUM, B. NASSIRI, L. ELMAHNI, and Y. OUBAIL. Platform for hands-on remote labs based on the esp32 and nod-red. *Scientific African*, 19:e01502, 2023.

[8] A. R. Al Tahtawi. Kalman filter algorithm design for hc-sr04 ultrasonic sensor data acquisition system. *IJITEE (International Journal of Information Technology and Electrical Engineering)*, 2(1):15–19, 2018.

[9] M. Amina. Conception et réalisation d'un robot mobile suivant une trajectoire commandé par une carte arduinouno. Master's thesis, 2014-2015.

[10] A. F. Amiri and M. D. Abdourahman. Robot mobile autonome. Master's thesis, University of Guelma, September 2020.

[11] B. Amos. *Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools*. Packt Publishing, 2020.

[12] N. Cameron. *Arduino Applied: Comprehensive Projects for Everyday Electronics*. Technology in action. Apress, 2018.

[13] G. Cen, N. Matsuhira, J. Hirokawa, H. Ogawa, and I. Hagiwara. Mobile robot global localization using particle filters. In *2008 International Conference on Control, Automation and Systems*, pages 710–713. IEEE, 2008.

[14] G. Colbach. *The WiFi Networking Book: WLAN Standards: IEEE 802.11 Bgn, 802.11n, 802.11ac and 802.11ax*. Independently Published, 2019.

[15] W. Gay and E. I. M. BV. *FreeRTOS for ESP32-Arduino: Practical Multitasking Fundamentals*. Onleihe. E-Book. Elektor, 2020.

[16] I. E. Guebli. Navigation et commande d'un robot mobile à 4 roues sous ros. Master's thesis, University of Guelma, October 2020.

[17] G. C. Hillar. *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.

[18] D. G. Holmes. *Pulse Width Modulation for Power Converters: Principles and Practice*. Wiley, 2003.

[19] D. Ibrahim and A. Ibrahim. *The Official ESP32 Book*. Learn, Design, Share. Elektor International Media, 2017.

[20] L. Jaulin. *Mobile Robotics*. Wiley, 2019.

[21] G. Klancar, A. Zdesar, S. Blazic, and I. Skrjanc. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Elsevier Science, 2017.

[22] N. kolban. *kolban's book on ESP32*. 2017.

[23] T. Kunasegeran. *Direct Current Motor Control Led by Microcontroller Created PWM*. UMP, 2012.

[24] S. Len. Locomotion d'un robot mobile. Master's thesis, Universite de Liege - Faculte des Sciences AppliqueesInstitut Monteoree, 2007-2008.

[25] E. Matthes. *Python crash course*. no starch press, 2023.

[26] M. M. Maung, M. M. Latt, and C. M. Nwe. Dc motor angular position control using pid controller with friction compensation. *International journal of scientific and research publications*, 8(11):149, 2018.

[27] S. Meyers. *Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14.* " O'Reilly Media, Inc.", 2014.

[28] E. J. Morgan. Hc-sr04 ultrasonic sensor. *Nov*, 2014.

[29] M. Nebbad and A. Lakhdari. Conception et réalisation d'un système de pilotage d'un robot mobile à base d'un micro-contrôleur 80c51. Master's thesis, Univ M'sila, 2019-2020.

[30] S. Noureddine. *SYSTEME DE LOCALISATION POUR ROBOTS MOBILES.* PhD thesis, 23 / 11 / 2005.

[31] G. M. Oliveira, D. C. Costa, R. J. Cavalcanti, J. P. Oliveira, D. R. Silva, M. B. Nogueira, and M. C. Rodrigues. Comparison between mqtt and websocket protocols for iot applications using esp8266. In *2018 Workshop on Metrology for Industry 4.0 and IoT*, pages 236–241. IEEE, 2018.

[32] J. Omidokum. *Design and Implementation of an Autonomous Driving System with a Deep Learning Approach on a Scaled Vehicle Platform.* PhD thesis, 2022.

[33] M. Oudini. Etude et réalisation d'un robot détecteur d'obstacles. Master's thesis, University of Guelma, June 2013.

[34] R. Santos. Complete guide for ultrasonic sensor hc-sr04 with arduino. *Random Nerd Tutorials, November*, 2013.

[35] J. Z. Sasiadek. Sensor fusion. *Annual Reviews in Control*, 26(2):203–228, 2002.

[36] Z. TELDJOUNE and I. GUENNICHE. Navigation autonome des robots mobiles. Master's thesis, Directeur: Mme SEBBAGH Hafidha/Co-Directeur: Mr ABDELLAOUI Ghouti, 2022.

[37] N. M. I. Zekhref Oussama. Localisation et navigation d'un robot mobile par webcam. Master's thesis, 2014-2015.