الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
THE PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمي
THE MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
AMAR TELIDJI UNIVERSITY OF LAGHOUAT
كـلـيـــــة الـتكنولوجيا
FACULTY OF TECHNOLOGY
قسم الالكترونيك
DEPARTMENT OF ElECTRONIC

Master's dissertation

Domain : Science and Technology
Field : Automatic
Option : Automatic and Industrial Informatic

By :
OUSSAMA GUETTAF ***MONCEF ZAKARIA MILOUDIA***

THEME

Q-Learning based Path Planning and Predictive Control for the Navigation of a Mobile Robot

Mme. DJERFAF FATIMA	**Pr.**	***President***
Mme. FEKNOUS SAFIA	***M.C. A***	***Examinator***
Mme . CHOUIREB FATIMA	***Pr.***	***Supervisor***

Academic year 2023/2024

Abstract

Our work aims to find the optimal path to enable a mobile robot to navigate from a starting point to a destination in a known environment, while avoiding obstacles. To achieve this goal, we started by studying and implementing the Model Predictive Control (MPC) framework in the first phase. Then, in a second phase, we explored various state-of-the-art planning algorithms, including Reinforcement Learning approaches. Among the latter, we studied and implemented the Q-Learning algorithm to perform the path planning according to the simulated scenarios. Ours simulations were conducted both using the Matlab environment and the MATLAB-ROS interface along with the Gazebo simulator. The results we obtained were highly reliable.

الملخص

يهدف عملنا إلى إيجاد المسار الأمثل لتمكين الروبوت المتنقل من التنقل من نقطة البداية إلى الوجهة في بيئة معروفة، مع تجنب العوائق. ولتحقيق هذا الهدف، بدأنا بدراسة وتنفيذ إطار نموذج التحكم التنبؤي (MPC) في المرحلة الأولى. ثم، في المرحلة الثانية، استكشفنا العديد من خوارزميات التخطيط الحديثة، بما في ذلك أساليب التعلم المعزز. ومن بين هذه الأخيرة، قمنا بدراسة وتنفيذ خوارزمية Q-Learning لإجراء تخطيط المسار وفقًا للسيناريوهات المحاكاة. تم إجراء عمليات المحاكاة الخاصة بنا باستخدام بيئة Matlab وواجهة MATLAB-ROS جنبًا إلى جنب مع جهاز محاكاة Gazebo. وكانت النتائج التي حصلنا عليها موثوقة للغاية. .

Résumé

Nos travaux visent à trouver le chemin optimal pour permettre à un robot mobile de naviguer d'un point de départ à une destination dans un environnement connu, tout en évitant les obstacles. Pour atteindre cet objectif, nous avons commencé par étudier et mettre en œuvre le la loi de commande Model Predictive Control (MPC) dans la première phase. Puis, dans une

deuxième phase, nous avons exploré divers algorithmes de planification de pointe, notamment des approches d'apprentissage par renforcement. Parmi ces derniers, nous avons étudié et implémenté l'algorithme Q-Learning pour effectuer la planification de parcours selon les scénarios simulés. Nos simulations ont été réalisées à la fois en utilisant l'environnement Matlab et l'interface MATLAB-ROS ainsi que le simulateur Gazebo. Les résultats que nous avons obtenus étaient très fiables.

Acknowledgments

First and foremost, we would like to express our gratitude to God for granting us health and patience to successfully complete this work. We also wish to extend our deep appreciation to our supervisor, M^{me}.CHOUIREB FATIMA, for her continuous support and valuable advice throughout the project.

Her encouragement and guidance have been a source of inspiration and constant support. It has been an honor for us to work under her supervision.

We would also like to thank and express our gratitude to all the professors and administrators in the Electronics Department at Amar Thelidji University of Laghouat for their continuous support and contribution to the success of this work.

Finally, we extend our sincere thanks to the committee members who will oversee the evaluation of our work and provide valuable scientific suggestions and guidance. We also wish to express our gratitude to all those who contributed in any way to the successful completion of this project.

Thank you all…

Contents

List of abbreviations

UAVs	unmanned aerial vehicles
LMRs	legged mobile robots
WMRs	wheeled mobile robots
MPC	Model Predictive Control
ROS	Robot Operating System

List of figures

Introduction

Since the invention of the first robot, the field of robotics has witnessed continuous advancements and numerous innovations, making it one of the most impactful fields due to rapid technological progress. Robots constitute a multidisciplinary field that includes mechanical engineering, mechatronics, electronics, automation, computer science, and artificial intelligence. They have become an integral part of our lives, transitioning from a mere vision of the future to a reality in the present.

Mobile robots hold particular significance in the realm of robotics, as they are capable of navigating their environment, enabling them to perform various tasks. Among the types of mobile robots, robotic vehicles are widely popular due to their diverse applications.

Path planning is one of the essential technologies of wheeled mobile robots, which designs a safe, collision-free and least-cost path based on one or more criteria. It has become an indispensable technique in a wide range of problems such as disaster rescue, autonomous navigation in robotic systems, where robots must navigate through complex environments without human intervention [11] .This technology is crucial in applications like warehouse automation, where robots optimize paths to retrieve and deliver goods efficiently, and in autonomous vehicles, ensuring safe travel while avoiding obstacles. Its applications extend also to agricultural robotics and planetary exploration [18].

In recent years, Reinforcement Learning (RL) is becoming a promising approach for solving path planning problem. In this thesis, we are going to study the Q-Learning algorithm to perform this task according to the simulated scenarios.

Once the path is planned, the robot must follow it to arrive at its destination. Traditionally, there is a wide variety of approaches solving path following problem. Most of these

techniques can be considered as a direct application of classic control theories on a geometry model. The most commonly used methods are pure pursuit, lineof-sight, and constant bearing guidance [11].

However, traditional techniques for the control of nonholonomic wheeled Mobile Robots (WMRs) often do not present good results, due to constraints on inputs or states that naturally arise. Model-based predictive control (MPC) appears therefore as an interesting and promising approach for overcoming the problems above mentioned.

Model Predictive Control (MPC) techniques are considered advanced techniques in control engineering, utilizing a model to predict the behavior of the system being controlled, taking into account possible constraints. This technique was pioneered in the early 1980s by the work of Dr. Clarke, but industrial interest in it grew in the late 1970s. It is characterized by its ability to consider both current and future system dynamics and potential constraints.

In this work, we are interested in the application of MPC to control our WMR in the problem of trajectory tracking.

The rest of this thesis is organized as follows:

The first chapter provides an introduction to the field of robotics by elucidating some key concepts about mobile robots: their definitions, classifications, characteristics, navigation, modeling, and so forth. Control is considered a crucial and necessary task in the study of robots. Therefore, two control techniques for a mobile robot will be proposed at the end of this chapter.

Chapter 2 describes the Q-learning method used in path planning, along with its advantages and disadvantages.

The final chapter elucidates the simulation results and analysis of the methods studied in the preceding chapters using Matlab and Gazebo.

Ultimately, we will conclude this thesis with a comprehensive conclusion summarizing the work done and presenting the prospects of this endeavor.

Chapter1 : Mobile Robots, Modeling and Control

1.1 Introduction

Because they make tough or repetitious everyday chores easier, robots have become an indispensable part of modern human existence, which is becoming harder to ignore.

One of the most significant categories of robots is mobile ones. Their value stems from their versatility, which enables them to solve a wide range of issues (heavy object transportation, space exploration missions, etc.).

This kind of mobile robot has been developed extensively in all of its forms, including the ground mobile robots that are the subject of our study here as we will be dealing with a differential wheeled mobile robot vehicle.

One of the few sophisticated control methods that has a major and pervasive influence on process control in industry is model-based predictive control. It was created and put to use in the sector for over twenty years before many automation researchers became interested in it.

Many researchers have concentrated on the adaptation of the MPC to the control of systems with fast dynamics where the frequencies of sampling are very high, such as robotics, aerospace, automotive, etc., due to the technological advancements in computers and the advent of methods for rapid optimization.

1.2 Typical mobile robots

The mobile robot has tools that enable it to move around its environment. It could have the ability to perceive and reason, depending on its level of autonomy or intelligence [3]. Below are some types of mobile robots.

1.2.1 Legged robots

Robots with legs are locomotion devices that are powered by motors. They can have two or more paws and frequently mimic humans and other creatures with legs.

These robots can traverse various terrains because of their multi-degree-of-freedom anatomy. Because they are designed to carry out a range of jobs for which site access is challenging, they are also adaptable. However, these benefits come at the expense of a vehicle's intricacy of motion, difficulty maintaining the vehicle's stability and balance, and high energy consumption [2].

Figure 1.1 - Examples of legged robots

1.2.2 Crawler robots

These types of robots utilize caterpillar tracks, which are articulated mechanical devices capable of transferring the vehicle's weight to the ground and distributing it across a surface higher than the point of contact of the wheels, to ensure their movement. A stretched band of treads on a series of aligned wheels is called a caterpillar track. They are suspended independently, not in contact with the ground, and they support the top of the band. The surface where the caterpillar tracks make contact with the ground is far more significant than

the area where the wheels or legs meet the ground, providing exceptional mechanical stability and adhesion. These benefits enable navigation across uneven terrain and natural obstacles such as rugged rocks, on the other hand, consistency and stability [3].

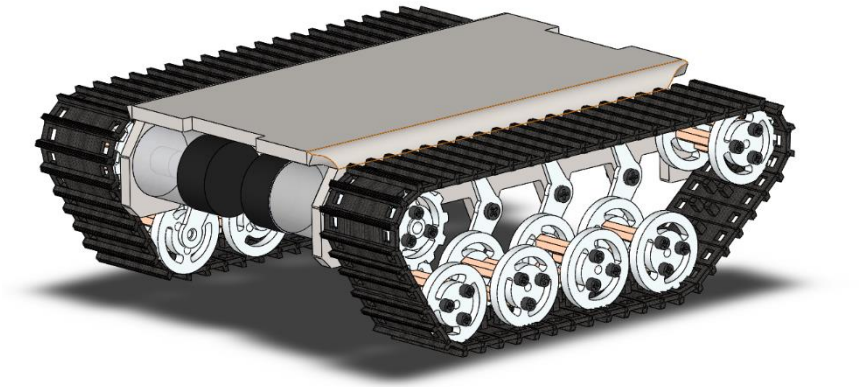


Figure 1.2 - Example of Crawler robots

1.2.3 Car-type robots

A robot that resembles a vehicle has four wheels placed at each of the chassis corners. The robot's traction is provided by two fixed wheels positioned on the same axis, and its steering is accomplished by the two steerable wheels situated on the same axis.

Within the family of wheeled robots, the car-type robot is regarded as stable due to its construction and four supports. These robots are mostly utilized outside as autonomous vehicles that are still in the research and development stage. It is important to remember that controlling a car-type robot is challenging because of the nature of propulsion. In fact, it cannot be turned or moved perpendicular to the fixed wheels [3].

Figure 1.3 - Example of Car-type robots

1.2.5 Differentially Driven WMRs

The most popular arrangement for wheeled mobile robots is differential drive. Its simplicity and adaptability make it useful. It is the simplest to use and manage. One or two castor wheels and two driving wheels make up a differentially driven Wheeled Mobile Robot (WMR). Figure 1.4 shows two popular differentially driven WMR.

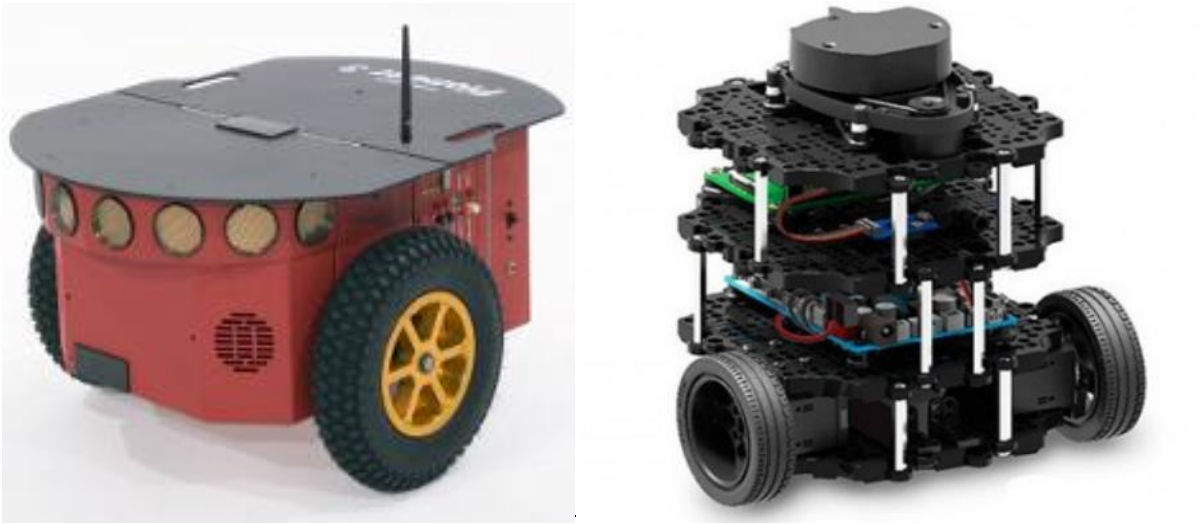


Figure 1.4 - P3-DX mobile robot (in the left) and Turtlebot3 robot (in the right)

The necessary motion in a differentially driven WMR is produced by the relative motion of the two driving wheels with regard to one another. The structure is the only thing supported by the caster wheels (see figure 1.5).

A differentially driven WMR moves in a straightforward manner as shown in Figure 1.6. When the two driving wheels of the robot revolve at the same speed, straight-line mobility is achieved. The wheels must rotate in the opposite direction in order to provide motion in the other direction. To turn, we apply brakes to one wheel and turn the other. The robot revolves around the stationary wheel. We may achieve sharp turning by turning the wheels in opposite directions. It is possible to move along an arc by using the differential motion of both wheels in relation to one another [4].

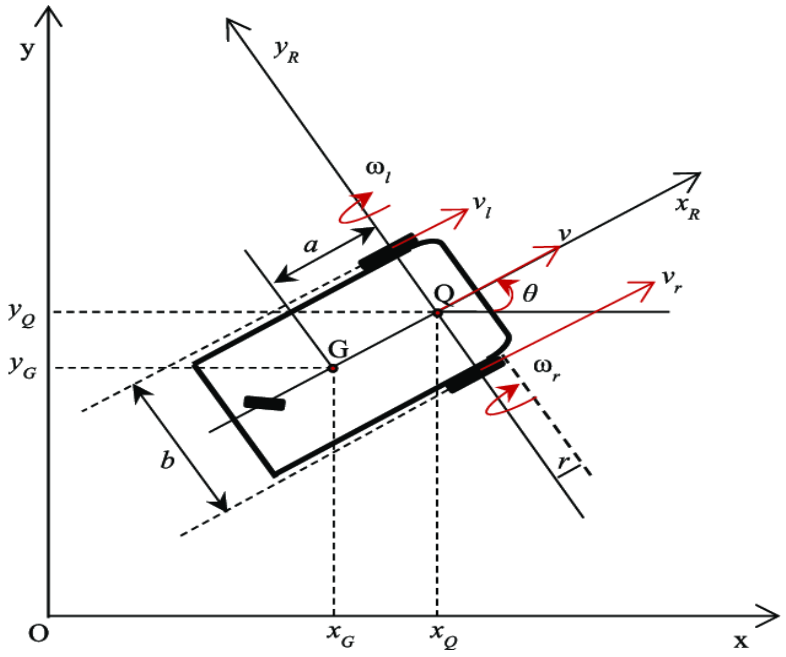


Figure 1.5 - Wheel differential drive mobile robot

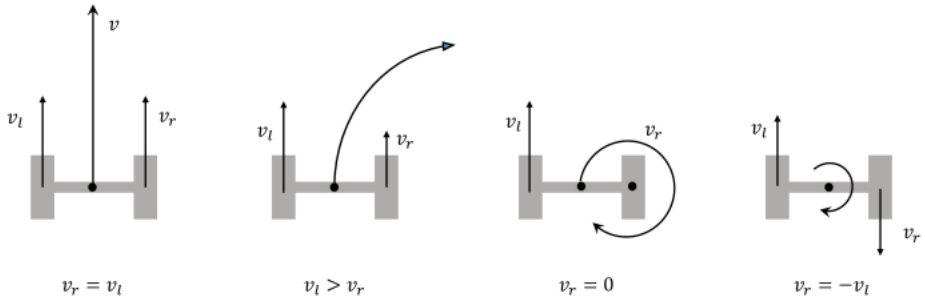


Figure 1.6: The different moving possibilities for unicycle mobile robot

1.3 Robot Kinematic model

Without considering the forces affecting the robot, kinematic modeling focuses on the geometric connections that govern its mobility. For instance, it describes the mobile robot system's location and speed progression without mentioning the robot's mass or coupling.

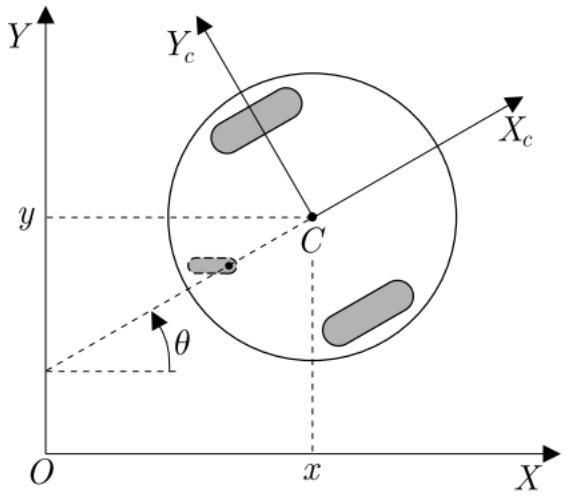


Figure 1.7 – Coordinate system of the WMR

It is assumed that the vehicle moves on a plane without slipping, i.e., there is a pure rolling contact between the wheels and the ground. The kinematic model of the WMR is then given by [11] :

$$\begin{aligned} \dot{x} &= v \cdot \cos\theta \\ \dot{y} &= v \cdot \sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{1.1}$$

The following is an alternative matrix representation of a differential drive robot model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1.2}$$

with $v = \frac{(v_r + v_l)}{2}$; $\omega = \frac{(v_r - v_l)}{b}$

$X = [x \ y \ \theta]^T$ is the state vector and $u = [v \quad \omega]^T$ is the control signal vector with v as the linear velocity and ω as the angular velocity. This model is called unicycle model, which represents the differential drive mobile robots and used in this research work. The model is sufficient to describe the nonholonomic constraints of this class of robots [9].

Considering a sampling period T and a sampling instant k, we obtain a discrete-time representation of mobile motion as follow:

$$\begin{cases} x(k+1) = x(k) + v(k)\cos\theta(k)T \\ y(k+1) = y(k) + v(k)\sin\theta(k)T \\ \theta(k+1) = \theta(k) + \omega(k)T \end{cases} \tag{1.3}$$

1.4 Posture Error

We are interested in calculating the error between the robot's location and the target position in a trajectory tracking issue for a mobile robot, as illustrated in figure (1.8), using the basic reference system.

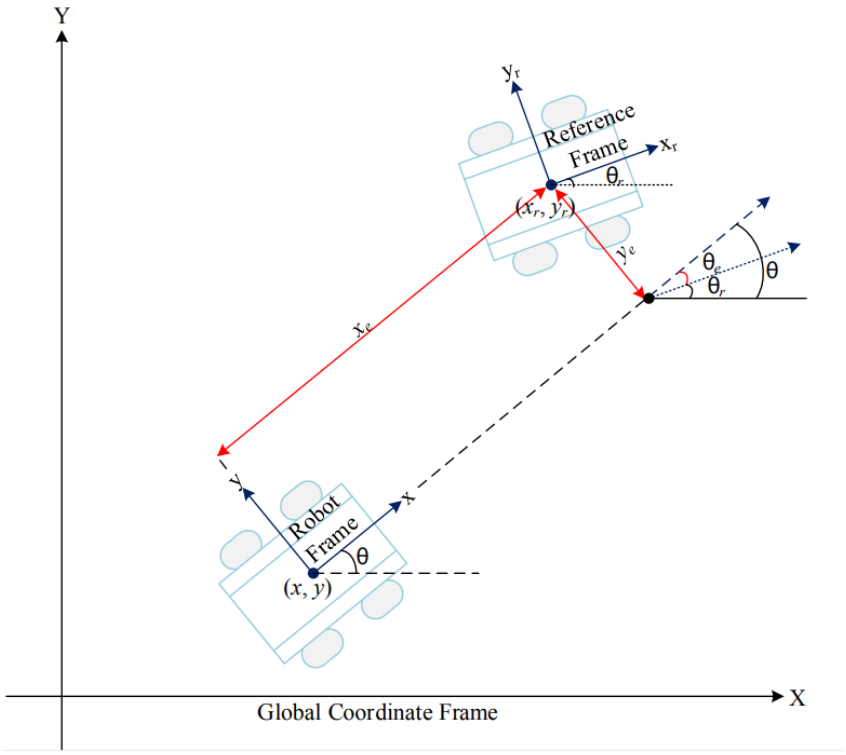


Figure 1.8 - The mobile robot tracking error

As illustrated in figure 1.8, a reference robot is considered. It is defined with the reference state vector $X_r = [x_r \ y_r \ \theta_r]^T$ and reference control vector $u_r = [v_r \quad \omega_r]^T$. The reference robot has the same model as (1.2). So, its kinematic model can be expressed as :

$$\dot{\mathbf{x}}_r = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{bmatrix} = \begin{bmatrix} v_r \cos\theta_r \\ v_r \sin\theta_r \\ \omega_r \end{bmatrix} = \begin{bmatrix} \cos\theta_r & 0 \\ \sin\theta_r & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} \tag{1.4}$$

The posture error is computed using the above graphic as a guide:

$$qr - q = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{1.5}$$

After implementing a coordinate change, we obtain:

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{1.6}$$

$$x_e = e_x\cos(\theta) + e_y\sin(\theta)$$
$$y_e = e_y\cos(\theta) - e_x\sin(\theta)$$
$$\theta_e = e_\theta$$

After derivation of the system of equations (1,6) and using equation (1,4) as well as the constraint of non-holonomy $\dot{x}\sin\theta_r = \dot{y}\cos\theta_r$, we obtain the dynamics of the error as follows:

$$\begin{bmatrix}\dot{x}_r\\ \dot{y}_r\\ \dot{\theta}_r\end{bmatrix} = \begin{bmatrix}\omega y_e - v + v_r\cos\theta_e\\ -\omega x_e + v_r\sin\theta_e\\ \omega_r - \omega\end{bmatrix} \quad \text{(I.7)}$$

1.5 Model Predictive Control

1.5.1. Principle of predictive control

The main idea of predictive control is based on [5]:

- The use of a model of the system to be controlled to predict its output over a certain horizon.
- The elaboration of a sequence of future commands minimizing a cost function.
- The application of the first element of the previous optimal sequence on the system and the repetition of the complete procedure at the next sampling period.

As shown in figure 1.9, MPC predicts the control inputs over the prediction horizon, p+1 in a way that the predicted output will merge with the reference trajectory. Then, it uses the predicted control input at time k only for actuation.

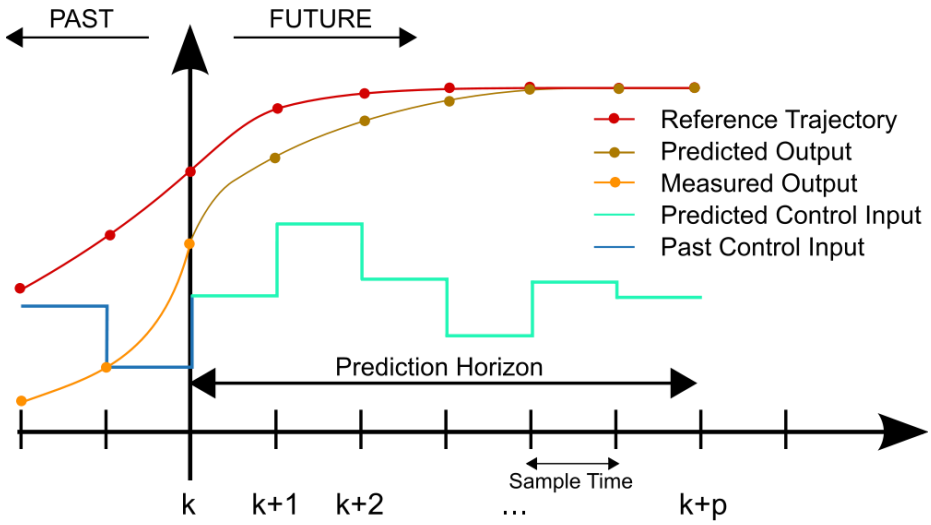


Figure 1.9 - Principle of model predictive control MPC [10]

1.5.2. Problem formulation

Unlike PID, Pure Pursuit or other Controllers, MPC is a predictive control method that predicts the future states of the vehicle and plans its control actions accordingly, bringing it closer to its desired trajectory.

MPC algorithm can handle non-linear and complex vehicle dynamics like tire force models and actuator models, allowing precise and accurate trajectory tracking.

Another powerful feature of MPC is its ability to take multiple constraints such as limiting jerks for comfortability, avoiding actuator saturation and setting the maximum speed. MPC can do all this by introducing constraints into the controller [10].

The control Block Diagram of MPC is given in figure 1.10. Using its two major blocks, optimizer and predictor, MPC computes control actions by minimizing a cost function based on the prediction of the vehicle states from a vehicle model over horizon, while considering the constraints.

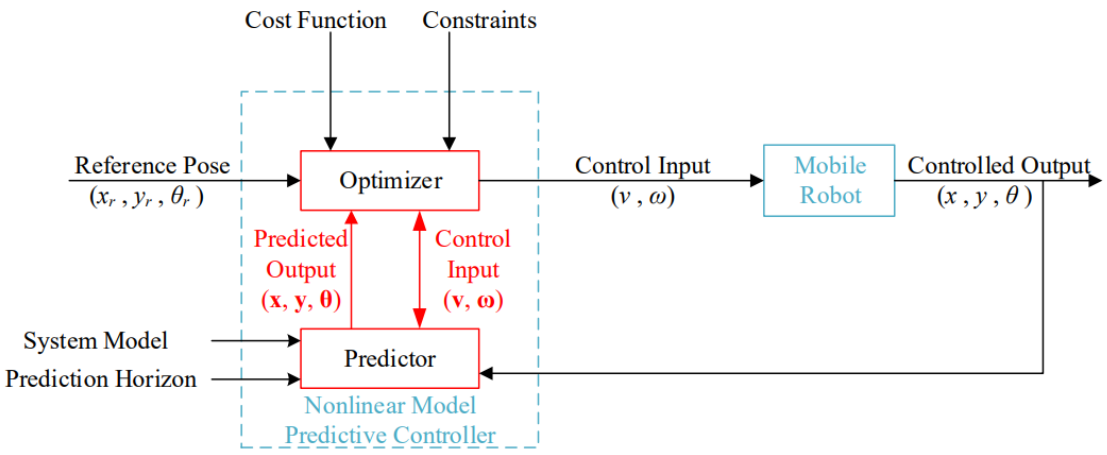


Figure 1.10: Control Block Diagram [9]

We can see in figure 1.9 how the MPC method may look over few time samples.

In our work a differential driven mobile robot with non-deforming wheels and a stiff body is taken into consideration. It is considered that the WMR drives on a plane with pure rolling contact between the wheels and the ground, meaning that it does not slide. We use the WMR's kinematic model provided in eq. (1.1) or, in a more compact form as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{1.8}$$

where $\mathbf{x} = [x \quad y \quad \theta]^{\boldsymbol{T}}$ describes the configuration (position and orientation) of the center of the axis of the wheels with respect to a global inertial frame $\{O, X, Y\}$. $\mathbf{u} = [v \ \omega]^{\boldsymbol{T}}$ is the control input, where v and ω are the linear and the angular velocities, respectively.

In MPC, a prediction model is employed, and the control law is computed in discrete time. So the discrete-time representation (1.3) of this model will be used. Its compact form is given by:

$$\mathbf{x}(k+1) = f_d(\mathbf{x}(k), \mathbf{u}(k)) \tag{1.9}$$

$\mathrm{x} = [\mathrm{x} \quad \mathrm{y} \quad \theta]^{\mathrm{T}}$

The problem of trajectory tracking consists of finding a control law such that

$$\mathbf{x}(k) - \mathbf{x}_r(k) = 0$$

where x_r is a predetermined, known reference trajectory. As explained in section 1.4, we will consider a virtual reference robot which has the same model as the robot to be controlled (eq. 1.4). Consequently, we have the compact representation:

$$\dot{\mathbf{x}}_r = f(\mathbf{x}_r, \mathbf{u}_r), \tag{1.10.a}$$

or, in discrete-time,

$$\mathbf{x}_r(k+1) = f_d(\mathbf{x}_r(k), \mathbf{u}_r(k))$$

1.5.3. Nonlinear Model Predictive Approach (NMPC)

This section presents the NMPC approach used to address the trajectory tracking problem.

The predicted robot motion derived from (1.9) [8]:

$$\mathbf{x}(k+j+1 \mid k) = f_d(\mathbf{x}(k+j \mid k), \mathbf{u}(k+j \mid k))$$

where $j \in [0, P-1]$ and the notation $a(m|n)$ indicates the value of a at the instant m predicted at instant n. By defining error vectors $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_r$ and $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_r$, the following objective function will be minimized:

$$\begin{aligned}\Phi(k) = \sum_{j=1}^{P} & \tilde{\mathbf{x}}^T(k+j \mid k)\mathbf{Q}\tilde{\mathbf{x}}(k+j \mid k) \\ & + \tilde{\mathbf{u}}^T(k+j-1 \mid k)\mathbf{R}\tilde{\mathbf{u}}(k+j-1 \mid k)\end{aligned} \tag{1.10.b}$$

where $\mathbf{Q} \geq 0$, $\mathbf{R} > 0$ are the weighting matrices for the error in the state and control variables, and the value of $\boldsymbol{P}$ is the prediction horizon.

We also take into account the possibility of limitations in the control variables' amplitudes:

$$\mathbf{u}_{min} \leq \mathbf{u}(k+j \mid k) \leq \mathbf{u}_{max} \tag{1.11}$$

where $\mathbf{u}_{max}$ denotes the upper bound and $\mathbf{u}_{min}$ denotes the lower bound. We can also express (1.11) as:

$$\mathbf{D}\mathbf{u}(k+j \mid k) \leq \mathbf{d}$$

with:

$$\mathbf{D} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} \mathbf{u}_{max} \\ -\mathbf{u}_{min} \end{bmatrix}$$

Also, state limitations can be similarly stated by: $\mathbf{Cx}(k+i \mid k) < \mathbf{c}$

Thus, the problem of nonlinear optimization can be expressed as follows:

$$\mathbf{x}^{\star}, \mathbf{u}^{\star} = \arg \min_{\mathbf{x},\mathbf{u}} \{\Phi(k)\} \tag{1.12}$$

with:

$$\mathbf{x}(k \mid k) = \mathbf{x}_0 \tag{1.13}$$

$$\mathbf{x}(k+j+1 \mid k) = f_d(\mathbf{x}(k+j \mid k), \mathbf{u}(k+j \mid k)), \tag{1.14}$$

$$\mathbf{Du}(k+j \mid k) \leq \mathbf{d} \tag{1.15}$$

where $j \in [0, P-1]$. $\mathbf{x}_0$ corresponds to the value of the states measured at the current instant k. Eq. (1.14) represents the discrete representation of the prediction model (1.3) and (1.15) is the control constraint. The decision variables are both state and control variables[8] The optimization problem (1.12)–(1.15) must be solved at each sampling time k. The result is a sequence of optimal states $\{\mathbf{x}^{\star}(k+1|k), \ldots, \mathbf{x}^{\star}(k+N|k)\}$ and optimal control inputs $\{\mathbf{u}^{\star}(k|k), \ldots, \mathbf{u}^{\star}(k+N-1|k)\}$. As we have said before, the MPC control law consists of applying only the first control action, $\mathbf{u}^{\star}(k|k)$, to the mobile robot.

1.5.4. Linear MPC Approach

The biggest hurdle in applying NMPC control is that the optimization problem is nonconvex, and is generally impossible to discover a global minimum [6]. Also, the computation time may be excessive when constraints are present and does not always allow for real time control. So, in order to decrease the computational load, the basic idea is to describe the system linearly and time-varyingly by applying a consecutive linearization strategy. Subsequently, the optimization issue to be addressed at each sample time can be transformed into a Quadratic Programming QP problem by taking the control inputs as the decision variables [7].

To do so, a linear model of the mobile robot dynamics can be obtained by computing an error model with respect to a reference robot [8]. The Taylor series expansion of (1.1) limited to the first order term around the point $(\mathbf{x}_r, \mathbf{u}_r)$ is given by:

$$\dot{\mathbf{x}} = f(\mathbf{x}_r, \mathbf{u}_r) \quad + \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\Big|_{\substack{\mathbf{x}=\mathbf{x}_r \\ \mathbf{u}=\mathbf{u}_r}} (\mathbf{x} - \mathbf{x}_r) + \\ + \frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\Big|_{\substack{\mathbf{x}=\mathbf{x}_r \\ \mathbf{u}=\mathbf{u}_r}} (\mathbf{u} - \mathbf{u}_r), \tag{1.16}$$

Or,

$$\dot{\mathbf{x}} = f(\mathbf{x}_r, \mathbf{u}_r) + f_{\mathbf{x},r}(\mathbf{x} - \mathbf{x}_r) + f_{\mathbf{u},r}(\mathbf{u} - \mathbf{u}_r), \tag{1.17}$$

where $f_{\mathbf{x},r}$ and $f_{\mathbf{u},r}$ are the jacobians of f with regard to $\mathbf{x}$ and $\mathbf{u}$, respectively. They are evaluated around the reference point $(\mathbf{x}_r, \mathbf{u}_r)$.

After that, subtracting (1.10) from (1.17) yields:

$$\dot{\tilde{\mathbf{x}}} = f_{\mathbf{x},r}\tilde{\mathbf{x}} + f_{\mathbf{u},r}\tilde{\mathbf{u}} \tag{1.18}$$

where $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_r$ is the error of the robot pose with respect to the reference robot one and $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_r$ is the corresponding error control input.

The discrete-time model of (1.18) is given by [8]:

$$\tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}(k)\tilde{\mathbf{u}}(k). \tag{1.19}$$

with

$$\mathbf{A}(k) = \begin{bmatrix} 1 & 0 & -v_r(k)\sin\theta_r(k)\,T \\ 0 & 1 & v_r(k)\cos\theta_r(k)\,T \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B}(k) = \begin{bmatrix} \cos\theta_r(k)\,T & 0 \\ \sin\theta_r(k)\,T & 0 \\ 0 & T \end{bmatrix}$$

where T is the sampling period and k the sampling time.

By introducing the two following vectors:

$$\bar{\mathbf{x}}(k+1) = \begin{bmatrix} \tilde{\mathbf{x}}(k+1 \mid k) \\ \tilde{\mathbf{x}}(k+2 \mid k) \\ \vdots \\ \tilde{\mathbf{x}}(k+P \mid k) \end{bmatrix}, \quad \bar{\mathbf{u}}(k) = \begin{bmatrix} \tilde{\mathbf{u}}(k \mid k) \\ \tilde{\mathbf{u}}(k+1 \mid k) \\ \vdots \\ \tilde{\mathbf{u}}(k+P-1 \mid k) \end{bmatrix},$$

the cost function (1.10.a) can be rewritten as:

$$\Phi(k) = \bar{\mathbf{x}}^T(k+1)\bar{\boldsymbol{Q}}\bar{\mathbf{x}}(k+1) + \bar{\mathbf{u}}^T(k)\bar{\boldsymbol{R}}\bar{\mathbf{u}}(k) \tag{1.20}$$

with

$$\bar{\mathrm{Q}} \triangleq \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q} \end{bmatrix} \quad \bar{\mathrm{R}} \triangleq \begin{bmatrix} \mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{R} \end{bmatrix}$$

This allows the optimization issue to be reformulated in the standard quadratic programming form.

From (1.19), we can demonstrate that $\bar{\mathbf{x}}(k+1)$ could be written as :

$$\bar{\mathbf{x}}(k+1) = \bar{\boldsymbol{A}}(k)\tilde{\mathbf{x}}(k \mid k) + \bar{\boldsymbol{B}}(k)\bar{\mathbf{u}}(k), \tag{1.21}$$

where

$$\bar{\boldsymbol{A}}(k) = \begin{bmatrix} \mathbf{A}(k \mid k) \\ \mathbf{A}(k+1 \mid k)\mathbf{A}(k \mid k) \\ \vdots \\ \alpha(k,2,0) \\ \alpha(k,1,0) \end{bmatrix},$$

$$\bar{\boldsymbol{B}}(k) = \begin{bmatrix} \mathbf{B}(k \mid k) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{A}(k+1 \mid k)\mathbf{B}(k \mid k) & \mathbf{B}(k+1 \mid k) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha(k,2,1)\mathbf{B}(k \mid k) & \alpha(k,2,2)\mathbf{B}(k+1 \mid k) & \cdots & \mathbf{0} \\ \alpha(k,1,1)\mathbf{B}(k \mid k) & \alpha(k,1,2)\mathbf{B}(k+1 \mid k) & \cdots & \mathbf{B}(k+P-1 \mid k) \end{bmatrix}$$

and $$\alpha(k,j,l) = \prod_{i=P-j}^{l} \mathbf{A}\,(k+i \mid k)$$

From (1.20), (1.21) and after some algebraic manipulations, we can rewrite the objective function in a standard quadratic form [8]:

$$\bar{\Phi}(k) = \frac{1}{2}\bar{\mathbf{u}}^T(k)\mathbf{H}(k)\bar{\mathbf{u}}(k) + \mathbf{f}^T(k)\bar{\mathbf{u}}(k) + \mathbf{d}(k) \quad (1.22)$$

With

$$\begin{aligned} \mathbf{H}(k) &= 2(\bar{\mathbf{B}}(k)^T\bar{\mathbf{Q}}\bar{\mathbf{B}}(k) + \bar{\mathbf{R}}) \\ \mathbf{f}(k) &= 2\bar{\mathbf{B}}^T(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k \mid k) \\ \mathbf{d}(k) &= \tilde{\mathbf{x}}^T(k \mid k)\bar{\mathbf{A}}^T(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k \mid k) \end{aligned}$$

A Hessian matrix is represented by the matrix $\boldsymbol{H}(\boldsymbol{k})$, which is always positive definite. It characterizes the objective function's quadratic component, while vector $\mathbf{f}$ characterizes its linear component. $\mathbf{d}$ is unrelated to $\tilde{\mathbf{u}}$ and has no bearing on how $\mathbf{u}\star$ is determined. Consequently, we define:

$$\bar{\Phi}'(k) = \frac{1}{2}\boldsymbol{u}^T(k)\mathbf{H}(k)\mathbf{t}(k) + \mathbf{f}^T(k)\mathbf{t}(k) \quad (1.23)$$

The optimization issue that needs to be resolved for every sample period is expressed as follows.

$$\tilde{\mathbf{u}}^\star = \arg\min_{\tilde{\mathbf{u}}}\{\bar{\Phi}'(k)\} \quad (1.24)$$

with $$\mathbf{D}\tilde{\mathbf{u}}(k + j \mid k) \leq \mathbf{d}, j \in [0, P-1] \quad (1.25)$$

1.6. Conclusion

In this chapter, we have described different categories of mobile. The most crucial components and ratings of mobile robots have been emphasized specifically. Two Model Predictive Controllers are studied in chis chapter. First, a nonconvex optimization problem was produced using a nonlinear MPC, and then, a linear MPC was produced by reformulating the issue into a single quadratic programming problem.

Chapter2: Reinforcement Learning and Path Planning

2.1. Introduction

The field of motion planning is about finding a safe path from point A to point B while avoiding obstacles. This area, along with navigation, has become very important in recent years due to the growing need for smart automation in industries and homes.

Since the 1970s, many different methods for planning these paths have been developed, like geometric algorithms, grid-based algorithms, potential field algorithms, neural networks, genetic algorithms, and sampling-based algorithms. Each method has its own strengths and weaknesses when it comes to finding the best path efficiently in terms of space, time, and optimization.

In this chapter we are going to study and explain the principle of Q-Learning based path planning algorithms and how it does help our robot to find the optimal path.

2.2. Path planning methods for mobile robots

What is path planning?

Path planning is the most critical concern in mobile robot navigation. It involves determining a geometric path from the current location of the robot to a target location while avoiding obstacles. This path must be navigable by the mobile robot and optimal in terms of at least one variable to be considered suitable. Depending on the target distance, the optimal path could be the smoothest, shortest, or the path allowing the mobile robot to move at the highest speed. In essence, the optimal path is determined based on these characteristics. A common method of path planning involves discretizing the space and considering the center of each unit as a movement point. Each movement point either contains an obstacle that needs to be avoided or is obstacle-free and can be traversed.

2.2.1. Global Planning and Local Planning

a) Global Path Planning

The global planner is used in environments with prior knowledge. Generally, it is an iterative algorithm designed to plan the path of the mobile robot in a known and static environment, allowing the robot to reach the final position from its initial position. The first step involves modeling the environment, and the second step involves planning an optimal path that avoids any type of collision with static obstacles. Examples of planners include Dijkstra, A*, D*, Ariane's thread, and cellular decomposition. Methods based on a global approach have the advantage of generating an optimal path while consuming a considerable amount of computation time and memory space.

b) Local Path Planning

Local methods are characterized by local knowledge of the environment. At each step of movement, the algorithm determines whether a collision exists; if there is a collision, the path is locally modified. This type of method allows problems to be solved within a reasonable time frame. The principle is to determine the robot's movements by considering only a local representation of the environment and to anticipate planning as an optimization problem. Several applications are possible in local planning, such as:

- **Planners for obstacle avoidance**: Probabilistic methods are generally used for planning. Information from exteroceptive sensors is necessary to generate a path that allows the robot to avoid a collision with a static or dynamic obstacle. These planners are commonly used in environments without prior knowledge and dynamic environments.
- **Planners for generating feasible paths for mobile robots**: These planners take into account certain non-holonomic and kinematic constraints specific to the mobile robot. There are several ways to generate a path. Generation depends on the robot's own constraints as well as those imposed on it, such as the path to be performed by the mobile robot may require it to perform maneuvers. In the literature, several methods can be found, such as deforming holonomic paths to make them feasible for non-holonomic

robots, using Dubins paths, Reeds and Shepp paths, continuous curvature paths generation, or using guidance methods.

The advantage of methods using this approach lies in their efficiency in terms of computation time, allowing their use in real-time applications. However, they have the disadvantage of falling into local minima and often generating a non-optimal path [11].

2.2.2. Path Planning Algorithms

Path planning is a frequently addressed topic in many applications, thus there are numerous algorithms available for such tasks. In this section, we will present some of the most commonly used methods for planning a path, briefly explaining their principles, as well as their advantages and disadvantages.

2.2.2.1. Grid based methods

Grid-based algorithm divides the entire map into a number of grid units or specific areas which reduce the complexity and difficulty to find the optimal path. A sample of grid-based path planning algorithm is shown in Figure II.1

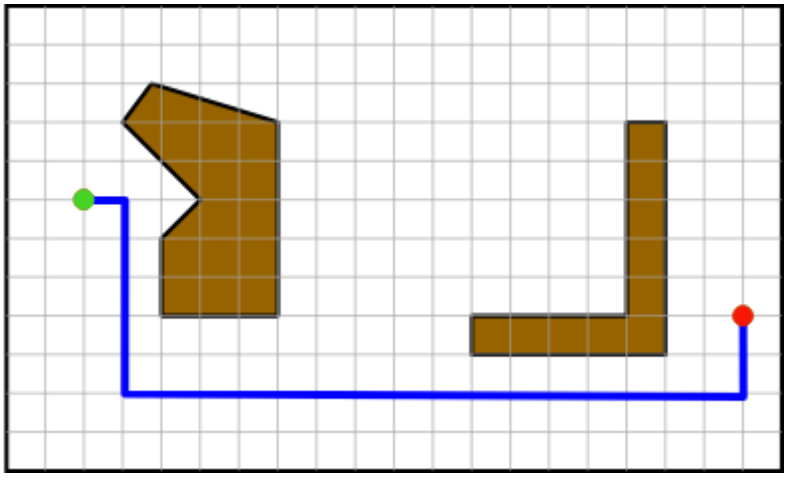


Figure 2.1: A sample of grid-based path planning method

Dijkstra's algorithm

Dijkstra's algorithm is a popular grid based algorithm for solving many single-source shortest path problems. The objective is to find the shortest distance between two vertices on a graph. It was conceived by Dutch computer scientist **Edsger W. Dijkstra** in 1956. The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited. The need for Dijkstra's algorithm arises in many applications where finding the shortest path between two points is crucial [12].

A* Algorithm

The A* Algorithm is also a popular graph traversal grid based path planning algorithm. A* operates similarly to Dijkstra's algorithm except that it guides its search towards the most promising states, potentially saving a significant amount of computation time . A* is the most widely used for approaching a near optimal solution with the available data-set/node. It is widely used in static environments; there are instances where this algorithm is used in dynamic environments. The base function can be tailored to a specific application or environment based on our needs. A* is similar to Dijkstra in that it works based on the lowest cost path tree from the initial point to the final target point. The base algorithm uses the least expensive path and expands it [13].

D* Algorithm

Path planning in partially known and dynamic environments in an efficient manner is increasingly critical, e.g., for automated vehicles. To solve this problem, the D* (or Dynamic A*) algorithm is used to generate a collision-free path amidst moving obstacles.

2.2.2.2. Artificial Potential Field (APF) based method

The domain of research and development concerning mobile robot obstacle avoidance continues to remain an active area of interest. Artificial potential fields (APF) are a common and effective method for obstacle avoidance path planning, where the robot is guided to the target location by a simulated environmental potential field. Traditional artificial potential field methods tend to trap robots in local minima, impeding their ability to reach the goal [14]. A sample of APF-based path planning algorithm is shown in Figure 2.2.

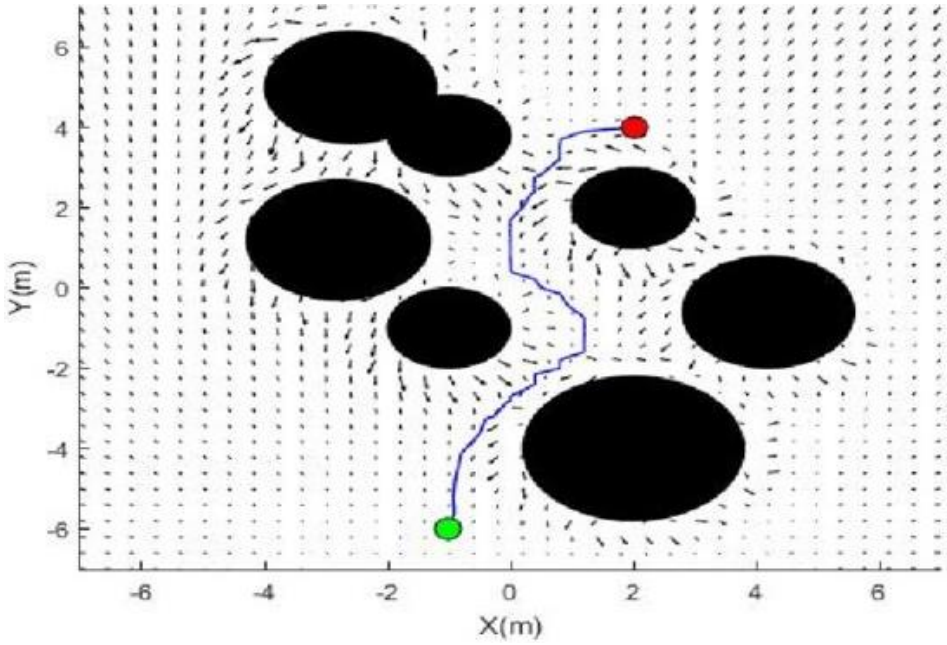


Figure 2.2: A sample of APF-based path planning method.

II.2.2.3. Probabilistic Methods

Probabilistic planners are part of a broader family of sampling-based methods. They are highly efficient, especially for problems defined in very high-dimensional configurations. One planning method called "Roadmap" is based on the use of graph theory. The principle is to construct a graph connecting the initial configuration to the final configuration while avoiding obstacles. These techniques have been among the most utilized and developed in recent years because they are global planners. In this section, we present the principle of two

main probabilistic approaches PRM (Probabilistic Roadmap) and RRT (Rapidly-exploring Random Tree).

- **RRT Method**

Initially proposed by Lavalle, RRT is one of the most popular methods in recent years. It involves building a tree starting from the initial position of the robot, which is the root node of the tree. The RRT algorithm gradually explores the configuration space to find a path to the target location. The principle of the RRT method is as follows:

The RRT planner develops the search tree rooted at the start state q_{start} following these steps:

1. The planner samples a random state q_{rand} from the state space.
2. The planner finds a state that is already in the search tree and is closest to q_{rand} in the existing tree.
3. Attempt to extend the tree from q_{near} towards q_{rand} by a certain length.
4. The new state q_{new} is added to the search tree.

This process is repeated until the tree reaches the q_{goal}. Each time a new node q_{new} is sampled, its connection with other nodes is checked for collisions to achieve a drivable path from q_{start} to q_{goal}. This principle is illustrated in Figure (2.3 (a)). Figure (2.3 (b)) shows the resulting path "R" among several candidate paths to the start position q_{start} and the goal position q_{goal}.

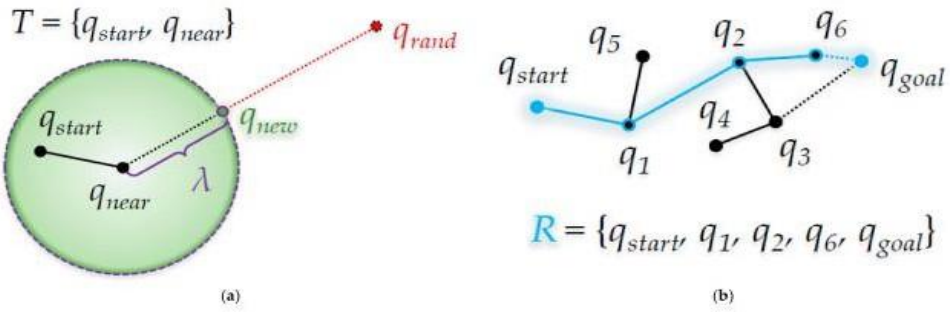


Figure 2.3: RRT Algorithm: (a) Illustration of the RRT extension principle ;
(b) after random sampling ends.

The Rapidly-exploring Random Tree (RRT) method consists of a phase of constructing a tree covering the free space (see Figure 2.4) and a query phase. The performance of this method stems from the fact that it does not require a pre-computation phase. An inherent and interesting property of this approach is that the growth of trees is strongly biased towards unexplored areas of the configuration space, enabling rapid exploration. Despite these advantages, RRT, like any approach, may have shortcomings; it does not take into account the solution cost during the search. Thus, the paths it produces are sometimes far from optimal.

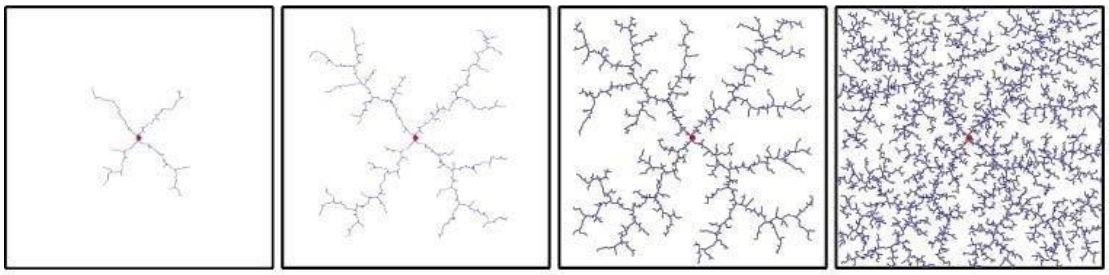


Figure 2.4: Evolution of a tree covering the free space by the RRT method

- **PRM Method**

Path planning by random sampling was first introduced by Kavraki under the name Probabilistic Roadmap (PRM).The principle of this method is to sample the configuration space C to find enough collision-free configurations to adequately represent the free configuration space. The algorithm determines whether the sampled configurations are in collision or not using a collision detector. Those in the free space are kept, and the algorithm attempts to connect them by an edge to the graph containing the other nodes. The connection is made using a local method to link one node to another along a free space. The graph initially consists of two nodes, the initial configuration and the final configuration. When we want to find a path between two nodes of the graph, it is traversed using an Astar-like algorithm, see Figure (2.5) where the starting configuration q_s and the arrival configuration q_b are indicated.

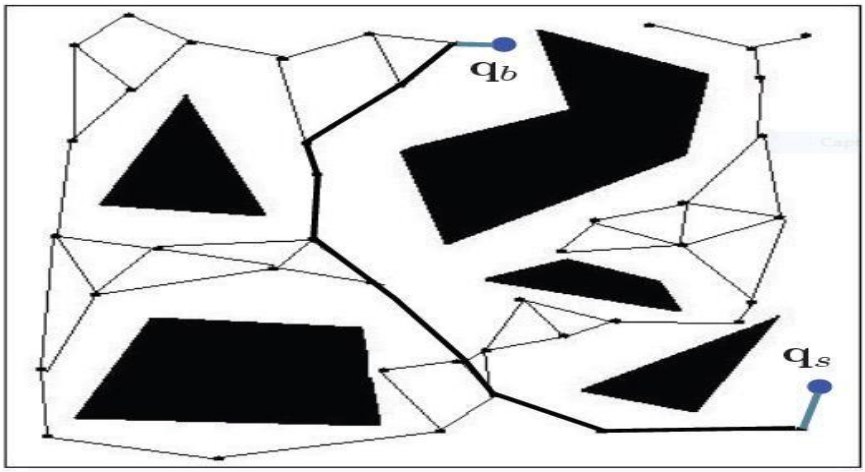


Figure 2.5: Path Planning by PRM.

The PRM method has the advantage of quickly finding a collision-free path. In high-dimensional spaces, the PRM method is efficient at rapidly finding a solution and it is straightforward to implement.

However, like all probabilistic motion planning methods, the representativeness of the free space graph C_{free} must be ensured before it is used for motion planning queries. This step is typically performed offline as generating a Roadmap is computationally expensive; the obtained environment graph is retained.

2.2.2.4. Path planning using Reinforcement Learning

Path planning using Reinforcement Learning (RL) is a technique where an agent learns to navigate through an environment to reach a goal while avoiding obstacles. RL is a type of machine learning where an agent interacts with an environment, taking actions and receiving feedback in the form of rewards or penalties.

In the context of path planning, the environment represents the space through which the agent moves, including obstacles and the goal location. The agent's objective is to learn a policy, a mapping from states (representing the agent's current situation) to actions (movements or decisions), that maximizes cumulative rewards over time.

The RL agent explores the environment, trying different actions, and learns from the feedback it receives. By repeatedly interacting with the environment, the agent improves its policy, ultimately finding optimal paths from a starting point to the goal while avoiding collisions. [11]

RL algorithms, such as Q-learning, Deep Q-Networks (DQN), or Proximal Policy Optimization (PPO), can be applied to path planning tasks. These algorithms use different approaches to learn the optimal policy, and they can adapt to various environments and scenarios.

Overall, RL-based path planning allows robots to learn to navigate autonomously in complex and dynamic environments, making it a powerful technique for robotics applications such as autonomous vehicles, drones, and mobile robots.

2.3 Reinforcement learning (RL)

What is Reinforcement Learning (RL)?

Reinforcement Learning is a part of machine learning. Here, agents are self-trained on reward and punishment mechanisms. It's about taking the best possible action or path to gain maximum rewards and minimum punishment through observations in a specific situation. It acts as a signal to positive and negative behaviors. Essentially an agent (or several) is built that can perceive and interpret the environment in which is placed, furthermore, it can take actions and interact with it. The figure 2.6 bellow illustrates the principle of RL.

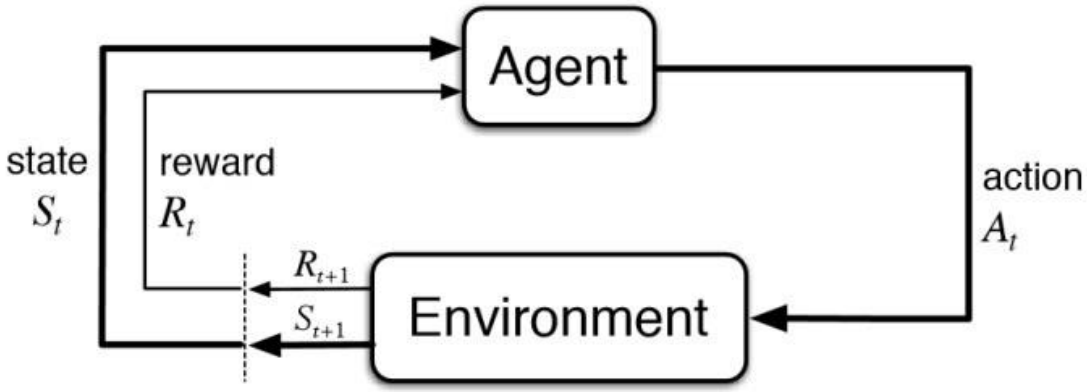


Figure 2.6: Basic Diagram of Reinforcement Learning

How Does Reinforcement Learning Work?

In RL, the agent follows the steps below:

1. Start in a state.
2. Take an action.
3. Receive a reward or penalty from the environment.
4. Observe the new state of the environment.
5. Update the policy to maximize future rewards.

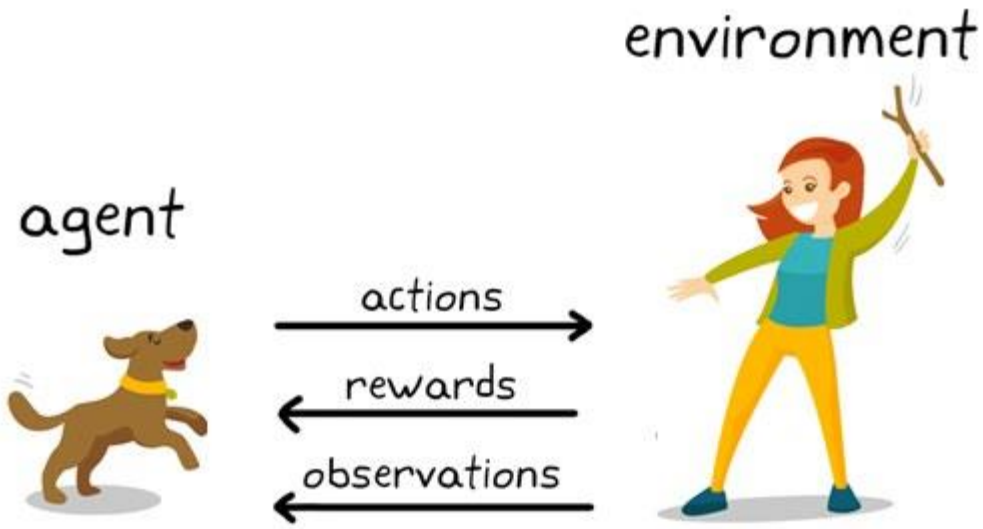


Figure 2.7: Reinforcement Learning Example

In the example of figure 2.7, we can see a dog and a master. Let's imagine the master is training her dog to get the stick. Each time the dog gets a stick successfully, she offered him a feast (a bone let's say). Eventually, the dog understands the pattern, that whenever the master throws a stick, it should get it as early as it can to gain a reward (a bone) from a master in a lesser time.

Terminologies used in Reinforcement Learning

Here are some terminologies used in RL:

Agent – is the sole decision-maker and learner.

Environment – a physical world where an agent learns and decides the actions to be performed.

Actions – a list of action which an agent can perform.

State – the current situation of the agent in the environment.

Reward – For each selected action by agent, the environment gives him a reward. It's usually a scalar value and nothing but feedback from the environment.

Policy – the agent prepares strategy (decision-making) to map situations to actions.

Value Function – The value of state shows up the reward achieved starting from the state until the policy is executed

Model – Every RL agent doesn't use a model of its environment. The agent's view maps state-action pairs probability distributions over the states

Reinforcement Learning Algorithms

There are 3 approaches to implement reinforcement learning algorithms

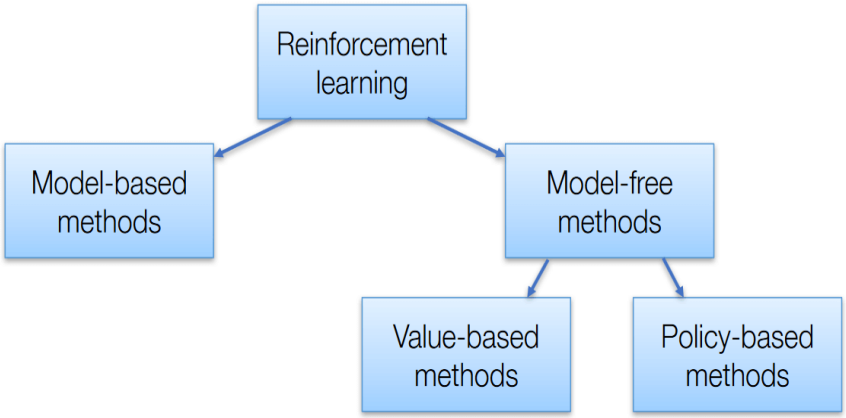


Figure 2.8: Reinforcement Learning Algorithms

Value-Based – The main goal of this method is to maximize a value function. The value-based method trains the value function to learn which state is more valuable and take action.

Policy-based – These methods train the policy directly to learn which action to take in a given state.

Model-Based – In this method, we need to create a virtual model for the agent to help him in learning to perform in each specific environment. The model-based algorithms use transition

and reward functions to estimate the optimal policy and create the model. In contrast, model-free algorithms learn the consequences of their actions through the experience without transition and reward function [15] .

2.4. Q-Learning method

What is Q-Learning?

Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The “Q” stands for quality. Quality represents how valuable the action is in maximizing future rewards.

Key Terminologies in Q-learning

Before we jump into how Q-learning works, we need to learn a few useful terminologies to understand Q-learning's fundamentals.

- **States (s)**: the current position of the agent in the environment.
- **Action (a)**: a step taken by the agent in a particular state.
- **Rewards**: for every action, the agent receives a reward and penalty.
- **Episodes**: the end of the stage, where agents can’t take new action. It happens when the agent has achieved the goal or failed.
- **Q** $_{(St+1,a)}$ **:** expected optimal Q-value of doing the action in a particular state.
- $Q_{(St,At)}$: it is the current estimation of $Q_{(St+1,a)}$.
- **Q-Table**: the agent maintains the Q-table of sets of states and actions.
- **Temporal Differences (TD)**: used to estimate the expected value of $Q_{(S_{t+1},a)}$ by using the current state and action and previous state and action. [16]

2.4.1. Q-Learning Algorithm

The **Q-learning algorithm** is given in in the Table II.1 below

Table 2.1: Q-learning Algorithm

Algorithm 1: Q-Learning

Input: positive integer num_episodes, small positive fraction α, γ and ε_i
Output: value function Q ($\approx q_\pi$ if num_episodes is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a)$= 0 for all $s \in S$ and $a \in A$(s), and Q(terminal-state, .) = 0)
for i ←1 to num_episodes do
  $\varepsilon \leftarrow \varepsilon_i$
  Observe S_0
  t←0
  repeat
    Choose action A_t using policy derived from Q (e.g., ε-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    t←t+1
  until S_t is terminal;
end
return

II.4.2. A Q-Learning example

To better understand Q-Learning, let's take a simple example: Let's say that a robot has to cross a maze and reach the end point. There are mines, and the robot can only move one tile at a time. If the robot steps onto a mine, the robot is dead. The robot has to reach the end point in the shortest time possible.

- We chose a Learning rate of 0.1.
- The discount rate (gamma) is 0.99

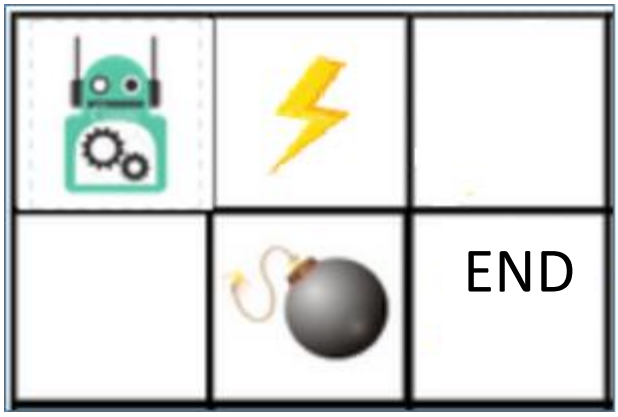


Figure 2.9: The robot environment

+0: Going to a state which is not the END or with no power in it.

+1: Going to a state with a power in it.

+10: Going to the END state.

-10: Going to the state with the mine and thus dying.

+0 If we take more than five steps

To train our agent to have an optimal policy (so a policy that goes right, right, down), we will use the Q-Learning algorithm. To do so, we follow the steps below:

Step 1: Initialize the Q-table

= 0 for all $\boldsymbol{s} \in S$ and $a \in A(s)$, and $Q(terminal - state) = 0\boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a})$

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
END	0	0	0	0

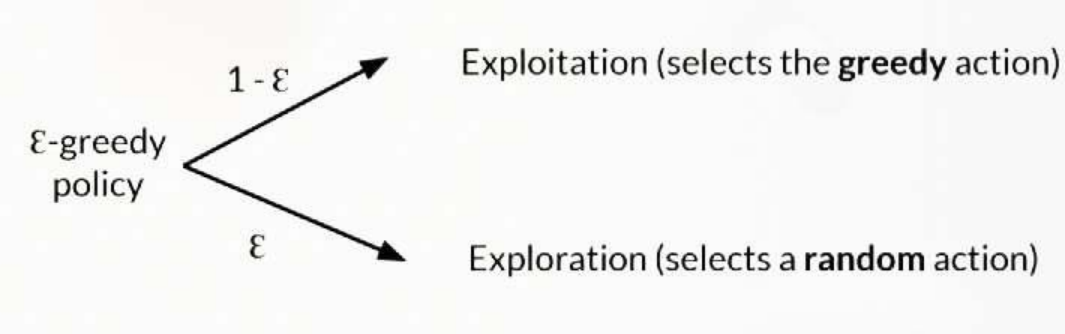
Figure 2.10: Initialize the Q-table

So, for now, our Q-table is useless; we need to train our Q-function using the Q-Learning algorithm.

We are going to do it here for 2 iterations (or episodes):

1st Training Episode

Step 2: Choose an action A_t using the epsilon-greedy strategy



The epsilon-greedy strategy is a policy that handles the exploration/exploitation trade-off. The idea is that, with an initial value of ε = 1.0:

- With probability 1 — ε : we do exploitation (aka our agent selects the action with the highest state-action pair value).
- With probability ε: we do exploration (trying random action).

At the beginning of the training, the probability of doing exploration will be huge since ε is very high, so most of the time, we'll explore. But as the training goes on, and consequently our Q-table gets better and better in its estimations, we progressively reduce the epsilon value since we will need less and less exploration and more exploitation.

In our case, because epsilon is big (= 1.0) so the robot has to do an Exploration; he takes a random action. In this case, he goes right.

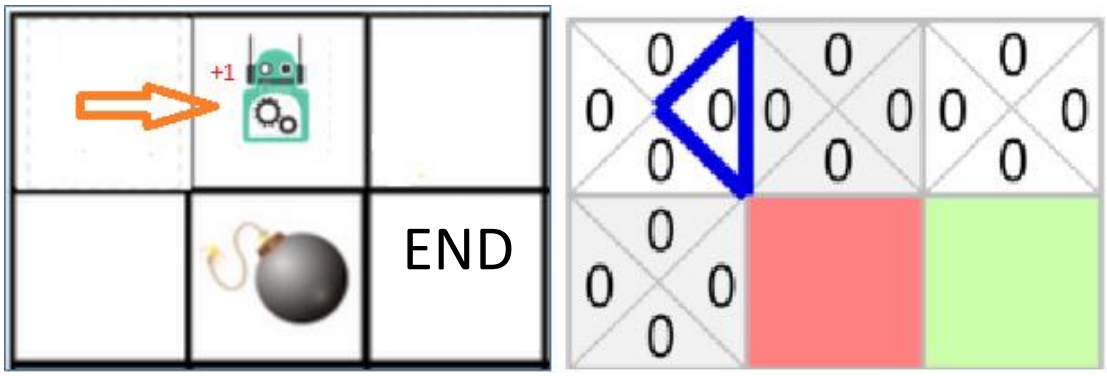


Figure 2.11: Random action taken by the robot

Step 3: Perform action A_t, get reward R_{t+1} and next state S_{t+1}

By going right, the robot gets a power, so $R_{t+1} = +1$ and he is in a new state as shown in the figure 2.11 above (Take action A_t and observe R_{t+1} and S_{t+1}).

Step 4: Update $Q_{(St,At)}$

We can now update $\boldsymbol{Q(S_t, A_t)}$ using the formula blow from the Algorithm:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation | Former Q-value estimation | Learning Rate | Immediate Reward | Discounted Estimate optimal Q-value of next state | Former Q-value estimation

$Q(Initial\ state, Right) = 0 + 0.1 * [1 + 0.99 * 0 - 0]$

$Q(Initial\ state, Right) = 0.1$

Hence, the Q-table becomes as follow:

	←	→	↑	↓
	0	0,1	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
END	0	0	0	0

Figure 2.12: the Q value updated

2nd Training Episode

Step 2: Choose an action using the Epsilon Greedy Strategy

We take a random action again, since epsilon = 0.99 is big (Notice we decay epsilon a little bit because, as the training progress, we want less and less exploration).

We took the action ‘down’. This is not a good action since it leads the robot to the mine.

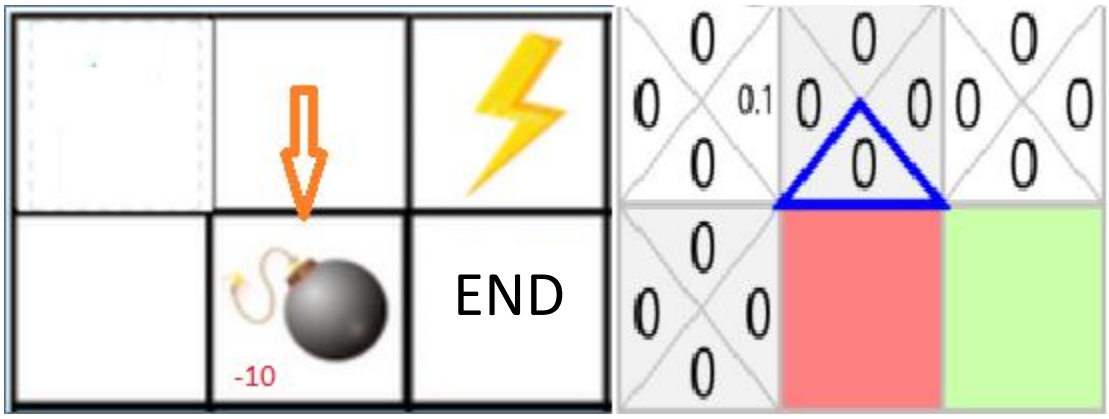


Figure 2.13: Random action (exploration)

Step 3: Perform action A_t get R_{t+1} and S_{t+1}

Because the robot ingested mine, he receives a reward of -10 denoted as $\boldsymbol{R_{t+1} = -10}$ and unfortunately, he perishes (Take action A_t and observe R_{t+1} and S_{t+1})

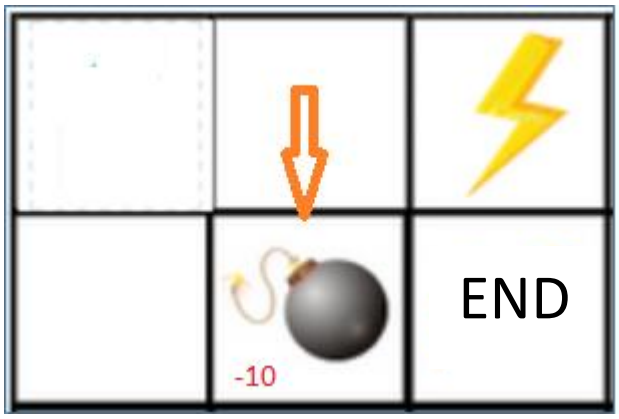


Figure 2.14: Action penalty taken by robot

Step 4: Update $Q_{(St,At)}$

$Q_{(St,At)} \leftarrow Q_{(St,At)} + \alpha(R_{t+1} + \gamma maxa\, Q_{(St+1,a)} - Q_{(St,At)})$

$Q(\,state\ 2, Down) = \ 0\ + 0.1 * [-10 + 0.99 * 0 - 0]$

$Q(\,state\ 2, Down) = \ -1$

The updated Q-Table is given below.

	←	→	↑	↓
	0	0,1	0	0
	0	0	0	-1
	0	0	0	0
	0	0	0	0
	0	0	0	0
END	0	0	0	0

Figure 2.15: the Q value updated after penalty

Because the robot is dead, we start a new episode. But what we see here is that, with only two explorations steps, our robot became smarter.

As we continue exploring and exploiting the environment and updating Q-values using the equation that update $Q_{(St,At)}$, the Q-table will give us a better and better approximation. At the end of the training, we'll get an estimate of the optimal Q-function.

2.5. Conclusion

In this chapter we learned about how Q-learning algorithm could be used in path planning problems in order to find optimal routes in various environments. It is effective for navigating through known or unknown spaces. Its efficiency is demonstrated in tasks like robotic navigation and game AI.

Chapter 3 : Results and Interpretations

3.1 Introduction

After modeling the mobile robot system and studying Model Predicive control methods, verification should be performed through simulation using Matlab and the ROS simulator Gazebo of the differentially driven wheeled mobile robot for different paths (circular, infinity, sinusoidal and the path planned using Q-Learning algorithm). For that, we present the obtained results by displaying and discussing the evolution curves of the robot's position, the evolution of x, y and θ coordinates over time, and the linear and angular speeds of the robot.

3.2 NMPC Simulation Results

In this section simulations were conducted to show the effectiveness of the NMPC controller in tracking problems with limited control signals. The results show how well the controller performs in the tracking issue. In the simulation, we choose a prediction horizon N=20, a sample $T_s = 0.1s$. The weighting matrices used are $R = diag(1,1)$ and $Q = diag(100,100,10)$. Constraints in the amplitude of the control variables are: V_{max}= 0.9m/s, $V_{min} = -0.9\ m/s$; $\omega_{max} = \frac{\pi}{2}\ rad/s$, $\omega_{min} = -\frac{\pi}{2} rad/s$ and their rates: $V_{ratemax}$ = 0.2m/s, $V_{ratemin} = -0.2m/s$, $\omega_{ratemax}$= $\frac{\pi}{4} rad/s$ and $\omega_{ratemin} = -\frac{\pi}{4} rad/s$.

Figure 3.1 illustrates the reference circular path to be followed by the robot, represented by a red line, while its actual path is represented by a blue solid line. Figure 3.2 shows the curves representing the evolution of reference locations and actual robot locations over time. While Figure 3.3 represents the linear and angular velocities of the robot over time.

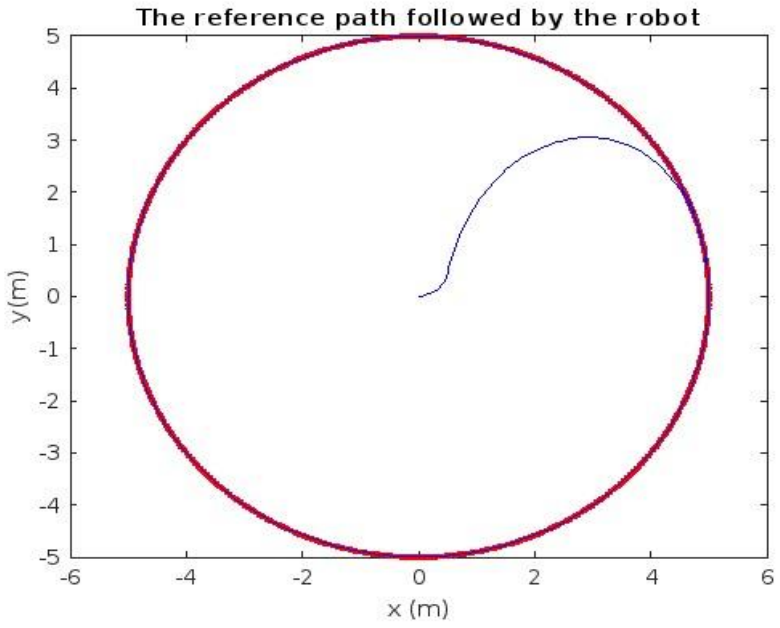


Figure 3.1: The circular trajectory of the robot in the XY plane..

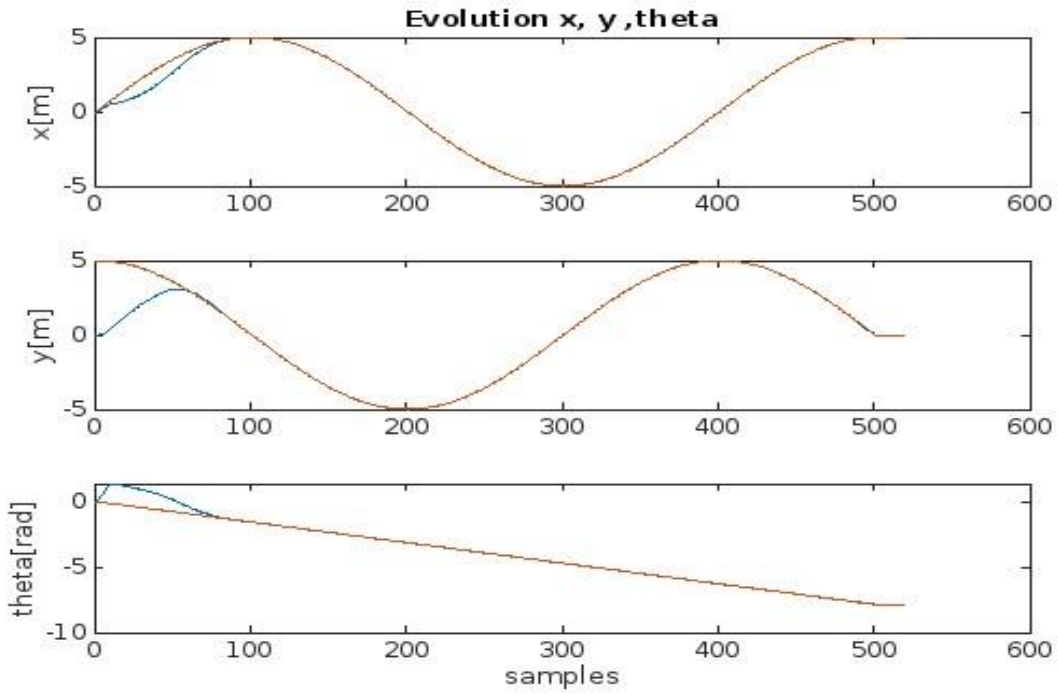


Figure 3.2: Evolution of the robot's components over time.

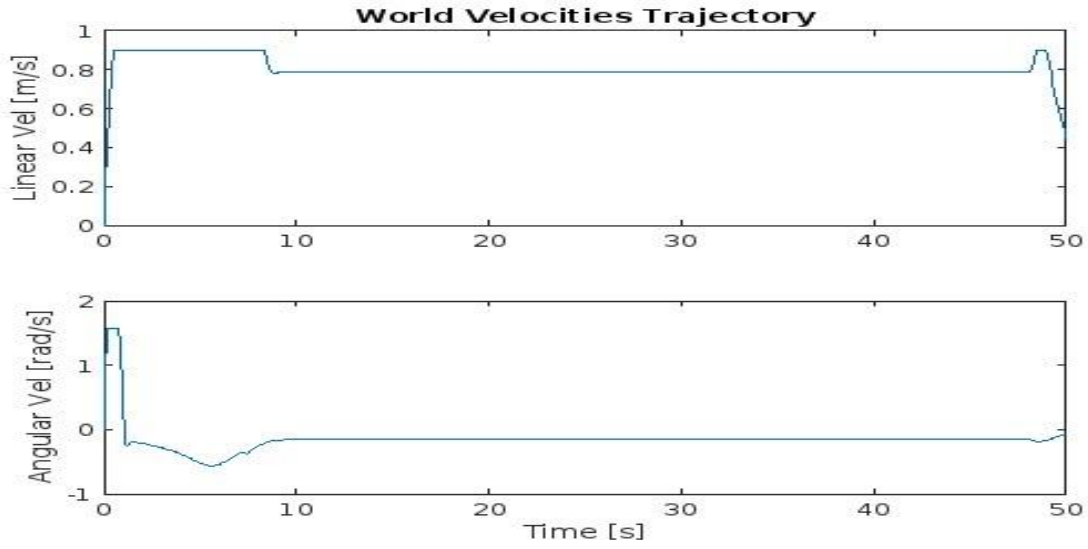


Figure 3.3: World Velocity Trajectory (v, ω).

From figure 3.1, It can be clearly seen that the actual trajectory of the robot converges to the reference. We can say the same thing about the curves of x and y coordinates and the orientation θ as shown in Figure 3.2. It must be noted that, even without error in the x state, the WMR must turn away from the reference trajectory due to the nonholonomic constraint. In Figure 3.3, it can be seen that the control inputs are inside the limits imposed by the constraints.

Figure 3.4 illustrates the lemniscate reference path to be followed by the robot, represented by a red dashed line, while its actual path is represented by a blue solid line. Figure 3.5 shows the curve which represents the evolution of reference locations and actual robot locations over time. While Figure 3.6 represents the linear and angular velocities of the robot.

It can be noted in Figure 3.4 that the problem is successfully solved, but with low convergence rate. Figure 3.5 shows that x and y position coordinates of the robot follow perfectly the reference coordinates but this is different for the orientation θ because its corresponding weight in the Q matrix is 50 which is lower than those of x and y coordinates.

Figure 3.6 shows that the generated control signals respect the imposed constraints.

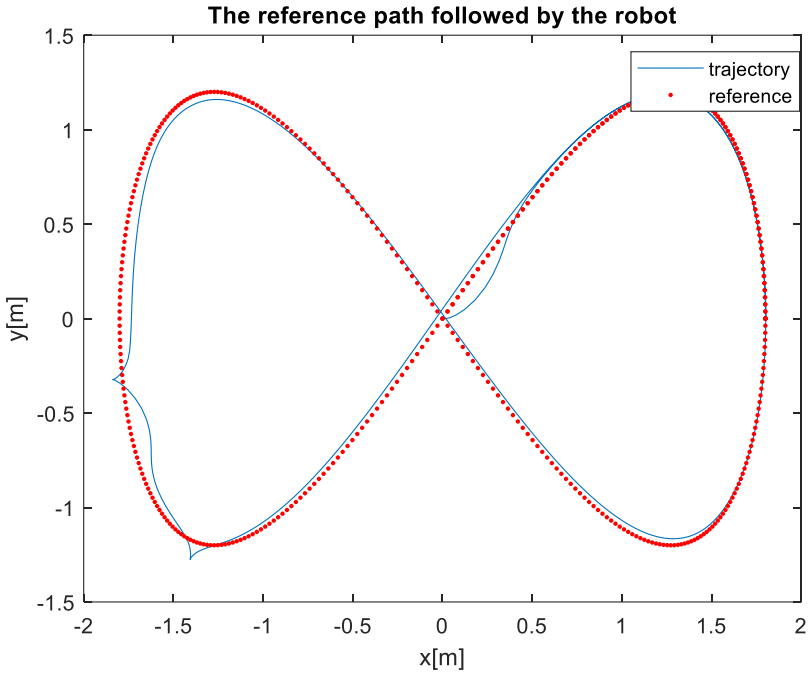


Figure 3.4: The lemnscate trajectory of the robot in the XY plane.

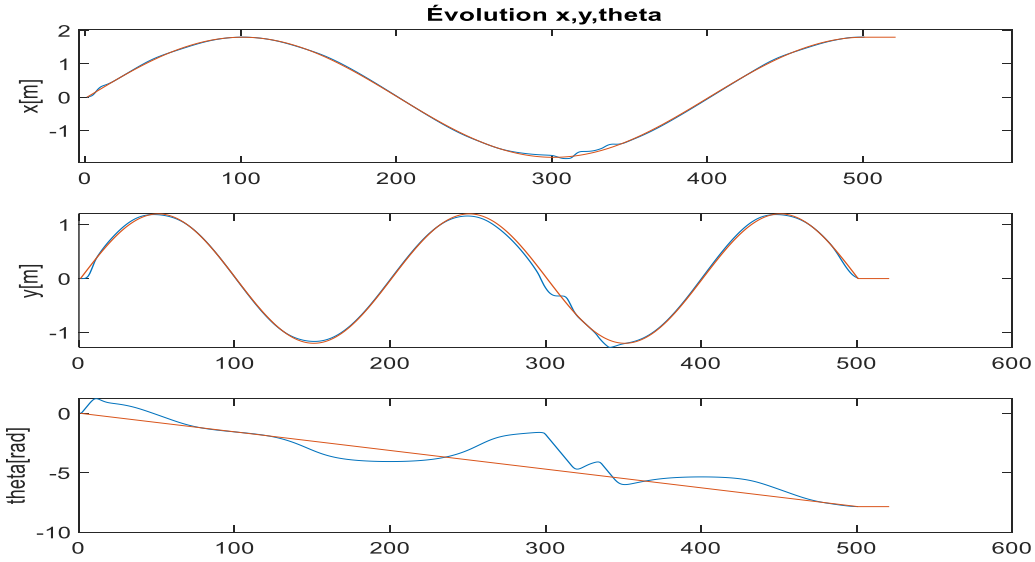


Figure 3.5: Evolution of the robot components over time in case of lemniscate reference trajectory.

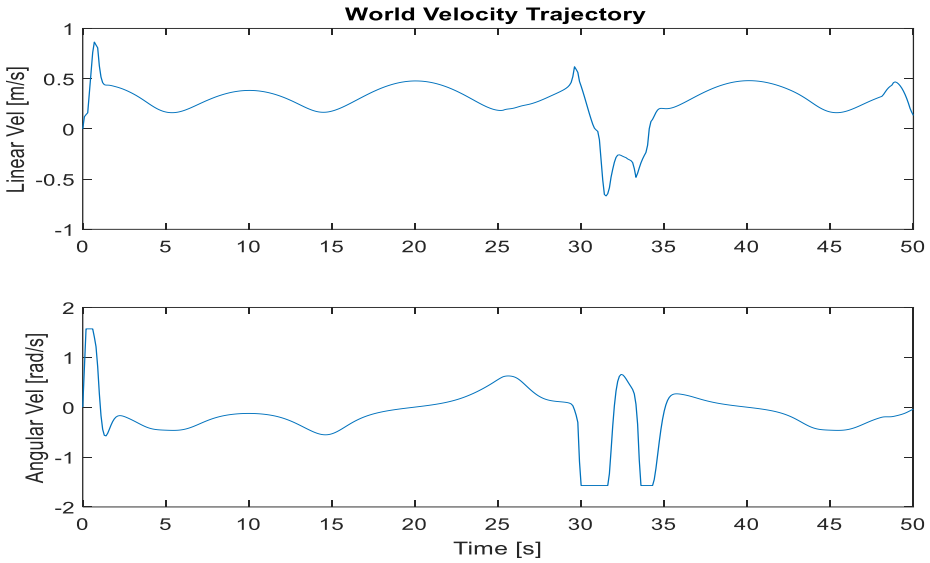


Figure 3.6: World Velocities of the robot (v, ω).

3.3 Linear MPC Simulation Results

In this section, simulation results are shown for the following parameters: The weighting matrices used are: The weighting matrices used are $R = diag(1,1)$ and $Q = diag(100,100,10)$. Constraints in the amplitude of the control variables are: V_{max}= 0.9m/s, $V_{min} = -0.9\ m/s$; $\omega_{max} = \frac{\pi}{2}\ rad/s, \omega_{min} = -\frac{\pi}{2} rad/s$. The prediction horizon is N=20.

As shown in Figure (3.7) for a sinusoidal reference path, it is clearly noticeable that the actual path of the robot converges to the reference path. The same thing can be said for the $\boldsymbol{x}$ and $\boldsymbol{y}$ coordinate curves and the angle $\boldsymbol{\theta}$, as illustrated in Figure (3.8), where there are slight deviations on the robot's actual path.

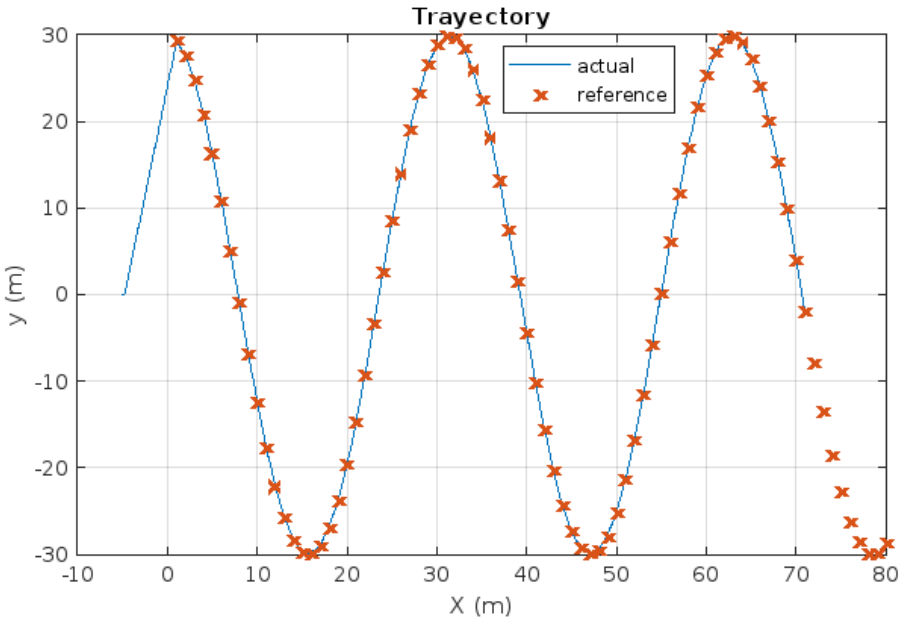


Figure 3.7: The sinusoidal trajectory of the robot in the XY plane.

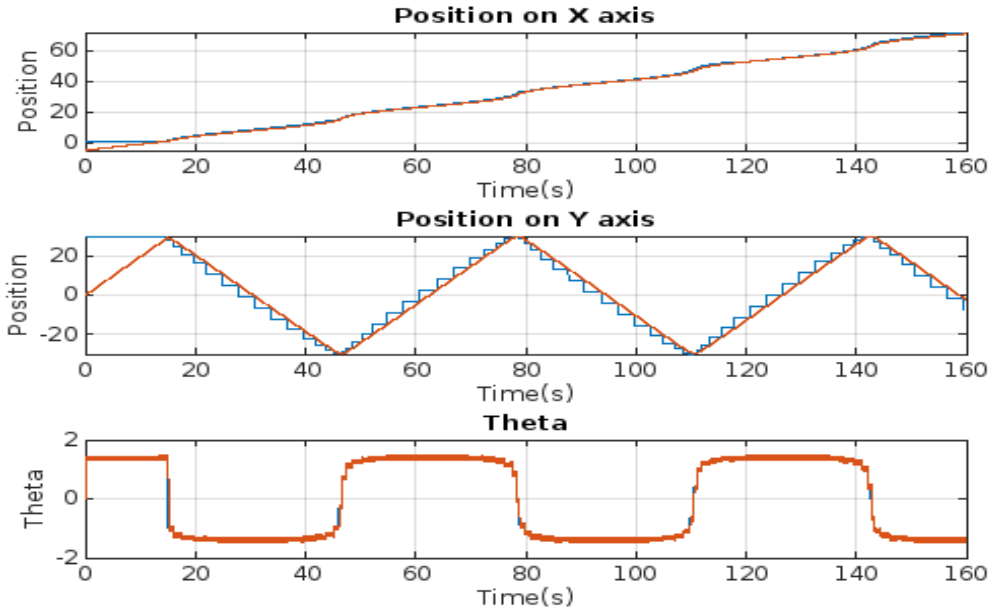


Figure 3.8: Evolution of the robot's components over time.

From Figure 3.9, it can be clearly observed that the circular path of the robot converges with the reference path. The same observation applies to the curves of the coordinates $\boldsymbol{x}$ and $\mathbf{y}$ as well as the angle $\boldsymbol{\theta}$, as illustrated in Figure 3.10. It should be noted that, there is no deviations on the $\boldsymbol{x}$ and $\boldsymbol{y}$ paths.

Figure 3.11 illustrates the reference Lemniscate path that the robot should follow, represented by a red dashed line, while the actual path is depicted by a solid blue line. Figure 3.12 shows the curves representing the evolution of the reference and actual positions of the robot over time and its rotational direction. All the results are very satisfactory. From all these results, we can say that there is not significate difference between the nonlinear and linear MPC.

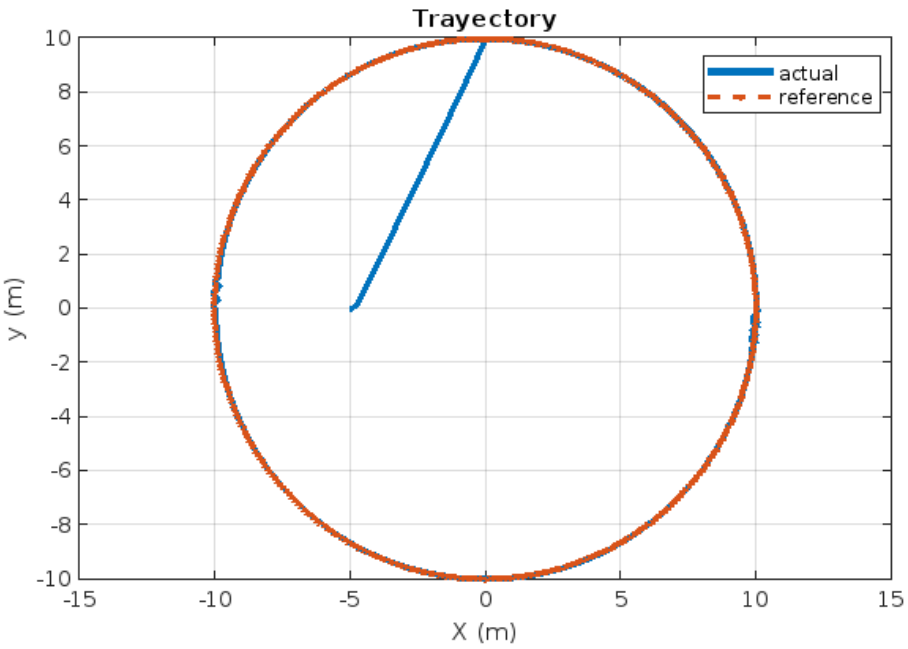


Figure 3.9: The circular trajectory of the robot in the XY plane.

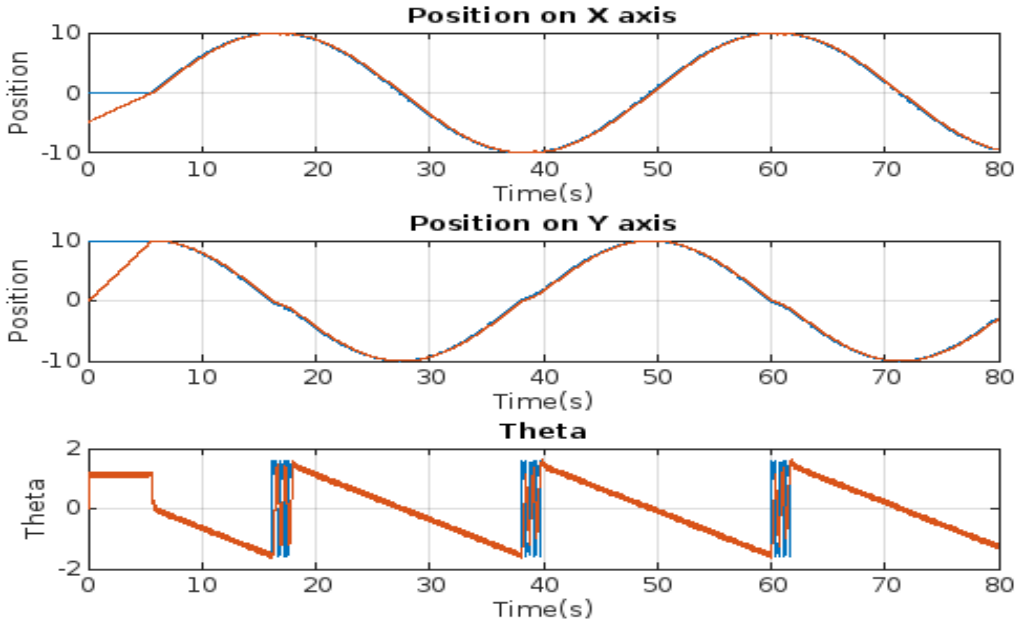


Figure 3.10: Evolution of the robot's components over time.

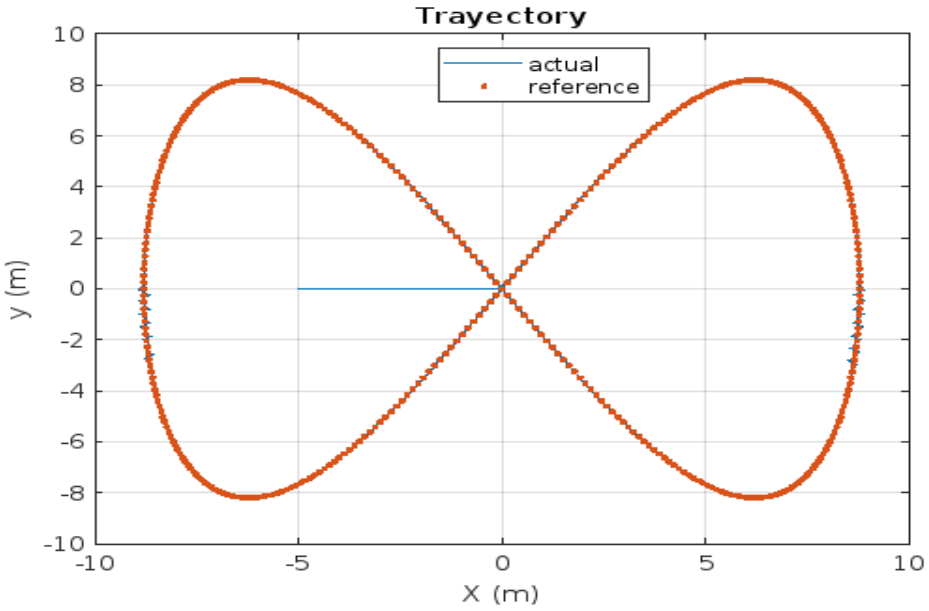


Figure 3.11: The lemniscate trajectory of the robot in the XY plane.

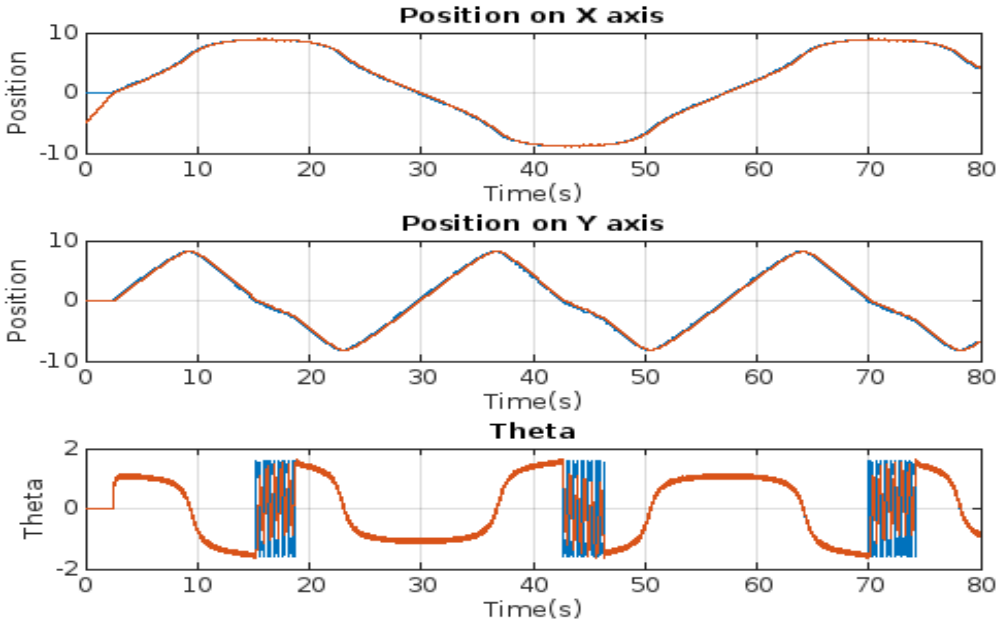


Figure 3.12: Evolution of the robot components over time in case of lemniscate reference trajectory.

3.4 Path planning Results using Q-Lerning

In this section, we used the environment SimpleMap from Matlab, in which we used the Q-Learning algorithm to plan a path allowing the robot to move from a start position (represented by a red circle) to a goal position (the green circle). Figure 3.13 shows the obtained result of the planned path to be followed by the robot in order to arrive to the goal state on the occupancy grid using only 4 types of actions: north, south, east and west as shown in figure 3.14. It will be noticed that the path has a stepped shape which is not very practical to a robot.

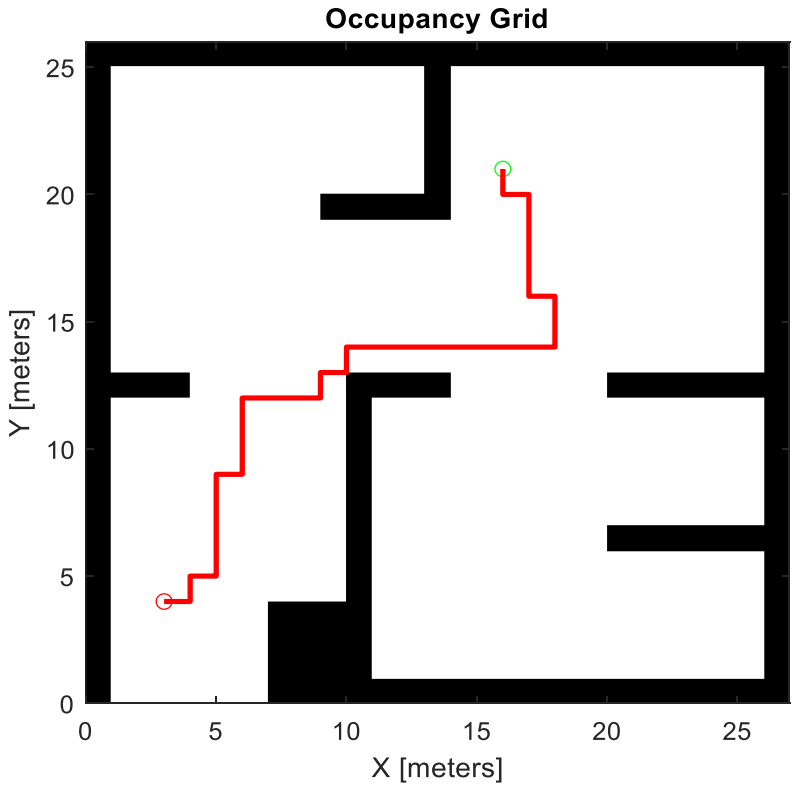


Figure 3.13: Trajectory in the XY plane using 4 actions.

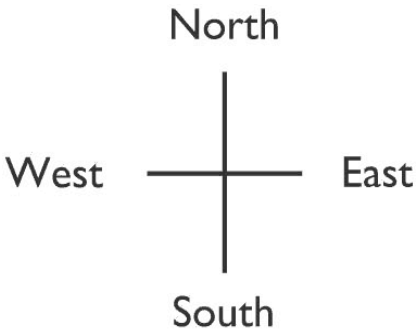


Figure 3.14: The 4 types of actions used

Figure 3.15 illustrates the obtained path when using 8 types of actions (north, south, east, west, north east, north west, south east, south west) as shown in Figure 3.16.

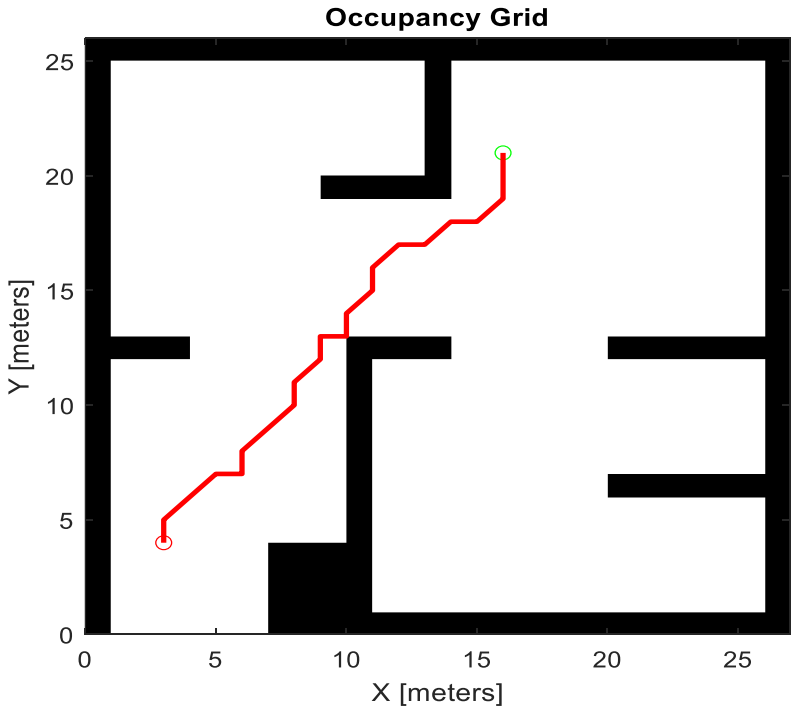


Figure 3.15: Trajectory in the XY plane using 8 actions.

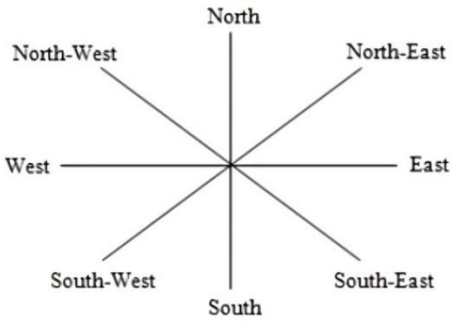


Figure 3.16: The 8 types of mouvements

In Q-learning implementations for grid worlds in MATLAB, the "inflate" function typically adjusts the occupancy grid representation by expanding the occupied space around obstacles. This expansion creates a buffer zone around obstacles, which helps path planning algorithms avoid getting too close to obstacles. By modifying the environment representation in this way, the agent perceives and interacts with the environment more effectively during learning, leading to safer and more robust navigation. This preprocessing step enhances the agent's ability to explore the environment and learn optimal policies.

Figure 3.17 shows the results after applying "inflate" function; the same result is represented in the real environment in Figure 3.18. We can notice that the path, in the real map of the environment, doesn’t touch any obstacle. It is to highlight that the dimension of the robot is taken into account while inflating the map.

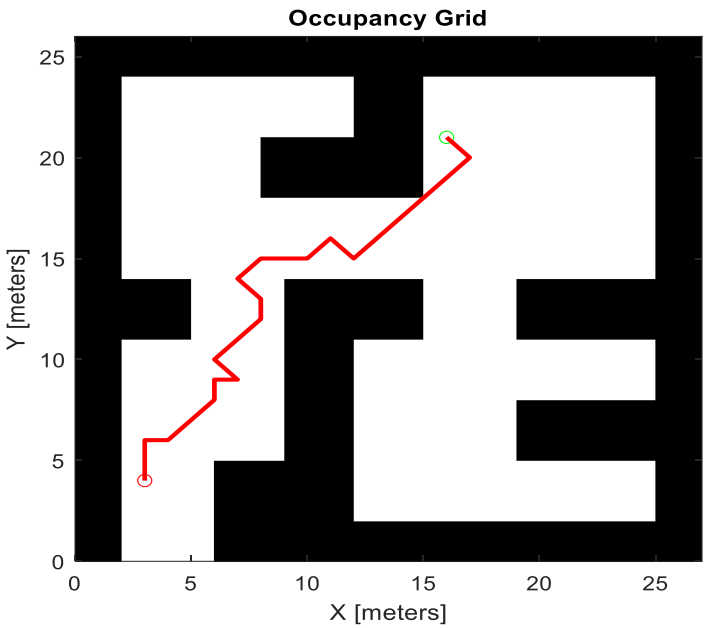


Figure 3.17: The planned path in the grid world after applying "inflate" function .

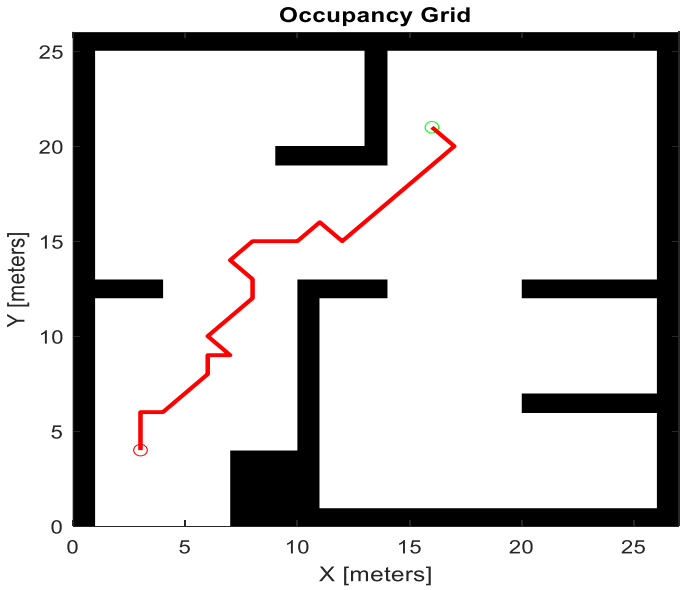


Figure 3.18: The planned path in the real grid world.

The following figure shows the result of the obtained reference path when we change the start and the goal positions. The path is shown in the inflated and the real map of the environment.

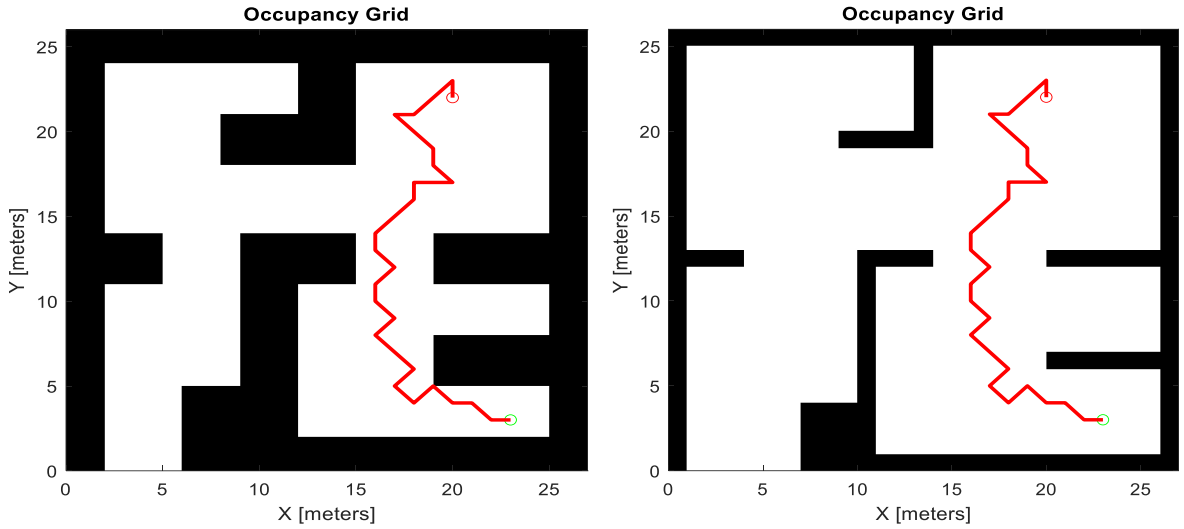


Figure 3.19: The planned path using different start and end positions

3.5 Path planning and Path Tracking Results

In this section, we considered the path obtained in the results presented in Figure 3.18. We applied the MPC control method on the robot to track the planned path. We choose a prediction horizon N=20, a sample$T_s = 0.1s$. The weighting matrices used are $R = diag(1,1)$ and $Q = diag(100,100,10)$, such that there will not be a significant importance to the robot's orientation against the importance given to X-Y position. Constraints in the amplitude of the control variables are: $V_{max} = 0.9m/s$, $V_{min} = -0.9\ m/s$; $\omega_{max} = \frac{\pi}{2}\ rad/s$ and $\omega_{min} = -\frac{\pi}{2} rad/s$. The results were as follows; Figure 3.20 presents the reference path in red and the actual trajectory of the robot in blue. In the left the reference path is represented by a set of waypoints obtained using the Q-Learning algorithm in the SimpleMap environment; while in the right figure, a linear interpolation is applied to the waypoints. It will be noticed that the actual trajectory of the robot follow perfectly the reference one.

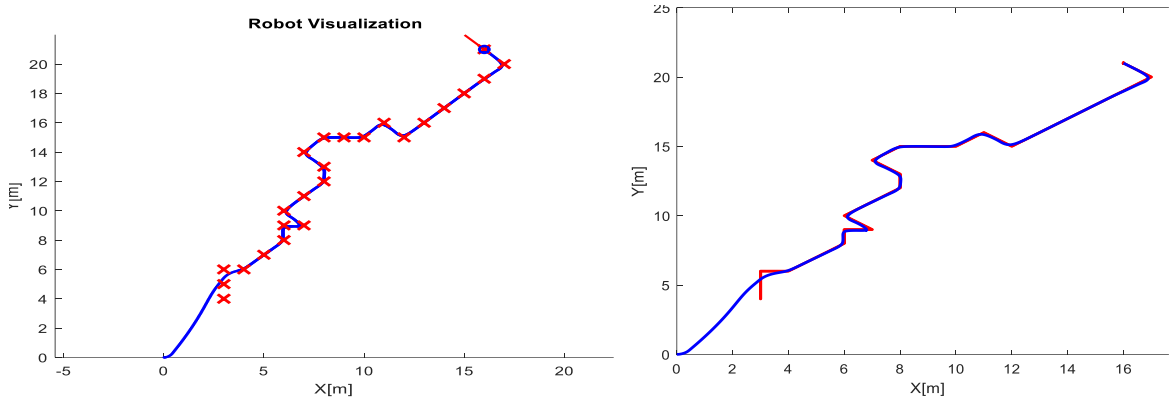


Figure 3.20: Trajectory of the robot in the XY plane.

Figure 3.21 shows that x and y position coordinates of the robot follow perfectly the reference coordinates. This is different for the robot's orientation since we did not give it a significant importance. Figure 3.22 presents the linear and angular velocities which are inside the limits imposed by the constraints.

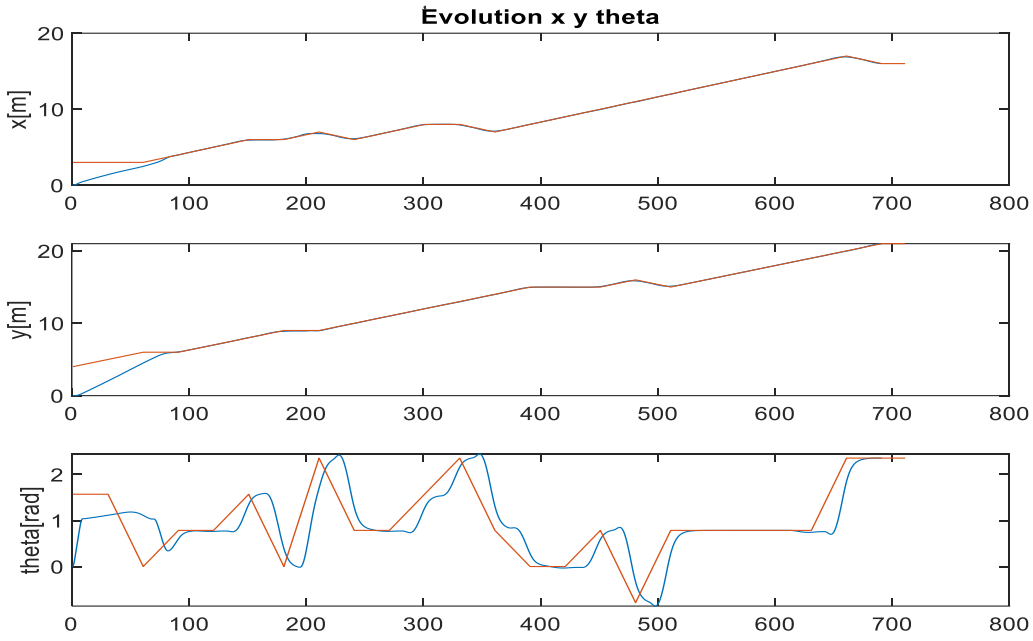


Figure 3.21: Evolution of the robot components over time.

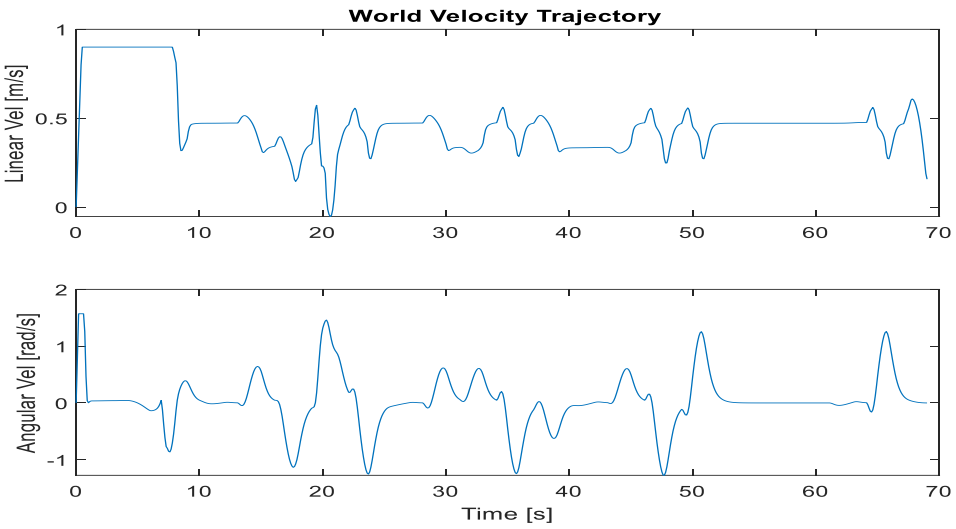


Figure 3.22: World linear and angular Velocities (v, ω).

In the case where we chose the weighting matrix $Q = diag(100,100,100)$, which means that the priorities are equal, indicating a significant importance of the robot's orientation. The result is as follows. We notice that the actual trajectory of the robot is smoother.

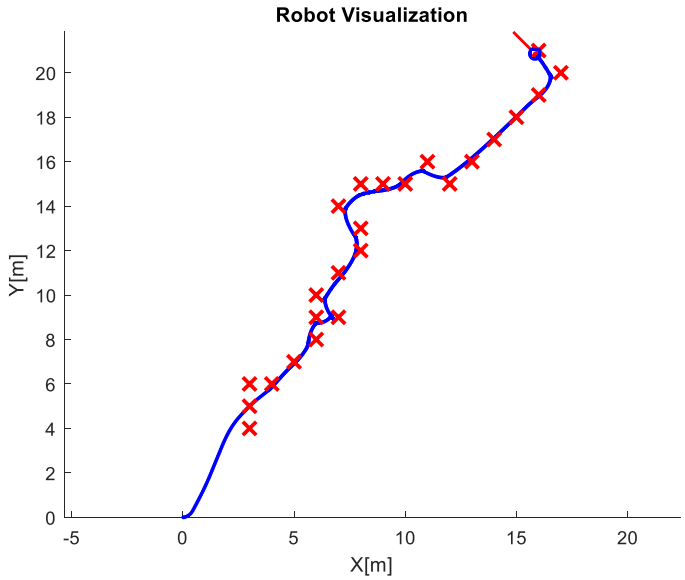


Figure 3.23: Reference and actual Trajectories in the XY plane in the case of equal weights of robot coordinates.

In the following Figure, we see a significant enhancement in the robot's orientation while the robot tracks the planned path. The linear and angular velocities are shown in Figure 3.25; we notice that they are inside the limits imposed by the constraints.

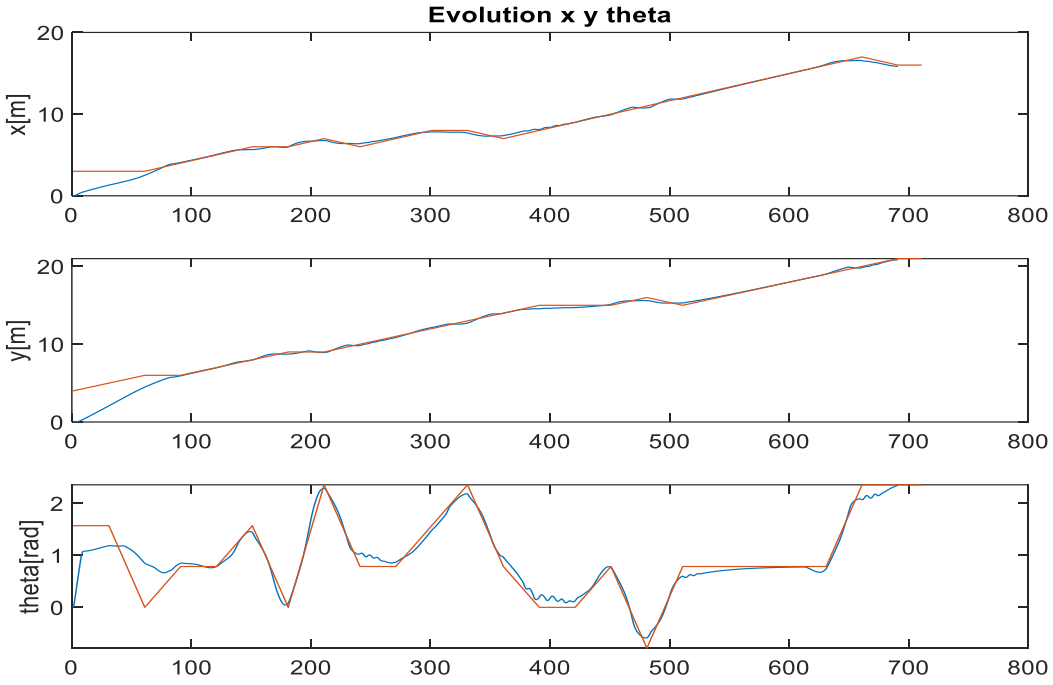


Figure 3.24: Evolution of the robot components over time.

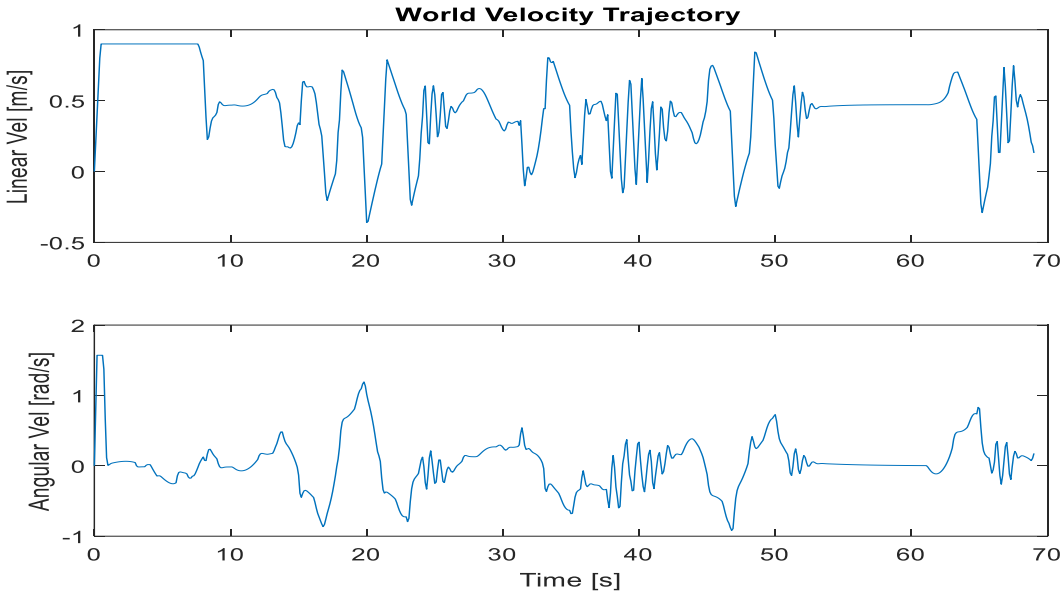


Figure 3.25: World linear and angular Velocities (v,w).

3.6 Results in Gazebo

In this section, we will perform simulations using the Matlab-ROS interface, Gazebo, and the simulated unicycle robot Turtlebot. It is worth noting that the real unicycle robot can be controlled by simply modifying some instructions in our code.

Gazebo offers the ability to accurately and efficiently simulate a group of robots in complex indoor and outdoor environments. In our work, we simulated a Turtlebot robot and, leveraging the Matlab-ROS interface, we were able to plan, using Matlab, its trajectory in the Office environment map using Q-Learning algorithm. Additionally, we controlled the Turtlebot robot by MPC control law to autonomously navigate along the planned path in the Gazebo office environment. Figure 3.26 shows the environment office and the Turtlebot robot in Gazebo simulator. In this simulation, we can see the robot moving in the gazebo office environment and at the same time it moves in the office map in Matlab.

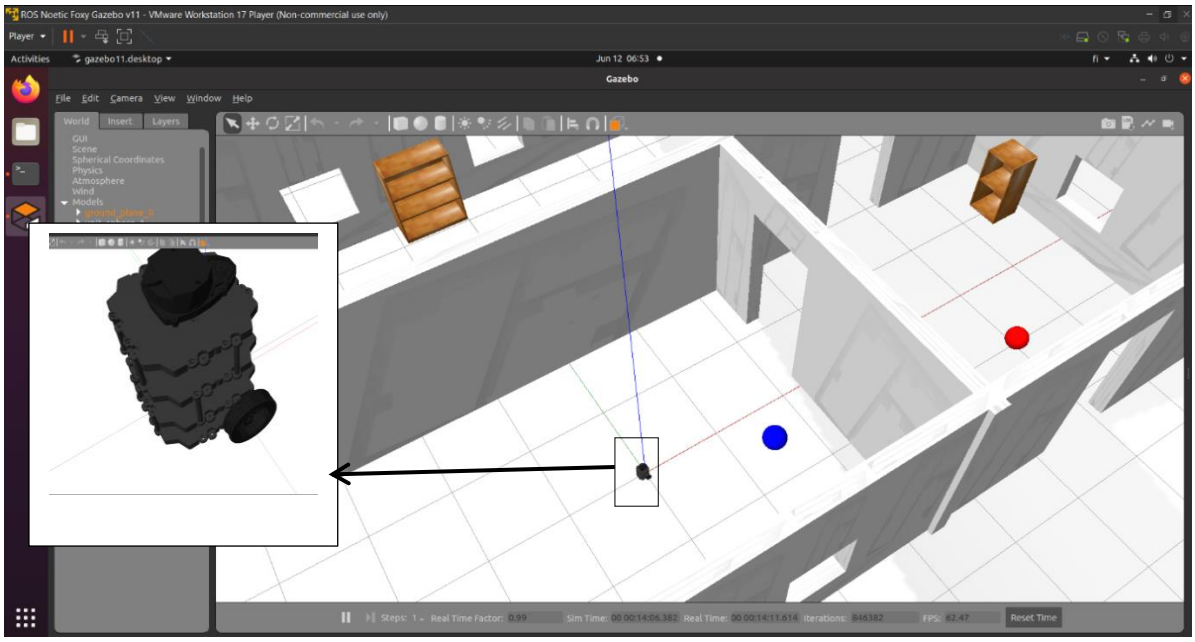


Figure 3.26: Office environment and the Turtlebot robot in GAZEBO

The weighting matrices used are R = diag(1,1) and Q = diag(100,100,10), such that there will not be a significant importance to the robot's orientation against the importance given to X-Y position. Constraints in the amplitude of the control variables are: V_{max}= 0.6m/s, V_{min} = 0 m/s ; $\omega_{max} = \frac{\pi}{2}$ rad/s and $\omega_{min} = -\frac{\pi}{2}$rad/s. The results were as follows; Figure 3.27 shows the path planned by using Q-Learning algorithm and Figure 3.28 represents the actual path of the robot after being controlled using the MPC algorithm. We can observe that the robot moved from the starting point, represented in red, to the endpoint, represented in green, smoothly and avoided the obstacles it encountered without any issues. Figure 3.29 illustrates the linear and angular velocities that satisfy the imposed constraints.

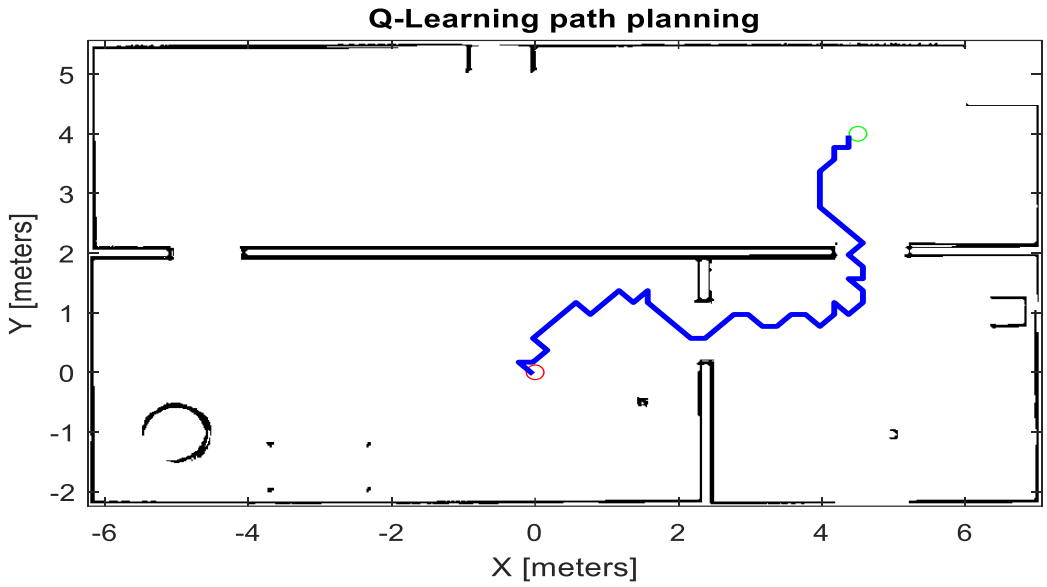


Figure 3.27: Q-Learning based planned path.

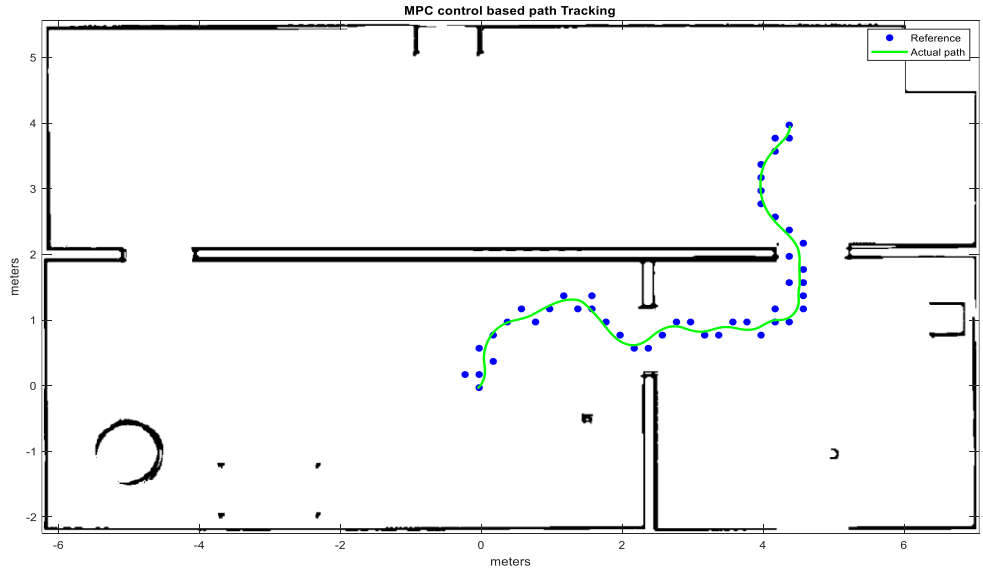


Figure 3.28: The actual path of the robot controlled by MPC

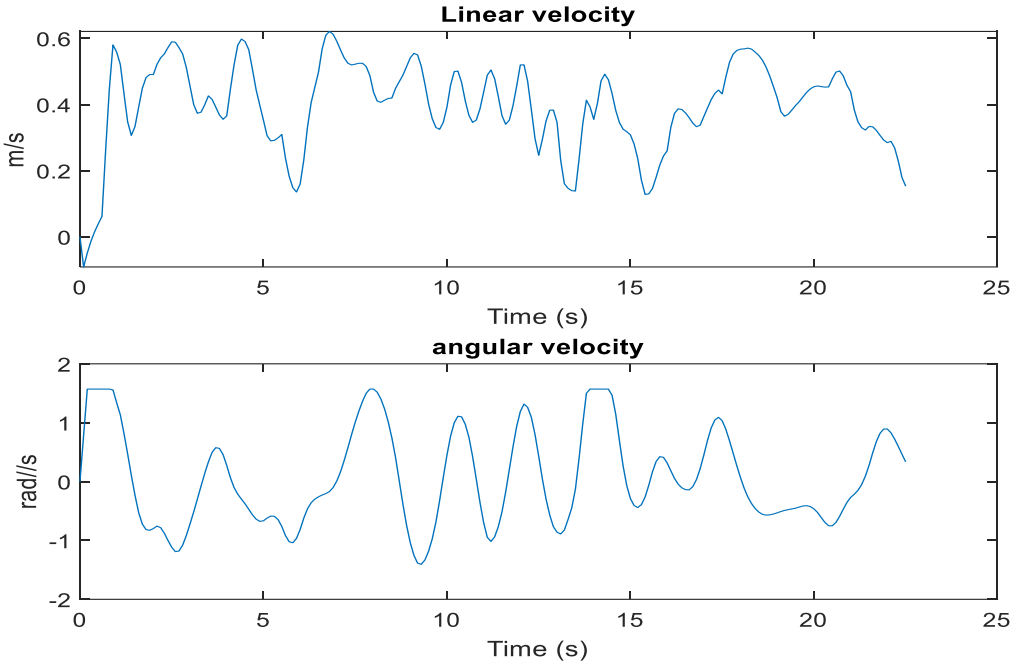


Figure 3.29: World linear and angular Velocities (v, ω).

In Figure 3.29, we can see the robot tracking the planned path within the office map in Matlab and simultaneously in Gazebo. It starts from the position [0, 0] in order to reach its destination [4.5, 4].

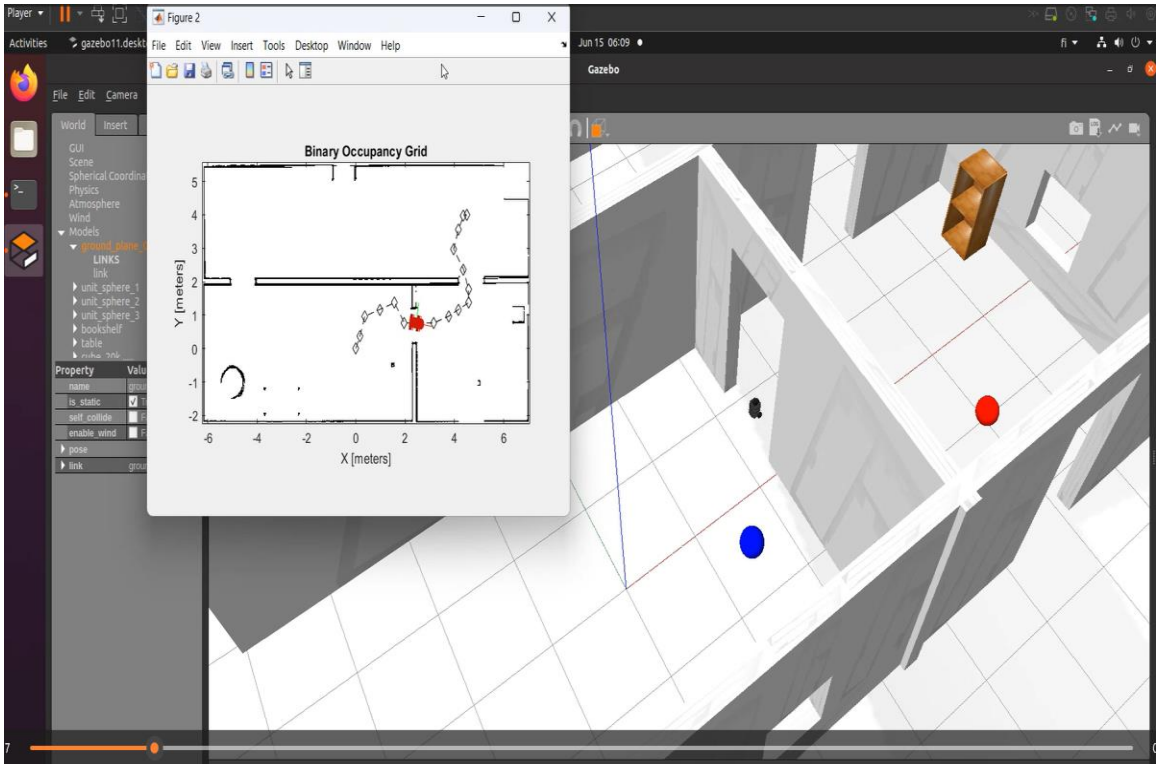


Figure 3.29: The robot tracks the planned path within the office map in Matlab and simultaneously in Gazebo. It starts from the position [0, 0] in order to reach its destination [4.5, 4].

Based on these results, we can say that the planning and path following of the unicycle robot in Gazebo were highly satisfactory and effective. It is possible to control the real unicycle robot using the same algorithms and code with just minor adjustments to some instructions.

3.7 Conclusion

In this chapter, we began by presenting the results of path tracking using Linear and nonlinear model predictive control for different types of trajectories (Circular, leminscate …). Then, we discussed the obtained results, which were very encouraging. After that, we implement the Q-Learning algorithm for the path planning problem; we obtained satisfactory results. After that, the robot was controlled using MPC to follow the planned path and navigate in its

environment while avoiding obstacles. The performances were evaluated through simulation, where it was observed that the actual path aligns well with the desired path. Finally, the simulation results of the unicycle robot were presented using the GAZEBO simulator.

Conclusions and Future works

This work was undertaken with the aim of applying advanced control law based on the principle of predictive control, which allows smooth tracking of a specific reference path. The predictive control model was developed and tailored to the wheeled mobile robot, and mathematical tools were utilized to facilitate law modeling. The good results demonstrated the effectiveness of this law in achieving the desired tracking with high accuracy. The system continued to track the specified path despite variations in initial conditions, and it was able to adapt to a new path without the need for major modifications to the original law.

In the other hand, path planning is very important and is a basic function for mobile robots.The use of the Q-learning algorithm, a reinforcement learning method, can solve many of the complex problems that traditional algorithms cannot solve. We applied this algorithm to allow a robot navigate in a complex environment in a safer manner without hitting obstacles. The simulation results show that the Q-learning algorithm effectively learns optimal paths over time, significantly reducing the collision rate compared to traditional path planning methods. These results highlight the potential of reinforcement learning techniques in enhancing the autonomy and safety of mobile robotic systems in various real-world applications.

We also presented the results of Co-simulation between Matlab and the ROS system using the Gazebo simulator to create environments and simulate the Turtlebot robot. The planning and trajectory monitoring results were very satisfactory and conclusive.

As future work, and based on the methods studied and implemented in this dissertation, we propose to make improvements allowing trajectories to be planned in real time in unknown environments and/or containing dynamic obstacles. We can also add SLAM and vision processing technologies in the path planning algorithm to make it more intelligent and broaden its application in our daily life.

Bibliography

[1] Spyros G. Tzafestas, "Introduction to Mobile Robot Control", School of Electrical and Computer Engineering, National Technical University of Athens Athens, Greec, 2014.

[2] ATLAS by Boston Dynamics Inc. https://www.bostondynamics.com/atlas. Last accessed on 15 February 2019.

[3] Tesla. Available: https://www.tesla.com

[4] R. Mathew and S. S. Hiremath, ‘‘Development of waypoint tracking controller for differential drive mobile robot,’’ in Proc. 6th Int. Conf. Control, Decis. Inf. Technol. (CoDIT), Paris, France, Apr. 2019.

[5] D.Q. Mayne, J.B. Rawlings, C.V. Rao and P.O.M. Scokaert,”Constrained model predictive control: Stability and optimality”, Automatica, Vol. , N° 36, pp. 789-814, 2000.

[5] Kühne, F. (2005). Predictive control of nonholonomic mobile robots (in portuguese), Dissertation (M.Sc.), Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil.

[6] João Manoel Gomes da Silva Jr. MOBILE ROBOT TRAJECTORY TRACKING USING MODEL PREDICTIVE CONTROL. . Oswaldo Aranha, 103 – CEP 90035-190 Porto Alegre, RS, Brasil.

[7] Zohaib H. Farooqi. Implementation of Nonlinear Model Predictive Control on all Terrain Mobile Robot:24

[9] Henson, M. A. (1998). Nonlinear model predictive control: current status and future directions, Computers and Chemical Engineering 23(2): 187–202.

[10] https://levelup.gitconnected.com/model-predictive-control-for-autonomous-vehicle-an- in-depth-guide-de984308ba10

[11] Planification de trajectoire et commande d’un robot mobile en
Utilisant l’interface MATLAB-ROS,(2022)

[12] What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/?ref=header_search)

[13] A Survey of Path Planning Algorithms for Mobile Robots by karthik karur,nitin sharma,chinmay dharmatti, joshoa E,siegel Submission received: 27 May 2021 / Revised: 19 July 2021 / Accepted: 25 July 2021 / Published: 4 August 2021

[14] Path planning algorithm based on Improved Artificial Potential Field method September 2023 by Eryi Zhang

[15] [https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners (updated)].

[16] https://www.datacamp.com/tutorial/introduction-q-learning-beginner-Tutorial

[17] LaValle, S. M. (2006), "Planning Algorithm".

[18] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D.. "Introduction to Autonomous Mobile Robots", 2011.