POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Implementation of path following and obstacle avoidance in two omnidirectional platforms

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Andrea Mauri, Giulia Fasoli**

Student ID: 945541, 953744
Advisor: Prof. Matteo Matteucci
Co-advisors: Simone Mentasti
Academic Year: 2021-22

Like I always say, can't find a door? Make your own.

- Full Metal Alchemist

Dedicate to our mums and dads.

Acknowledgements

We would like to thank the AirLab staff and the stimulating atmosphere created; we came out enriched, not only as engineers. A special thanks to professors Matteucci and Mentasti for their advice and the opportunity to work on this experimental thesis together. We learned a lot.

We want to thank all the true "Iguanine": Anto, Edo, Fil and Max. The best gambling den of these years. Many thanks to all our friends outside the Politecnico for all the patience and moments spent together. A special thank to Andrea and Zoe for believing in us.

A final thank to our families for helping us, supporting us, and accompanying us through all our experiences.

Abstract

The thesis presented here has the main objective of the realization of a position controller for two omnidirectional platforms, with different wheels and different configurations, then integrated with ROS navigation stack through an intermediate planner. This work presents an analysis of the kinematics of the two robots and the estimation of a complete model through a black-box approach. The adopted controller is a PID, based on feedback measurement of the output according to OptiTrack™motion capture system, in this way the two platforms are able to directly reach a single goal or to follow multiple waypoints before arriving at the final destination. The intermediate planner implemented generates several close waypoints on the global pre-determined path according to a look-ahead distance. An obstacle avoidance algorithm is also developed to allow the motion of the robots in an environment characterized by static unmapped obstacles whose presence is read according to the onboard lidar sensor; a cost is assigned to each obstacle depending on its distance from the robot so that it is possible to create a map based on these values and the platforms can choose the path with a cost lower than a defined threshold. Finally, a double mode of navigation is implemented allowing to switch between the omnidirectional motion and another similar to the differential drive robot one. All the previous techniques are tested in a real environment and experimental results of both two platforms are presented.

Keywords: Omnidirectional platforms, PID, Waypoints, Navigation, Intermediate planner, Obstacle avoidance

Abstract in lingua italiana

La tesi presentata ha come obiettivo principale la realizzazione di un controllore di posizione per due piattaforme omnidirezionali, con ruote e configurazioni diverse, integrato poi con il pacchetto di navigazione di ROS attraverso un pianificatore intermedio. Questo lavoro presenta un'analisi della cinematica dei due robot e la stima di un modello completo attraverso un approccio black-box. Il controllore adottato è un PID, basato sulla misurazione in retroazione dell'output secondo il sistema di motion capture OptiTrack, in questo modo le due piattaforme sono in grado di raggiungere direttamente un singolo obiettivo o di seguire più waypoint prima di arrivare alla destinazione finale. Il pianificatore intermedio implementato genera diversi waypoint ravvicinati lungo il percorso globale predeterminato in funzione di un orizzonte spaziale. Viene inoltre sviluppato un algoritmo di obstacle avoidance per consentire il movimento dei robot in un ambiente caratterizzato da ostacoli statici non mappati, la cui presenza viene letta in base al sensore lidar presente sui robot; a ogni ostacolo viene assegnato un costo in base alla sua distanza dal robot, in modo che sia possibile creare una mappa basata su questi valori e le piattaforme possano scegliere il percorso con un costo inferiore a una soglia definita. Infine, viene implementata una doppia modalità di navigazione che consente di passare dal movimento omnidirezionale a un altro simile a quello dei robot a trazione differenziale. Tutte le tecniche precedenti sono state testate in un ambiente reale e sono stati presentati i risultati sperimentali di entrambe le piattaforme.

Parole chiave: Piattaforme Omnidirezionali, PID, Waypoints, Navigazione, Pianificatore Intermedio, Obstacle Avoidance

Contents

1 | Introduction

1.1. Introduzione

Negli ultimi decenni, l'automazione e la robotica stanno acquisendo sempre più importanza, non solo in ambito industriale ma anche nella vita di tutti i giorni. La ragione principale di questo successo è legata alla maggiore produttività garantita dai robot, dall'enorme efficienza nel completare i compiti e, infine, dalla possibilità di operare in situazioni potenzialmente pericolose. L'origine della robotica moderna è strettamente legata alll'applicazione industriale, come ad esempio i bracci meccanici, capaci di completare compiti velocemente e con grande precisione ma isolati e senza la possibilità di interagire con operatori umani.

Questa limitazione è la ragione principale dello sviluppo di robot mobili, tipicamente veicoli a guida autonoma (AGV), che forniscono uno spazio di lavoro maggiore in aggiunta alla possibilità di operare in svariati campi come, ad esempio, ospedali, alberghi, magazzini, e applicazioni militari o agricole. Inoltre, i robot mobili sono in grado di lavorare a contatto con operatori umani o altri robot in modo completamente autonomo e sicuro.

Gli aspetti più importanti di un AGV sono: l'abilità di lavorare in completa autonomia, un'ampia flessibilità per essere capaci di svolgere diversi compiti e muoversi lungo diversi percorsi e, infine, essere in grado di evitare oggetti statici, altri robot, e operatori in movimento.

Considerando i vantaggi menzionati, la ricerca su questa tipologia di robot è aumentata portando numerose alternative tecnologiche come, ad esempio, differenti tipologie di ruote e possibili configurazioni permettendo un numero considerevole di movimenti eseguibili. In particolare sono stati effettuati numerosi studi ed esperimenti sui robot omnidirezionali. Questi sono in grado di muoversi in ogni direzione sfruttando i tre gradi di libertà che hanno a disposizione, i.e. le due coordinate planari x ed y e l'orientamento θ. I robot omnidirezionali sono adatti ad ambienti molto dinamici o a posti caratterizzati da spazi stretti e corridoi, proprio per la loro capacità di eseguire qualsiasi manovra.

L'obiettivo di questo lavoro di tesi è l'implementazione di un controllore di posizione PID in un ambiente dotato di un sistema motion capture, che viene integrato ai preesistenti metodi di navigazione attraverso l'utilizzo della piattaforma ROS. Successivamente, lo sviluppo di un algoritmo per evitare gli ostacoli adatto alla elevata mobilità dei robot omnidirezionali. Infine, ad ogni analisi affrontata in questo lavoro di tesi, viene aggiunto un confronto tra le due piattaforme adottate basato sul loro modello e sui risultati sperimentali ottenuti.

Gli obiettivi del controllo di posizione sono i seguenti:

- Il robot deve raggiungere una singola meta partendo da una determinata posizione soddisfando le specifiche.
- Il robot deve seguire una serie di punti intermedi che possono simulano possibili stazioni di lavoro in applicazioni future.

Il controllore PID nasce nei primi anni del 1900 ed è ancora una delle strategie di controllo più usate, questo grazie alla sua semplicità e alla sua facilità di implementazione anche in microcomputer che hanno limitate capacità computazionali. Un'altra ragione per utilizzare questo approccio nel nostro lavoro è la sua adattabilità, poichè, operando con due piattaforme utilizzabili in diversi ambienti, la facilità di regolazione dei parametri del controllore è preferibile rispetto ad altre strategie.

Per integrare il PID con i metodi di navigazione presenti nell'ambiente ROS, è necessario generare punti intermedi lungo il percorso predefinito. Per farlo personalizziamo un algoritmo nato nel mondo del deep reinforcement learning, capace di generare subgoals secondo un certo orizzonte spaziale. Infine, viene presentato un algoritmo usato per evitare ostacoli con l'ausilio dei sensori presenti a bordo dei robot.

L'intero sistema è testato su piattaforme reali in diverse situazioni possibili e sotto diverese condizioni iniziali, ottenendo risultati soddisfacenti che garantiscono un buon punto di partenza per sviluppi più avanzati di entrambi i robot.

Tutti gli algoritmi e i modelli definiti in questo lavoro di tesi sono sviluppati utilizzando Python, Matlab®, e Simulink®.

1.2. Introduction

In the last decades, automation and robotics are gaining more importance, not only in the industrial field but also in everyday life. The main reasons behind this success are related to the higher productivity guaranteed by robots, the huge efficiency in accom-

plishing tasks, and finally, their possibility to operate in conditions that are potentially dangerous. The origin of modern robotics is strictly related to industrial applications, for example, mechanical arms, able to complete tasks quickly and with high precision but isolated and without interactions with human operators.

This limitation is the main reason for the development of mobile robots, typically Automated guided vehicles (AGV), that provide a larger working environment in addition to the possibility of operating in several fields, for example, hospitals, hotels, warehouses, and military or agricultural applications. Moreover, mobile robots are able to operate with humans or other robots in a completely autonomous and safe way.

The most important aspects of an AGV are: the ability to work with complete autonomy, large flexibility to be able to perform different tasks and travel on different paths and, finally, the ability to avoid static obstacles, other robots, and moving operators.

Considering the mentioned advantages, the research on these kinds of robots is increased bringing several technological alternatives such as different types of wheels and different possible configurations, allowing a considerable number of executable motions. In particular, numerous studies and experiments are executed on omnidirectional robots. These are able to move in any direction exploiting their three available degrees of freedom, i.e. the two planar coordinates x and y and the orientation θ. Omnidirectional robots are suitable for very dynamic environments or places characterized by narrow spaces and corridors because of their ability to perform any manoeuvre.

The aim of this thesis work is the implementation of a PID position controller in an environment equipped with a motion capture system, which is integrated with the pre-existing navigation methods through the ROS platform. Subsequently, an algorithm for obstacle avoidance suitable for the elevated mobility of omnidirectional robots is developed. Finally, to each analysis faced in this thesis work is added a comparison between the two platforms based on their model and on the obtained experimental results.

The position control objectives are the following:

- The robot must reach a single goal from a certain starting position fulfilling specifications.
- The robot has to follow a series of intermediate points that can simulate possible workstations in future applications.

PID controller is born in the first years of 1900 and it's still one of the most used control strategies, this is due to its simplicity and ease of implementation also on microcomputers with limited computational capabilities. Another reason to adopt this approach in our

work is its adaptability because, since we are operating with two platforms that can be used in various situations and environments, the ease of tuning the controller parameters is preferable with respect to other strategies.

In order to integrate the PID with the navigation methods present in the ROS environment, it is necessary to generate intermediate points along the predefined path. To do so we customize an algorithm that is born in the world of deep reinforcement learning and that is able to generate subgoals according to a spatial horizon. Finally, an algorithm able to avoid obstacles through onboard sensors is presented.

The full system is tested on real platforms in different possible situations and under different initial conditions, obtaining satisfactory results that guarantee a good starting point for more advanced implementation of both robots.

All the algorithms and the models defined in this thesis work are developed through Python, Matlab®, and Simulink®.

1.3. Contribution

The contribution of this work are listed below:

- Implementation of a position control through a feedback loop mechanism and the Proportional Integral Derivative (PID) controller, starting from robot kinematics and the theoretical model of the system obtained using a black-box approach.
- Implementation of an algorithm that allows path following-like motion of the robots and the possibility to adopt the PID controller with the commonly used navigation approaches.
- Implementation of an algorithm that allows obstacle avoidance based on a customized vector field histogram approach enabling the robot to move in an environment with unmapped obstacles.
- Application of the overall developed system on two omnidirectional platforms that move in a real working environment equipped with a motion capture system.
- Comparison of the performances of the two platforms based on their experimental results.

1.4. Thesis outline

Chapter 2 presents an overview of the technological background of mobile robots, focusing on the different typologies of wheels and configurations with a particular interest in omnidirectional robots. Then, the two most adopted control strategies are explained describing their advantages and disadvantages. Lastly, the starting point of the algorithms and methods used to allow the platforms to navigate and avoid obstacles are shown.

Chapter 3 illustrates the set-up, both hardware and software, used in the entire work. Starting with a deeper analysis relative to the two platforms' mechanics and going on with a brief presentation of actuators, sensors, onboard computer, and control boards, concluding with a description of the adopted software: ROS, MATLAB®, and Simulink®.

Chapter 4 introduces the kinematic model of the two platforms expressing all the required relationships. Moreover, it shows the black-box technique adopted to estimate the parameter of an approximated model starting from the measured data.

Chapter 5 presents the implementation of the PID controller explaining the meaning of each gain, the tuning strategy adopted, and the discretization technique used. Finally, it shows the experimental results relative to the control position and the development of a double navigation motion.

Chapter 6 describes the move_base package functioning as well as the algorithm that generates subgoals to simulate trajectory tracking and the algorithm used for obstacle avoidance so that it is possible to perform complete navigation. Moreover, the experimental results for both platforms are depicted.

Chapter 7 concludes the thesis by outlining the future works and possible upgrading of the robots.

2 | State of The Art

The purpose of this chapter is to provide a general description of wheeled mobile robots starting from the different types of wheels, discussing their properties from the mobility point of view, introducing different control strategies and navigation algorithms, and describing the most commonly encountered realizations of such robots.

2.1. Wheeled mobile robots

In this Section, we describe a variety of wheels and their possible configurations in mobile robots. We discuss the restriction of robot mobility implied by the use of some of these wheels and the overcome of this limitation with the development of omnidirectional robots.

2.1.1. Types of wheel

Wheeled mobile robots are widely used to achieve robot locomotion and, although it is difficult to overcome uneven ground conditions, they are suitable for several target environments in practical applications. In general, wheeled robots are characterized by lower energy consumption and faster motion than other locomotion mechanisms (e.g., tracked vehicles or legged robots) [5].

Nowadays multiple types of wheels with different properties and for various purposes are available. The primary distinction to be made is between standard design wheels and unconventional design wheels. In a single-wheel standard design there are two main conditions to be defined:

- a mechanical design to allow steering motion (i. e. to fix the wheel orientation or not)
- the determination of steering and driving actuation (i.e. active or passive drive)

As standard designs, we focus on the conventional wheel, the steering wheel and the caster wheels. These wheels have the classical round shape that is commonly seen [29].

Conventional wheels, Figure 2.1a, are simple and functional providing forward and backward rotation of the wheel. The steering axis is fixed, i.e. the orientation of the wheel does not change, moreover, the displacement is limited on the driving direction.

Steering wheels are similar to conventional ones from an aesthetic point of view, but with different mechanical structures. They also allow rotation around the vertical axis: a driving motor provides forward and backward locomotion, while a steering motor controls the rotation around the axis so that this kind of wheel can behave as a conventional wheel or as a steering wheel.

Caster wheels help to achieve near-omnidirectional mobility of a vehicle. There exist two categories of them: rigid wheels, Figure 2.1b, and swivel wheels, Figure 2.1c. The first ones can rotate only forward and backwards while a swivel wheel can also passively rotate 360 degrees around the vertical axis.

Conventional, steering and caster wheel have simple designs and for this reason, are cheaper, moreover, they are robust for rough surfaces up to some degree of roughness. As for the unconventional design wheel, we describe the two kinds of wheels that are mounted on the platforms used in this thesis.

Omni wheels are characterized by a combination of a main active wheel and passive freely rotating rollers in which the axes of passive rollers are orthogonal to the main wheel axis as it is shown in Figure 2.1d. Free rollers are employed in order to eliminate the non-holonomic velocity constraint. Merging the active rotation of several active wheels with the passive rotation of the rollers, it is possible to move a vehicle in any direction.

Mecanum wheels are similar to the Omni ones but rollers are mounted with their axis at an angle of 45 degrees relative to the axis of the active wheelbase as we can see in Figure 2.1e. As said, passive rollers are free to rotate around the axis of rotation, which enables the lateral motion of the wheel. As a result, the driving velocity should be controlled by an actuator, while the lateral velocity is passively driven. Applying different velocities to each wheel a robot can move in any direction.

Omnidirectional wheels are suggested in indoor environments due to their high sensitivity to rough and unequal surfaces. The mechanical design of unconventional wheels is more complex compared to the standard wheels, as a result, their cost of production is more elevated. An important feature to consider when analysing different kinds of wheels is their restriction on the motion. Each wheel has a defined number of degrees of freedom that impact the robot's range of motion. Conventional wheels only have one degree of freedom; steering wheels can guarantee also rotation around a vertical axis, which means

(a) Conventional (b) Rigid Caster (c) Swivel Caster

(d) Universal Omni (e) Mecanum

Figure 2.1: Types of wheel

that they have two degrees of freedom and motion is still limited. Finally, universal Omni wheels and Mecanum wheels have three degrees of freedom allowing motion in any direction.

2.1.2. Robot configuration

Table 2.1 compares the degrees of freedom of wheels and their most used configurations. The abbreviation DW stands for Different Wheel, it is used to indicate that the configuration requires one or more different wheels together with the one specified, while

Type of Wheel	Possible wheels configuration	Minimal required number of wheel	DoF
Conventional	2 or more wheels	2	1
Steering	2 (1 DW) 3, 4 wheels	2 (1 DW)	2
Caster	1 or more (as SW)	1 (as SW)	1, 2
Omni	3 or 4 wheels	3	3
Mecanum	4 or more wheels	4	3

Table 2.1: Mechanical parameters and configuration [29].

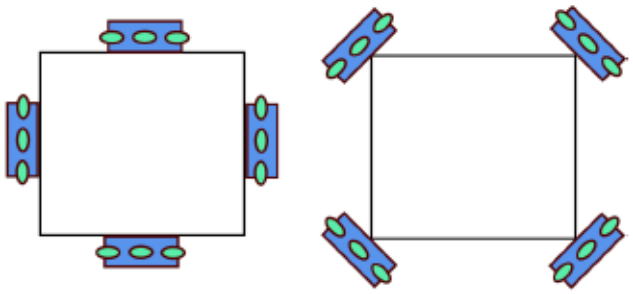


Figure 2.2: Omni wheels configurations.

SW means Support Wheel.

As reported in 2.1, Caster wheels are used in a passive way and, in order to obtain omnidirectional mobility, other wheels are required. A common configuration with caster wheels is the one present in the differential drive robot, Figure 2.3a, in which conventional wheels are used as active wheels while a caster wheel adds stability to the robot without limiting the robot rotation and therefore its degrees of freedom. It is possible to extend the two differential drive robot to a four wheel robot by mounting additional passive caster wheels. The major advantages can be summarized as:

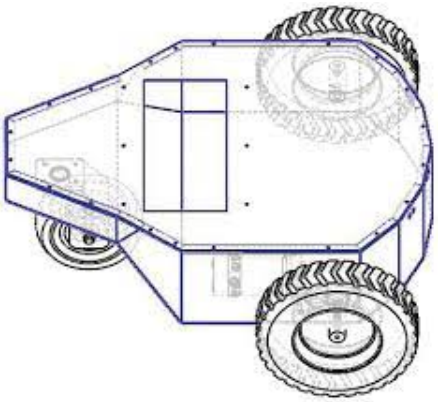
- A simple mechanical design with a simple kinematic model
- Low fabrication costs
- Errors can be easy to calibrate.

On the other hand, the disadvantages are:

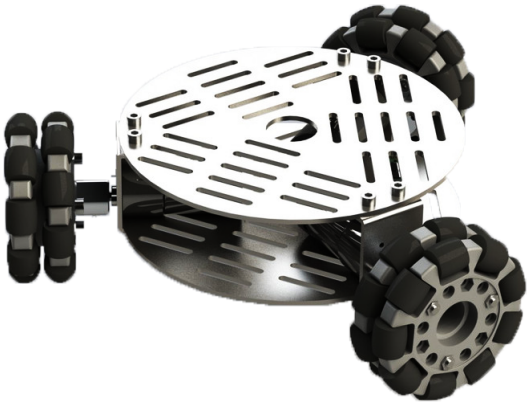
- Difficulty of moving irregular surfaces as a consequence of the possible loss of contact with the ground of one or more active wheels.
- Only forward and rotational movement allowed.

Steering wheels are typically used with four-wheeled platforms, such as car-like robots, three-wheeled vehicles such as triangular cart-like platforms in which two active steering wheels are combined with a single caster passive wheel, and two-wheeled configurations such as bicycles. Focusing on the car-like structure, the front two steering wheels must be synchronised to keep the same instantaneous centre of rotation. Typically, this solution is kinematically approximated to a bicycle model. Car-like robots are more stable during high-speed motion whereas a major disadvantage is reduced motion related to the steering mechanism.

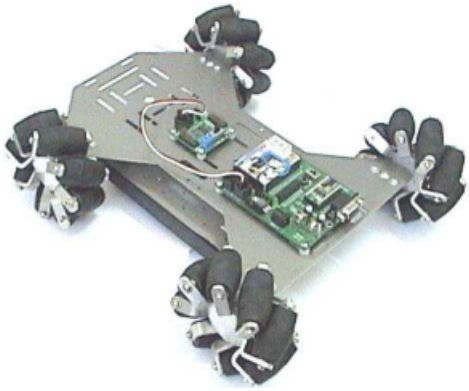
Universal Omni wheels can be used in three-wheeled configurations or four-wheeled



(a) Differential drive robot [8]



(b) Three Omni wheels configuration robot

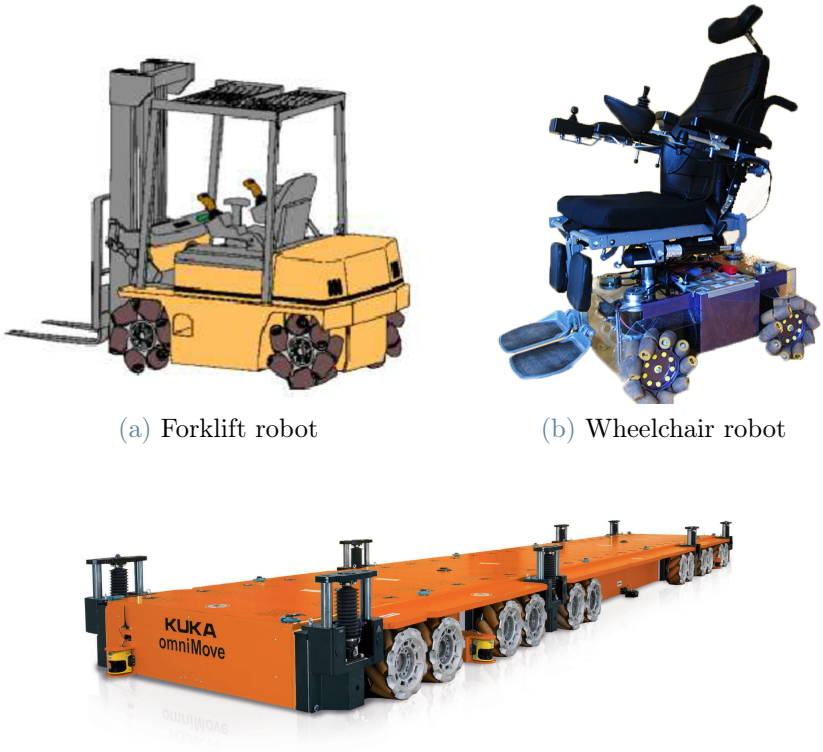


(c) Classical mecanum wheels configuration

Figure 2.3: Configuration examples

configurations. Three universal wheels are mounted on a triangular platform equally spaced every 120 degrees, as presented in Figure 2.3b. In four-wheeled design, instead, there are two possibilities: the first one in which wheels are located symmetrically on the sides of square mobile platforms with 90 degrees angle between the wheels, and the second one in which the wheels are mounted symmetrically at the corner of the square mobile platform with axis inclined by 90 degrees relative to each other. This difference is depicted in Figure 2.2.

Finally, Mecanum wheels are typically used in car-like platforms, in which wheels are mounted in line with each other as depicted in Figure 2.3c.



(a) Forklift robot (b) Wheelchair robot

(c) Industrial agv

Figure 2.4: Omnidirectional robot application examples

2.1.3. Omnidirectional robots

Omnidirectional mobile robots are continuously gaining more popularity, they are largely used in industrial productions, such as the forklift shown in Figure 2.4a or the AGV in Figure 2.4c, and in congested environments such as houses, offices, and hospitals but also in domestic, military, agricultural, and many other fields. Possible examples are service robots, wheelchairs (Figure 2.4b), and other devices used to help physical disabilities' daily life or the vehicle used in storage facilities with the ability to move, inspect and handle products.

These robots can guarantee greater manoeuvrability and efficiency, indeed, they are able to move instantaneously in any direction from any configuration in a two-dimensional plane.

A torque is applied to each wheel by an independent actuator, so their respective direction of rotation can be arbitrarily performed. In addition, omnidirectional systems can slide perpendicularly to the torque vector, thanks to the passive rollers mounted on the edge of the wheels, allowing greater flexibility in a congested environment. As a con-

sequence, translational and rotational motions are decoupled so that the robot can change its motion direction without changing its orientation. On the other hand, omnidirectional robots are inefficient in terms of energy consumption because the wheels can generate opposing forces and, therefore, can wear out faster than conventional wheels [32].

These robots are typically characterized by three and four wheels platforms. Three wheels configurations are simpler, cheaper from an equipment point of view, and have a lower energy consumption. On the other hand, four wheels configurations can have higher acceleration with the same actuators and have better floor traction meaning that the slippage is lower if we assume that all the wheels are in contact with the ground [20]. Four-wheeled robots may require some kind of suspension to distribute forces equally among the wheels.

2.2. Control Strategies

Position control is a crucial topic in autonomous mobile robotics. The precision of the positioning of the platforms together with the repeatability of the results is a fundamental requirement to be satisfied when the robots need to move around a working environment. This Section details the main control strategies used in industrial applications concerning omnidirectional mobile robots while underlining the advantages and disadvantages of both regulators.

Proportional Integral Derivative (PID) control is the most commonly used control algorithm due to its effectiveness in many operation conditions, its simplicity, and ease of implementation. It acts on the error between the set-point and the controlled variable, for this reason, it does not require measurements of the internal state and, as a consequence, few sensors are needed and therefore lower costs. Moreover, this kind of controller does not need an elevated knowledge of the plant and it is efficient against common uncertainties around an operating region. Finally, PID is very easy to implement in continuous-time or digital-time form into microcomputers, because is characterized by a single equation. The main disadvantage of this category of controllers is the high sensitivity to variable conditions of operation like changes of transported mass or resistance forces. Further analysis of this regulator is conducted in Chapter 5.

The term Model Predictive Control does not refer to a specialised control strategy, but rather to a wide range of advanced control methodologies that make explicit use of a model, which describes the observed process, to obtain a control through the minimisation of an objective function. Compared to standard PID, MPC is a very effective and advanced control strategy, that has a significant impact on industrial process control [22]-

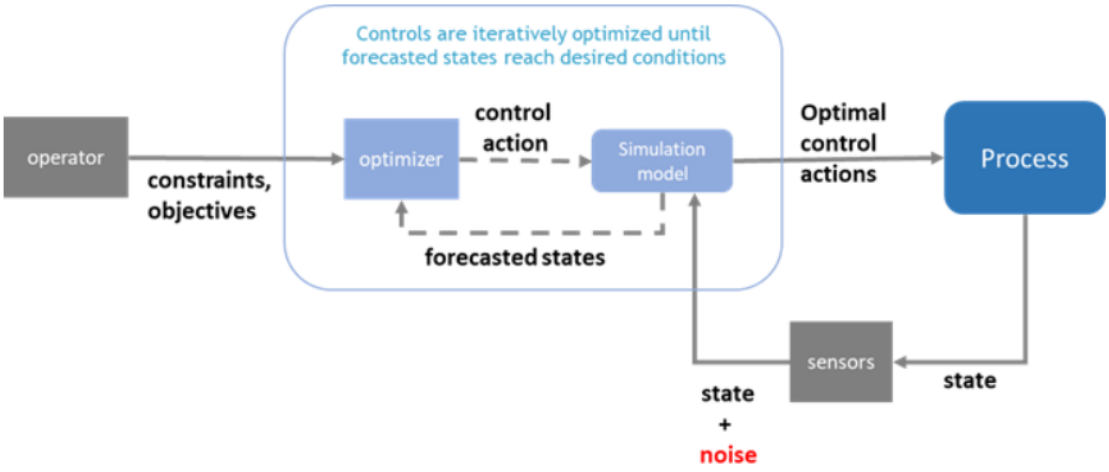


Figure 2.5: MPC block diagram [11].

[18]. The model is used to predict the future output based on historical information about the process, as well as anticipated future input. The aim of MPC is to compute an optimal control input that, while ensuring the satisfaction of given system constraints, minimizes a priori defined cost function. It guarantees adequate control in presence of object constraints and delays in the control system, plus it is able to respond to set point changes without the need for the occurrence of control error resulting from actual measurement, only based on the predicted output values with the possibility of correct operation of the control system in case of temporary unavailability of the measured quantities. Finally, it can use velocity information in the future, based on a predefined velocity profile at each time instant. The main problems with MPC are the high computational load and the elevated algorithmic complexity in addition to the large number of control parameters required.

2.3. Robot navigation

Robot navigation is the robot's ability to determine its own position in its frame of reference and then plan a path toward some goal location. As depicted in Figure 2.6 localization is a fundamental part of robot navigation since it provides the robot's current pose that is then used by the path planning and navigation algorithms [12].

Robot localization is the process of determining the position and orientation of a mobile robot situated in an environment. Localization is one of the most fundamental capabilities required by an autonomous robot since it is crucial to determine the robot's future actions. In a typical scenario, the map of the environment is available and thanks

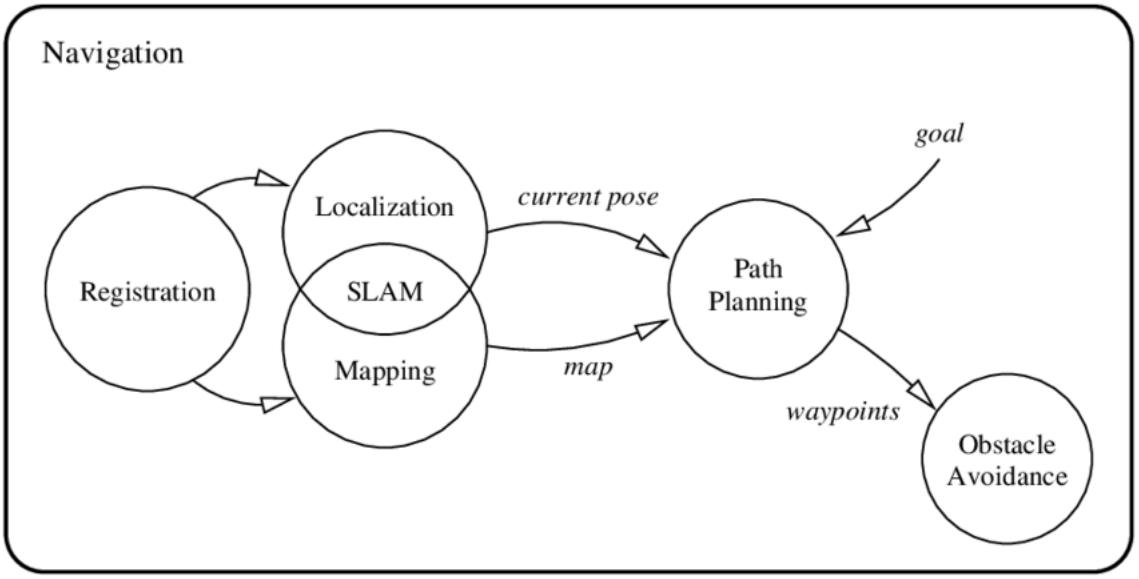


Figure 2.6: Navigation block diagram [1].

to the adoption of internal and external sensors the robot observe the environment and monitors its own behaviour.

In order to solve the navigation problem, ROS, a middleware that is better presented in Chapter 3, relies on move_base. It employs hierarchical planners consisting of a global planner, which calculates the optimal path using search-based approaches, and a local planner, which executes it considering local observations and unknown obstacles.

2.3.1. Global planner

The global planner uses a priori information from the environment to create the best possible path between point A on the map and point B. Usually, the path planning problem is addressed in the configuration space (C-space), in which the platforms are represented as a mobile point at each time instant. Obstacles are also mapped from the workspace to the C-space. The subset of the C-space that describe the collision-free zone is called C_{free}. If a path completely belongs to C_{free}, then is called a free path. The two main categories of path planning algorithms are search-based planning and sampling-based planning. Search-based algorithms are characterized by the:

- Generation of a systematic discrete graph of C_{free}
- Guarantee of finding a path if it exists (i.e. resolution complete)
- Addition of nodes when necessary

- Possible high computational cost when problem dimension increases

While Sampling-based algorithms differ according to the following characteristic:

- The generation of a sparse sample-based graph of C_{free}
- The guarantee that the probability of finding a path if it exists approaches 1 as the number of iterations tends to infinity (i.e. probabilistically complete)
- The addition of samples only when necessary
- The reduced memory usage and adaptability for high-dimensional planning

The move_base global planner is typically one of these three different possibilities. The first one is carrot_planner, it is the simplest global planner, which makes the robot get as close to a user-specified goal point as possible. The planner checks whether the user-specified goal is an obstacle. If it is, then it moves back along the vector between the robot and the goal. Otherwise, it passes the goal point as a plan to the local planner or controller (internally). A possible alternative is navfn which uses Dijkstra's algorithm to determine the global path.

Finally, the most used one is global_planner which, in addition to Dijkstra's algorithm, supports the A* algorithm, toggles quadratic approximation, and toggles grid path.

2.3.2. Intermediate planner

As described in [16], many robotic navigation systems, employ hierarchical planners consisting of a global planner, using search-based approaches e.g. A-star or Random Rapid Tree(RRT) search, and a local planner, which executes the task considering local sensor information and unknown obstacles. Current systems can operate in static environments, while dynamic ones are still challenging. To achieve efficient navigation in highly dynamic environments, Deep Reinforcement Learning (DRL) has gained importance as a planning method thanks to its ability to navigate effectively using raw sensor input. The main disadvantages of planning through DRL are its incapability to recover from a complex situation, e.g. corners and corridors, since it is affected by local minima; and for the difficult training required.

To generate waypoints in a more dynamic way and provides a shorter goal horizon two algorithms are introduced: Spacial Horizon (SH) and Landmark-based Waypoint Generator (LM-WP).

Landmark-based Waypoint Generator (LM-WP) main aim is to smooth and improve the efficiency of the path created with the global planner. To achieve these results, LM-

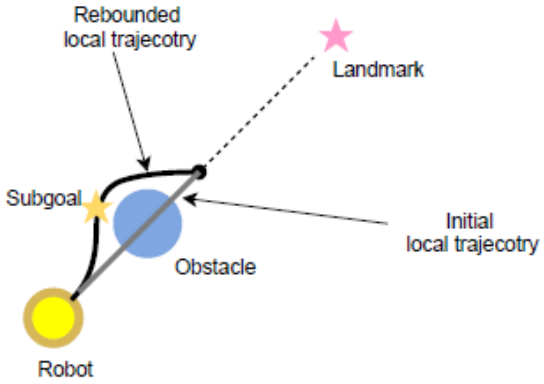


Figure 2.7: Waypoint calculation with LM-WP algorithm

Algorithm 2.1 Landmark selection pseudocode algorithm

Input **:** Start position p_s, goal position p_g, occupancy grid-map
Output **:** queue of landmarks L
Parameter: ψ_{tresh};
Global path $\pi_g \leftarrow$ kinodynamicAstar(p_s,p_g);
Parameterize global path π_g as uniform b-spline path π_g^*;
for t from t_q to t_{M-q} **do**
    Evaluate position p_t, velocity v_t, acceleration a_t on π_g^* at time t using De Boor's algorithm;
    Calculate steering angle velocity ω_t and steering angle ψ_t;
    if $\psi_t > \psi_{tresh}$ **then**
        L.pushback(p_t);
    end if
end for
L.pushback(p_g);

WP is based on the assumption that just a subset of points of the global path must be reached by the robot, these are denoted as landmarks. To navigate on the map, the landmarks correspond to turning points, which can be described as points where the robot's steering angle is greater than a specified threshold. Once the landmarks are selected, a path is calculated using b-spline interpolation based on the kinodynamic A-star search global path planner method. From the parameterized path are then evaluated the position, velocity and acceleration. The landmark selection is shown in Algorithm 2.1. Subsequently, the waypoints are found taking into account the robot's current pose, the following landmark position, and the obstacles' locations in the occupancy grid-map. To ensure a path in C_{free}, the b-spline path is optimised using the ESDF-free gradient-based method. After the local path optimization and using De Boor's algorithm [21] is possible to find the sub-goal from the b-spline path.

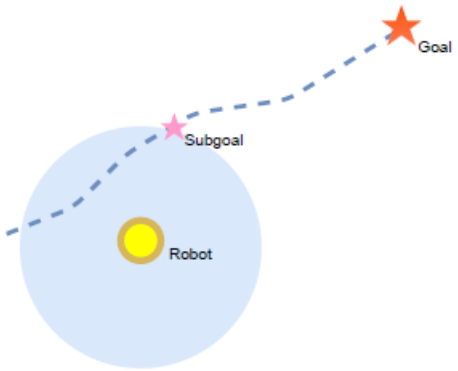


Figure 2.8: Spatial Horizon example

Algorithm 2.2 Spatial horizon pseudocode algorithm

Input **:** Global path π_g, Robot position p_r
Output **:** Subgoal g_sub, Robot position p_r
Parameter d_{ahead};
Find the set of intersection points of R(p_r,d_{ahead}) and global path π_g as Φ;
if Φ *is not empty* **then**
    select the intersection point near to the global goal as g_{sub}
else
    call GLOBAL REPLAN $\rightarrow$ new global path π_g^{new};
    Subgoal calculation (π_g^{new},p_r);
end if

Spatial Horizon (SH) provides, not only a shorter goal horizon but also an additional security layer in preparation for the local planner that only has access to local sensor data. More precisely, starting from the global path, SH defines a subset of local goals based on the robot's position and on a pre-defined look-ahead distance named spatial horizon as depicted in Figure 2.8. These are considered local sub-goals and are given as input to the local planner. Finally, SH possesses a way to recompile the global path when the robot is too far away from it or after a certain time interval with no movements of the robot. A sketch of the algorithm is shown in Algorithm 2.2.

2.3.3. Obstacle avoidance

Obstacle avoidance is a key phase during navigation, it consists in preventing collisions with obstacles using information from the static map and onboard sensors. The development of collision avoidance algorithms has encountered a great improvement in the last decades, from the simpler ones that, when detecting an obstacle, stop the robot to recalculate a possible path; to more advanced ones that steer the robot to avoid the

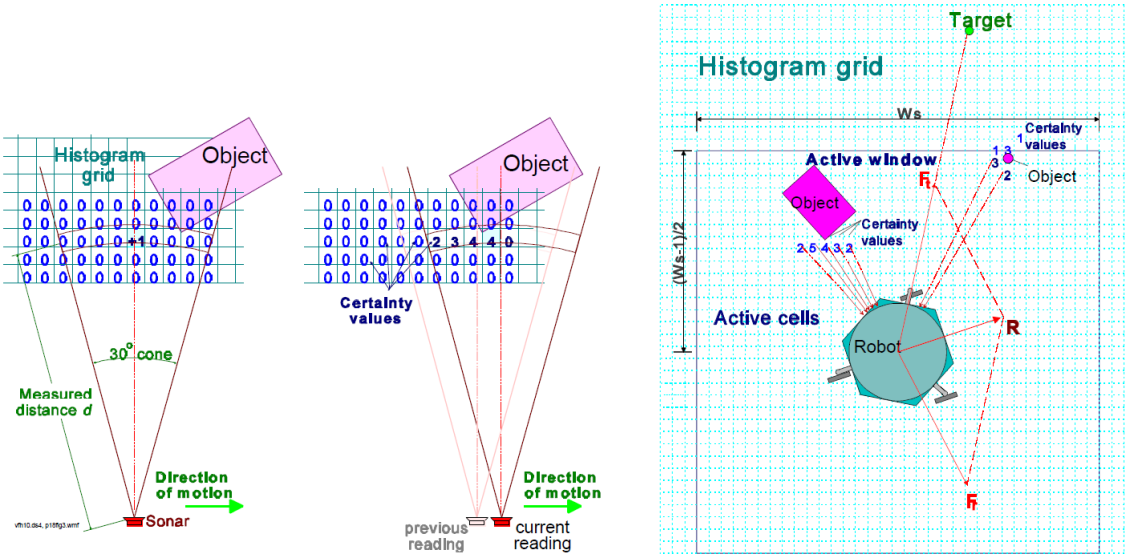
collision. The increase in the complexity of algorithms results in necessary additional measurements of the occupied dimensions of the obstacles. Obstacle avoidance also called local path planning, does not guarantee, differently from global path planning, an optimal solution due to the lack of a priori information about the environment.

In literature is possible to find many relevant obstacle avoidance methods, we focus on the potential field methods. In this class of methods, obstacles produce repulsive imaginary forces, while the goal position applies an attractive force to the robot. The resultant is a force vector, obtained as the sum of all the forces, calculated at each time instant.

One of the first real-time obstacle avoidance methods is Virtual Force Field (VFF), as presented in [3], which allows fast, continuous, and smooth motion of the controlled vehicle among unexpected obstacles without requiring it to stop the vehicle. It is composed by two main phases:

- To represent the obstacles, it is used a two-dimensional Cartesian histogram grid, in which each cell holds a value representing the probability of the presence of an obstacle in that specific location. This method creates a likelihood distribution modifying the value on only one cell in the histogram for each sensor reading. The data are gathered continuously and rapidly to calculate a precise probability (i.e. likelihood) distribution. Iterating the process, the cells in the neighbour of the obstacle's actual position have high certainty values.
- To implement the potential field to the histogram grid it is needed to define $activecells$ as the cell inside the sensor's range. Each active cell applies a repulsive force in the direction of the robot, whose intensity is inversely proportional to the distance between the cell and the centre of the robot and directly proportional to the certainty value of the cell. In addition, a constant virtual attractive force is applied to the chassis toward the target. The sum of all the repulsive and the attractive forces results in the force vector.

By combining the first two phases of this method the robot is able to react rapidly in front of unknown obstacles. One of the problems with the VFF method is encountered when the robot has to pass between two obstacles, like a doorway for example, because the two sides generate a force that pushes the robot away. Moreover, the histogram grid is built in a discrete way, this can create a problem when the robot changes its position from one cell to another because there is a variation of a unit in the distance between the robot and the obstacle, determining a huge variation in the resultant force. A low-pass filter able to smooth the control signal can reduce this effect but, on the other hand, it creates



(a) Histogram creation (b) Potential Field idea

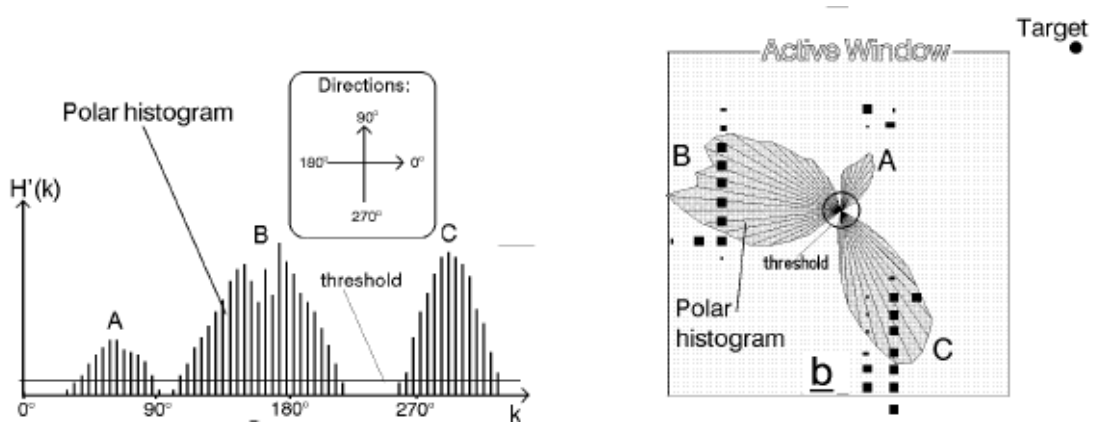
Figure 2.9: VFH phases

a delay that impacts the robot's reaction to an unexpected obstacle. However, the main issue with VFF is the huge data reduction performed, the large amount of information provided by sensors is reduced, in one step, to the direction and intensity of the resultant force.

Due to the problems mentioned above, another method called Vector Field Histogram (VFH) has been implemented. It uses a two stages data reduction resulting in three levels of data representation. The three-level of data are described below.

- In the highest level the two-dimensional Cartesian histogram is continuously updated in real-time, similar to VFF first phase, in addition, it also holds a detailed representation of the environment.
- In the middle level, at every time instant the robot's position is used to generate a one-dimensional polar histogram as depicted in 2.10a. The results are a predetermined number of sectors, with the same width, each one with a value that represents the polar obstacle density in the direction corresponding to the specific sector.
- In the lowest level the reference values for the drive and steer controllers of the vehicle are produced.

In conclusion, it is possible to summarize the behaviour of a VFH-controlled mobile robot as the response of the platforms depending on the likelihood of the presence of an



(a) VFH obstacle density histogram (b) Polar histogram in the histogram grid

Figure 2.10: Results of VFH information reduction

a priori unknown object. Two histograms are then created: the histogram grid, in which the probability is expressed as a certainty value; and a polar histogram, which transforms the previous information into the width and height of the column of the graph. Vector Field Histogram is appropriate in situations where inaccurate sensor data are present, e.g. sensor fusion.

3 | Experimental set-up

This Chapter aims to describe the setup of the platforms used. First, it outlines the prototype robots' main hardware features and then describes the software used in the project during the phase of theoretical simulation and the physical experimental tests.

3.1. Hardware

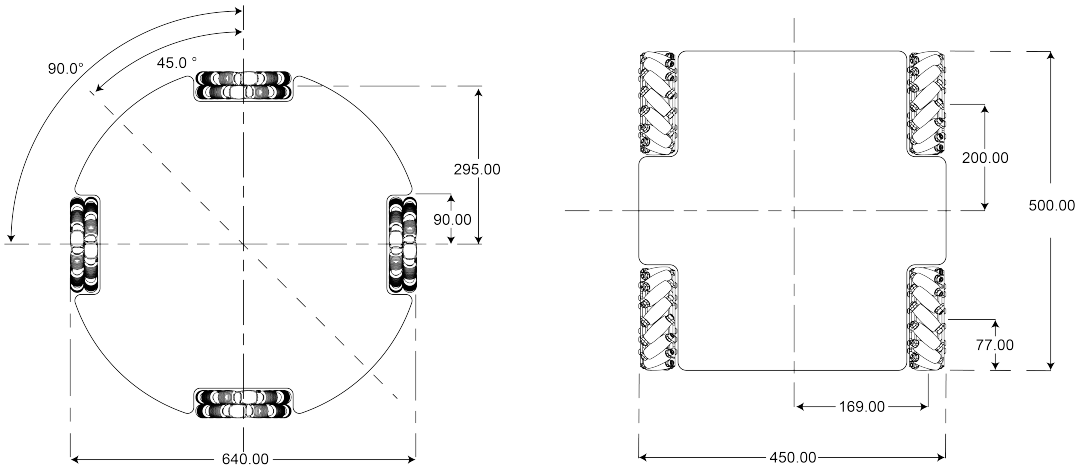
The hardware Section presents a description of the two platforms' dimensional features, the actuators, the electronic components and the used sensor (internal and external).

3.1.1. Mobile platforms

The mobile platforms adopted in this work are two omnidirectional holonomic robots, i.e. the degrees of freedom of the platform are equal to the controllable degrees of freedom. As introduced in Chapter 2 the omnidirectional robots allow simultaneous and independent rotational and translational displacements granted by four electric motors, one for each wheel.

The limit velocity of the platforms is set to 1.2 m/s, e.g. velocity of a walking person, as it is a reasonably safe value for an indoor environment. The robots differ in two main aspects: the type of wheel and their spatial arrangement; the latter consequently influence the shape of the chassis.

The first robot, as shown in Figure 3.1b, presents four Omni wheels with their axes inclined 45 degrees with respect to the chosen forward direction and equally spaced every 90 degrees; while the second robot (Figure 3.1a) adopts four Mecanum wheels mounted parallel to the forward chassis axes.



(a) Omni wheeled robot.

(b) Mecanum wheeled robot.

Figure 3.1: Platforms' dimensions and configurations.

Robot	M	r	d_x	d_y	L	l
Omni wheeled robot	30	90	295	295	640	640
Mecanum wheeled robot	30	77	200	169	500	450

Table 3.1: Robots weights and dimensions

3.1.2. Actuator

The four electric motors mounted on both platforms are TORQUE BOARD 6355-190KV. The motors have a shaft diameter of 8 mm and a usable shaft length of 32 mm. They are equipped with a 200 mm Silicone 12AWG wire with 5.5 mm Male Gold Bullet Connectors. These motors are equipped with CNC vent holes to improve efficiency and limit overheating due to inefficiencies [31] and use high-temperature neodymium magnets to allow higher overheating limits. Moreover, all bearings are replaced with rubber-sealed bearings to extend the life of the motors, since they prevent lubricants from seeping out and external contaminants from damaging the bearing.

The technical specifications are reported in Table 3.2. Of those parameters, motor resistance and inductance are calculated using the Vesc®Tool software.

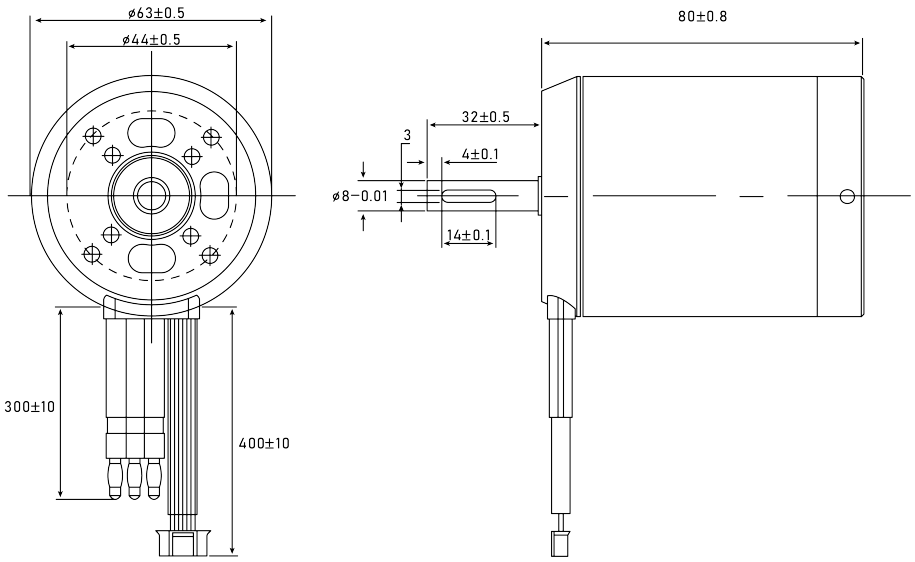


Figure 3.2: TORQUE BOARD 6355-190KV dimensions.

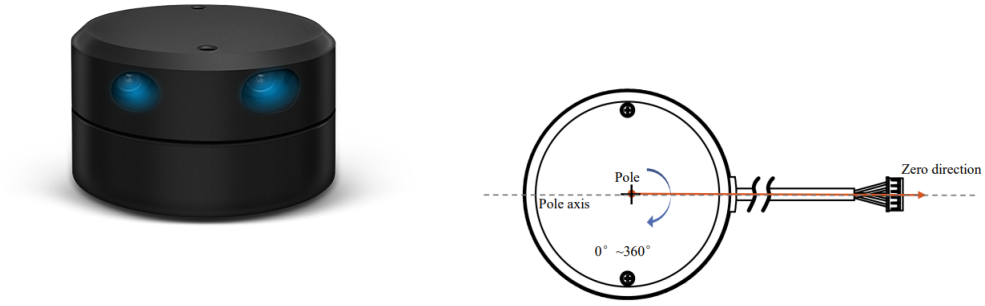
Max Power	2500 W
Max current	80 A
Nominal voltage	44.4 V
Max Torque	2.83 Nm
Motor resistance	14.4 $m\Omega$
Motor inductance	8.75 μH
Motor poles	14
Gear ratio	5
Weight	639,5652 g

Table 3.2: TORQUE BOARD 6355-190KV technical specification

3.1.3. Laser Scanner

Each robot is equipped with multiple laser scanners YDLIDAR G4, shown in Figure 3.3a. YDLIDAR G4 is a 360 degrees 2D Lidar based on the triangulation principle. It uses an infrared laser and optical lens to transmit and receive the laser signal.

The structure rotates 360 degrees to continuously output the angle information and the data of the environment. These sensors can be used singularly if the robot setup does not limit the visual of the sensor creating a shadow cone, or if multiple laser sensors have to be installed, a merging and synchronization of data are recommended to have a better reconstruction of the environment.



(a) YDLIDAR G4. (b) Lidar top view.

Figure 3.3: Lidar scanner.

Ranging frequency	9000 Hz
Motor frequency	7 Hz
Ranging distance	0.28-16 m
Field of view	0-360 deg
Angle resolution	0.28 (frequency @7Hz) deg
Baud rate	230400 bps

Table 3.3: YDLIDAR G4 technical specification

YDLIDAR G4 internally defines a polar coordinate system. Its reference frame takes the centre of the rotating core of the sensor as the pole and the angle is positive clockwise. The zero angle is located in the direction of the outlet of the plug interface line as shown in Figure 3.3b.

In Table 3.3 are reported the technical specifications considering the typical values; all information is gathered from the data-sheet of the sensor [14].

3.1.4. OptiTrack™

To gather information about the position of the robots inside the laboratory arena we use the OptiTrack™system. OptiTrack™includes motion capture software and high-speed tracking cameras. Multiple synchronized cameras are installed around the target area, each of them capturing 2D images. From the comparison and the overlapping of the positions derived by the cameras is possible to compute the 3D positions via triangulation. Optical motion capture systems obtain the data by detecting emitted or reflected light, indeed cameras track surfaces covered with retro-reflective material. The IR light emitted

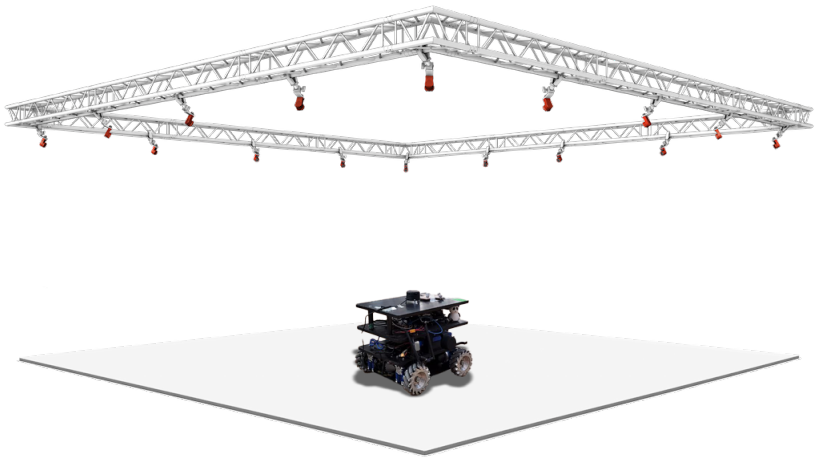


Figure 3.4: OptiTrack setup.

from the camera is reflected and detected by the camera's sensor. The captured reflections are used to calculate the 2D marker position. Rigid body movement can be tracked when at least three markers are visible [6].

To track the mobile platforms we use an arrow-shaped surface with five circular passive markers fixed on top. The configuration is labelled and saved on a custom project in Motive,i.e. the software part of OptiTrack™. The cameras are able to locate markers with millimetre precision when the light noise is reduced to the minimum, the 3D and 2D poses are then broadcast at a maximum of $350Hz$, but our set-up publishing frequency is set to $120Hz$.

The main parameter that can be configured from Motive are LED illumination, which sets the brightness of the cameras; Exposure, which controls how long the shutter remains open; Threshold, which limits the minimum brightness for a pixel to be considered; FPS, which determine the number of images a camera can catch per second; and Gain, which can improve the tracking at very long distances.

3.1.5. On board computer

The calculations are performed by a NUC Intel mounted on both robots. The computer is screwed on the top layer of the platforms and wired to the other components.

Processor	Inter(R) Core tm i7-1165G7
Graphic card	Intel (R) Xe Graphics (TGL GT2)

Table 3.4: NUC Intel main technical specification

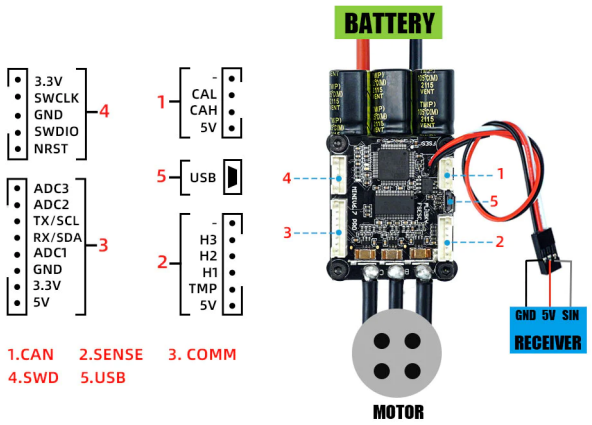


Figure 3.5: FLIPSKY Mini FSESC6.7 PRO 70A.

Continuous current	70 A
Voltage	14 V- 60 V
Control Interface Ports	USB,CAN,UART, SPI, IIC
Motor wire	12AWG
Power cable	12AWG
ERPM	150000
Size	67x39x18.7 mm (Including heatsink)
Weight	130 g

Table 3.5: FLIPSKY Mini FSESC6.7 PRO 70A technical specification

3.1.6. Control boards

The four electric motors are actuated and controlled by the FLIPSKY Mini FSESC6.7 PRO 70A. The ESC (Electronic Speed Controller) is based on a open source project called VESC. The Vesc project

The FLIPSKY Mini FSESC6.7 PRO 70A support four control modes: current, duty cycle, speed, and position control mode [10]. The Vesc project allows the user to tune the current and speed control of the motors. The current controller implemented is a Field Oriented Control (FOC) that guarantees higher top speed and higher efficiency by pushing the motor to achieve maximum torque at a given speed.

Figure 3.6: ROS icon.

3.2. Software

This Section illustrates the software used in this thesis work. It includes Robotic Operating System (ROS) as middle-ware and Python as the programming language of the algorithms. To create and simulate the robots' models we use Matlab® and Simulink®.

3.2.1. ROS: Robotic Operating System

ROS is defined as an open-source, meta-operating system that provides services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management [23].

Simplifying the definition is possible to state that ROS is a framework that manages the communication between distributed processes, and enables the executable to be designed independently. This characteristic benefits the overall system by decreasing code complexity while increasing fault tolerance.

ROS supports TCP/IP-based and UDP-based protocols. The small processes that perform computations are called *Nodes*, which can be grouped in *Packages* and *Stacks*. The inputs/outputs are called *Messages* that are written on specified channels, called *Topics*. Topics have anonymous publish/subscribe semantics to decouple the production of information from its consumption [24]. A node can publish a message on topics and/or read a message from topics. Another entity largely used is *Services* which are similar to functions that can be called inside the nodes to perform different tasks. Services are based on request/reply communication which is often required in a distributed system.

To easily represent the state of the robots and their perception of the environment we use Rviz. Rviz (short for "ROS visualization") is a 3D visualization package. It uses sensor data, e.g. laser scan data, to recreate an accurate depiction of the robot's perception of the world (real or simulated). Rviz allows the representation of different reference frames and the footprint of the robot, i.e. the projection of the occupancy area of the robot on the ground plane. It also shows the environment map and relative occupancy grid: a 2D grid map, in which each cell indicates the likelihood the cell contains an obstacle. Figure 3.7 shows the AIRLab arena mapped using the laser sensor data and Rviz.

Figure 3.7: AIRLab arena map.

A widely used ROS package that maintains the relationship between multiple reference frames over time is tf. This is typically used to publish the relative pose with respect to any defined coordinate frame. In our specific case, we use a unitary tf to link the map frame to the Optitrack™world frame. We use other two tf: one that corresponds to the marker pose, denoted as "base_link", in the world frame and one that links the laser scan position to the base_link frame. Thanks to this structure we are able to observe the motion of the robot in the map on Rviz.

3.2.2. MATLAB® and Simulink®

MATLAB® (an abbreviation of "MATrix LABoratory") is a programming platform designed to analyze data, develop algorithms, create models, and simulate systems. The core of MATLAB® is a matrix-based language to allow the expression of computational mathematics. Simulink® is a graphical programming language, with a block diagram interface for modelling and simulating dynamical systems. Simulink® offers a tight correlation with MATLAB® and its environment, allowing it to incorporate MATLAB® functions into models and export simulation results to MATLAB for further analysis [34].

All the mathematical relationships and models used in this work are created using MATLAB® and Simulink®. The kinematic models, expressed in detail in Chapter 4, for both robots are developed using the symbolic language of MATLAB®, while the black-box all-inclusive model is created in Simulink®.

To estimate the parameters of the models' transfer function we use a specific Simulink® tool: the Parameter Estimator. Starting from real measured data, saved as time-series,

Figure 3.8: MATLAB® and Simulink® icon.

i.e. a set of data points ordered in time, a model, and a set of targeted tests, the parameter estimator adjusts the transfer functions gains and time constant so that the simulated model matches the output measured one. Further description of the application of the Parameter Estimation Toolbox is carried out in Section 4.2.1.

4 | Models

The following Chapter presents two models: the first is mathematically derived from the robots' motion relationships, whereas the second is estimated from empirically measured data.

Motion planning and control are core components for mobile robot mobility. Both of them require an analysis of the physical behaviour of the robot to be able to develop the best possible plans and controls. In the context of mobile robotics, a platform's physical behaviour is generally characterized by its dynamics and kinematics, [30].

Since the dynamic model of the system is not crucial for the study developed in this thesis, and since the robots' speeds are limited; the system is approximated in a $Black - Box$ model. Given the technical complexity of creating specific tests to estimate the system's dynamic response alone, it is decided to group the entire system: kinematic and dynamic, within the black-box model. This further simplification made it possible to carry out tests aimed at isolating the responses of the three degrees of freedom and to independently define the parameters of the three transfer functions of the model. Starting from the simulation of the overall model, it is then possible to define the initial values of the position controller, which is dealt with in detail in Chapter 5.

4.1. Kinematic model

Kinematic models are a subcategory of mechanical models describing limitations on a robot's motions, these can describe the spatial position of a rigid body, or system of bodies, neglecting the forces involved.

The definition of the kinematic model is a bottom-up process, indeed, every single wheel contributes to the movement of the platform and, simultaneously, imposes constraints that limit the robot's motion. Furthermore, the geometrical configuration links wheels together, determining a constraint on the overall motion of the chassis.

Starting from [4] we develop the kinematic model. The configuration of a mobile platform as a rigid body can be generally described with six variables, three Cartesian

coordinates (x, y, z) and three Euler Angles ($roll$, $pitch$, yaw), related to the inertial external reference frame. If a mobile robot, such as the ones treated in this thesis, moves on a planar surface, this reduces the degree of freedom of the body to two Cartesian coordinates x and y, and one orientation angle θ, i.e. yaw angle.

Considering the configuration of the n wheels of the robot and the chassis pose, the variable vector that define the model can be written as:

$$q = \begin{bmatrix} q_r \\ q_w \end{bmatrix} \tag{4.1}$$

Where:

- q_r represents the position of the robots' centre of mass in the global reference frame defined by the coordinates x and y and the orientation θ.
- $q_w = \begin{bmatrix} \varphi_1 & \dots & \varphi_n \end{bmatrix}$ is the configuration vector of the n wheels.

These are not independent because their time derivatives are related according to a kinematic constraint.

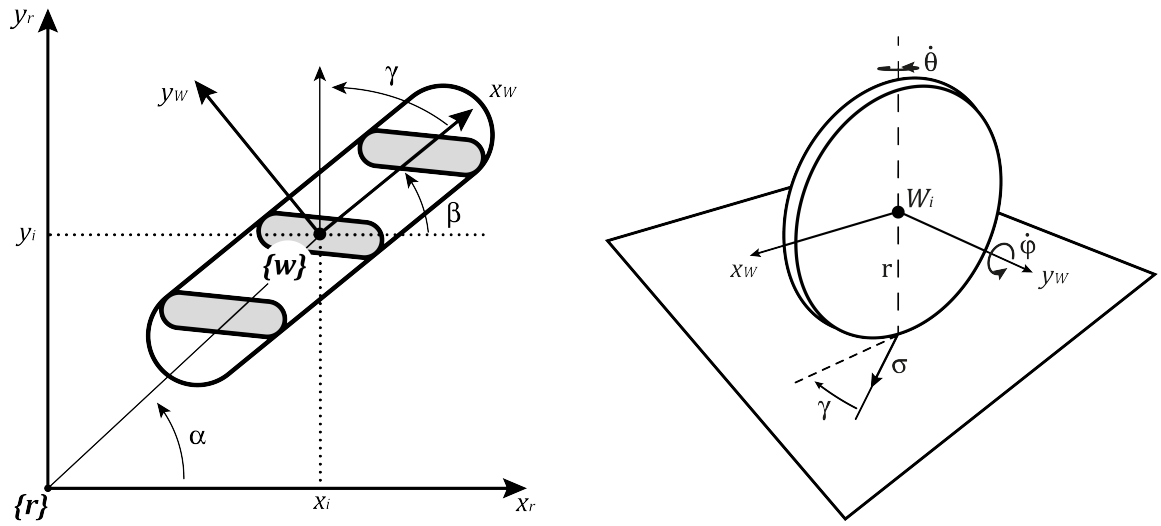
To derive the kinematic model of the robot we start focusing on the point W_i, which is the centre of a wheel. This point can be considered both as a point belonging to the robot chassis or as a point of the wheel. By equating the velocity's in the different reference frames, the relationship between the velocity of the chassis and the angular velocity of the wheels is found.

Fixing the reference frame $\{w\}$ on the contact point between the centre of the wheel and the robot chassis it is possible to relate the velocity of point W_i with respect to the angular velocities of the wheel φ_i and the sliding velocity of the passive roller in contact with the ground σ. The velocity of point W_i is determined based on the linear velocity of the wheel in the contact point and the angular velocity of both the chassis and the wheel.

$$\begin{bmatrix} v_W \end{bmatrix}_{\{w\}} = \begin{bmatrix} \sigma\cos(\gamma) \\ \sigma\sin(\gamma) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\varphi} \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ r \end{bmatrix} = \begin{bmatrix} \sigma\cos(\gamma) + \dot{\varphi}r \\ \sigma\sin(\gamma) \\ 0 \end{bmatrix} \tag{4.2}$$

keeping only the xy component:

$$\begin{bmatrix} v_{xw} \\ v_{yw} \end{bmatrix} = \begin{bmatrix} r & \cos(\gamma) \\ 0 & \sin(\gamma) \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \sigma \end{bmatrix} \tag{4.3}$$



(a) Wheel model. (b) Wheel detail.

Figure 4.1: Wheel representation.

Fixing a reference frame on the robot chassis $\{r\}$ and considering the centre of the wheel belonging to this frame, the velocity of W_i can be deduced from the linear and the angular velocity of the centre of mass of the robot.

The position of each wheel in the robot chassis reference frame is defined by the coordinates x_i and y_i.

$$\left[v_W\right]_{\{r\}} = \begin{bmatrix} v \\ vn \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \\ 0 \end{bmatrix} = \begin{bmatrix} v - \dot{\theta} y_i \\ vn + \dot{\theta} x_i \\ 0 \end{bmatrix} \tag{4.4}$$

$$\begin{bmatrix} v_{xr} \\ v_{yr} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} \tag{4.5}$$

The orientation of reference frame $\{w\}$ with respect to the reference frame $\{r\}$ can be described by the angle β around the z axis. Therefore, considering the corresponding rotation matrix relative to the x and y terms we can write the equation that represent the dependencies between the two frames:

$$\begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \left[v_W\right]_{\{w\}} = \left[v_W\right]_{\{r\}} \tag{4.6}$$

By substituting Equation (4.2) and Equation (4.4) into Equation (4.6), the relationship between wheels' angular velocity and the velocity of the chassis of the robot becomes:

$$\begin{bmatrix} \dot{\varphi} \\ \sigma \end{bmatrix} = \begin{bmatrix} r & \cos(\gamma) \\ 0 & \sin(\gamma) \end{bmatrix}^{-1} \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix}$$

Finally, by rearranging the equation we obtain:

$$\begin{bmatrix} \dot{\varphi}_i \\ \sigma_i \end{bmatrix} = \begin{bmatrix} \frac{\sin(\gamma_i+\beta_i)}{r\sin(\gamma_i)} & \frac{-\cos(\gamma_i+\beta_i)}{r\sin(\gamma_i)} & \frac{-y_i\sin(\gamma_i+\beta_i)-x_i\cos(\gamma_i+\beta_i)}{r\sin(\gamma_i)} \\ \frac{-\sin(\beta_i)}{\sin(\gamma_i)} & \frac{\cos(\beta_i)}{\sin(\gamma_i)} & \frac{y_i\sin(\beta_i)+x_i\cos(\beta_i)}{\sin(\gamma_i)} \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} \tag{4.7}$$

Assuming that there is no wheel slipping on the ground we can neglect the terms related to σ, and, considering a robot with n wheels, the transformation matrix can be rewritten as:

$$\begin{bmatrix} \dot{\varphi}_1 \\ \vdots \\ \dot{\varphi}_n \end{bmatrix} = \begin{bmatrix} \frac{\sin(\gamma_1+\beta_1)}{r\sin(\gamma_1)} & \frac{-\cos(\gamma_1+\beta_1)}{r\sin(\gamma_1)} & \frac{-y_1\sin(\gamma_1+\beta_1)-x_1\cos(\gamma_1+\beta_1)}{r\sin(\gamma_1)} \\ \vdots & \vdots & \vdots \\ \frac{\sin(\gamma_n+\beta_n)}{r\sin(\gamma_n)} & \frac{-\cos(\gamma_n+\beta_n)}{r\sin(\gamma_n)} & \frac{-y_n\sin(\gamma_n+\beta_n)-x_n\cos(\gamma_n+\beta_n)}{r\sin(\gamma_n)} \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} \tag{4.8}$$

Let's define with T_k the transformation matrix in equation (4.8). The rotation matrix that links the global inertial frame to the frame of the robot chassis is a rotation of angle θ around the z axis and will be defined as R_θ.

$$\begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} = R_\theta \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{4.9}$$

where:

$$R_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

In conclusion, the inverse kinematic of a generic $n - wheel$ robot can be rewritten as:

$$\dot{q}_w = T_k R_\theta^\top \dot{q}_r \tag{4.11}$$

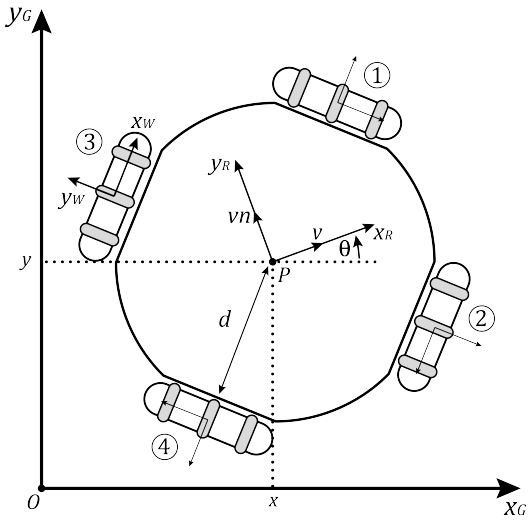


Figure 4.2: Omni wheeled robot

Wheel	$\boldsymbol{\alpha_i}$	$\boldsymbol{\beta_i}$	$\boldsymbol{\gamma_i}$	$\boldsymbol{d}$
1	45°	−45°	90°	d
2	−45°	−135°	90°	d
3	135°	45°	90°	d
4	−135°	135°	90°	d

Table 4.1: Omni wheeled robot parameters.

4.1.1. Omni wheeled robot

The first robot considered is designed to have four Omni wheels equally spaced every 90° in space. The distance between the geometrical centre of the robot to the centre of each wheel is d.

Starting from Equation (4.8) and considering the specific geometric parameters described in Table 4.1, the direct and inverse kinematic models become respectively:

$$\begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} = -\frac{r}{4} \begin{bmatrix} -\sqrt{2} & \sqrt{2} & -\sqrt{2} & \sqrt{2} \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ \frac{1}{d} & \frac{1}{d} & \frac{1}{d} & \frac{1}{d} \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{bmatrix} \tag{4.12}$$

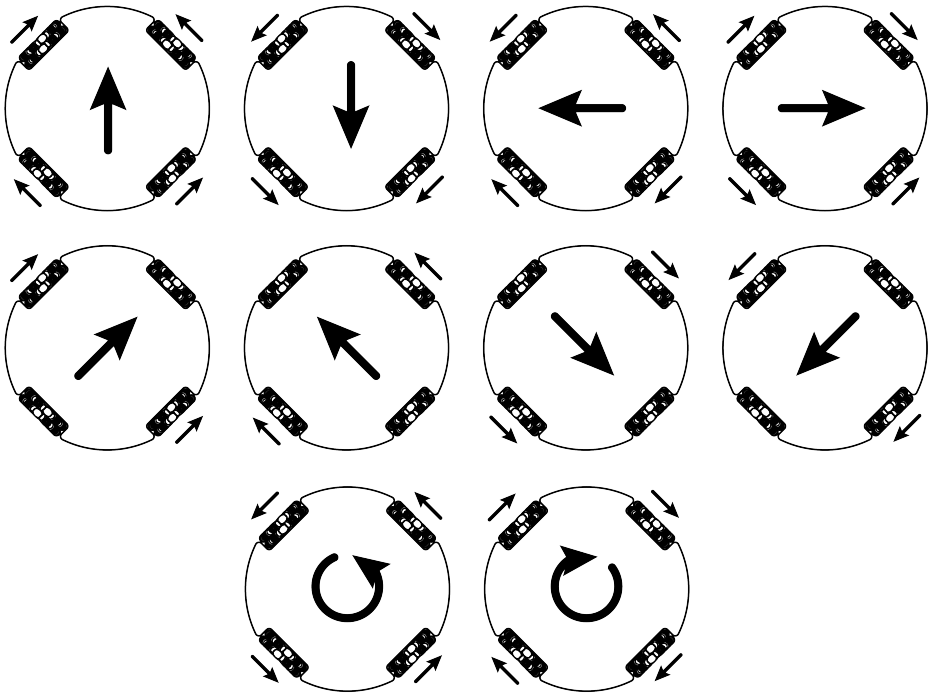


Figure 4.3: Moving direction of Omni wheeled robot

$$\begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{bmatrix} = -\frac{1}{r} \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & d \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & d \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & d \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & d \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} \tag{4.13}$$

To better understand the mathematical relationship, Figure 4.3 shows the rotation direction of the wheels required to perform different manoeuvres. It is important to notice how, in diagonal displacement, two wheels are not actuated. The passive roller mounted on these wheels rotates without any sliding.

4.1.2. Mecanum wheeled robot

The second platform configuration considered has four Mecanum wheels, i.e. with 45° inclined rollers around the wheel's edge. The distances of the wheels' centre with respect to the centre of the chassis can be redefined as:

$$d_x = d * \cos(\alpha) \quad d_y = d * \sin(\alpha) \tag{4.14}$$

The geometrical parameters are reported in Table 4.2.

Wheel	α_i	β_i	γ_i	x_i	y_i
1	$40°$	$0°$	$45°$	d_x	d_y
2	$130°$	$0°$	$-45°$	d_x	$-d_y$
3	$-130°$	$0°$	$-45°$	$-d_x$	d_y
4	$-40°$	$0°$	$45°$	$-d_x$	$-d_y$

Table 4.2: Mecanum robot parameters.

The direct and inverse kinematic models are expressed in Equation (4.15) and (4.16).

$$\begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{d_x+d_y} & \frac{1}{d_x+d_y} & -\frac{1}{d_x+d_y} & \frac{1}{d_x+d_y} \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{bmatrix} \tag{4.15}$$

$$\begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(d_x+d_y) \\ 1 & 1 & d_x+d_y \\ 1 & 1 & -(d_x+d_y) \\ 1 & -1 & d_x+d_y \end{bmatrix} \begin{bmatrix} v \\ vn \\ \dot{\theta} \end{bmatrix} \tag{4.16}$$

Figure 4.5 shows the chassis motion concerning the wheel's linear velocity direction. Unlike the Omni wheeled robot, the passive rollers in diagonal displacement do not roll

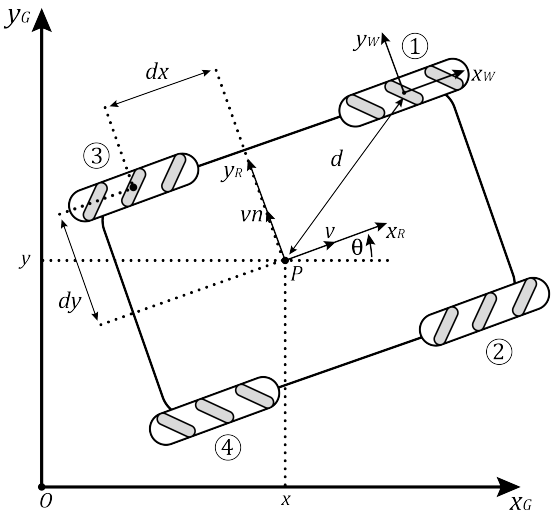


Figure 4.4: Mecanum wheeled robot

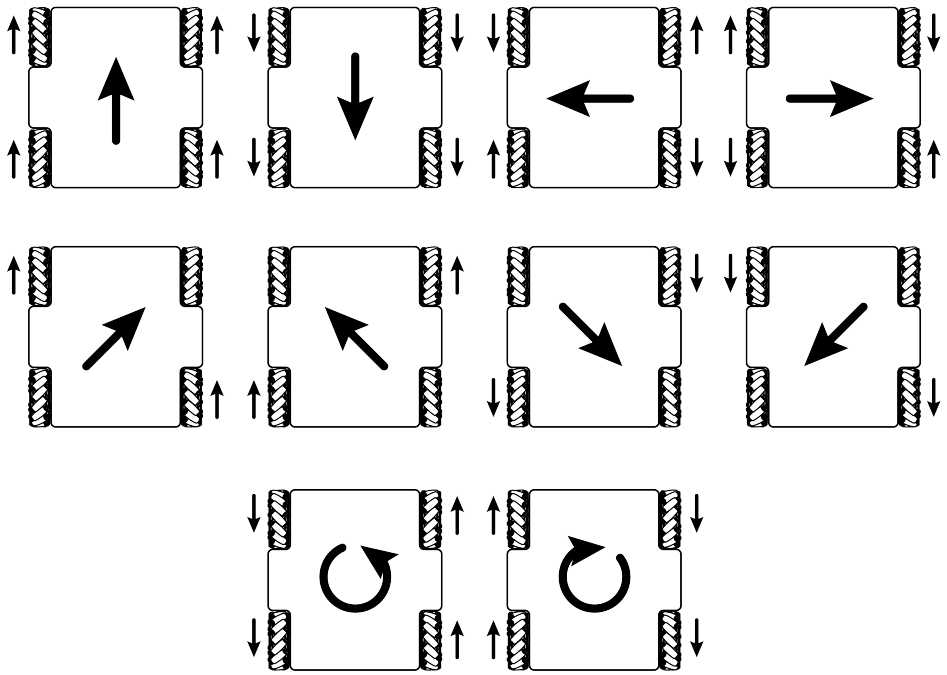


Figure 4.5: Moving direction of Mecanum wheeled robot

but slide, this condition causes an increase in energy consumption and wheel wear.

4.2. Black-box model

Black-box modelling is mostly used when the focus is on fitting the data regardless of the mathematical relationships of the model. It is usually a trial-and-error process, where the parameters of multiple structures are estimated and the results are compared. It is common usage to start from a simple and linear model structure and then progress by adding complexity, boundaries and non-linearity to create more advanced models if necessary [33].

Between different black-box structures, e.g. Linear ARX model, State-space model; we opt for a transfer function representation. Transfer function representation aims to describe relationships between the output of a controllable system and the input signal for all the input values. It is defined as the ratio of the Laplace transform of the output variable to the Laplace transform of the input variable with the assumption of zero initial value.

Having the two platforms' deep similarities in the movement's relationship, the Black-box models developed are similar for both robots.

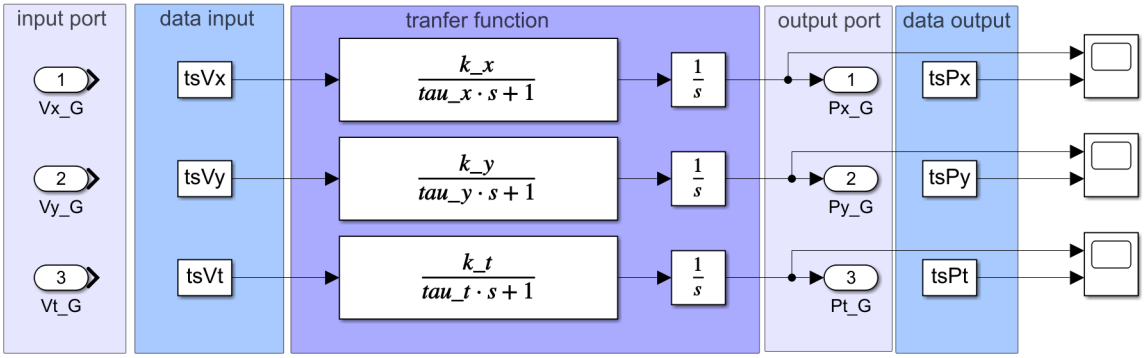


Figure 4.6: Parameter estimator transfer functions

4.2.1. Parameter Estimation

The parameter estimation is carried out using a specific Simulink® tool. The data used to estimate the parameters are collected by parsing the bags, i.e. "a file format in ROS for storing ROS message data" [25], recorded during the experiments. The recorded tests deeply influence the precision and representativeness of the model's parameters.

There are a few main features to supervision during gathering the data: the test duration and the velocity of the operating point. The test we consider is a step response of the output, i.e. the response of a system to make the output change from an initial state to a final one and remain constant. This is because the position control, explained in Chapter 5, receives as input set-points that it has to reach starting from an initial position, this corresponds to a step response for each of the three degrees of freedom (x, y, θ) of the system.

In a step response if the time series recorded is too long the estimation represents better the steady state portion of the response, e.g. the position set-point, while if the test is too short then the model estimated is more faithful to the transient part, e.g. the position variation between the initial state to the final one. The data need to be collected as a trade-off between the two conditions mentioned above to have a good representation of the overall response and obtain a realistic model.

The other main feature we take into consideration is robots' speed. Knowing that the model is non-linear, the behaviour of the system changes with the speed of the chassis and because of that the parameters estimated during a test with a specific speed, might not be accurate enough if used in different velocity conditions. Since our focus is restricted to a small set of possible velocities and we want the system to be precise in the neighbourhood of an operating point we estimated the parameters of a subset of tests and then calculate

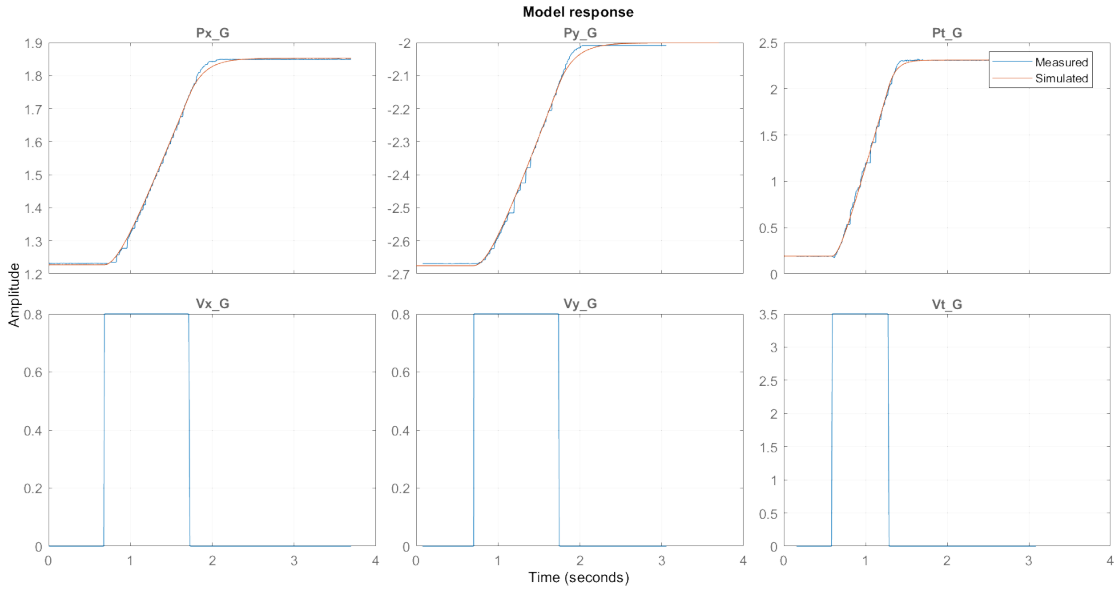


Figure 4.7: Parameter estimator window

the mean to have the final approximation.

As shown in Figure 4.6, to operate, the parameter estimator needs a time series as input for each transfer function, represented in the box "data input", and a time series for each output, named "data output". To collect decoupled data we set different maximum speeds on the joystick and then move the robot remotely along a single axis each time.

In the command window of the parameter estimator, we set the input and the output time series, the initial value as the first data collected in each time series of the output (position of the robot) and which parameter of the transfer function to evaluate. We command the Simulink® tool to estimate three gains, i.e. k_x, k_y, k_t, and three time constants, i.e. tau_x, tau_y, tau_t. The transfer functions link the velocity of the platforms' chassis in the global reference frame to the position of the robots' centre in the same reference frame.

For both robots, we test different linear velocities from 0.6 m/s to 1.2 m/s and angular velocities from 2.0 rad/s to 3.5 rad/s, and for each test, we estimate the parameters.

4.2.2. Omni wheeled robot estimation

The data estimated are reported in Table 4.4. From these values, we calculate the mean to find the final approximation parameters of the simplified black-box model.

Figure 4.7 shows the model response: the first row presents the output, i.e. the

$\mathbf{V}_x$	k_x	τ_x
0.6	0.804	0.177
0.8	0.751	0.188
1.0	0.813	0.216
1.2	0.861	0.248
final	0.807	0.207

$\mathbf{V}_y$	k_y	τ_y
0.6	0.695	0.180
0.8	0.811	0.197
1.0	0.812	0.212
1.2	0.820	0.233
final	0.784	0.206

$\mathbf{V}_\theta$	k_θ	τ_θ
2.0	0.789	0.090
2.5	0.834	0.070
3.14	0.869	0.103
3.5	0.873	0.097
final	0.841	0.090

Table 4.3: Estimated parameters.

position of the robot both measured and simulated, and the second row represents the input, i.e. the velocity. Each column is related to one degree of freedom, respectively x, y, θ.

4.2.3. Mecanum wheeled robot estimation

The Table 4.4 report the parameters estimated from the measured data. The final approximation parameters of the simplified black-box model are found by calculating the mean of each set of data.

The model step response, both measured and simulated, of the system is shown in Figure 4.8 representing the three different degrees of freedom. The difference between the

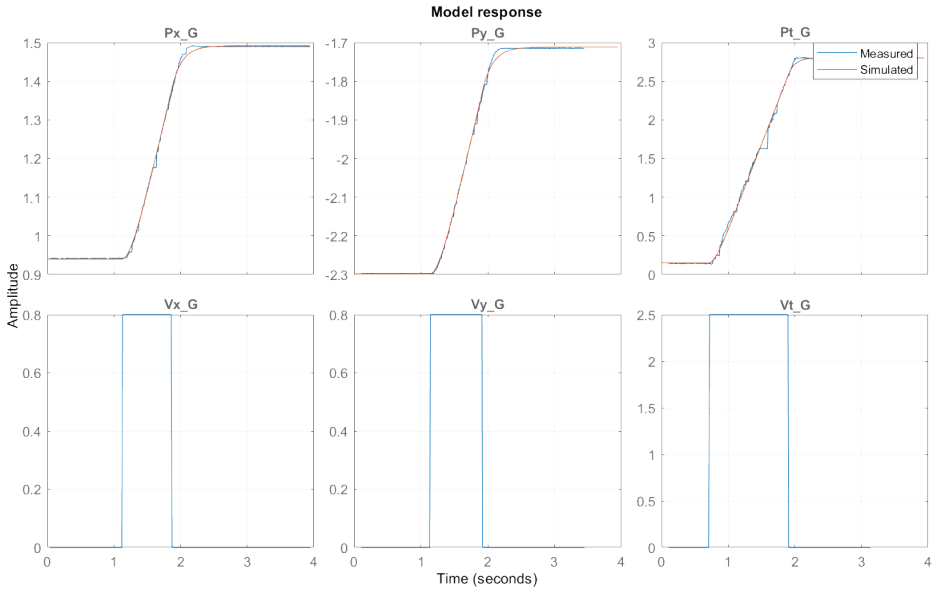


Figure 4.8: Parameter estimator window

$\mathbf{V}_x$	k_x	τ_x
0.6	0.891	0.123
0.8	0.930	0.154
1.0	0.966	0.209
1.2	0.109	0.214
final	0.724	0.175

$\mathbf{V}_y$	k_y	τ_y
0.6	0.903	0.145
0.8	0.940	0.156
1.0	0.971	0.199
1.2	0.102	0.203
final	0.729	0.176

$\mathbf{V}_\theta$	k_θ	τ_θ
2.0	0.887	0.098
2.5	0.892	0.099
3.14	0.934	0.089
3.5	0.950	0.950
final	0.916	0.309

Table 4.4: Estimated parameters.

real and theoretical estimated model is higher at the end of the transient part, this is due to the contribution of dynamic effects that are harder to approximate given a simple model. Nevertheless, it can give a sufficiently accurate starting point for the controller tuning, which is better analysed in Chapter 5.

5 | Position Control

This Chapter presents the first target of our thesis work, which is the control of the position of the robots. It means that, given a final position,i.e. the goal, the platforms are able to reach it while fulfilling some pre-defined specifications.

5.1. Controller and Specifications

Considering the advantages presented in Chapter 2, in particular its ease of implementation and its robustness, we decide to adopt the PID controller. Moreover, we are working with omnidirectional platforms, which means that it is possible to control independently the linear position on the $x-axis$, the linear position on the $y-axis$, and the angular position around the $z-axis$. For this reason, we close three independent control loops with three different PIDs, one for each variable as it is shown in Figure 5.1.

Each controller receives as input the error between the set-point, i.e. the goal in our situation, and the measured actual position of the robot, obtained from OptiTrack™, producing as output the linear and angular velocities in the two-dimensional global reference frame. These are then converted into the robot's reference frame through a rotation matrix around the $z-axis$ and finally, according to each robot's kinematic, transformed into wheel velocities that are actuated by the four motors as specified in Chapter 4.

5.1.1. Control and Environment Specifications

As already mentioned above, during its motion, the robots have to satisfy some specifications. These can be related to the performances or the environment. Settling time, rise time, percent overshoot, and steady-state error values are correlated to the desired platforms' behaviour. They can be defined as:

- Settling time is the time required for the output to settle within a certain percentage which is typically 2% or 5%.
- Rise Time is the amount of time the system takes to go from 10% to 90% of the

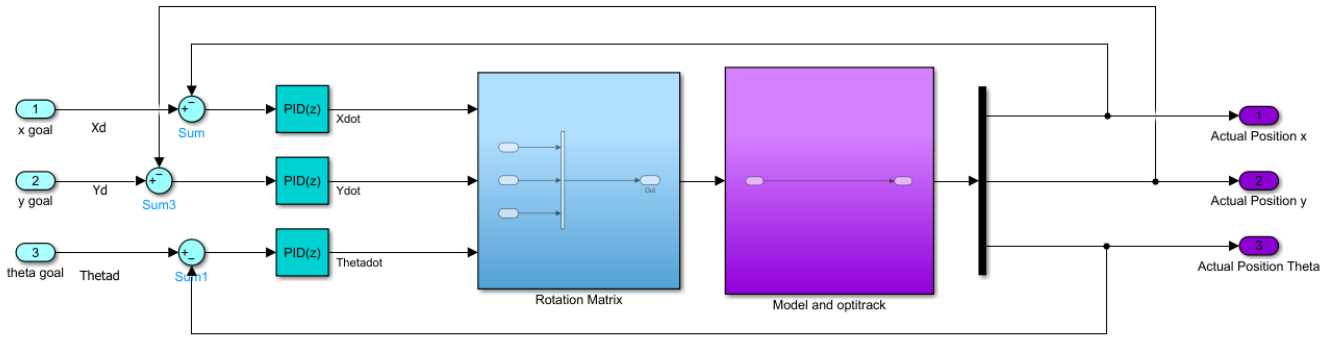


Figure 5.1: Simulink representation of independent PIDs

steady-state, or final value.

- Percent overshoot is the amount that the output exceeds the final value, expressed as a percentage of the final value.
- Steady-state error is the final difference between the output and the desired value.

To better visualize the specification Figure 5.2 shows a step response of a generic system highlighting them.

Since the two platforms are probably going to operate in environments characterized by unknown obstacles, narrow places, and workstations, we decide to give more importance to percent overshoot and to steady-state error trying to reduce both to zero. In this way, the robots reach the goal in a slower but safer way, managing to arrive at the exact desired position without the risk of hitting the workstation.

To avoid an aggressive proportional action, better explained in Section 5.2, that can cause a high peak of current, and therefore limit the lifetime of the actuator we set an error saturation, i.e. a maximum value that the position error can assume. Another reason to proceed at a lower speed is related to the environmental specifications. Indeed, robots work indoors where high velocities are dangerous because of the possible presence of people. For this reason, we also introduce a maximum linear saturation on the output of the PID controller equal to 1.2 m/s and saturation on the maximum angular velocity equal to 1.2 rad/s.

The self-imposed specifications are listed in Table 5.1 .

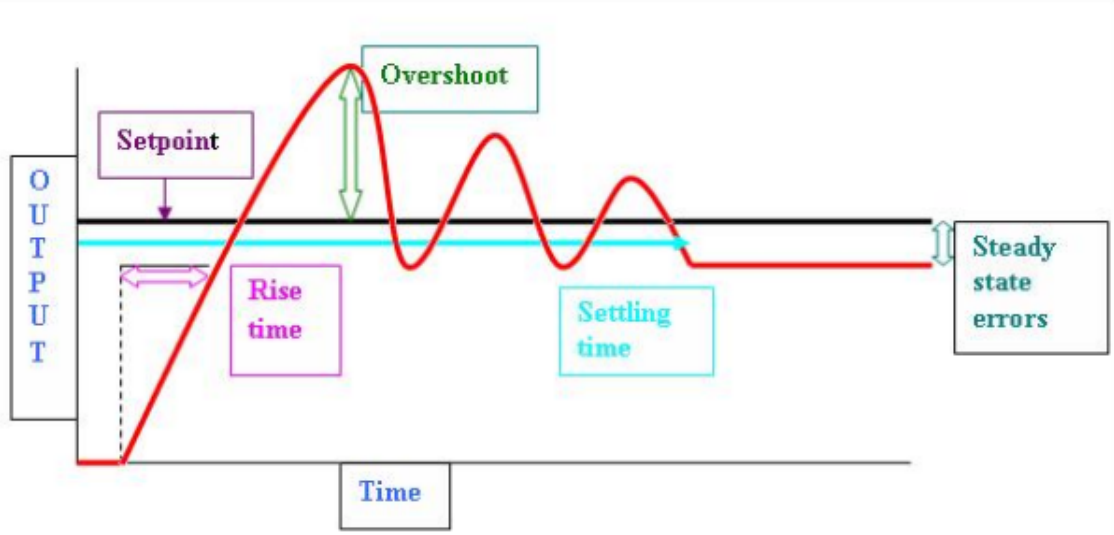


Figure 5.2: Controller specifications

5.1.2. Control Discretization

As stated in Section 3.1.4 the Optitrack™motion sensor publish the pose of the platform at 120 Hz. To implement the regulator on ROS we are bound to a callback, i.e. a function triggered by the subscriber every time a message is sent on a topic. Due to the possibility of the sensor losing information, we decide to associate an independent timer with our controller. We set the timer frequency at 100 Hz, being a reasonably high value without exceeding the sensor's one.

Given a SISO (i.e., Single Input Single Output) system, a typical controller is described by the transfer function:

$$C(s) = \frac{b_o s^{n-1} + \cdots + b_{n-1}}{s^n + a_o s^{n-1} + \cdots + a_{n-1}} \tag{5.1}$$

overshoot	5%
steady-state error	0.04 m
error saturation	0.5 m
output saturation	1.2 m/s

Table 5.1: Control specification

The core idea is to take the continuous time controller $C(s)$ and discretize it into $C(z)$ so that its output is as close as possible to the output $u(t)$ of the continuous time controller for a specific time instant kT. So, the integrators that take as input the continuous output of the controller and give the continuous output of the plant $y(t) = \int_0^t u(\tau)d(\tau)$ are substituted with blocks returning a discrete output $\tilde{y}(k) = y(Tk) = y(k)$.

There are three possible approaches to carry out this task: Forward Euler discretization, Backward Euler discretization, and Tustin discretization. We decide to use the last one. The main idea is to sample two consecutive instants: kT and $(k+1)T$, as presented in [28], obtaining the integral:

$$y((k+1)T) = \int_0^{(k+1)T} u(\tau)d(\tau) \tag{5.2}$$

This can be split in two obtaining:

$$y((k+1)T) = \int_0^{(k+1)T} u(\tau)d(\tau) = \int_0^{kT} u(\tau)d(\tau) + \int_{kT}^{(k+1)T} u(\tau)d(\tau) \tag{5.3}$$

The first integral corresponds to $y(kT) = y(k)$, the continuous signal $y(t)$ sampled with period T. The second integral, instead, is the area of $u(t)$ between $u(kT)$ and $u((k+1)T)$ which can be approximated as the coloured area in Figure 5.3. Tustin's approach linearly connects the two consecutive samples, so, we rewrite equation (5.2) as:

$$\tilde{y}(k+1) = \tilde{y}(k) + h\frac{u(k+u(k+1)}{2} \tag{5.4}$$

we apply Z-transform and obtain:

$$\tilde{Y}(z) = \frac{h}{2}\frac{1+z^{-1}}{1-z^{-1}}U(z) \tag{5.5}$$

and the change of variables is the following one:

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{5.6}$$

We select Tustin's method because is the only one able to map the pole of the unitary circle as depicted in Figure 5.4

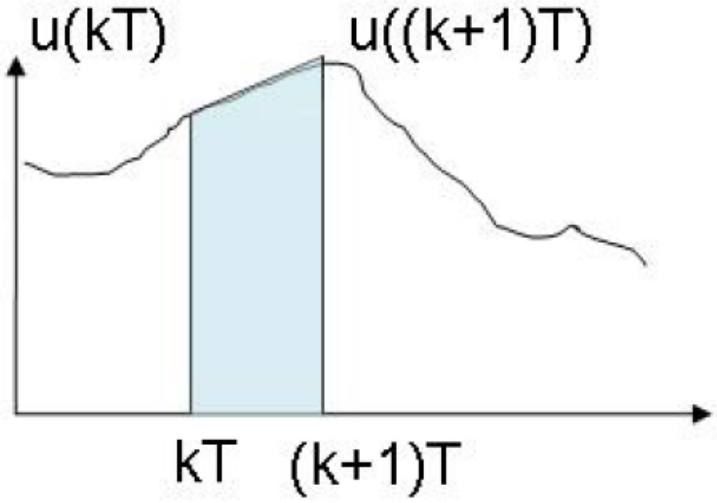


Figure 5.3: Tustin discretization

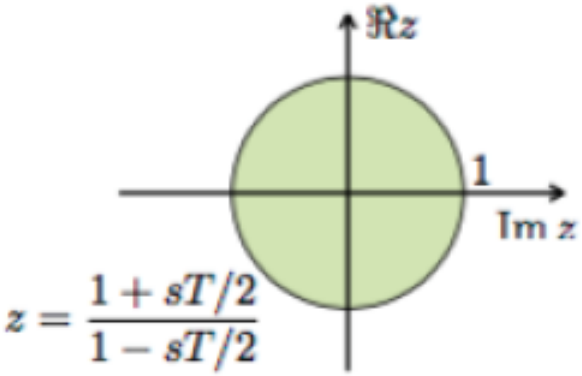


Figure 5.4: Pole mapping with Tustin method

5.2. Controller Implementation

The output of the PID controller in the time domain is:

$$u(t) = K_p e(t) + K_i \int_0^t u(\tau)d(\tau) + K_d \frac{de}{dt} \tag{5.7}$$

where $e(t)$ is the tracking error between the set-point and the measured output, while K_p, K_i, K_d are respectively the proportional, integral, and derivative gain which are the parameters to be tuned. The simulink representation of a basic continuous time PID controller is represented in Figure 5.5. Evaluating the Laplace transform of the previous

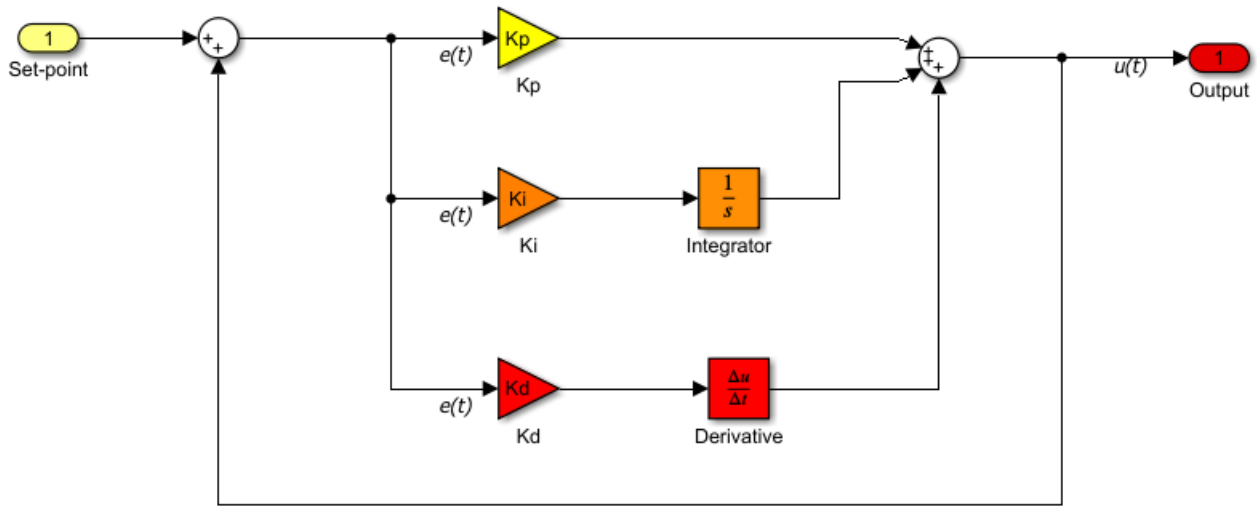


Figure 5.5: Basic PID simulink representation

equation we obtain the PID transfer function as:

$$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{2} \tag{5.8}$$

5.2.1. Parameters Description

The proportional term $K_p \cdot e(t)$ generates an output that is directly proportional to the error so that the greater the error, the greater the output. This means that a high proportional gain implies an elevated change in the output as a consequence of a large input error determining a quicker reaction but also a greater overshoot and can cause instability and oscillations. On the other hand, if K_p is small, the controller is less sensitive and responsive. Finally, the proportional term is able to reduce the steady-state error but not completely eliminate it.

The integral term $K_i \cdot \int_0^t u(\tau)d(\tau)$ depends on the cumulative error and defines the offset that must be corrected. For this reason, it is able to eliminate the steady-state error but, it can cause a huge increment of the overshoot. The main problem with the integral term is the phenomenon called windup. If the initial position error is so high that the actuator saturates, the integral term continuously increases reaching its maximum value when the error is zero. From this point on, the output can not leave the saturation condition until the error remains negative for a sufficiently long time allowing the integral

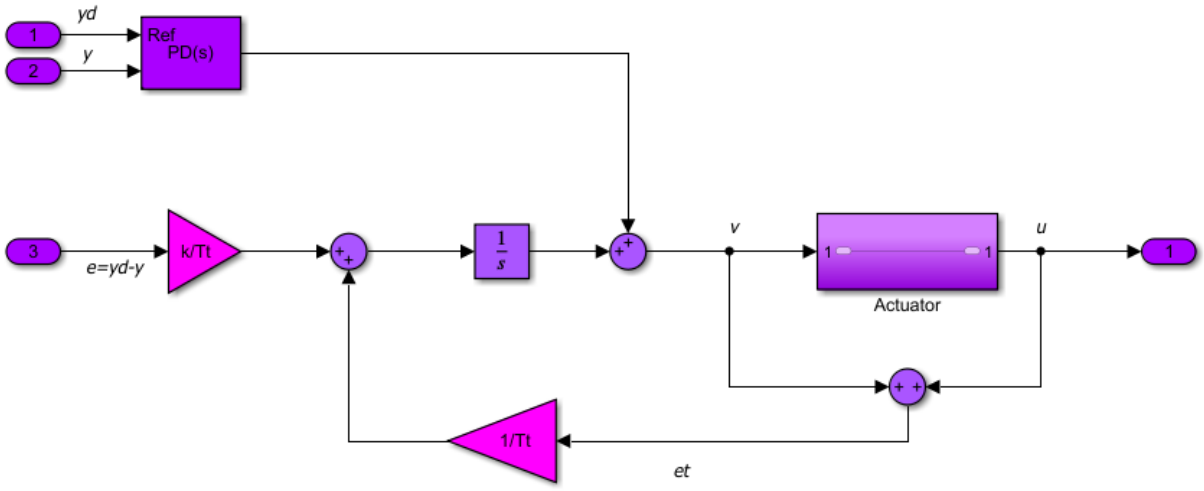


Figure 5.6: Anti-windup implementation on simulink

term to decrease. This can generate a huge overshoot and a dump oscillation.

There are several possibilities to avoid this issue, we decide to adopt the back-calculation method as presented in [2]. In this case, when the output saturates, K_i is recomputed in order to give a value at the saturation limit and the integrator is reset with a time period T_t. A good rule of thumb consists in setting $T_t = \sqrt{T_i \cdot T_d}$ where $T_i = \frac{K_p}{K_i}$ and $T_d = \frac{K_d}{K_p}$ that are respectively the integrator and the derivative time constants.

Figure 5.6 illustrates the block diagram with the implementation of the back-calculation method, the difference between the output of the PID and the output of the actuator becomes the input of the integrator through a constant $\frac{1}{T_t}$. In this way, the regulator works as if it has two modes: the normal condition and the one operating when saturation is reached to avoid windup.

The derivative term $K_d \cdot \frac{de}{dt}$ depends on the rate of error change. Its aim is to predict the system behaviour in order to improve settling time and reduce overshoot. The main problem is the high sensitivity to noise of the derivative term that can cause the system to vibrate. Moreover, to create a derivative action in a real situation it is necessary to increase the number of poles of its transfer function, for this reason, we must add a high-frequency pole. In this way, the transfer function describing the derivative term becomes:

$$\frac{K_d s}{1 + \frac{K_d}{K_p N} s} \tag{5.9}$$

	K_p	K_i	K_d
P	$0.50K_{max}$	-	-
PI	$0.45K_{max}$	$1.2K_p \cdot f_o$	-
PID	$0.60K_{max}$	$K_p \cdot f_o$	$0.125K_p/f_o$

Table 5.2: Ziegler-Nichols tuning rule.

The value of N is chosen to obtain a pole $s = -\frac{N}{T_d}$ which is outside of the control bandwidth. The control bandwidth measures the reactivity of the system to the input command changes. Typical values of N are between 5 and 20. We decide to consider $N = 10$ for both robots.

5.2.2. Tuning

The first PID is developed in 1911 by Elmer Sperry but introduced as a tunable proportional controller in 1933 by the Taylor Instrumental Company. In order to eliminate the steady-state error, some control engineers introduce, a few years later, a way to reset the integrated error from the proportional controller, introducing the first PI. In 1940 Taylor Instrumental Company, in order to reduce overshoot, creates the first derivative action to be added to the controller. However, the first tuning rules for PIDs are introduced in 1942 by Ziegler and Nichols defining a method that is still one of the most used. It is a heuristic method in which all the gains are initially set to zero. Then, the proportional gain is increased until the system is unstable and the output starts to oscillate. In this way, the maximum gain K_{max} and the frequency of oscillation f_o are found, so that K_i and K_d can be expressed as a function of the [9]. Depending on the type of PID required, Ziegler and Nichols define a set of rules that are described in Table 5.2

This approach may generate gains that are so elevated to possibly be dangerous in a real application and, in particular, in an indoor environment like the one our robots are moving in. For this reason, as explained in Chapter 4, we create a black-box model of the full system, kinematic and dynamic, and we apply the tuning method in simulation through MATLAB® and Simulink® to obtain the first set of value for the gains, as depicted in Figure 5.7. The main problem of the Ziegler and Nichols tuning rule is related to low robustness and to high sensitivity to parameters variation, for this reason, we settle the final value of the proportional, integral, and derivative gains according to a trial-and-error approach considering how a change in each gain affects the response of the system,

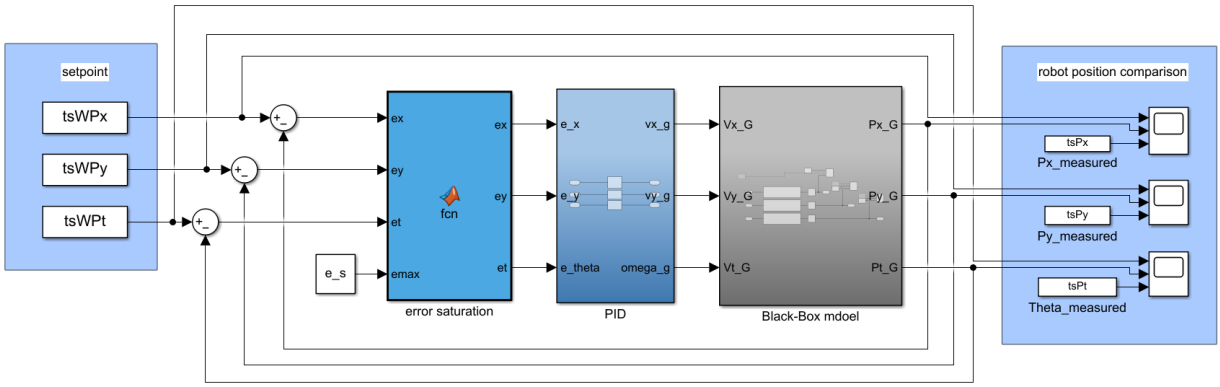


Figure 5.7: PID and black-box model

	Rise Time	Overshoot	Settling Time	Steady-State error
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Minor Change	Decrease	Decrease	None

Table 5.3: Effects of increasing the three gains.

as it is summarized in Table 5.3.

Furthermore, in order to have a diagonal motion even in situations where the difference between the set-point and the actual position is much greater in one of the two directions, we modify the x and y errors according to Algorithm 5.1

Finally, if the robots arrive at the last waypoint all the velocity commands are published equal to zero, and the PIDs are reset. Moreover, to be safe in case of loss of connection with optitrack, if no messages from it arrive for $50ms$ the robots stop.

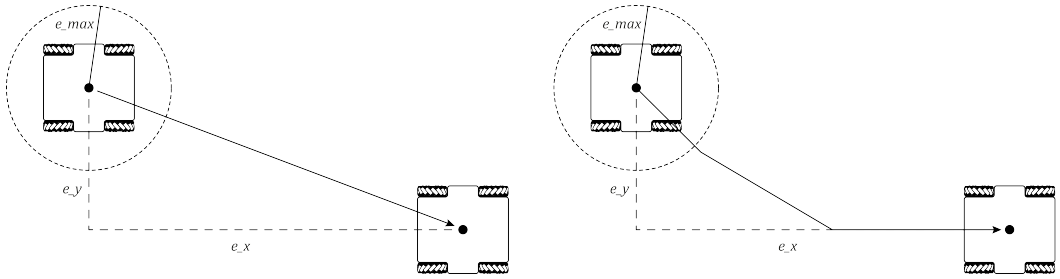
5.2.3. Double Navigation

In order to allow greater modularity during navigation for different possible applications, and to be able to use, in the best way, the in-front camera, in future works, we decide to implement the possibility to switch between two modes: the first one in which the platforms behave as a classical omnidirectional robot and another one in which the two robots are similar to differential drive ones.

In the second case, when a waypoint is generated, we evaluate the direction between the actual position of the robot and the x and y position of the set-point. During the

Algorithm 5.1 Error modification algorithm

1: **if** $abs(error.x) > e_{max}$ or $abs(error.y) > e_{max}$ **then**
2: **if** $abs(error.x) > abs(error.y)$ **then**
3: error.y = sign(error.y)·e_{max}·abs($\frac{error.y}{error.x}$)
4: error.x = sign(error.x)·e_{max}
5: **else**
6: error.x = sign(error.x)·e_{max}·abs($\frac{error.x}{error.y}$)
7: error.y = sign(error.y)·e_{max}
8: **end if**
9: **end if**



(a) Desired motion. (b) Undesired motion.

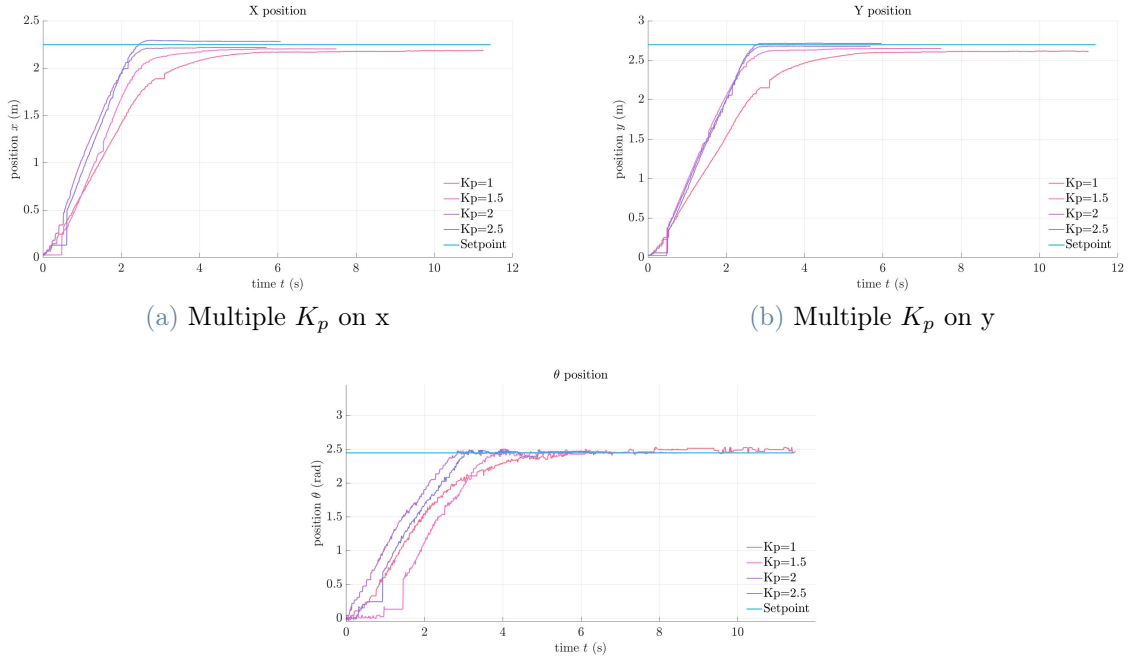
Figure 5.8: Error saturation algorithm

motion, we control the orientation of the robot considering as the set-point the angle evaluated in this way and removing the independence between the three PIDs so that the robots can move only forward. When they reach a user-determined distance from the set-point, the desired angle becomes the orientation of the waypoint and the control is performed as happens during the omnidirectional motion.

5.3. Experimental Results

In this Section, we present the experimental results of the tuning of the PID controllers on both platforms with a position step signal as input. We then proceed to validate the overall system with multiple waypoints and with different starting positions, to test different possible distances to be travelled.

In order to choose the best set of gains with the trial and error approach, we test the motion of the robot by changing the values of the parameters. We adopt a step-by-step method: starting from pure proportional controllers we then increase complexity if



(a) Multiple K_p on x

(b) Multiple K_p on y

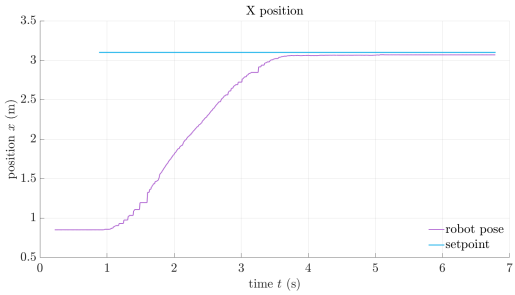
(c) Multiple K_p around theta

Figure 5.9: Proportional term tuning

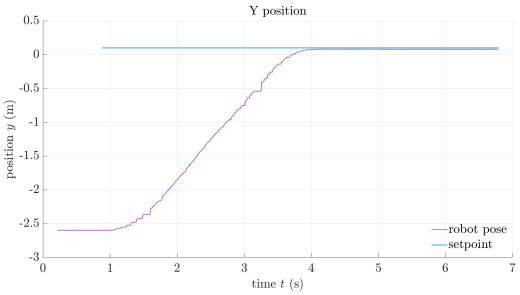
necessary. Given the considerations elaborated in Section 5.2.2 we choose to set the initial P gain equal to a fifth of the Ziegler and Nichols value.

5.3.1. Omni wheeled robot

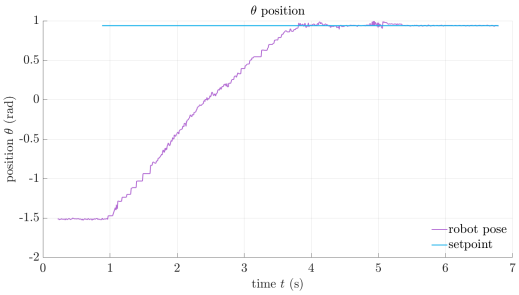
Starting with the proportional term, we change it from a value of 1.0 to 2.5 at regular intervals of 0.5, moving from the same initial position to the same goal and analyzing the response for the three directions x, y and θ. The results of these tests are shown in Figure 5.9. Starting from the response on the x direction, which is depicted in Figure 5.15a, we notice that, with a value of $K_p = 1$ the response is slow and the steady-state error does not satisfy the specifications. Increasing the gain to $K_p = 1.5$ we obtain a faster response but with a steady-state error that can still be improved. The response with $K_p = 2$ guarantees a response that is satisfactory both from the point of view of the percent overshoot and of the steady-state error. Trying to obtain an even greater improvement in the performance we set the proportional gain to a value of 2.5 but the response presents an overshoot and the steady-state error is still present. For this reason, we decide to adopt the third possibility. Analogous reasoning can be done for the proportional gain of the PID regarding the y direction, Figure 5.15b. Finally, we analyze the responses for the rotation around *theta* that are represented in Figure 5.15c, these measurements



(a) x response



(b) y response



(c) θ response

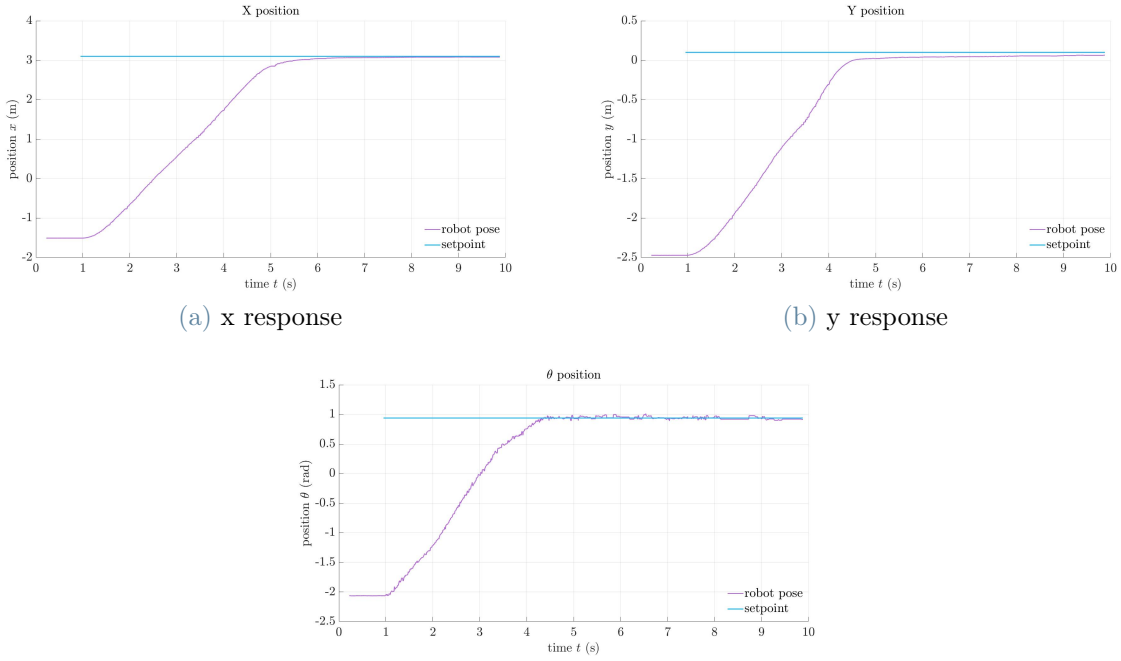
Figure 5.10: Response of the robot with proportional gain K_p=2

	Steady-State error	Overshoot	Settling Time	Rise Time
$\boldsymbol{x}$	0.0311 m	0 %	3.1 s	1.864 s
$\boldsymbol{y}$	0.019 m	0 %	3.3 s	1.87 s
$\boldsymbol{\theta}$	0.0058 rad	0 %	3.285 s	2.186 s

Table 5.4: Omni wheeled specification with K_p=2.

are affected by the noise from Optitrack, in any case, it is possible to notice that the response with $K_p = 1$ is very slow and presents a large steady-state error, for this reason, is excluded. The performances obtained with the other tests are similar, as a consequence, we decide to assign to the proportional term a value of 2 because is the best trade-off between fast response and safety behaviour.

The detailed response of the platform with a proportional gain equal to 2.0 is shown in Figure 5.10, starting from the initial pose $q_{r,start} = [0.8512, -2.5971, -1.5089]^T$, the robot must reach pose $q_{r,goal} = [3.1, 0.1, 0.9389]^T$. In this way, for the control along the x direction, we obtain a steady-state error of 0.0311 m and zero percent overshoot. Moreover, the rise time is equal to 1.864 s while the settling time set to 5% is 3.1 s. Also along the y direction, the percent overshoot is null, and the steady-state error reaches a



(a) x response

(b) y response

(c) θ response

Figure 5.11: Response of the robot when travelling a longer distance

	Steady-State error	Overshoot	Settling Time	Rise Time
$\boldsymbol{x}$	0.0245 m	0 %	4.36 s	3.033 s
$\boldsymbol{y}$	0.0356 m	0 %	4.059 s	2.489 s
$\boldsymbol{\theta}$	0.0245 rad	0 %	3.525 s	2.362 s

Table 5.5: Validation specification.

value of $0.019m$. The rise time is 1.87 s and the settling time is 3.3 s. Finally, we analyze the behaviour around θ, in this case, we obtain a response that is characterized by no overshoot, steady-state error equal to 0.0058 rad, a rise time of 2.1860 s and settling time of 3.285 s. In conclusion, with this tuning configuration, the result of the motion of the platforms completely fulfils the specifications. Since the response of the Omni wheeled robot satisfies the limitations that we impose and has satisfactory behaviour with just a proportional controller, we decide to not add the integral and the derivative terms so that we have a regulator that is as simpler as possible. The specification found are summarize in Table 5.4. In order to validate our controller choice, we decide to perform the same test with different starting conditions, i.e. with a different starting point, analyzing the behaviour of the robot when it has to travel longer or shorter distances.

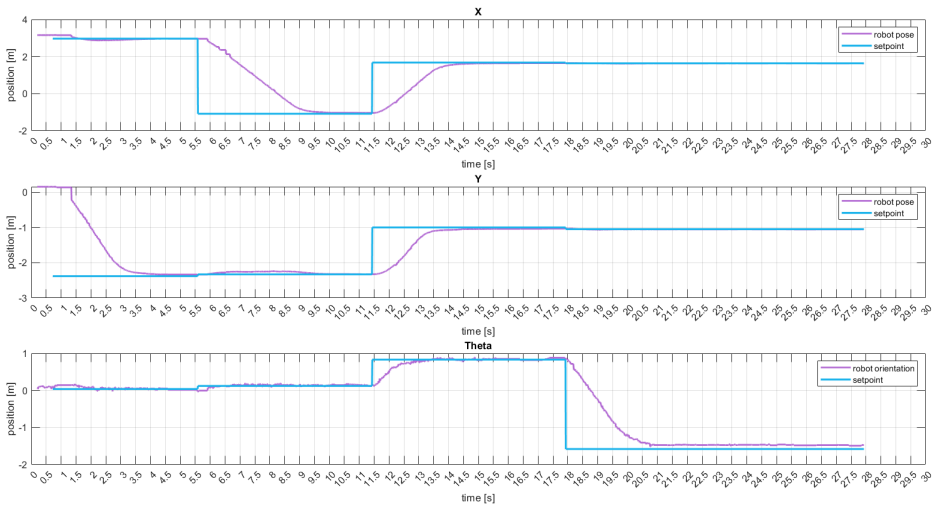


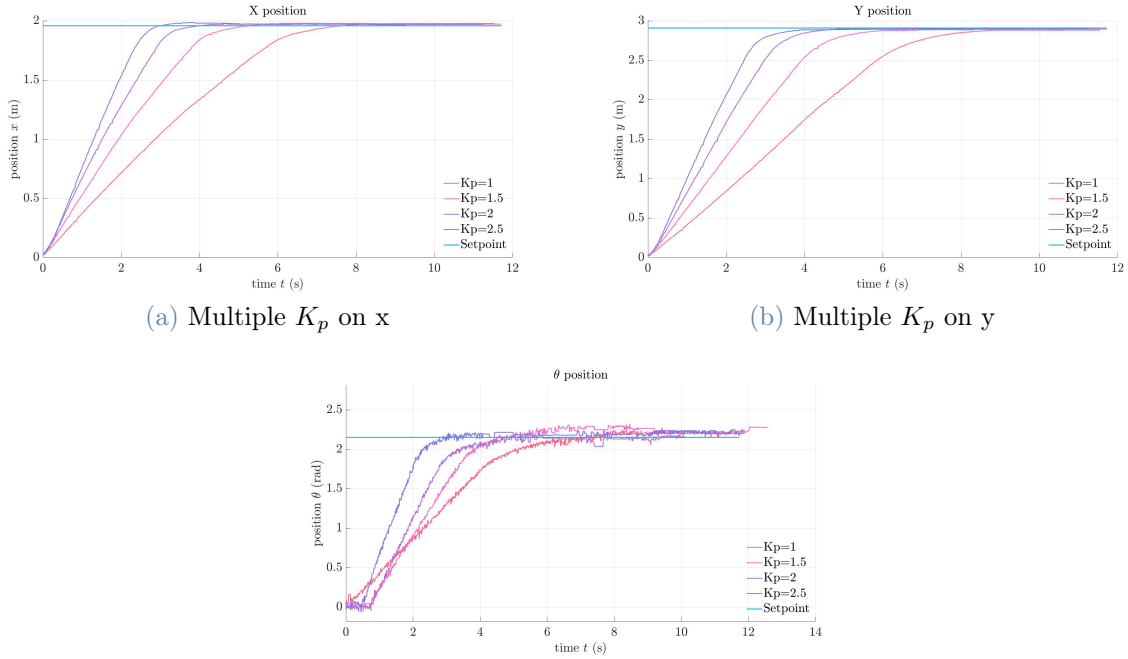
Figure 5.12: Response of the robot when travelling multiple waypoints

The first situation is shown in Figure 5.11, the robot has to move from $q_{r,start} = [-1.5081, -2.4719, -2.0633]^T$ to $q_{r,goal} = [3.1, 0.1, 0.9389]^T$. It is possible to notice that, also in this case, overshoot is absent in all three responses. Furthermore, the steady-state error is 0.0245 m along x, 0.0356 m along y and 0.0245 rad around θ meaning that the controller is valid. Regarding the other response information we obtain a rise time of 3.033 s and a settling time of 4.36 s for x, a rise time of 2.489 s and a settling time of 4.059 s for y, and, finally, we obtain on θ 2.362 s and 3.525 s respectively for the rise time and the settling time.

As a final remark, it could be possible to obtain a faster motion by increasing the saturation limits or the maximum permitted error but, it would affect the steady-state error and the percent overshoot of the responses, since we decide to give more importance to these two aspects, we still prefer to have a slower robot but with higher precision.

As our last test, we create a possible real-life situation for the robot, identifying multiple waypoints that simulate working stations the robot has to cross before reaching the final goal. In order to do that, we use a joystick to collect the different waypoints and then we let the robots move in autonomous. We introduce a tolerance value of 5 cm on x and y and of 0.05 rad on θ to allow the robot to move to the next set-point, this represents the admitted steady-state error.

In particular, the platform has to perform a first motion that is prevalent along the y direction, followed by sliding in the x direction. Then it moves diagonally and concludes with a rotation as depicted in Figure 5.12.



(a) Multiple K_p on x

(b) Multiple K_p on y

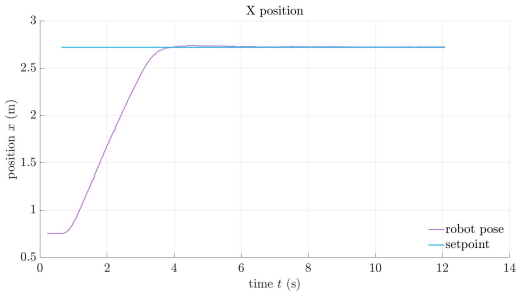
(c) Multiple K_p around theta

Figure 5.13: Proportional term tuning

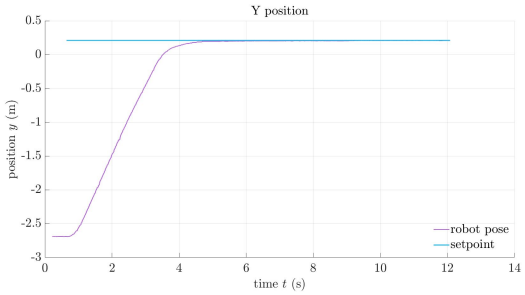
The robot is able to reach each set-point with the required precision, as already noticed during the validation test with the short step, the robot presents overshoot when it has to travel small distances along a certain direction or it has to perform small rotations. It would be possible to reduce the time the platform stops at a waypoints by increasing the tolerance value. It is important to underline the high precision of the robot when it arrives at the goal, the steady-state error on both x and y is lower than 1 cm.

5.3.2. Mecanum wheeled robot

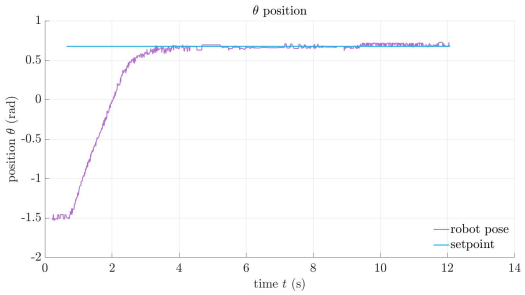
We decide to adopt the same approach used before also with this second robot, in order to obtain the best controller for our purpose we tune the proportional gain as our first step. Considering Ziegler Nichols as the starting point we analyze the response of the robot to the same goal with different values of K_p. The results are visible in Figure 5.17. Along x direction can be noticed that all the responses present a small percent of overshoot and a low steady-state error but, anyway, the specifications on these values are satisfied. For this reason, we base our final decision on how fast the robot is able to reach the goal. As expected, an increase in the proportional gain leads to a smaller rise time, as a consequence, we decide to impose $K_p = 2.5$. Regarding the behaviour along y direction, none of the responses present overshoot, our choice is based on the steady-state error,



(a) x response



(b) y response



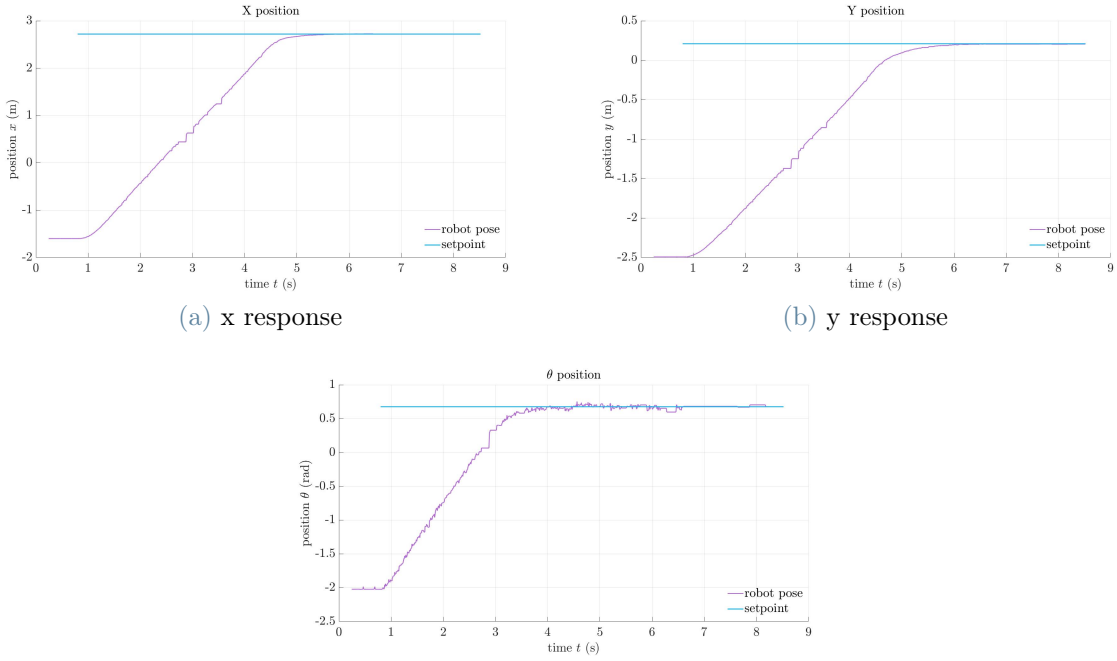
(c) θ response

Figure 5.14: Robot response with the implemented controller

	Steady-State error	Overshoot	Settling Time	Rise Time
$\boldsymbol{x}$	0.0047 m	1 %	3.1350 s	2.032 s
$\boldsymbol{y}$	0.0011 m	0.0449 %	3.417 s	2.226 s
$\boldsymbol{\theta}$	0.0062 rad	2.3347 %	2.6420 s	1.476 s

Table 5.6: Mecanum wheeled specification with K_p=2.5.

which is acceptable with all the possibilities but definitely lower with the higher values of K_p, and, secondly on the rise time which is lower with $K_p = 2.5$; for this reason, the latter is our final choice. Finally, we analyze the rotation around θ, this is particularly affected by the noise from Optitrack™but it is possible to see that, as in the x situation, any choice of the proportional gain causes overshoot and steady-state error, and, like before, we consider the rise time to determine our final decision, also in this case we set $K_p = 2.5$. We test our platform with the chosen controller to analyze with more precision the results and the specifications, starting from $q_{r,start} = [0.7543, -2.6889, -2.5201]^T$, the robot must reach the goal placed in $q_{r,goal} = [2.72, 0.21, 0.6761]^T$, Figure **??**. Regarding x, we obtain a steady-state error of 0.0047 m and a percent overshoot of 1% that are much below our imposed limits, furthermore, with a rise time of 2.032 s and a settling time of 3.135 s the controller is satisfying. Moving along y the response is characterized by an overshoot



(a) x response

(b) y response

(c) θ response

Figure 5.15: Robot response when travelling longer distances

	Steady-State error	Overshoot	Settling Time	Rise Time
$\boldsymbol{x}$	$\simeq 0\ m$	0.15040 %	4.274 s	2.926 s
$\boldsymbol{y}$	0.0043 m	0 %	4.5780 s	3.0740 s
$\boldsymbol{\theta}$	$\simeq 0\ rad$	2.8851 %	2.8960 s	1.9910 s

Table 5.7: Mecanum wheeled specification .

of 0.0449% and the robot reaches its final value with an error of 0.0011 m, the rise time and the settling time are respectively 2.226 s and 3.417 s, this means that also in this case the proportional gain is set correctly. We conclude by focusing on the θ response, in this case, the numeric results related to the specifications are strongly affected by noise, indeed, the percent overshoot is 2.3347%, and the steady-state error is 0.0062 rad, but in reality, they are lower. It's important to underline that, despite the noise, the results are still satisfactory. Analyzing the obtained values, we conclude implementing a position controller composed of only the proportional term with a gain equal to 2.5. Also with this robot, as before, this type of controller is enough for our purpose, if it would be necessary to reduce the steady-state error or to obtain a faster response, an integrative term can be added even if this would increase the overshoot.

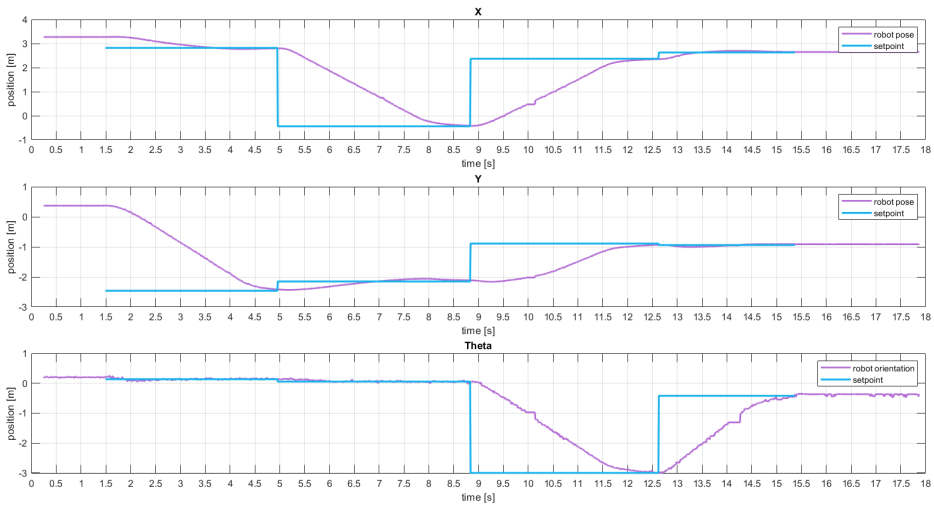
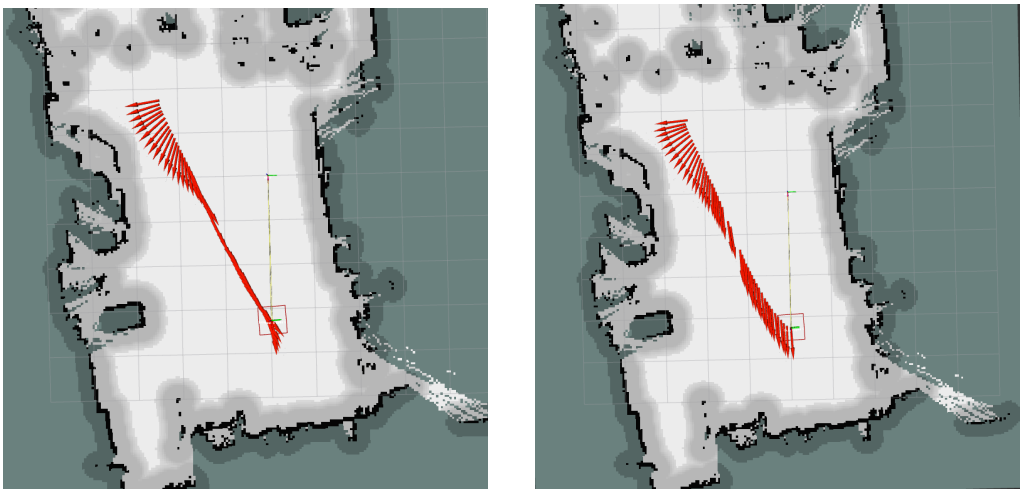


Figure 5.16: Multiple waypoints following

In order to verify if the selected controller works well in different situations, we test the robot with longer and shorter distances.

The result from the first case is presented in Figure 5.15, the starting pose is $x = -2.0218$, $y = -2.4925$, $\theta = -2.0218$, while the goal is placed in $x = 0.6761$, $y = 0.2100$, $\theta = 0.6761$. In this case, the error is almost eliminated considering the motion along x with a percent overshoot of 0.1504%. On the other hand, the response is slow with a rise time of 2.926 s and a settling time at 5% of 4.274 s. Also the behaviour on y is satisfactory, it's important to underline the complete absence of overshoot and the steady-state error equal to just 0.0043 m, even if this implies a slow response also in this direction, indeed, the rise time is 3.0740 s and the settling time 4.5780 s. Finally, the response on θ reports again a steady-state error very close to zero and an overshoot of 2.8851%, much lower than the specification. We can conclude that this controller is perfectly valid for longer distances. In order to be coherent, we decide to conduct a concluding test, also on the Mecanum wheeled robot, the multiple waypoints following. Using the joystick we collect waypoints with coordinates $x = 2.82, -0.44, 2.37, 2.63$, $y = -2.46, -2.15, -0.89, -0.94$, and $\theta = 0.1341, 0.0548, -3.0022, -0.421$. As depicted in Figure 5.16, thanks to the proportional controller, the robot is able to reach all the intermediate points and the final goal with the required accuracy. In this case, the robot stops at a waypoint adjusting its position in less time thanks to the higher value of the proportional gain. It is possible to notice that when the step dimension is very small, a certain amount of overshoot appears, but it is still much lower than our required specifications. This test completely demonstrates that our proportional controller is sufficient to fulfil all the specifications.



(a) Differential navigation (b) Omnidirectional navigation

Figure 5.17: Navigation modes

5.3.3. Double navigation

Before moving to the next Chapter, we want to analyze the differences, also from the experimental point of view, between the two types of possible navigation. As depicted in Figure 5.17a, when the mode is set to "differential", the robot is oriented along the direction that connects the actual position and the goal during its motion, then it turns and reaches the desired orientation when it is closed to the desired position. Figure 5.17b, instead, represents the classical omnidirectional motion, the x, y, and θ directions are corrected independently and the robots perform a rotation during its translation along

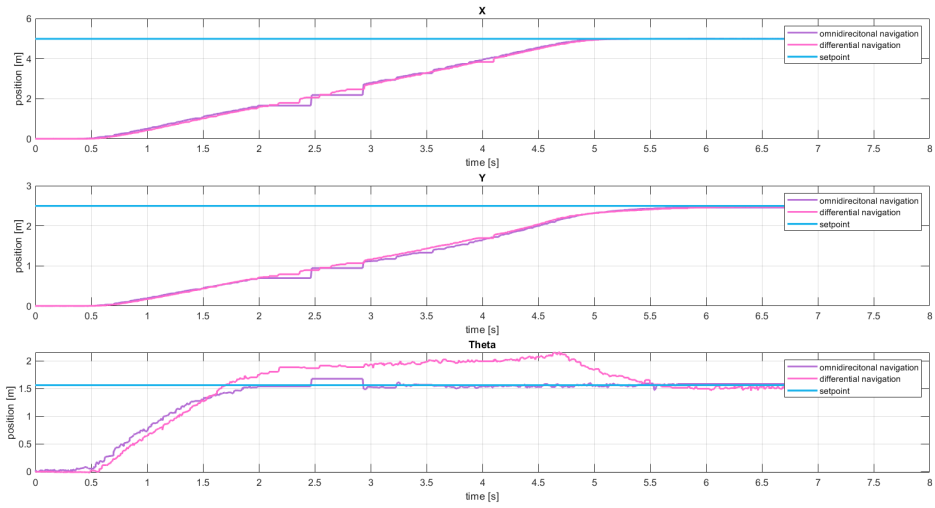


Figure 5.18: Double navigation experimental results

x and y axes. It is important to underline, as shown in Figure 5.18 that, from the performance point of view, the two modes guarantee the same results, the only difference stands in the behaviour of θ. In conclusion, the unique decision factor is the required movement specification in the robot application.

6 | Robot Navigation

This Chapter presents the two algorithms that we develop in order to perform navigation through the PID controller tuned in Chapter 5, together with the experimental results obtained implementing them on our real platforms. As already mentioned in Chapter 2, navigation consists in moving without collision with obstacles, from an initial position to a specific goal. The starting point to achieve this is move_base.

6.1. Move_Base

Move_base is a package, used in ROS, providing an implementation of an action that, as explained in [26], given a goal in the world, attempts to reach it with a mobile base, connecting the global and the local planners. This package provides the move_base node that implements a ROS interface for configuring, running, and interacting with the navigation stack on a robot.

The navigation stack takes information from odometry and sensor streams to be able to compute the velocity commands that are sent to a mobile robot[27]. The other required inputs are the information from amcl and the map from the map server. Figure 6.1 illustrates how move_base node operates and interacts with other components.

The map is created by moving the robot in the environment while considering the sensor readings and the odometry from the encoders. Odometry is used to estimate the position of the robot with respect to its starting localization. Amcl, instead, estimates the pose of the robot taking as input the laser-based map and the laser scans. Their different operating behaviours are shown in Figure 6.2. It is possible to notice that localization performed through odometry is based on dead reckoning which means calculating the current position by using a previously determined position and then integrating estimates of heading direction and speed, for this reason, this method is subjected to uncertainties related to odometry drift. Since our working environment is equipped with OptiTrack™, localization is performed with this external sensor, and no odometry or estimation through amcl is required avoiding the problems mentioned above.

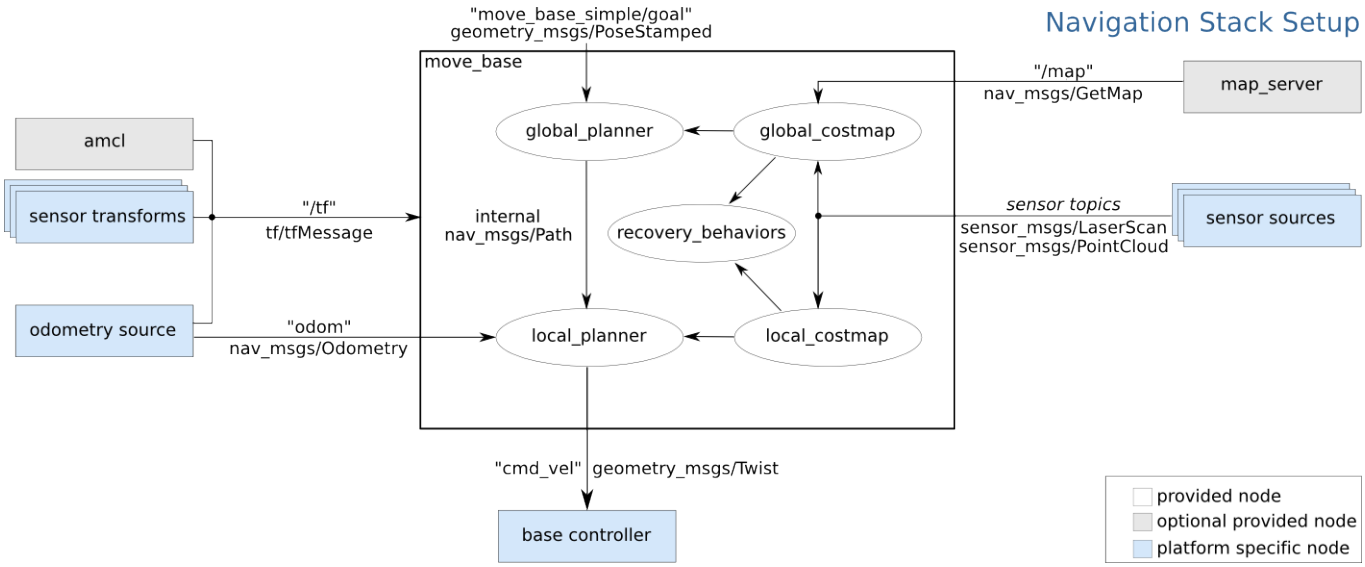


Figure 6.1: move_base structure

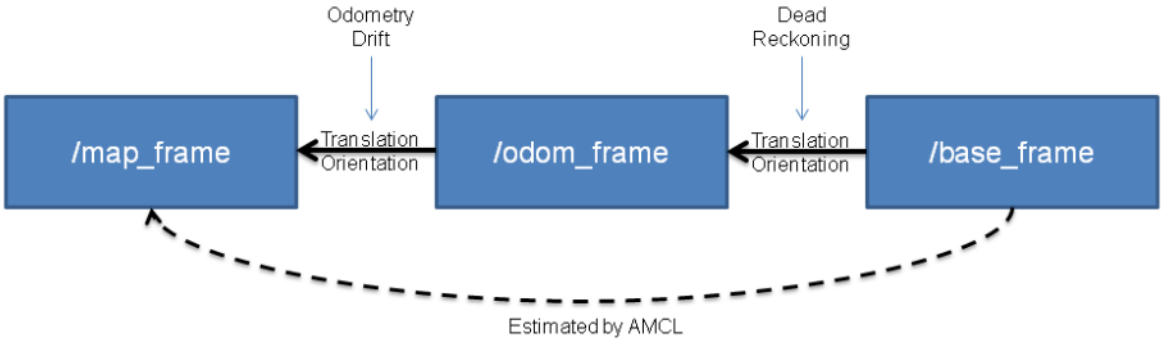


Figure 6.2: Odometry and amcl comparison

The global planner uses offline information about the environment, described by the map, to create the best possible path. Between the three options presented in Chapter 2, we adopt global_plan thanks to its flexibility. In fact, it can support both Dijkstra's and A* algorithms. As clarified in [19], the first one is a graphic search method that uses information from a grid cell map. It assigns a value to all the nodes, i.e. two elements to be connected, in the free space, and then, starting from the initial position, the value of a cell is the number of nodes that has to be crossed to reach that. The minimum value of the sum of all edges, i.e. line connecting nodes, from the starting position to the goal is the shortest path. Dijkstra's is based on a greedy algorithm in the sense that it finds the next best solution hoping to obtain as a final result the best solution to the whole problem, i.e. it finds the locally optimal choice at each step. The main advantage is the small complexity but, on the other hand, it may consume a lot of time analyzing the costs to all the nodes. A* algorithm, instead, is based on Dijkstra's method but it

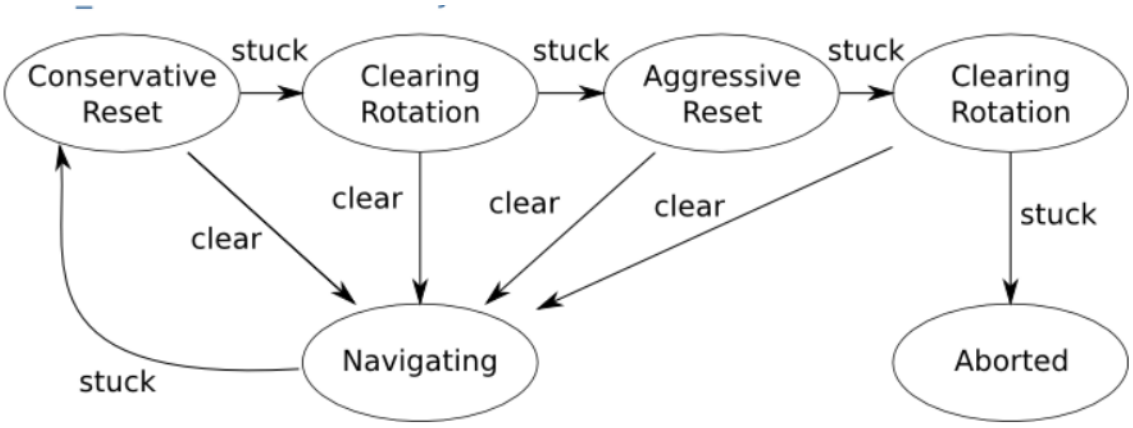


Figure 6.3: Recovery behaviors

uses a heuristic function to determine an estimate of the path between each node and the goal [13]. A heuristic function ranks alternatives based on available information, typically distance. There are three possible methods to evaluate a distance: the Euclidean distance, the Manhattan distance and the great circle distance. The first one is the measure of the segment connecting two points, the second one is the sum of absolute differences of the Cartesian coordinates of the points [7], and finally, the last one is the shortest measured distance on the surface of a sphere containing the two points [17]. In this way, the algorithm is faster and more efficient.

The local planner considers obstacles detected with sensors, and vehicle constraints in order to create new suitable waypoints while trying to follow the global path. In our work, we do not use a pre-defined local planner but implement an intermediate planner and a custom obstacle avoidance algorithm as presented in the next sections.

The navigation stack uses two costmaps to be able to read information about obstacles. The first one is used for global planning and considers the whole environment while the second one is used for local planning and obstacle avoidance and is continuously updated through sensor readings.

There are cases in which the robot can perceive itself as stuck, for this reason, the move_base node can perform recovery behaviours. In the beginning, all the obstacles that are outside a specific region are removed from the map and then the robot tries an in-place rotation to clear out space. If the robot fails again, it tries to clear its map another time, removing all the obstacles outside of the rectangular region in which it rotates in place. Finally, the robot tries to rotate again, and if it fails the goal is considered infeasible and it is aborted. This procedure is sketched in Figure 6.3

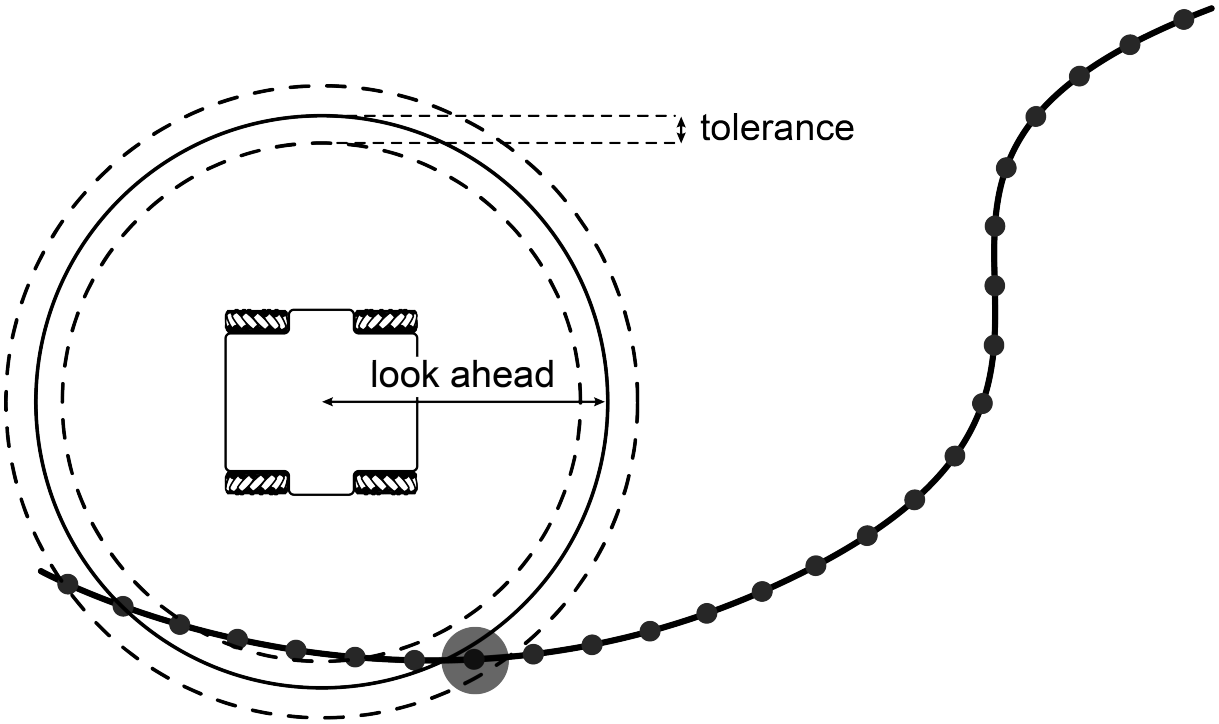


Figure 6.4: Spatial Horizon

6.2. Spatial Horizon

In order to allow the robot to perform a motion that can approximate a path following, but using the waypoint following strategy with PIDs developed in Chapter 5 we implement an intermediate planner. It is able to generate subgoals on the global path according to a user-determined look-ahead distance; these become the new set-points of the controller.

6.2.1. Algorithm explanation

The pose of the robot in each time instant is obtained by subscribing to OptiTrack™topic, furthermore, the initial position and orientation of the robot are saved and given as input to the global path generator.

The other input required by move_base is the goal, we generate it through the 2D Nav Goal button on Rviz, then it is saved and the path is created according to the make_plan service.

The global_path is composed of numerous close points, in order to generate the subgoal, we check all of these points evaluating the norm between the actual position of the robot and each of them. If this distance is greater than the difference between the look-ahead distance and the pre-determined linear tolerance, but it's lower than the sum

of the two, the farthest subgoal from the pose of the robot is selected and then published on the specific topic.

From this point on we implement two possible approaches: the new subgoals are continuously published so that the robots are characterized by a smoother motion, or a new subgoal is determined only when the robots are close enough to the previous one. When the distance between the robot and the goal is lower than the look-ahead horizon, the new subgoal is set equal to the end pose and published.

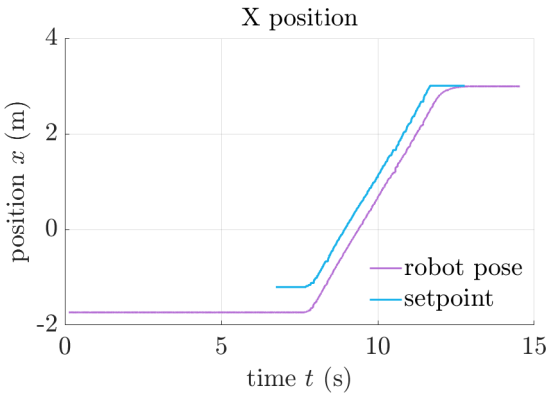
Finally, if the distance between the platform and the subgoal is greater than a user-defined threshold, which means that the robot is too far away from the path, the global route is planned again.

While in the classical Spatial Horizon algorithm the subgoals are sent to the local planner, in our case they are read from the PID controller and set as the new waypoints the robot has to follow.

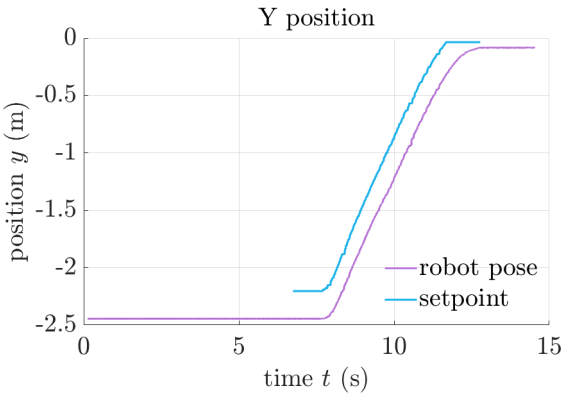
6.2.2. Experimental Results

This section shows the results obtained by implementing the algorithm on the two real platforms. We focus our analysis on the difference that the systems' responses present by changing the look-ahead distance; concluding with the comparison between the motion of the robots when they have to reach the subgoal before moving to the next one and when, instead, they are continuously updated.

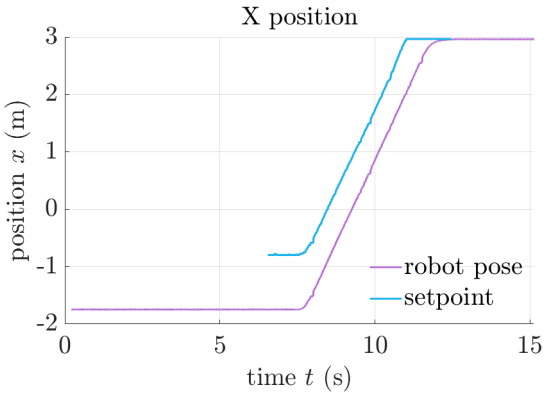
Starting with a value of 0.5 m, we perform the tests incrementing the horizon up to a value of 2.5 m. The results are depicted in Figure 6.5 where the line on the top represents the updated waypoints while the line below corresponds to the position measured by OptiTrack™. It is possible to notice that, as the look-ahead increases, the constant distance from the set-points increases and the motion is less precise, on the other hand, if we decide to set a more aggressive controller, with a small horizon, the robot reaches the subgoal before it is updated, this is due to the update frequency at which they are computed. It is important to underline that, using the same position controllers implemented in Chapter 5, the two platforms are able to move and reach the final goal with the required precision.



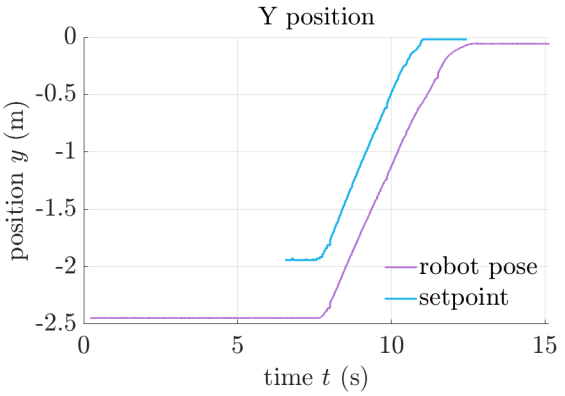
(a) x response with 0.5



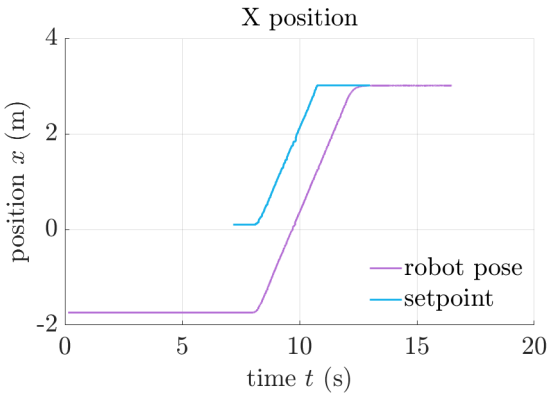
(b) y response with 0.5



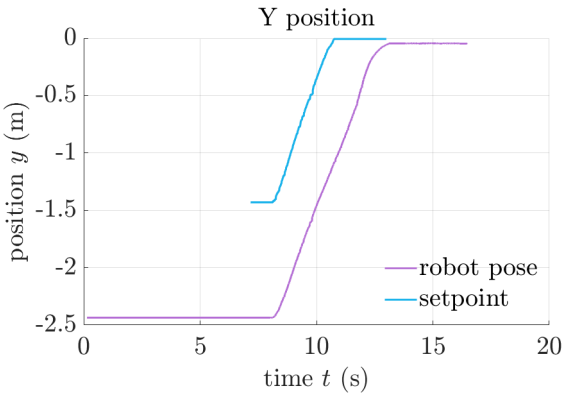
(c) x response with 1



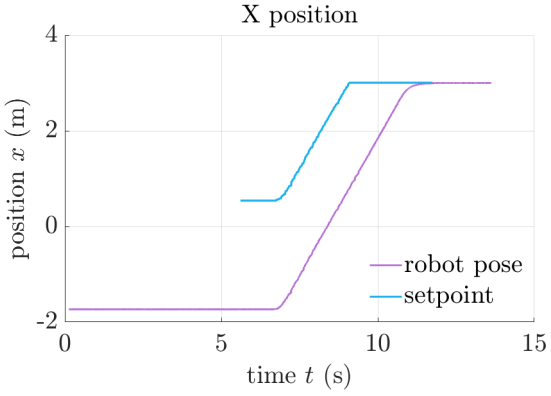
(d) y response with 1



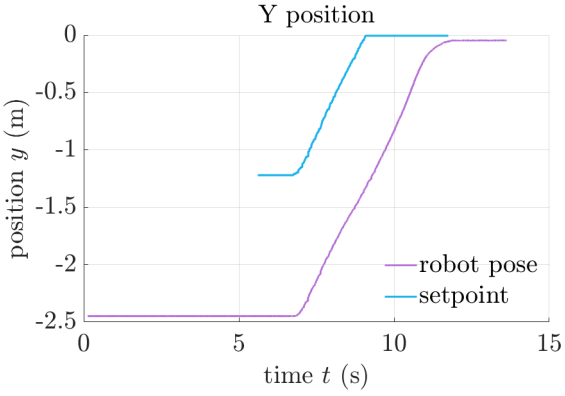
(e) x response with 2



(f) y response with 2

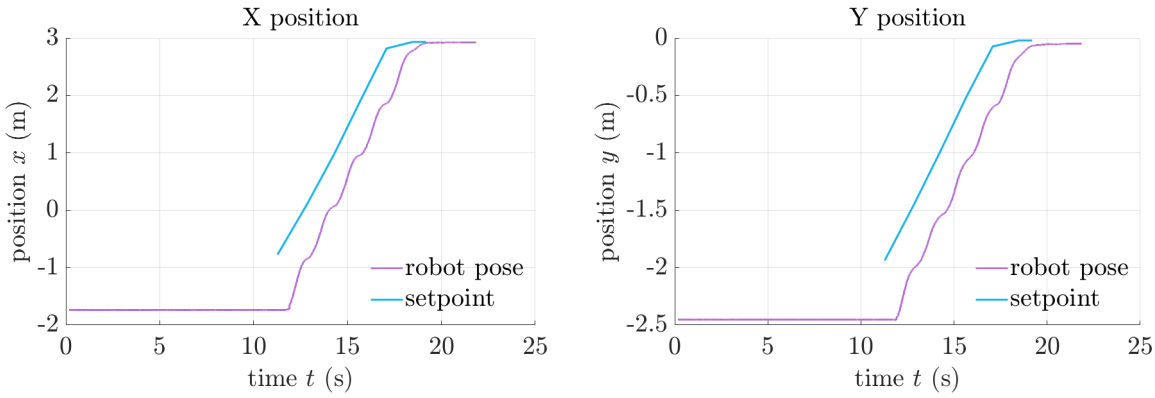


(g) x response with 2.5



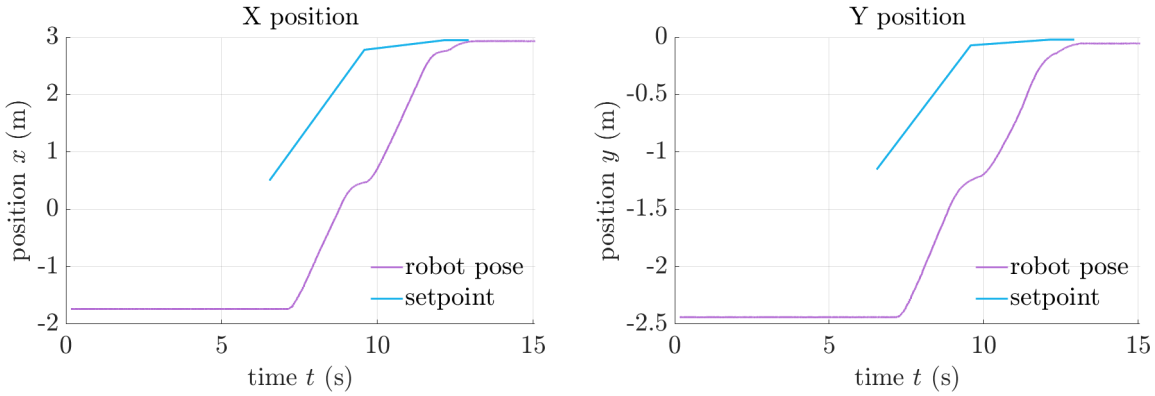
(h) y response with 2.5

Figure 6.5: Robot response with different look-ahead distances



(a) Second strategy x response (b) second strategy y response

Figure 6.6: Subgoal generation second approach



(a) Second strategy x response with horizon=2.5 (b) Second strategy y response with horizon=2.5

Figure 6.7: Effect of increasing the look-ahead distance

Figure 6.6 shows that using the second strategy of subgoals generation, the response is composed of multiple small steps, and the behaviour of the robots is less smooth. For this reason, we decide to use the first method.

For completeness, we show in Figure 6.7 the consequences of changing the look-ahead distance when subgoals are generated when robots are close to the previous one; it is possible to notice that the dimension of the steps is larger.

6.3. Vector Field Histogram

The last step of our work is related to obstacle avoidance. To perform that we decide to customise the VFH algorithm introduced in Chapter 2 since it is simple, requires a small amount of calculation, has high real-time performance and does not require the

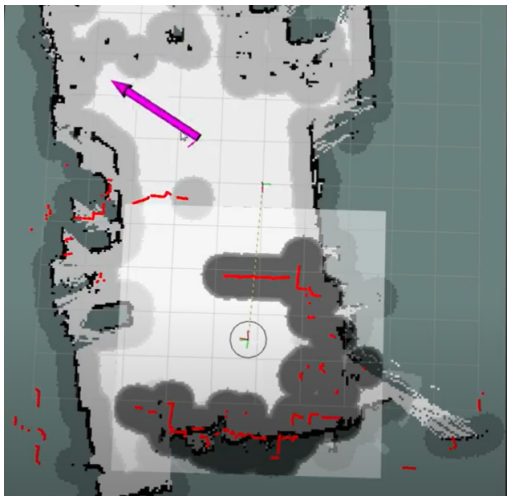


Figure 6.8: Goal definition with Rviz

robot to stop in front of an obstacle. In addition, it perfectly suits omnidirectionality.

6.3.1. Algorithm explanation

The actual 2D pose of the robot in each time instant is obtained and saved by subscribing to the OptiTrack™topic, then we determine a final goal through Rviz as illustrated in Figure 6.8.

When VFH is enabled, the subgoals from Spatial Horizon, obtained as described in Section 6.2, are not given directly to the PID controller but are first analyzed to check if they are on a path that can cause a collision.

When the intermediate goal is defined and the local costmap is ready, we build a circumference with a radius equal to the distance between the actual pose and the subgoal and centred with the chassis of the robot as represented in Figure **??**. The circumference is defined as a list with 360 elements, corresponding to 360 points in the 2D plan separated by 1 degree, where the first one is the subgoal from Spatial Horizon.

For each of these points, we apply Bresenham's algorithm to approximate the line between the centre of the robots and the 360 points of the circumference so that we can identify all the pixels in between.

As explained in [15], the algorithm identifies the driving axis (DA) as the one in which the difference between the two points is greater, and the passive axis (PA). The first one becomes the axis of control of the algorithm and the one of maximum movement. Starting from the initial point the coordinate corresponding to DA axis is incremented one by one, the other only as needed. Bresenham's algorithm keeps an error bound ϵ at each stage,

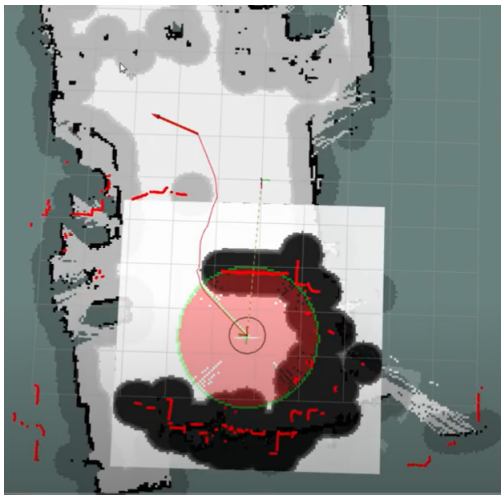


Figure 6.9: Circumference creation

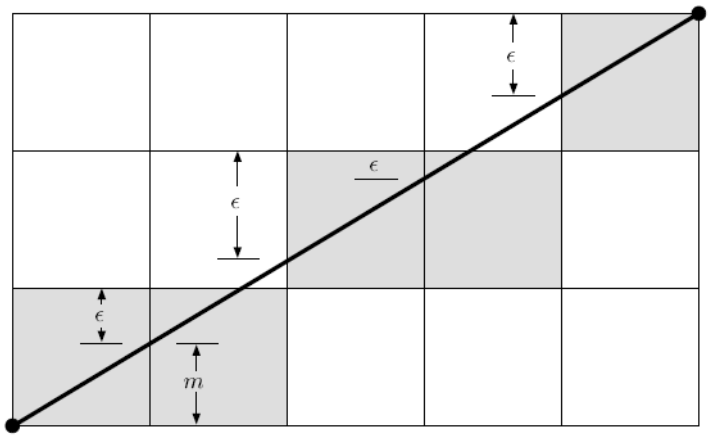


Figure 6.10: Bresenham algorithm example

which represents the negative of the distance from the point where the line exits the pixel to the top edge of the pixel as shown in Figure 6.10

In the beginning, its value corresponds to $m - 1$, and then it is incremented by m each time the DA coordinate is incremented by one. When ϵ becomes greater than zero it means the line has moved upwards one pixel, and it is necessary to increment the PA coordinate and decrement ϵ by one unit.

Once the pixels of all these lines are determined we have to define the corresponding index of the costmap since its data are represented in a tuple. This is found as

$$dy * rowstride + dx \tag{6.1}$$

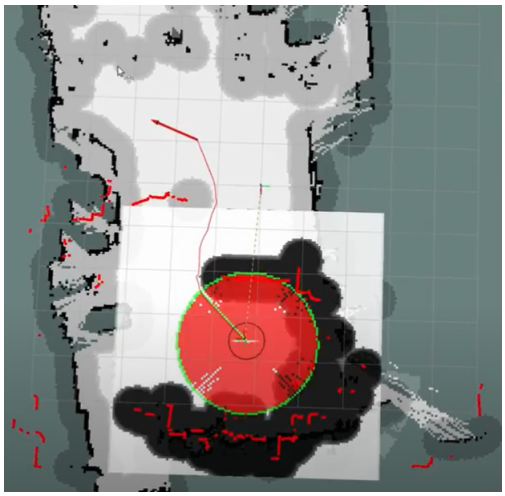


Figure 6.11: Lines connecting robot's centre and circumference

Where dy and dx are respectively the vertical and the horizontal distance (in pixels) from the origin of the costmap while the rowstride is the number of cells between the first pixel in a row of image data and the first pixel in the next row. It corresponds to the height of the costmap. As a result, we know the cost of each line connecting the centre of the robot to the points of the circumference, these are represented by the internal coloured circle in Figure 6.11.

In this way, thanks to the costmaps data, we can determine the cost of each pixel, defined as the probability of having an obstacle at that point so that we can obtain an list of 360 elements where the first one is the cost to reach the defined subgoal and the last is the cost to reach the point of the circumference on its right.

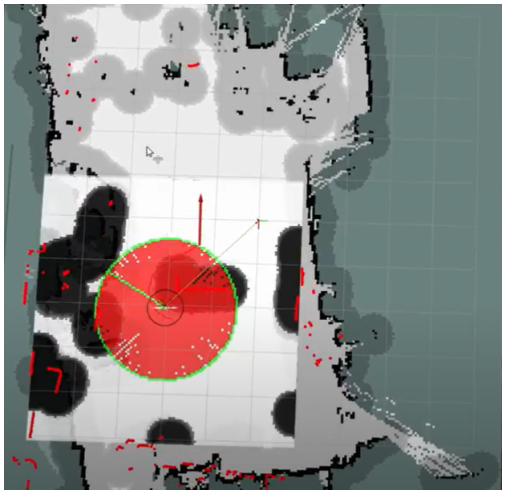
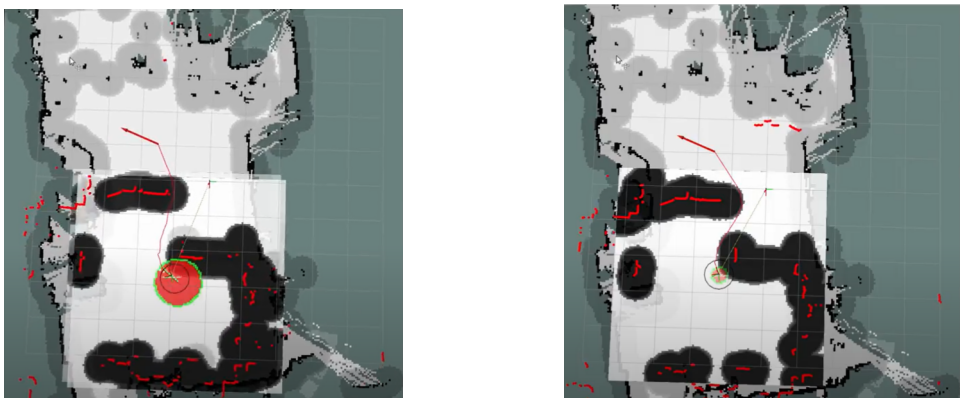


Figure 6.12: Subgoal selection



(a) Smaller circunference (b) Further radius reduction

Figure 6.13: Effect of increasing the look-ahead distance

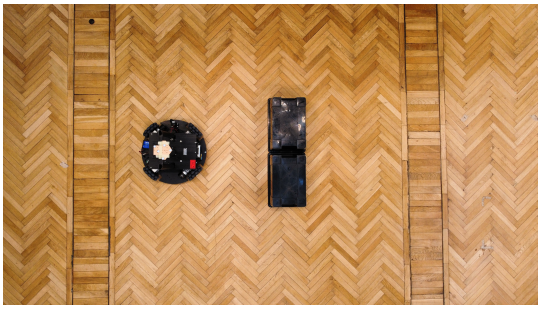
Finally, the algorithm begins scanning the list by alternating an element from the top and an element from the bottom corresponding to alternating left and right points on the circumference starting from the subgoal. When a point with a cost lower than the threshold is found, it becomes the new subgoal and it is published on a new topic so that the PID can use it. Figure 6.12 shows the line connecting the centre of the robot to the chosen subgoal, it is important to underline that the chosen set-point might not belong to the global path.

If the robot is stuck, i.e. no suitable points are found, we implement a procedure to recover. It consists of determining new circles with decreasing radius until the robots are able to determine a feasible path as represented in Figure 6.13. If a new subgoal can not be found, an error message is shown on the screen and the robot remains on the post.

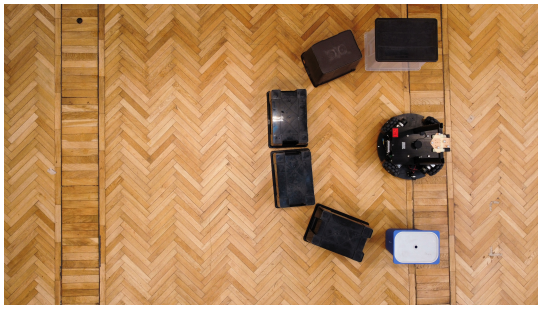
6.3.2. Experimental Results

This Section presents the results of implementing the obstacle avoidance algorithm. To test the reaction of the robots to different situations we select three main set-ups.

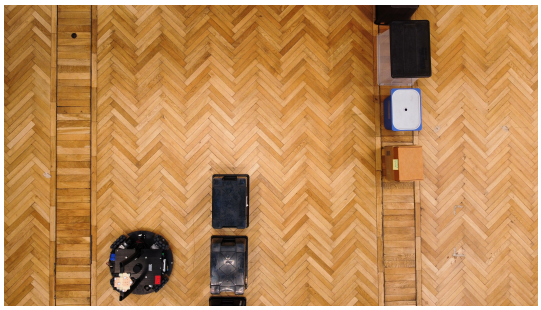
First of all, the platforms must be able to avoid a single obstacle as represented in Figure 6.14a. While the omnidirectional robot is able to perform well with the regulator implemented in Chapter 5, we have to reduce the proportional gain to a value of $K_p = 2$ for the mecanum one otherwise it is too fast to allow the Lidars to recompute the costmap. Subsequently, we want to test the robots with a more complex obstacle configuration as the S-shape depicted in Figure 6.14c. Both platforms are able to complete the path in a satisfactory way. Finally, we focus on a configuration that starts in a stuck condition, Figure 6.14b; both the robots are able to recover and reach the goal situated beyond the



(a) single obstacle



(b) [U-shape obstacle



(c) S-shape obstacle

Figure 6.14: Obstacle avoidance configurations

obstacles.

We want to conclude the experimental phase by analyzing the precision with which, during obstacle avoidance, the robot is able to follow the best path generated by the global planner. Starting with the already implemented regulators, the results are not satisfactory

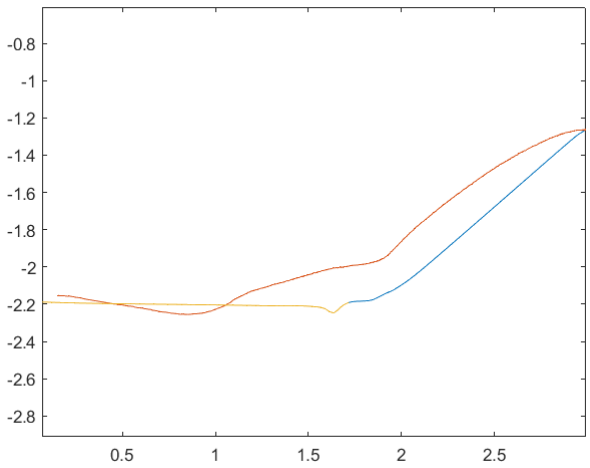


Figure 6.15: Path following with proportional controller

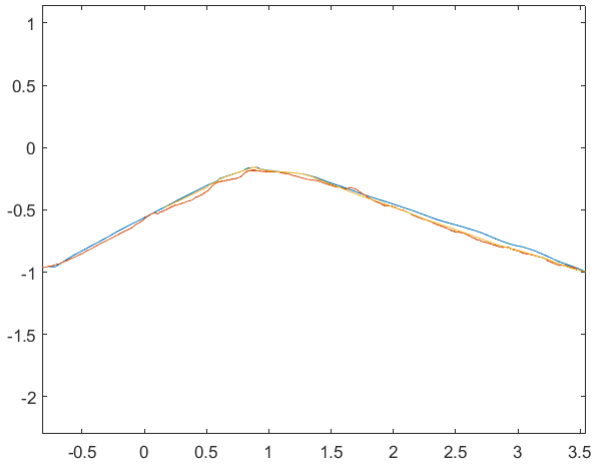


Figure 6.16: Path following with integral action

as shown in Figure 6.15. For this reason, we decide to add an integral action to both the controllers in order to reduce the steady-state error and to decrease the proportional term to $K_p = 1.5$ to be able to follow the path with more accuracy. The final behaviour is represented in Figure 6.16. With this implementation, it is possible to obtain a response characterized by higher precision, the robots follow the path with an error of 3 cm.

7 | Conclusion and Future Works

In this thesis work, we show the development of a control strategy that enables efficient navigation and obstacle avoidance of two omnidirectional platforms. In particular, we adopt a position controller, based on feedback measurement of the output from a Motion Capture system, that is able to reach the position and orientation of waypoints.

Starting from the complete model of the system that comprehends the kinematic and the dynamic model, estimated according to a black box approach, we evaluate the first set of gains for the controller by applying the Ziegler-Nichols tuning rule. Since with these values, the specifications are not satisfied, we adjust them according to an experimental trial and error approach. With the final set of gains, it is possible to obtain a satisfactory system response, and the controller choice is validated considering different initial conditions.

The obtained position controller is integrated with the traditional navigation approaches from ROS package move_base. Experimental results show that the intermediate planner that we implement guarantees a motion that is close to a path following, moreover the customized obstacle avoidance algorithm allows the robot to perform well and safely with simple and common obstacle shapes.

In addition, all the tests are performed with both the motion possibilities that we implement: the omnidirectional one and the differential-like one. The results are similar meaning that the user can choose the mode according to its necessities with the same performances.

The themes developed in this thesis allow several possible future works. First of all the definition of a precise dynamic model with the evaluation and the estimation of variables like wheel's and platforms' inertia and friction coefficients, so that it is possible to obtain an accurate theoretical model and improve the tuning of the controller. Moreover, in this way, it will be possible to perform tests with changing masses, simulating the behaviour of a manipulator taking objects that can be mounted on the two platforms.

The measurement of the output is obtained through the motion capture system, this

will not be available in all the environments the two platforms will work. For this reason, it could be useful to substitute this strategy with other localization methods. A possible solution can be the exploitation of the methods that are present in the navigation stack from ROS: amcl and odometry.

The PID controller chosen in this thesis guarantees good performances of the two robots, but, it can become useful, depending on the application field, to implement a more advanced controller. A future solution can be the MPC that minimizes a cost function, e.g. energy consumption or platforms' vibrations. Nevertheless, more complex strategies require a higher computational cost.

In cases the two robots are going to be used in dynamic environments such as warehouses, hospitals or hotels, characterized by moving obstacles or interaction with people, it would be useful to develop an obstacle avoidance algorithm that can substitute the customized VFH presented in this thesis. Moreover, there are obstacle configurations that cause the robot to oscillate without finding a feasible path, for this reason, it can be necessary to implement algorithms that overcome this problem.

Since the robot is equipped with an onboard camera that is not used in the present thesis work, a possible future implementation integrates an algorithm for image recognition with the currently onboard sensors.

Finally, from the hardware point of view, the contact point between a wheel and the ground is discontinuous creating higher vibrations with respect to other types of wheels, e.g. conventional wheels. Moreover, the platforms are characterized by rigid structures which means that vibrations propagate from the wheel to the top of the robots. A possible future solution is the implementation of a structural component that isolates the highest part of the robot from the lower ones preventing damage to the electrical devices and sensors.

Bibliography

[1] H. Andreasson. *Local visual feature based localisation and mapping by mobile robots*. PhD thesis, Örebro universitet, 2008.

[2] K. Åström and L. Rundqwist. Integrator windup and how to avoid it. 1989. American Control Conference, 1989, ACC '89 ; Conference date: 21-06-1989 Through 23-06-1989.

[3] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[4] I. M. Caireta. Model predictive control for a mecanum-wheeled robot in dynamical environments. *Universitat Politècnica de Catalunya Facultat de Matemàtiques i Estadística, Universitat Politècnica de Barcelona: Barcelona, Spain*, 2019.

[5] G. Campion and W. Chung. Wheeled rob 17 . wheeled robots. 2008.

[6] N. Corporation. Optitrack documentation wiki, 2022. URL `https://v30.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki`.

[7] Z. Cui. *Towards Interpretable Machine Learning with Applications to Clinical Decision Support*. Washington University in St. Louis, 2019.

[8] C. Delgado-Mata, R. Velázquez, and C. A. Gutiérrez. A differential-drive mobile robot driven by an ethology inspired behaviour architecture. *Procedia Technology*, 3: 157–166, 2012.

[9] G. Ellis. Chapter 6 - four types of controllers. In G. Ellis, editor, *Control System Design Guide (Fourth Edition)*, pages 97–119. Butterworth-Heinemann, Boston, fourth edition edition, 2012. ISBN 978-0-12-385920-4. doi: https://doi.org/10.1016/B978-0-12-385920-4.00006-0. URL `https://www.sciencedirect.com/science/article/pii/B9780123859204000060`.

[10] Flipsky. Flipsky mini fsesc6.7 pro 70a base on vesc6.6 with aluminum anodized heat sink, 2018. URL `https:`

//flipsky.net/collections/electronic-products/products/flipsky-mini-fsesc6-7-pro-70a-base-on-vesc6-6-with-aluminum-anodized-heat-sink.

[11] T. A. Gaffoor. Ai for process control series (part 1) - introduction to control strategies, 2022. URL https://www.innovyze.com/en-us/blog/ai-for-process-control-series-part-1-introduction-to-control-strategies.

[12] S. Huang and G. Dissanayake. Robot localization: An introduction. *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–10, 1999.

[13] ICS. A* search algorithm, 2022. URL https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all.

[14] Jane. *YDLIDAR G4 Data Sheet*. Shenzhen EAI Technology Co., 16th Floor, Block 7A,International innovation Valley,Nanshan District, Shenzhen, Guangdong, China, 6 2022.

[15] K. I. Joy. On-line computer graphics notes: Breshenham's algorithm, 2022. URL https://www.cs.put.poznan.pl/swilk/pmwiki/uploads/Dydaktyka/bresenham-int.pdf.

[16] L. Kästner, T. Buiyan, X. Zhao, Z. Shen, C. Marx, and J. Lambrecht. Connecting deep-reinforcement-learning-based obstacle avoidance with conventional global planners using waypoint generators. *CoRR*, 2021.

[17] B. D. Kifana and M. Abdurohman. Great circle distance methode for improving operational control system based on gps tracking system. *International Journal on Computer Science and Engineering*, 4(4):647, 2012.

[18] J. M. Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[19] P. Marín, A. Hussein, D. Martín Gómez, and A. de la Escalera. Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018:1–10, 02 2018. doi: 10.1155/2018/6392697.

[20] H. P. Oliveira, A. J. Sousa, A. P. Moreira, and P. J. Costa. Dynamical models for omni-directional robots with 3 and 4 wheels. In *ICINCO-RA (1)*, pages 189–196, 2008.

[21] K. Qin. General matrix representations for b-splines. In *Proceedings Pacific Graphics' 98. Sixth Pacific Conference on Computer Graphics and Applications (Cat. No. 98EX208)*, pages 37–43. IEEE, 1998.

[22] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[23] ROS. Introduction, 2018. URL `http://wiki.ros.org/ROS/Introduction`.

[24] ROS. Topics, 2019. URL `http://wiki.ros.org/Topics`.

[25] ROS. Bags, 2020. URL `http://wiki.ros.org/Bags`.

[26] ROS. move_base, 2020. URL `http://wiki.ros.org/move_base`.

[27] ROS. navigation, 2020. URL `http://wiki.ros.org/navigation`.

[28] L. Schenato. Control laboratory: Discretization of continuous systems. 4 2016.

[29] K. Shabalina, A. Sagitov, and E. Magid. Comparative analysis of mobile robot wheels design. *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*, pages 175–179, 2018.

[30] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[31] D. E. Skateboard. Electric skateboard motor 6355 190kv, 2022. URL `https://diyelectricskateboard.com/products/electric-skateboard-motor-6355-190kv`.

[32] H. Taheri, B. Qiao, and N. Ghaeminezhad. Kinematic model of a four mecanum wheeled mobile robot. *International journal of computer applications*, 113(3):6–9, 2015.

[33] I. The MathWorks. Black-box modeling, 2022. URL `https://it.mathworks.com/help/ident/ug/black-box-modeling.html`.

[34] I. The MathWorks. Simulation and model-based design, 2022. URL `https://it.mathworks.com/help/simulink/`.