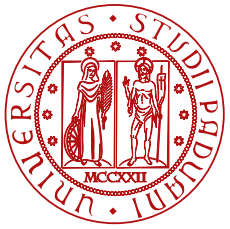


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master Degree in Control Systems Engineering

Autonomous navigation in unstructured environments using an arm-mounted camera for target localization

Master Candidate

Nicola Cigarini

ID 2019128

Supervisor

Prof. Angelo Cenedese

University of Padova

Co-supervisor

Dott. Nicola Lissandrini

University of Padova

Academic Year: 2022/2023

Graduation Date: 24th october 2023

To my family,
my friends,
and everyone who ever supported me

Abstract

Autonomous mobile robots became over the recent years a popular topic of research mainly for their capacity to perform tasks in complete autonomy without the constant intervention of an human operator. In this context, autonomous navigation represents one of the main studied branch of autonomous robotics.

Autonomous navigation in both structured and unstructured environments have been widely researched over years, with the development of several techniques that tries to solve this problem. In this context, there are several components that are required to get the proper solution to the navigation problem, and one of these is represented by the knowledge of the final position that an autonomous robot has to reach inside an environment.

In this thesis, the goal is to enhance the autonomous capabilities of a robot by making it able to detect and follow constantly a target placed inside an unstructured environment. This result is obtained using a camera installed as end-effector of a robotic arm, which in turn is installed on top of a mobile robot. All the methodologies as well as the tools that have been used in the development of this project are presented in this thesis.

The evaluation of the performances of the algorithm are performed both in a static context, where the robot is fixed and the target is free to move, and in a dynamic context where the robot moves and the target is fixed. The motion of the robot is obtained using an innovative algorithm for navigation in unstructured environments, NAPVIG.

The proposed approach has been implemented using ROS and been tested both in a simulated environment using Gazebo as well as in a real world scenario. The results obtained from both type of experiments will be presented and discussed.

Sommario

I robot autonomi sono diventati in questi anni un argomento di ricerca popolare principalmente per la loro capacità di eseguire compiti in completa autonomia, senza la presenza costante di un operatore umano. In questo contesto, la navigazione autonoma rappresenta uno dei principali settori di studio per la robotica autonoma.

La navigazione autonoma sia in ambienti strutturati che non è stata ampiamente studiata negli anni, portando allo sviluppo di diverse tecniche che cercano di risolvere questo problema. In questo contesto, vi sono differenti componenti richiesti per ottenere la soluzione adatta al problema della navigazione, e uno di questi è rappresentato dalla posizione finale che il robot deve raggiungere all'interno di un ambiente.

In questa tesi, l'obiettivo è di aumentare le capacità autonome di un robot rendendolo in grado di riconoscere e seguire costantemente un target all'interno di un ambiente non strutturato. Questo risultato è ottenuto utilizzando una videocamera installata come utensile di un braccio robotico, installato a sua volta sulla sommità di un robot mobile. Tutte le metodologie e gli strumenti utilizzati nello svilippo di questo progetto sono presentati nella tesi.

La valutazione delle performance dell'algoritmo è effettuata sia in un contesto statico, in cui il robot è fisso e il target è libero di muoversi, sia in un contesto dinamico in cui il robot è in movimento ed il target è fisso. Il movimento del robot è ottenuto utilizzando un algoritmo innovativo per la navigazione in ambienti non strutturati, NAPVIG.

L'approccio proposto è implementato in ROS ed è stato testato sia in un ambiente simulato utilizzando Gazebo sia in uno scenario nel mondo reale. I risultati ottenuti da entrambi i tipi di esperimento saranno presentati e discussi.

Contents

List of Figures

List of Tables

List of Acronyms

DDR Differential Drive Robot

DoF Degree of Freedom

FoV Field of View

GVD Generalized Voronoi Diagram

NAPVIG Narrow Passage Navigation

PID Proportional Integrative Derivative

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

1 Introduction

In the context of mobile robots, autonomous mobile robots represent nowadays one of the most interesting and challenging topic of research in the scientific community. Their application find place in different and various fields in the daily lifetime, spacing from the domestic use (as for example vacuum cleaner robots or lawn mower robots) to their usage inside extreme or harsh environments where the presence of an human operator is prohibitive. In all their applications, autonomous mobile robots help and support human activities by accomplishing tasks which are considered difficult or extreme to do. Moreover, in the recent years, the development of new technologies, such as high-performances microprocessors or faster sensors, allowed to improve even more the capabilities of autonomous mobile robots, leading them even further towards the fully autonomous direction.

Within the autonomous mobile robots world, the research topic concerning the autonomous navigation represents one of the biggest issues when perceiving the fully autonomy of a robot. Ideally, the aim of autonomous navigation is to make a robot capable of travelling from a starting point towards a destination in a completely independent way, with the robot being able to determine on its own the path it has to follow and avoiding at the same time the presence of any obstacle that would compromise the whole navigation process.

The knowledge of the destination, or target, that an autonomous mobile robot has to reach plays a central role in the determination of the path it has to follow. One way to determine the desired position to reach is to specify it by hand, which, in a fully autonomous context, seems quite contradictory. In this

sense, for an autonomous mobile robot the most logical thing to do should be to use its sensors to detect an object which can be interpreted as a target inside the environment.

The project presented in this thesis focuses mainly on this side of the autonomous navigation process, dealing with the recognition of a target and the following of it during the overall navigation of the robot. The proposed approach for the target detection makes use of a camera sensor which has been installed on a robotic manipulator, mounted in turn on an autonomous mobile robot. The usage of the robotic arm allows the constant following of the target, keeping always the camera fixed on it even where the mobile base of the robot is moving inside the environment or when both the target and the robot are moving together. With this approach, the purpose is to extend further the autonomous capabilities of a robot excluding even more the intervention of an external human operator.

1.1 Thesis structure

The present thesis work is structured into five chapters, where the first two will give a state of the art about autonomous mobile robots, while the remaining ones will focus onto the methodologies used to solve the target-locking problem as well as the presentation of the results obtained from a campaign of experiments. In particular:

- in chapter 2 is presented in detail a logical structure in which an autonomous robot can be divided into, giving explanations on what each substructure does to make an autonomous mobile robot such. There is also presented the detailed modelling of a particular type of autonomous mobile robot as well as the main control techniques that can be employed to drive it;

- in chapter 3 an innovative algorithm for the autonomous navigation of a robot is presented, which has been used in all the experiments performed. The last part of this chapter deals with an odometric position estimation algorithm that can be used to extend the autonomous capabilities of a robot. There are briefly presented the results obtained from a campaign of simulations;

- chapter 4 presents how the target-locking mechanism has been developed, starting from a brief description of robotic manipulators and illustrating the procedure that leads to the final algorithm;
- in chapter 5 the results obtained from a campaign of both simulated and real-world experiments are presented, discussed and used to prove the working of the target-locking algorithm described in chapter 4;
- at last, in chapter 6, there are given the conclusions about this project as well as some suggestions for future works.

2

Autonomous mobile robot

An autonomous robot can be defined as an intelligent machine which has the capability to operate without the constant presence of a human operator. In this context, autonomous mobile robots, with their capability of moving autonomously inside an environment and their ability in task solving, represents one of the most innovative and vastest scientific field of research of the recent years. The correct design of autonomous mobile robot may lead potentially to the solution of every task it is asked to solve, however, the designing process could result difficult in relation with the characteristic of the robot itself and of the unpredictable nature of the surrounding environment.

2.1 Structure of autonomous mobile robots

As mentioned in the introduction of this chapter, an autonomous mobile robot is considered to be so if it has the ability of determining what are the actions to undertake in order to move autonomously inside an environment with a little or even no intervention of an human operator. To achieve this result, an autonomous mobile robot requires a series of components that, working together, allows the correct motion of the robot. Considering the hardware point of view, an autonomous mobile robot is composed by:

- Actuators: are the components that allows the motion of the robot by converting energy into mechanical form. On an autonomous mobile robot electric motors are the most diffused way to move it, but it is possible to

find other types of actuators on it, such as hydraulic actuators, pneumatic actuators and so on;

- Sensors: are used to perceive the surrounding environment and get information from it;

- Controllers: are considered the brain of the robot, usually are computers, microprocessors or embedded micro-controllers;

- Power system: stores the energy that needs to be supplied to the other components of the robot.

The choice of the components that will be installed on the robot are made depending on the necessities and tasks it has to accomplish and is typically demanded to the manufacturer of the robot. This choice, in most of the cases, is tricky and requires always a trade-off between performances and costs. Instead of considering directly the hardware that will be used for an autonomous mobile robot, it is possible to define a more general architecture that divides the robot into different subsystems, which are identified as *locomotion*, *perception*, *control system* and *navigation* [29][27] and that will be helpful in the designing process of the robot. Each one of these subsystems is responsible of managing one aspect in the entire motion process of the autonomous robot. In the following sections there will be reported a description of these subsystems as well as their functions inside a robot.

2.1.1 Locomotion

Locomotion is the first aspect in autonomous mobile robot design that will be analyzed. The locomotion system allows the motion of the robot inside environments of various type, which can space from known and controlled ones (for example factories, laboratories) to the extreme and inhospitable ones (underwater exploration, space missions). The design of an efficient locomotion system however relies not only on the medium in which the robot will travel, but depends also on other technical aspects such as stability, controllability, maneuverability, terrain conditions, efficiency. Depending on their locomotion system, it is possible to divide autonomous mobile robots into different categories.

Stationary

As the name mentions, these robots do not move inside an environment. They are fixed to the world by means of a base and are composed of an open kinematic chain with an end-effector as last component, which is typically used for special tasks (grasping, welding, assembling, painting and so on)[1]. Inside this category of robots it is possible to find manipulators and industrial arms that are typically used inside industrial environments. However, in the recent years it is possible to find stationary robots mounted on mobile robots, with the aim to increase their capabilities in terms of tasks they are able to perform [26]. In chapter 4 there will be given a deeper analysis of the stationary robots since a robotic manipulator will be used for the purposes of the present thesis work.

(a) Example of industrial arm

(b) Example of manipulator on mobile robot

Figure 2.1: Examples of stationary arm

Ground-based

Ground-based mobile robots are nowadays one of the most diffused and studied category of mobile robots. In literature, they are also defined as Unmanned Ground Vehicle (UGV)s. To allow motion inside ground environment, these type of robots rely on different kinds of locomotion systems, and therefore can be categorized as:

- *Wheeled mobile robot*: wheels represent one of the most diffused locomotion system for mobile robots. The usage of a wheel instead of other locomotion systems finds its advantages in cheaper cost of implementation and simpler design, as well as simple control of them. On the other hand, wheeled locomotion tends to not perform well in cases where there is low friction

between the wheel and the terrain or when the robot has to overcome obstacles. In literature it is possible to find different type of wheels (see figure 2.2) and they can be arranged in different number and configurations. For thesis purposes, in section 2.2 there will be presented a detailed analysis of one particular type of ground-based wheeled mobile robot;

- *Legged mobile robot*: legged mobile robots (also called walking mobile robots) take inspiration from human motion, and, at the price of higher implementation costs, provides higher adaptability and maneuverability in presence of rough or uneven terrain with respect to the wheel implementation. As for the wheeled locomotion, it is possible to find robot with different number and arrangement of legs, starting from the single-leg robot (also called hopping robot [5]) to robots with six or more legs [2];

- *Tracked mobile robot*: can be used as an alternative to the wheeled locomotion, with high efficiency in environments with low friction or harsh conditions. However, the usage of treads leads to issues when steering, requiring a skid movement to steer the robot. This makes the dead-reckoning of the robot very imprecise and difficult, requiring other systems to provide the correct position of the robot (for example the usage of a GPS system, when available, could solve the problem);

- *Hybrid mobile robot*: to overcome problems in some environments, as well as to use advantages coming from the previously cited locomotion systems, it is possible to find ground mobile robots with hybrid locomotion systems [6].

Air-based

An Unmanned Aerial Vehicle (UAV) (also commonly known as "drone" [30]) is a type of a mobile robot which can operate in the aerial space without human interaction. Their first usage were in military applications but in the recent years they have found applications in other fields, such as photography, commercial, agricultural and so on. It is possible to divide UAVs into:

- *Fixed-wing*: they look and behave like a plane;

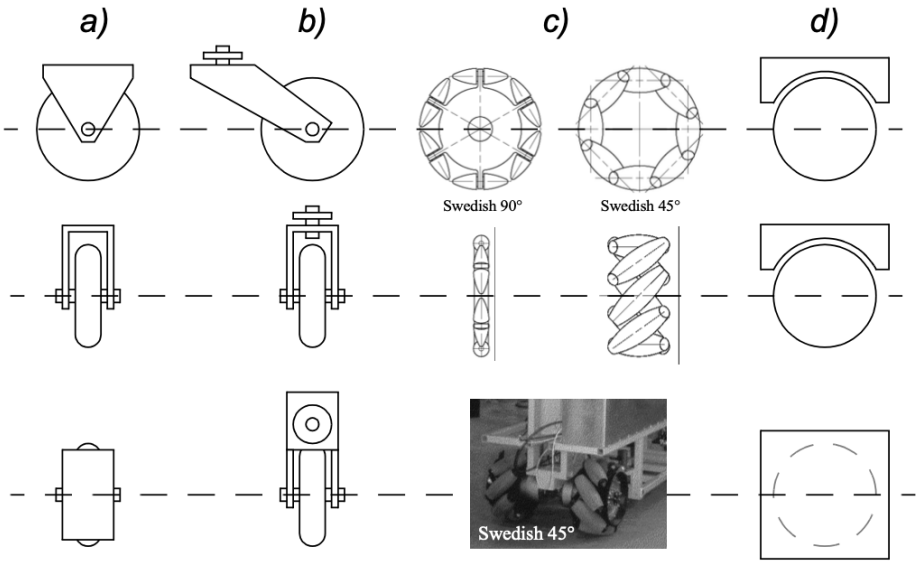


Figure 2.2: Wheels type: (a) Standard wheel. (b) Castor wheel. (c) Swedish wheel. (d) Spherical wheel

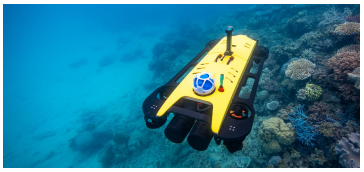
- *Rotor-craft*: similar to helicopters, based on their application they can be found in different sizes and with one to multiple rotors.

Water-based

Underwater environments are one of the most difficult places to reach for man. An important branch of autonomous mobile robotics studies the development of underwater mobile vehicles that allows the exploration of these environments safely. Other than the submerging robots, it is possible to find robots that works on the water surface like boats (they are called unmanned surface vehicles).

(a) Example of UGV

(b) Example of UAV



(c) Example of underwater mobile robot [19]

Figure 2.3: Examples of mobile robots

2.1.2 Perception

For an autonomous mobile robot, the knowledge of the surrounding environment as well as the state of the robot itself are fundamental. Perception tries to solve these problems using different types of sensors and the information that are obtained using them. The usage of sensors allows the mobile robot to determine both its positioning inside an environment and a mapping of the environment. Also, sensors are useful for what concerns the detection and recognition of objects [20]. It is possible to categorize sensors by means of two functional axes. The first one divides sensors by means of *where* measures takes place:

- *Proprioceptive*: measures values internal to the robot, such as joint states, battery voltage, temperature and so on;
- *Exteroceptive*: measures data from the surrounding environment, such as distances, sounds, light measurements and so on. These measures are then processed by the robot to get an crucial environment features.

while the second functional axes instead divides sensors by their working functionality:

- *Active*: works by emitting energy into environment and then measuring the environmental reaction.
- *Passive*: measure environmental energy entering the sensor (for example microphones, CMOS or CCD cameras).

Sensors' performances may vary depending on the environment they are working in. For example, in a laboratory setting, some sensors may provide better results rather then other sensors. It is important to characterize the performances of a sensor by means of basic variables, which are explained in the following:

- *Dynamic range*: it measures the spread between the lower and the upper value measurable by the sensor in normal operating conditions. Dynamic range is also important since many times sensors operate in environments where they are subject to input values beyond their normal working range. In these cases, it is difficult to predict how the sensor will react;

- *Resolution*: represents the minimum difference between two measurable values by a sensor. Typically, resolution coincides with the lower limit of the dynamic range;
- *Linearity*: it governs the behaviour of the sensor's output signal as the input signal varies;
- *Bandwidth*: it measures the speed with which a sensor can provide a stream of readings. For autonomous mobile robots the usage of limited bandwidth sensors could limit the maximum operating speed of the robot, therefore the increase of the bandwidth of sensors is still an active research topic. Formally, the number of measurements per seconds of the sensor is defined as its *frequency* and is measured in Hertz [Hz];
- *Sensitivity*: is defined as the ratio of output change to input change. Sensitivity measures the degree with which a change in the input signal influences and changes the relative output signal.

At last, when dealing with perception and sensors, it is important to take into account sensor errors. *Error* is defined as the difference between sensor's output and the true measured value. It is possible to divide errors into two possible sources:

- *Systematic errors*: these errors depends on factors that theoretically can be modeled (in fact it is possible to predict this kind of errors). Systematic errors affects the *accuracy* of the sensor, which is formally defined as the degree of conformity between sensor's output measurements and the true value;
- *Random errors*: these errors can be described only in a probabilistic way, since it is impossible to define an accurate model that describes them. *Precision*, which is defined as how close are measurements to each other, is affected by random errors.

In literature there is a consistent number of sensors that can be used in a robotic context, for example it is possible to find:

- *encoders*: are used to know the robot's part position and speeds;

- *gyroscopes*: are used to measure angular velocities and orientation of the robot;
- *laser range finder*: it uses a laser beam to generate high-precision distance measurements;
- *vision-based sensors*: they process data from any modality and use the electromagnetic spectrum to produce an image.

2.1.3 Cognition and control

The mechanical structure of an autonomous mobile robot (in particular the locomotion system defined in 2.1.1) requires to be controlled in order to perform correctly tasks.

The control system of an autonomous mobile robot is responsible of this process and usually implements one of the most common used paradigm in autonomous mobile robot applications, the *Sense-Plan-Act* paradigm [11], where:

- the perception system described in 2.1.2, which provides information about the robot and the environment, is in charge of the *sense* part;
- the *cognition* system, which will be described later, is responsible of the *plan* part;
- the locomotion system, with the appropriate commands from the control system, allows the motion of the robot and therefore is in charge of the *act* part.

The planning and decision part of the sense-plan-act paradigm is managed by the cognition system of the robot. Its goal inside the whole architecture is firstly to get the correct interpretation of the data provided by the perception system, and, for this reason, the robot could require a cognitive model of the robot, the environment and of the way they interact. Using the information from the perception system and the high-level objectives of the robot, the cognition system is then able to plan the correct path the robot has to follow to achieve them by taking the appropriate actions. At last, in cooperation with the control system, the cognition system deliberates the suitable commands to the locomotion system allowing the motion of the robot.

Figure 2.4: The Sense-Plan-Act paradigm

2.1.4 Navigation

For an autonomous mobile robot, navigation skills represent one of the most important aspect when designing it. The objective for an autonomous mobile robot is to move from one place to another taking into account not only the final position it has to reach but also all the information received from the sensors about its surroundings. For these reasons, the definition of the correct path to follow, united with the correct representation of the surrounding environment and an appropriate method of avoiding obstacles, are crucial in the design of an autonomous mobile robot.

Localization and mapping

One of the key aspect in navigation is localization. In order to obtain the navigation in an environment, a robot must determine what is its position inside it. On top of that however knowing only the position of the robot may not be enough for navigation. For an autonomous mobile robot, the knowledge and the representation of the surrounding environment plays a major role in navigation. Therefore, it is possible to describe these two main aspects in navigation as:

- *Localization*: it is the exact knowledge of the pose of a mobile robot. In literature, there have been developed various technologies and techniques (odometry, inertial navigation, landmark navigation and so on [4]) to solve the localization problem;

- *Mapping*: the representation of the map should be consistent with the accuracy which the robot is working with but also with the accuracy of the sensors providing information for map reconstruction. As for the localization problem, there have been developed several techniques to solve the map representation problem (for example probabilistic map-based localization, Monte Carlo localization, Landmark-based localization [23]).

Path, trajectory and motion planning

With the localization and mapping, an autonomous mobile robot is able to determine its position inside an environment and at the same time to determine what are its surroundings. To ensure mobility, it is necessary to define how the robot should move inside the environment, namely by defining a path or trajectory to follow. Motion planning tries to solve this problem by defining a series of action that an autonomous mobile robot must follow to move from a starting point to reach the final target. Inside motion planning problem, it is possible to define what are the two main issues to solve for a robot to move:

- *Path planning*: the aim of path planning is to determine what is the best path to follow in order for the robot to reach the final destination without hitting any obstacle, neglecting the temporal evolution and without considering any velocity or acceleration;

- *Trajectory planning*: determines what is the force input necessary to move the actuators of the robot, in order for it to follow a trajectory. Differently from the path planning process, trajectory planning takes into account both the temporal evolution and the forces (and therefore accelerations) needed to obtain the motion of the robot.

Over the years, many motion planning techniques and algorithms have been developed. It is possible to categorize them as:

- *Classic methods*: were the first developed methods in autonomous mobile robotic industry. They are based on roadmaps (for example Voronoi diagram [24] and visibility graphs), cell decomposition [7] and potential functions [15];

- *Probabilistic methods*: some classic methods presented some problems, such as high computational complexity and trapping of the planner in a local minima solution. To overcome these problems, planning methods evolved into "probabilistic methods". PRM (probabilistic roadmap) planners [13] are one of the most diffused probabilistic planning methods. They are based on single-dimensional roadmaps and the goal of the path planner is to determine paths that link the initial and final point to the roadmap;

- *Heuristic methods*: to speed up performances of previous methods, it is possible to adopt heuristic approaches to solve the planning problem. The most commonly used algorithms are A* algorithm [28] and D* algorithm [31];
- *Evolutionary algorithms*: in the recent years this class of algorithms arise in the field of motion planning. Evolutionary algorithms solve the motion problem by mimicking behaviours of living things. Several of the most common used techniques used in recent years are for example Genetic Algorithms (GA) [9], Particle swarm optimization (PSO) [14] and Ant colony optimization [10];
- *Sensor-based algorithms*: as the name mentions, these algorithms work using sensors to perceive surroundings and determine the correct path to follow. These methods have been used over years and continue to evolve in the present. Their main advantage resides in the online computation of the path.

Obstacle avoidance

The last aspect an autonomous mobile robot has to deal with when planning the movement is obstacle avoidance. As already mentioned, the motion of the robot from its current position to a final position requires a map of the environment (which can be known or unknown), the goal location and the robot current position (which is determined using the localization system). Other than that, the motion planner should take into account the presence of any obstacle which could be present in the trajectory and modify it suitably in order to avoid collisions. Obstacle avoidance algorithms can be divided into two main categories:

- *Map based*: these algorithms rely on the knowledge of the surrounding environment provided by a map of it. Then, knowing the localization of the robot at each time instant, its is possible to determine the detect collisions by evaluating distances from obstacles and plan a suitable trajectory;
- *Mapless based*: there is no explicit knowledge of he surrounding environment, which is observed by means of the sensors mounted on the robot.

The position of the obstacles as well as their distances from the actual position of the robot can be retrieved using from the sensors of the robot. In this way, the path planner can take into account any obstacle in the planning process and modify the final trajectory accordingly in order to avoid the obstacles. Over the years, there have been developed several techniques to solve the obstacle avoidance problem, such as the *Bug algorithm* [22], *vector field histogram* [3], *potential field methods* [17].

2.2 Unmanned Ground Vehicle (UGV)

As introduced in section 2.1.1, an UGV is an autonomous mobile robot which is specialized in moving in different ground terrains. For the purposes of this thesis, there will be given a brief description of a model that can be used for UGV as well as the control techniques used to drive it.

2.2.1 Representation of an Unmanned Ground Vehicle (UGV)

An UGV which moves inside a 2D environment can be considered as a rigid body whose pose can be described by means of two different reference frames:

- world reference frame ($\mathcal{F}_w$) which is inertial, fixed and could be known or unknown;
- body reference frame ($\mathcal{F}_b$) whose origin typically coincides with the center of mass of the robot.

Considering these two reference frames, it is possible to express the pose of an UGV in the world reference frame using the vector:

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \theta \end{bmatrix} \in \mathbb{R}^3 \tag{2.1}$$

where, the first two components of the vector $\mathbf{q}$ ($\mathbf{p} = [x, y]^T \in \mathbb{R}^2$) represents the position of the robot in Cartesian coordinates, while the third component ($\theta \in (-\pi, \pi], \theta \in \mathbb{S}^1$) accounts for the relative orientation of the robot with respect ot the world reference frame.

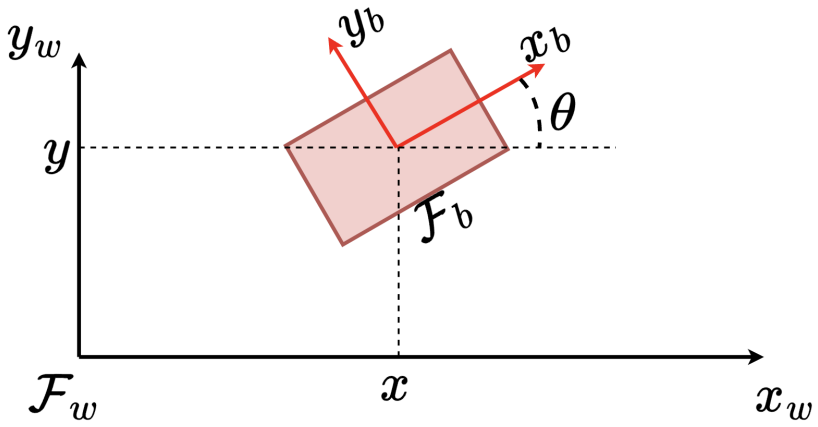


Figure 2.5: Representation of world and body reference frames

2.2.2 Unicycle model and Differential Drive Robot (DDR)

In order to obtain a simple model of an UGV, there will be considered the unicycle vehicle. The unicycle is a planar vehicle built with a single orientable wheel, whose pose is represented using equation (2.1). The determination of the kinematic model of the unicycle starts by relating the translational and rotational coordinates using the following equations:

$$\dot{x} = v\cos\theta \tag{2.2}$$

$$\dot{y} = v\sin\theta \tag{2.3}$$

$$v^2 = \dot{x}^2 + \dot{y}^2 \tag{2.4}$$

where v is the linear velocity of the UGV. By combining equations (2.2), (2.3) and (2.4) it is possible to define the non-holonomic pure rolling constraint affecting the UGV in the form $\Phi(\mathbf{q}, \dot{\mathbf{q}}) = 0$:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{2.5}$$

At last, by taking into account also the angular velocity of the unicycle (ω), it is possible to define the kinematic model of the UGV as:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta\\ \sin\theta\\ 0\end{bmatrix} v + \begin{bmatrix}0\\ 0\\ 1\end{bmatrix} \omega = \begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix} \begin{bmatrix}v\\ \omega\end{bmatrix} \tag{2.6}$$

Differential Drive Robot (DDR)

A Differential Drive Robot (DDR) is a particular type of UGV which is composed by two standard wheels which are responsible for the motion of the robot and typically one or two wheels (which can be spherical or castor) used for stability purposes. The assumption of a pure rolling movement without any slide causes the wheels to describe always arcs in the plane, in such a way that the robot moves around a point which is defined as Instantaneous Center of Rotation (ICR). If both driving wheels have the same velocity (namely ω_l and ω_r for left and right wheel) the ICR is placed at infinity, on the other hand if wheels have different velocities then the robot will describe a circular trajectory. For a DDR it is possible to derive the control inputs of the robot, namely the driving (linear) speed v and the steering (angular) velocity ω from the wheels velocity as:

$$v = \frac{r(\omega_r + \omega_l)}{2} \qquad \omega = \frac{r(\omega_r - \omega_l)}{d} \tag{2.7}$$

where r represents the radius of the driving wheels and d is the distance between them. In this way, it is possible to adopt the same model defined for the unicycle also for the DDR.

2.2.3 Unicycle control

For the unicycle, there are introduced and presented three different types of control:

Cartesian control

With the Cartesian control, the objective is to drive the unicycle to a desired position $\mathbf{p}^{des} = [x^{des}, y^{des}]^T$ regardless of the orientation θ. Without loss of generality, it is possible to consider the following as the desired target pose:

$$\mathbf{q}^{des} = \begin{bmatrix} \mathbf{p}^{des} \\ \forall \end{bmatrix} = \begin{bmatrix} x^{des} \\ y^{des} \\ \forall \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \forall \end{bmatrix} \tag{2.8}$$

The error with respect to the current position $\mathbf{q} = [x, y]^T$, $\mathbf{e}_p$, as well as the unit vector sagittal axis $\mathbf{n}$, are defined in equation (2.9):

$$\mathbf{e}_p = \begin{bmatrix} 0 - x \\ 0 - y \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix} \qquad \mathbf{n} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{2.9}$$

In figure 2.6, it is reported the graphical meaning of equation (2.9). From that, it is possible to define the projection of the error on the sagittal axis and the angles presented in figure 2.6 as:

$$< \mathbf{e}_p, \mathbf{n} >= \mathbf{e}_p^T \mathbf{n} = -x\cos\theta - y\sin\theta \tag{2.10}$$

$$\alpha = \text{atan2}(y, x), \quad \delta = \alpha + \pi \tag{2.11}$$

and at last, the angle between the position error vector and the sagittal axis is defined as:

$$\gamma = \delta - \theta = \text{atan2}(y, x) + \pi - \theta \tag{2.12}$$

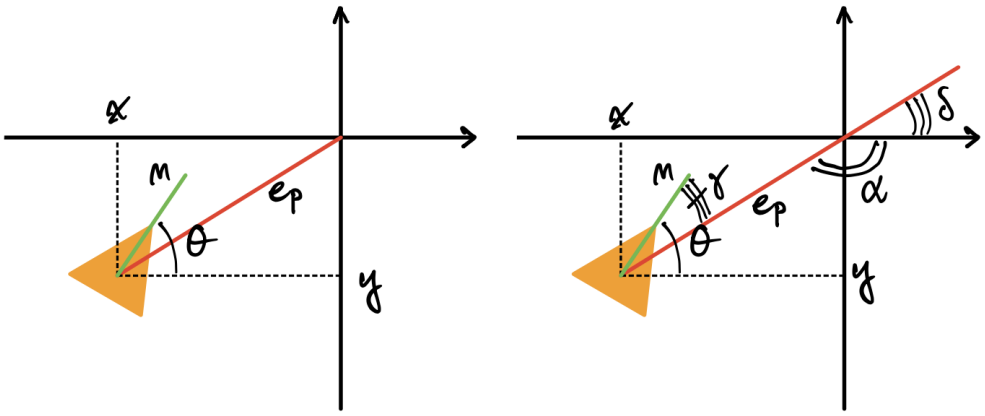


Figure 2.6: Cartesian control: angles and vectors definition

The goal of the control action is to bring to zero both the projection error defined in equation (2.10) and the angle defined in equation (2.12). To achieve these results, it is possible to adopt the following control law acting on both the

driving and steering velocities:

$$v = k_v \mathbf{e}_p^T \mathbf{n} = k_v(-x\cos\theta - y\sin\theta) \tag{2.13}$$

$$\omega = k_\omega \gamma = k_\omega(\text{atan2}(y, x) + \pi - \theta) \tag{2.14}$$

where k_v and k_ω are positive control gains.

Posture control

Differently from the Cartesian control, in posture control also the final orientation of the robot is important. Therefore, without loss of generality, the desired target pose is defined as:

$$\mathbf{q}^{des} = \begin{bmatrix} \mathbf{p}^{des} \\ \theta^{des} \end{bmatrix} = \begin{bmatrix} x^{des} \\ y^{des} \\ \theta^{des} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.15}$$

By considering the length of the position error e_p defined in equation (2.9) $\rho = \sqrt{x^2 + y^2}$, united with the angles γ and δ defined respectively in equations (2.11) and (2.12), it is possible to give a polar representation of the unicycle pose. The aim of the control action is to bring those quantities to zero. One possible control law is:

$$v = k_v \rho \cos\gamma \tag{2.16}$$

$$\omega = k_\omega \gamma + k_v \frac{\sin\gamma\cos\gamma}{\gamma}(\gamma + k_\delta \delta) \tag{2.17}$$

where again the parameters k_v, k_ω and k_δ are the positive control gains of the controller.

Tracking control

The aim of this controller is to drive the unicycle so that it follows a trajectory over time. The desired trajectory is expressed as:

$$\mathbf{q}^{des}(t) = \begin{bmatrix} \mathbf{p}^{des}(t) \\ \theta^{des}(t) \end{bmatrix} = \begin{bmatrix} x^{des}(t) \\ y^{des}(t) \\ \theta^{des}(t) \end{bmatrix} \tag{2.18}$$

while, the trajectory error can be expressed both in the world reference frame and in the body reference frame as:

$$\mathbf{e}_W = \begin{bmatrix} x^{des} - x \\ y^{des} - y \\ \theta^{des} - \theta \end{bmatrix} \quad \Rightarrow \quad \mathbf{e}_B = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \mathbf{R}_z^T(\theta)\mathbf{e}_W = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{e}_W \tag{2.19}$$

Considering now the derivative of equation (2.19), united with the invertible input transformation:

$$\begin{aligned} v &= v^{des}\cos e_3 - u_1 \\ \omega &= \omega^{des} - u_2 \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} u_1 &= -v + v^{des}\cos e_3 \\ u_2 &= \omega^{des} - \omega \end{aligned} \tag{2.20}$$

it holds that:

$$\dot{\mathbf{e}} = \begin{bmatrix} 0 & \omega^{des} & 0 \\ -\omega^{des} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e} + \begin{bmatrix} 0 \\ \sin e_3 \\ 0 \end{bmatrix} v^{des} + \begin{bmatrix} 1 & -e_2 \\ 0 & e_1 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.21}$$

At last, it is possible to linearize the error dynamics described in equation (2.21) around the point $\mathbf{e} = 0$:

$$\dot{\mathbf{e}} = \begin{bmatrix} 0 & \omega^{des} & 0 \\ -\omega^{des} & 0 & v^{des} \\ 0 & 0 & 0 \end{bmatrix} \mathbf{e} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.22}$$

and a possible control law is:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} -k_1 & 0 & 0 \\ 0 & -k_2 & -k_3 \end{bmatrix} \mathbf{e} \tag{2.23}$$

Using this control law makes the closed-loop error dynamics become:

$$\dot{\mathbf{e}} = \begin{bmatrix} -k_1 & \omega^{des} & 0 \\ -\omega^{des} & 0 & v^{des} \\ 0 & -k_2 & -k_3 \end{bmatrix} \mathbf{e} \tag{2.24}$$

where the positive control gains k_1,k_2 and k_3 can be selected such that the error dynamics asymptotically convergences to zero.

3

Autonomous navigation - NAPVIG

In section 2.1.4 it was introduced the concept of autonomous navigation for an autonomous mobile robot, as well as the main components of a navigation system and the principal algorithms and techniques that find solution to this problem. In this chapter, there is presented an innovative algorithm for the autonomous navigation called Narrow Passage Navigation (NAPVIG) [18].

3.1 Introduction

As seen in section 2.1.4, the path planning problem presents different ways to solve it. Despite the various number of techniques presented, nowadays the solutions depend mainly on the application of *graph-search methods* and *sensor-based algorithms*. For what concern sensor-based methods, the Rapidly-exploring Random Trees (RRT) [16] and its derivation (for example RRT* [12]) can be found nearly in most of the scenarios. On the other hand, graph-search methods like A* [28] are commonly adopted especially when dealing with unknown environments.

Despite their efficiency in finding a solution to the navigation problem, both methods tends to be computationally highly demanding in cases of complex scenarios and in presence of obstructed or cluttered passages. Moreover, when dealing with unknown or dynamic environments, real-time requirements are a challenging aspect to deal with, and the aforementioned methods, due to their working process, are not well suited for reactive navigation.

Reactive navigation is a diffused paradigm for controlling an autonomous mobile robot in unknown, dynamic and cluttered environments, in which the designed algorithm has to adapt to the variations around the robot making use of any priori global information and using the information provided by the sensors of the robot. Typically, in a reactive approach, the goal is to determine only the next command of the robot instead of computing all the path to the final target. The computation of the command starts by specifying a cost function for all the possible movements that the robot can achieve in terms of the surrounding obstacles and, if present, the desired target position. At each time instant, the algorithm looks for the minimum in the cost function and executes the command associated to it. As mentioned before, in order to ensure real-time performances and allow the robot to be fast enough to the environment changes, in the evaluation of the command are considered only local information over a short period of time.

In this context, the NAPVIG algorithm deals with the reactive navigation paradigm by computing a single point of the local Generalized Voronoi Diagram (GVD) of the environment, which is obtained from the raw measurements of a LiDAR scanner. The algorithm is written using C++ and Python languages and is implemented using ROS (Robot Operating System), which is a set of software libraries and tools widely used in robotic applications.

3.2 Algorithm description

The main goal of the NAPVIG algorithm, in order to achieve safe navigation, is to determine a trajectory which is at maximum distance for every possible obstacle in the movement direction of the robot. Due to its low computational costs, it is possible to utilize this algorithm in environments where high reactivity is required. As mentioned at the end of section 3.1, NAPVIG works using the noisy data measurements provided by a LiDAR sensor installed on the robot, which are collected with a certain sampling time $T_m \in \mathbb{R}$. All the measurements are expressed in a frame $\mathcal{F}_k$, while the pose of the robot at each time instant is expressed by the frame $\mathcal{F}_t$ with respect to the inertial world frame $\mathcal{F}_w$. The measurements collected are used by NAPVIG to define what constitutes one of the three pillars of which the algorithm is based on, the *landscape function*.

The landscape is a function whose goal is to map every point in $\mathbb{R}^2$ accordingly to a value related to the distance of the nearest measurement provided by

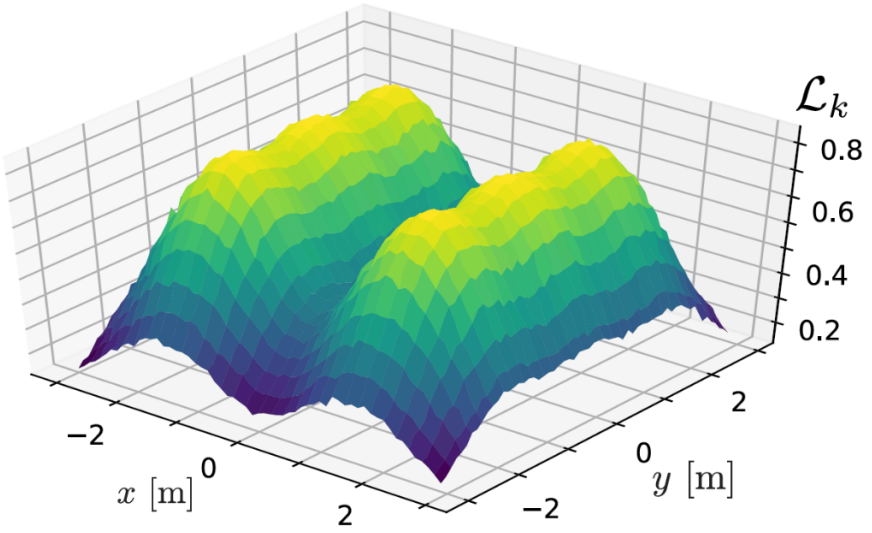


Figure 3.1: Example of a landscape function [18]. The yellow zone represents the obstacles, while the blue areas are the safe zone to navigate in which the trajectory will be computed

the LiDAR. The evaluation of the landscape function starts from the computation of Gaussian-like function centered on each point of the measurements m_k at time instant t, as:

$$\Gamma : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow (0, 1] : (\mathbf{x}, \mathbf{m}) \rightarrow e^{-\frac{||\mathbf{x}-\mathbf{m}||^2}{2\sigma^2}} \tag{3.1}$$

The superposition the Gaussian peaks defined in equation (3.1) allows the definition of the *raw landscape function* $\check{\mathcal{L}}_a$. To obtain the real landscape function, the raw landscape function $\check{\mathcal{L}}_a$ is convoluted with a Gaussian kernel, getting therefore a smooth version of it which is then used in the algorithm. The obtained function gives a proportional representation of the surrounding space around the robot, with higher values corresponding to obstacles and lower values to the free space where the robot navigation is safe (in fact the robot moves in the minimum of the landscape function).

The second key aspect in the NAPVIG algorithm are *landmarks*. They are used to keep track of the movements made by the robot by storing useful information about areas that have been already explored. A landmark is expressed using the triplet $l = (\mathcal{F}_l, t_l, \mathbf{x}_l) \in \mathbb{SE}(2) \times \mathbb{R} \times \mathbb{R}^2$, where:

- $\mathcal{F}_l \in \mathbb{SE}(2)$ is the last measurement frame where the landmark is created;

- $t_l \in \mathbb{R}$ is the timestamp when the landmark is created;
- $\mathbf{x}_l \in \mathbb{R}^2$ is the position of the robot at the landmark creation, expressed in the frame $\mathcal{F}_l$.

Landmarks are stored in a limited size batch which is treated as a First In Last Out (FILO) queue, so that the oldest landmarks are replaced by the newest and useful ones. Each landmark is used to evaluate a cost factor that penalizes the navigation inside areas that have been already explored and encourages therefore the exploration of different paths. The presence of the decaying factor $e^{-\lambda(t-t_l)}$ in the evaluation of the cost factor takes into account the timestamp in which the landmark had been created. This is done to allow the exploration of already visited zones in a different timestamp, which results helpful especially when dealing with dynamic scenarios (for example the removal of an obstacle in a certain zone could make easier the reaching of the target through that specific path).

The third and last key aspect for NAPVIG is the target position. Target is a position $x_f(t) \in C_{free}$, where $C_{free} \subset \mathbb{R}^2$ is the complementary set of $C_{coll} \subset \mathbb{R}^2$, the set of all the possible collisions in the environment. Target plays a key role in the working functionality of the NAPVIG algorithm, especially in the policy switching phase as will be discussed later.

All the components described so far are necessary to determine the safest trajectory (the one which is farthest from each possible obstacle) for the robot to follow during the navigation. Moreover, it is important to point out that the trajectory computed by NAPVIG always follows the GVD of the environment. The GVD can be defined as the set of points which are equidistant from two or more obstacles and it divides the space into cells defined Generalized Voronoi Cells (GVCs). Each cell contains exactly one object (or seed) and the set of points composing the GVC is such that each point of the cell is closer to the seed rather than any other point in the space.

At last, the NAPVIG algorithm decides the trajectory to follow based on a policy-switching method. The method is based on six different policies which can be regrouped into three main classes: *predictive*, *reactive* and *auxiliary*. Depending on the status of the robot, each policy plays a specific role in the moment of the computation of the trajectory. The policy-switching criterion is not discussed but there is given only a brief description of each one of the six policies used in the algorithm.

Predictive policies

Predictive policies are responsible for the navigation task. Each one of these policies can predict one or more trajectories to follow $\xi^{(h)}$, which are associated to an exit status $c_{\xi}^{(h)}$ and a cost value $J(\xi^h, c_{\xi}^{(h)})$. The decision of the best trajectory to follow then relies on the minimization of the following optimization problem:

$$\xi^* = \underset{\xi^{(h)}, h=0,\dots,H}{\arg\min} \; J(\xi^{(h)}, c_{\xi}^{(h)}) \tag{3.2}$$

The NAPVIG algorithm makes use of three predictive policies:

- *Fully-exploitative policy*: generates a trajectory that points directly towards the target. It is activated when it is possible to determine a direct path to the target, however, while the navigation could be very efficient, the trajectory computed might not be the safest or in the worst case be a valid one;

- *Fully-explorative policy*: it is used when the target is not in sight and the goal of the navigation task is to explore as much map as possible. This policy makes large use of the landmarks by applying a penalty factor in the evaluation of the trajectory to areas that have been already explored by the robot and encouraging the exploration of new areas of the map;

- *Partly-exploitative policy*: is used when the target is in sight but there is no direct road towards it, therefore the robot is required to explore the map in order to find a free path that reaches the target.

Reactive policy

Predictive policies are used to determine the best trajectory in cases where more than one GVD branches are available (for example the presence of a bifurcation induces two possible trajectories). After choosing the path to follow, it could be unnecessary to compute again all the possible routes toward the target with the robot being required to follow only one direction. In these cases, the *legacy* policy is adopted.

Auxiliary policies

The remaining two policies are used in special states for the robot:

- *Free-space policy*: is used when the robot is sufficiently near to the target, where it is possible to consider the space around it with a certain threshold all safe to navigate;
- *Halt policy*: is used when the robot is required to stop, for example in cases where all the other policies have been tried and rejected or when the robot reached its final destination.

3.3 Odometric position estimation algorithm

In section 2.1.4 there was introduced the localization problem for an autonomous mobile robot. In the experimental setup for NAPVIG, the robot has no knowledge at all of its position inside the global reference frame $\mathcal{F}_w$, and the localization of the robot is entrusted other localization mechanism such as a motion capture system [21]. This setup works finely only in a restricted environment such as laboratories, however, in the perspective of an usage of the robot in uncontrolled environment, the usage of a motion capture system is nearly impossible. Therefore, in order to extend the motion capabilities of the robot, it was decided to implement a simple odometric position estimation algorithm which exploits the characteristics of the DDR to retrieve the pose of the robot, as well as the travelled path, from its initial pose.

3.3.1 Algorithm description

For a DDR, it is possible to estimate its pose $\mathbf{q} = [x, y, \vartheta]^T$ using an iterative odometric position estimation algorithm which makes use of the distance traveled by each wheel. The rotation of each wheel (that will be denoted as $\Delta\vartheta_l$ and $\Delta\vartheta_r$ for left and right wheel respectively) is provided at each time instant Δt by the optical wheel encoders mounted on each driving wheel. The odometric position estimation algorithm determines the pose of the robot by making use

of the incremental travel distance evaluated at each time instant, which can be determined starting from the rotation of each wheel as:

$$\Delta s = r\frac{\Delta\vartheta_r + \Delta\vartheta_l}{2} \tag{3.3}$$

$$\Delta\vartheta = r\frac{\Delta\vartheta_r - \Delta\vartheta_l}{d} \tag{3.4}$$

where the quantities d and r are respectively the distance between the two driving wheels and the radius of each wheel. The quantities Δs and $\Delta\vartheta$ are the total linear and angular displacement of the robot and are expressed with respect to the center point between the two driving wheels of the DDR. The incremental travel distance for the x and y axis can be determined from equation (3.3) as:

$$\Delta x = \Delta s \cos(\vartheta + \Delta\vartheta/2) \tag{3.5}$$

$$\Delta y = \Delta s \sin(\vartheta + \Delta\vartheta/2) \tag{3.6}$$

and finally, the updated pose of the robot is determined as:

$$\mathbf{q}' = \begin{bmatrix} x' \\ y' \\ \vartheta' \end{bmatrix} = \mathbf{q} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\vartheta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \vartheta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\vartheta + \Delta\vartheta/2) \\ \Delta s \sin(\vartheta + \Delta\vartheta/2) \\ \Delta\vartheta \end{bmatrix} \tag{3.7}$$

With this algorithm it is possible to get a simple estimation of the pose of the robot, however with this method the estimate obtained is quite rough and could presents some error in the final position of the robot. Some of the causes of the errors can be identified in:

- *limited resolution during integration*;
- *unequal floor contact*;
- *misalignment of wheels*;
- *uncertainty in wheels dimension*.

To overcome these errors it could be possible to define a more accurate model, however, for the purposes of this thesis, the estimate provided with this simple method works finely. To prove this, in section 3.3.2 are reported

the results obtained from a campaign of experiments conducted in a simulated environment.

3.3.2 Simulation results

For what concerns the simulations, the test of the odometric position estimation algorithm were performed using Gazebo, which is an open-source 3D robotic simulator. The robot in use is a modified version of the "Locobot_wx200" provided by *Trossen Robotics*. The modifications made will be discussed later in chapter 4 since they are not useful for the tests performed in this section. The efficiency of the algorithm was tested by making the robot follow some trajectories, obtained by applying directly to it different linear and angular velocities. All the tests were performed in a completely free environment without any kind of obstacle. The tested trajectories are:

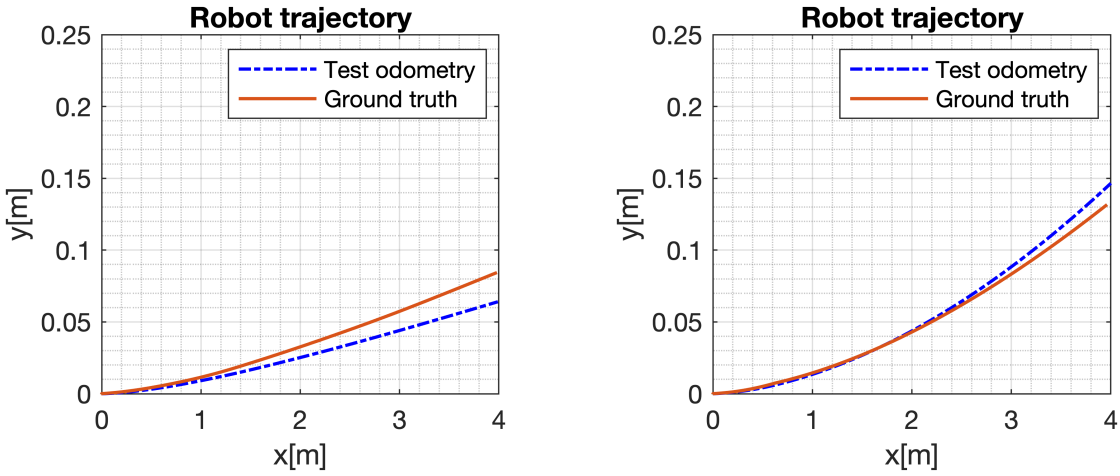
- Linear trajectory (application of linear velocity only);
- Rotation on axis (application of angular velocity only);
- Circular trajectory (application of both linear and angular velocity);
- Random trajectories obtained by the application of linear and angular velocity in different ways.

To prove the goodness of the algorithm, the results obtained are put in comparison with the odometric position estimation provided by Gazebo itself.

Linear trajectory

The first tests were performed by applying to the robot a constant linear velocity, obtaining therefore ideally a linear trajectory until the robot reached approximately the final position $\mathbf{p} = [4, 0]^T$. From the results reported in figure 3.2, the robot starts drifting from the expected perfectly linear trajectory, with a more marked effect as speed grows. However, this drifting phenomenon in the evaluation of the effectiveness of the odometric position estimation algorithm is out of concern since it is caused by other problems related to the robot itself (e.g. slippage of the wheels).

By taking a look at the error graphs reported in figure 3.3, the error growth rate can be considered acceptable when considering all the possible sources



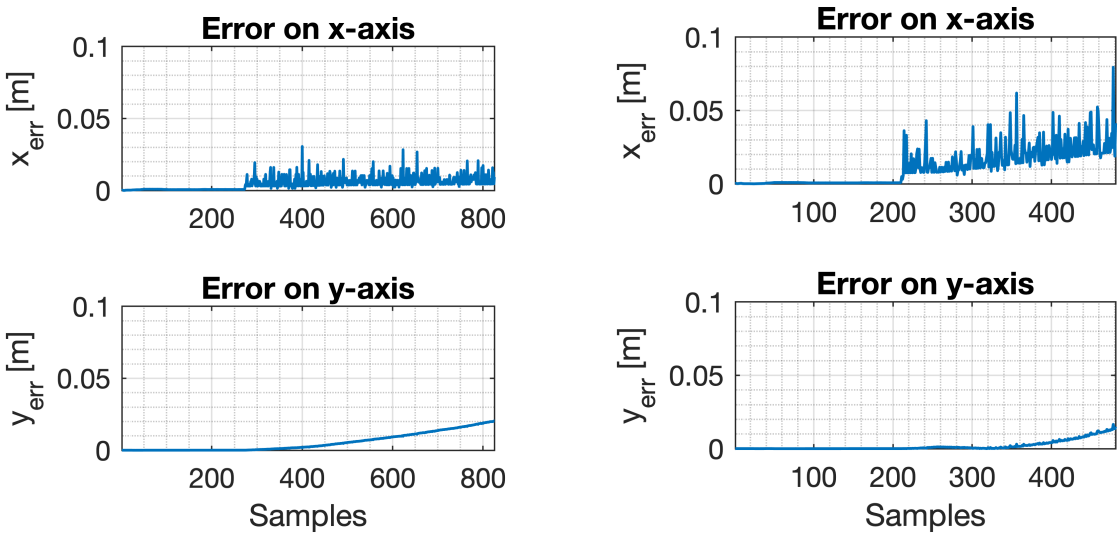
(a) $v = 0.1\,[m/s]$ (b) $v = 0.2\,[m/s]$

Figure 3.2: Results with constant linear velocity

of errors mentioned in section 3.3.1. The detail of the error obtained in the estimation of the final position is reported in table 3.1.

$v\,[m/s]$	$x_{err}\,[m]$	$y_{err}\,[m]$	Euclidean distance between endpoints $[m]$
0.1	0.0093	0.0203	0.0224
0.2	0.0413	0.0150	0.0439

Table 3.1: Errors in final position with constant linear velocity



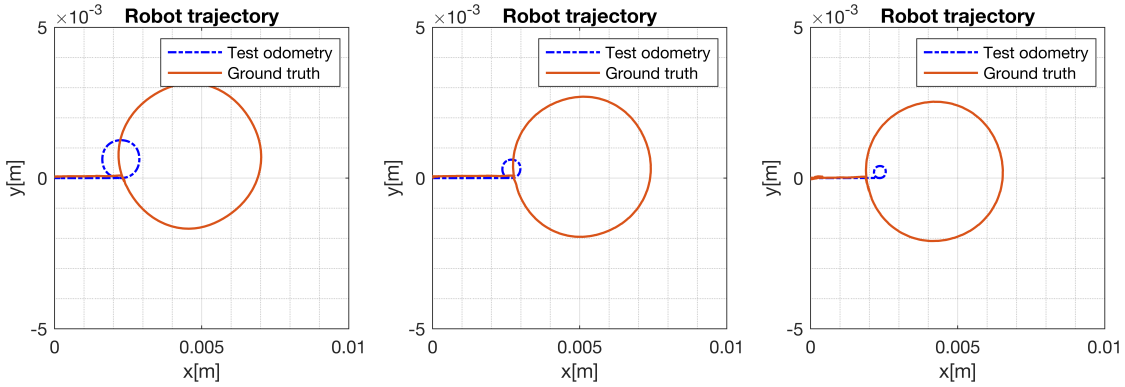
(a) $v = 0.1\,[m/s]$ (b) $v = 0.2\,[m/s]$

Figure 3.3: Error with constant linear velocity

Rotation on axis

The second type of tests were performed by applying a constant angular velocity to the robot, to evaluate its behaviour with respect to the rotation on its vertical axis.

As shown in the results reported in figure 3.4, the odometric position estimation algorithm provides good results in comparison with the ground truth, with a difference in the order of millimeters. This difference can be imputed to an incorrect definition of the robot parameters (wheel distance and radius), united also with errors in the measurements. Also, in each test there is the presence of a small linear displacement from the starting position. However, this problem has to be imputed to the simulation itself, since at the spawn of the robot in the environment it starts moving slightly from the start position.



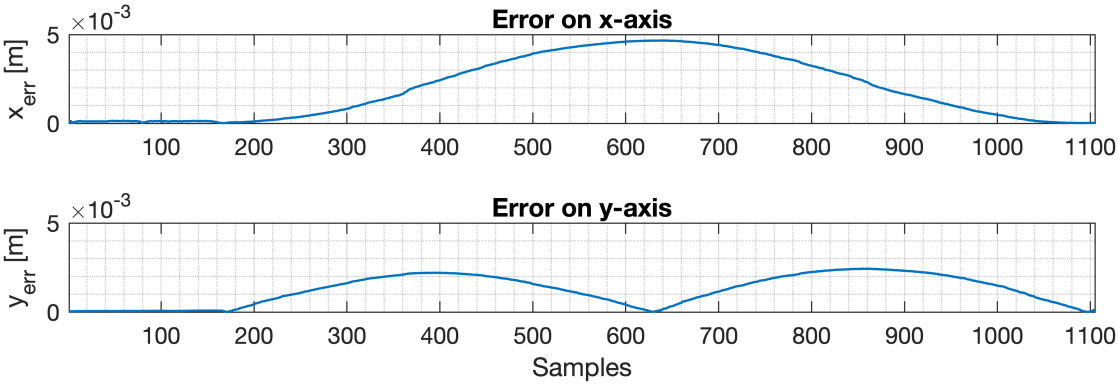
(a) $\omega = 0.1\,[rad/s]$ (b) $\omega = 0.2\,[rad/s]$ (c) $\omega = 0.3\,[rad/s]$

Figure 3.4: Results with constant angular velocity

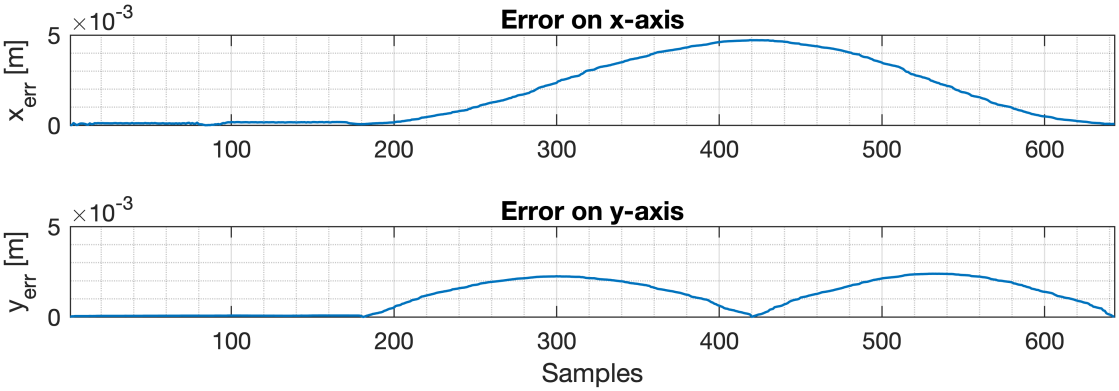
As done for the tests with only linear velocity applied, the effectiveness of the odometric position estimation algorithm is evaluated by taking a look at the error graphs reported in figure 3.5 and at the error in the final position reported in table 3.2. Differently from the linear case, the error assumes here a different behaviour which is however consistent with the trajectory described. To eliminate the misalignment between the odometric position estimation algorithm and the ground truth, one possible solution is to modify slightly the parameters used in the odometric position estimation (wheel distance and wheel radius). This however would influence the estimation process with other trajectories leading to higher errors in the final position, therefore since the error obtained is under the order of millimeters (as reported in table 3.2) it was decided to keep the estimation parameters fixed.

$\omega\,[rad/s]$	$x_{err}\,[m]$	$y_{err}\,[m]$	Euclidean distance between endpoints $[m]$
0.1	0.30e-04	1.21e-04	1.24e-04
0.2	0.80e-04	0.37e-04	0.88e-04
0.3	4,19e-04	4.42e-04	6.09e-04

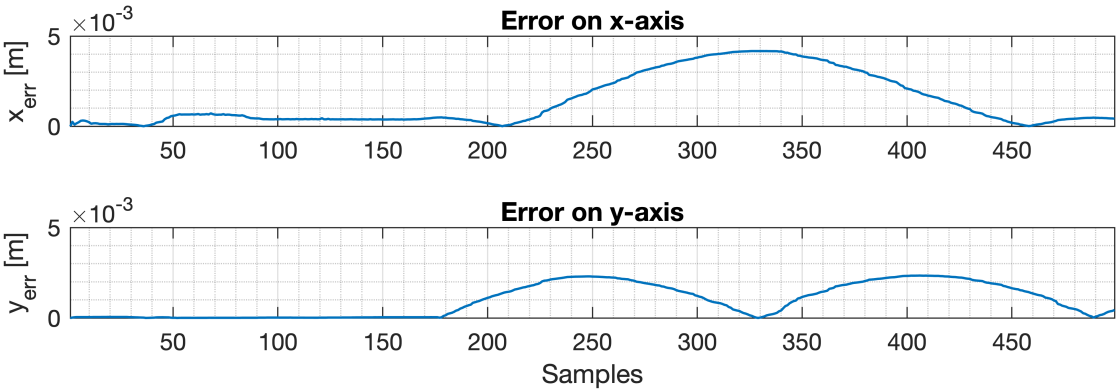
Table 3.2: Errors in final position with constant angular velocity



(a) $\omega = 0.1\,[rad/s]$



(b) $\omega = 0.2\,[rad/s]$

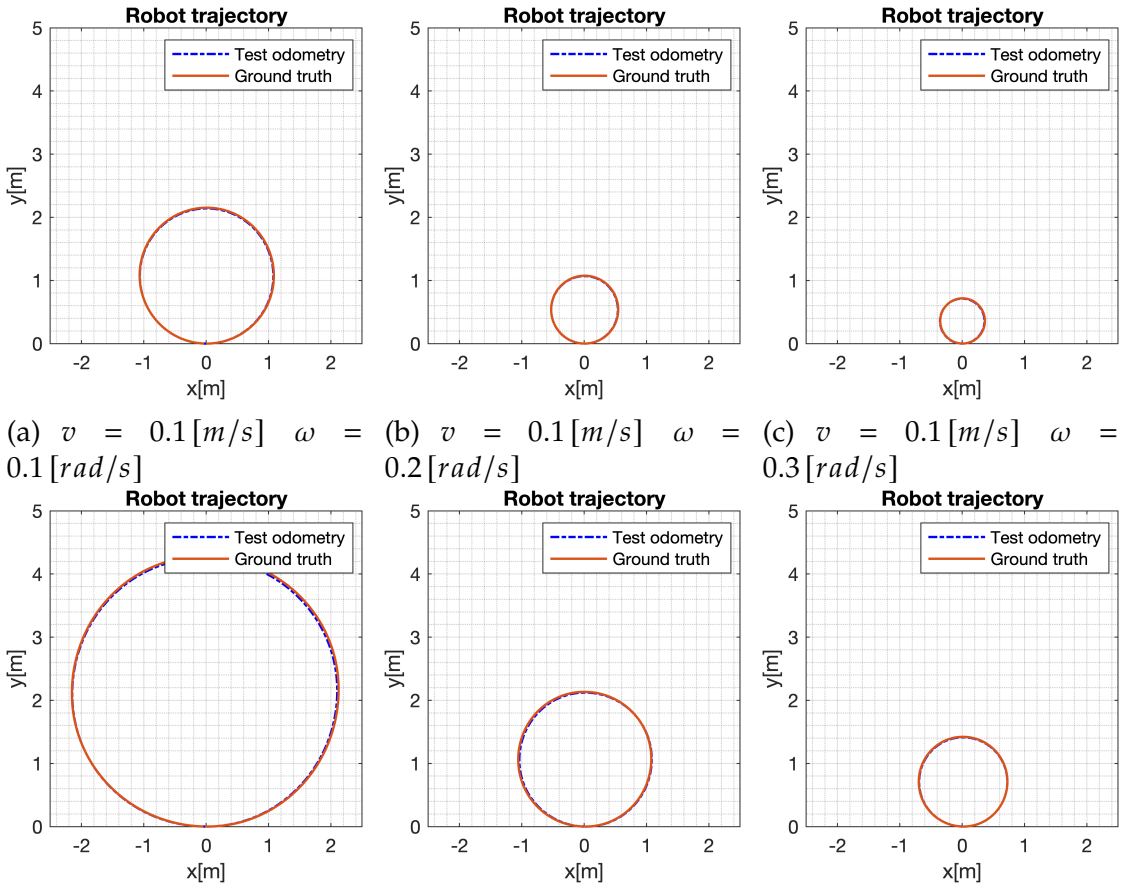


(c) $\omega = 0.3\,[rad/s]$

Figure 3.5: Error with constant angular velocity

Circular trajectory

The third typology of tests were performed by applying to the robot a constant linear and angular velocity simultaneously, getting perhaps a circular trajectory. There were tested out every possible combination of the velocities used in the test of linear and angular velocity taken singularly. The results obtained, reported in figure 3.6.



(a) $v = 0.1\,[m/s]$ $\omega = 0.1\,[rad/s]$ (b) $v = 0.1\,[m/s]$ $\omega = 0.2\,[rad/s]$ (c) $v = 0.1\,[m/s]$ $\omega = 0.3\,[rad/s]$

(d) $v = 0.2\,[m/s]$ $\omega = 0.1\,[rad/s]$ (e) $v = 0.2\,[m/s]$ $\omega = 0.2\,[rad/s]$ (f) $v = 0.2\,[m/s]$ $\omega = 0.3\,[rad/s]$

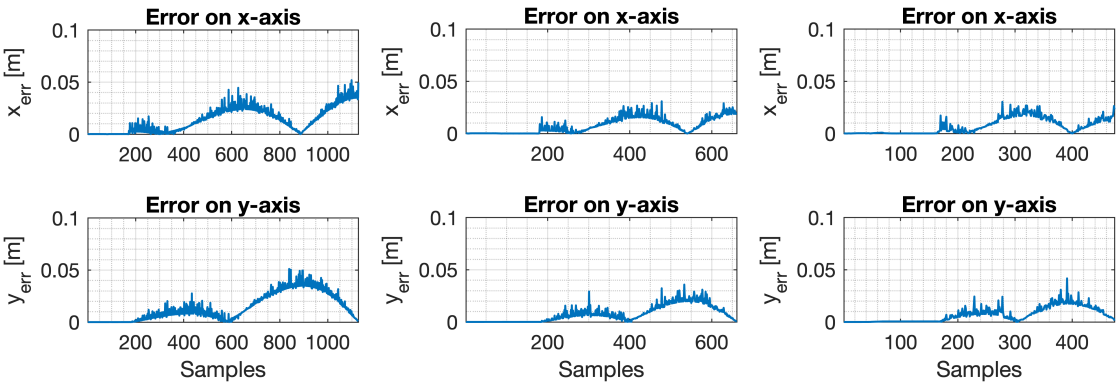
Figure 3.6: Results with circular trajectory

As done for the previous tests, there are reported in figure 3.7 the error graphs obtained. The shape of the error is nearly similar to the ones reported in figure 3.5, but with these simulations, the error obtained in the final position is much higher. This is related directly to the fact that the circular trajectory is obtained as superposition of both an angular and linear speed, and the presence of the last one causes an higher error in the final positioning. Table 3.3 reports the error in final position for each test case. As for the previous case, the error

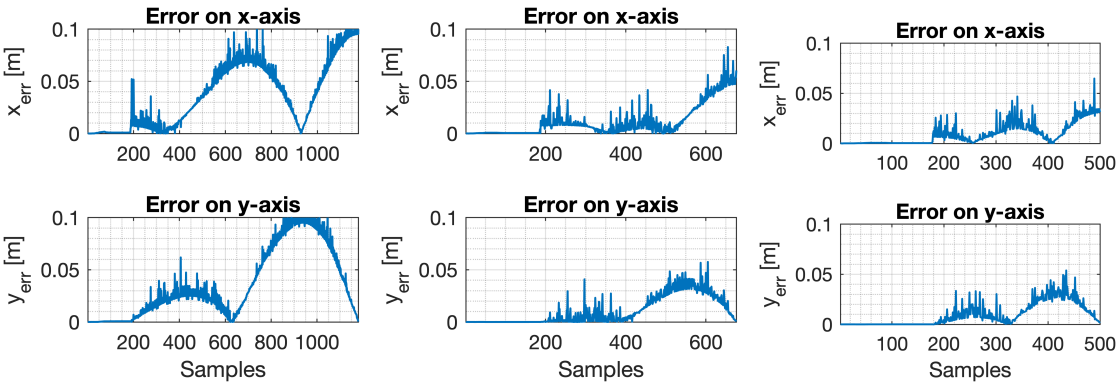
obtained for our purposes is acceptable, even for the cases where the error is above 5cm.

$v\,[m/s]$	$\omega\,[rad/s]$	$x_{err}\,[m]$	$y_{err}\,[m]$	Euclidean distance between endpoints$[m]$
0.1	0.1	0.0317	0.0003	0.0317
0.2	0.1	0.0314	0.0001	0.0314
0.1	0.2	0.0138	0.0003	0.0138
0.2	0.2	0.0604	0.0004	0.0604
0.1	0.3	0.0232	0.0012	0.0233
0.2	0.3	0.0246	0.0012	0.0247

Table 3.3: Error in final position with circular trajectory



(a) $v = 0.1\,[m/s]$ $\omega = 0.1\,[rad/s]$ (b) $v = 0.1\,[m/s]$ $\omega = 0.2\,[rad/s]$ (c) $v = 0.1\,[m/s]$ $\omega = 0.3\,[rad/s]$

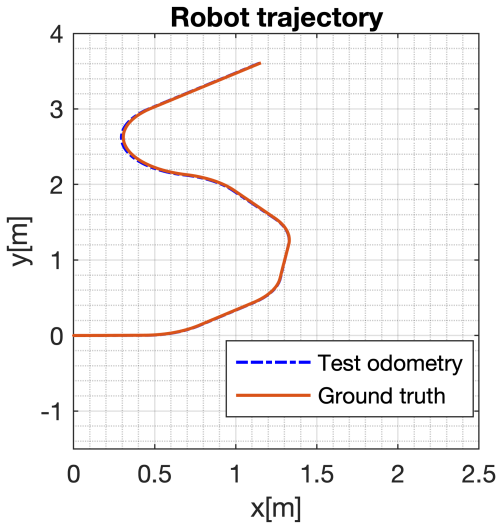


(d) $v = 0.2\,[m/s]$ $\omega = 0.1\,[rad/s]$ (e) $v = 0.2\,[m/s]$ $\omega = 0.2\,[rad/s]$ (f) $v = 0.2\,[m/s]$ $\omega = 0.3\,[rad/s]$

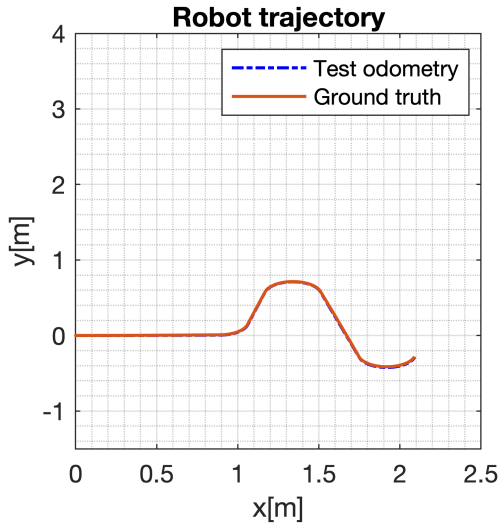
Figure 3.7: Error with circular trajectory

Random trajectory

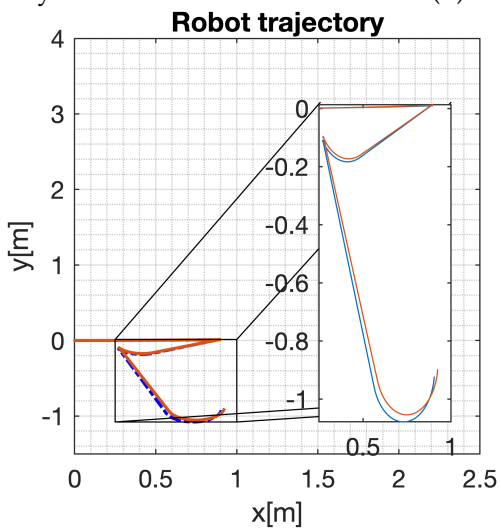
The last type of tests were performed by applying to the robot a random combination of both linear and angular velocities to test the odometric position estimation algorithm performances with different trajectories.



(a) First trajectory



(b) Second trajectory

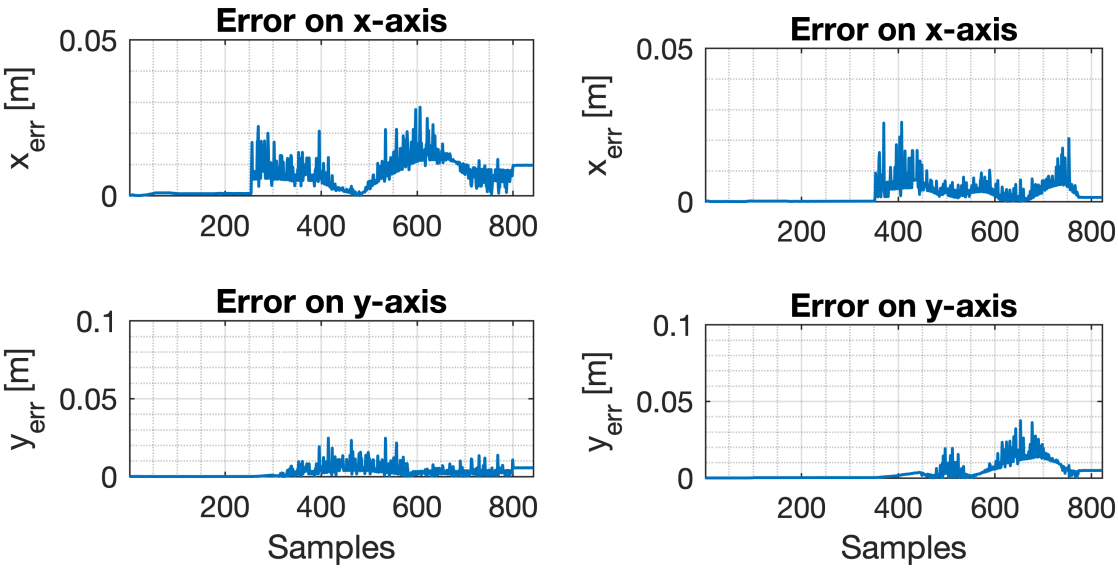


(c) Third trajectory

Figure 3.8: Results with random trajectory

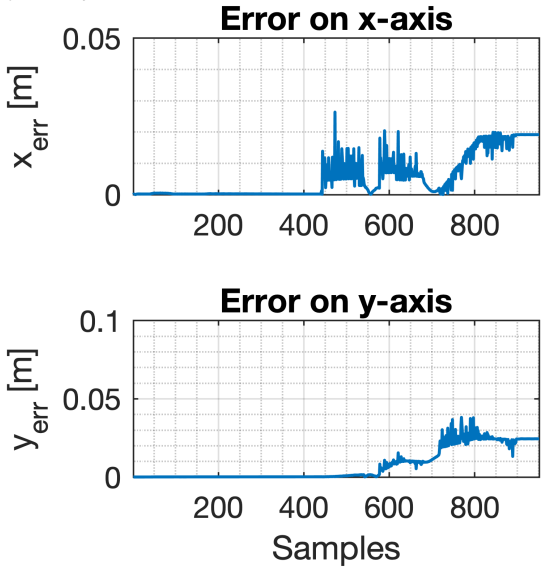
The results obtained, shown in figure 3.8, prove that the odometric position estimation algorithm is able to estimate correctly the trajectory and the pose of the robot even with more complex trajectories. However, the case reported in figure 3.8c presents a problem. As highlighted from the error graphs of figure 3.9, the error in figure 3.9c tends to grow more rapidly with respect to the other

tested trajectories. This problem is due to the rapid turns that the robot makes in the trajectory, causing a slight difference between the odometric position estimation algorithm and ground truth, which integrated over time leads to an higher error. However, for this case and for the remaining cases, the error we obtain in final position, reported in table 3.4, is again low and acceptable for our purposes.



(a) First trajectory

(b) Second trajectory



(c) Third trajectory

Figure 3.9: Error with random trajectory

Trajectory	$x_{err}[m]$	$y_{err}[m]$	Euclidean distance between endpoints $[m]$
1	0.0093	0.0056	0.0111
2	0.0013	0.0049	0.0051
3	0.0193	0.0244	0.0311

Table 3.4: Error in final position with random trajectory

3.3.3 Conclusions

The tests highlighted how the odometric estimation algorithm described in chapter 3.3.1 works but with some issues. The first and more important issue is the continuous integration of errors over time, which causes the odometric position estimation algorithm to diverge from the real robot position. In these tests the effects are minimal, but over a long period of time this problem could lead to an incorrect localization of the robot. The second aspect to highlight is the speed of the robot (both linear and angular). As it increases, we obtain an error on the odometric position estimation algorithm which grows faster as speed increases, therefore as solution one idea is to keep the speed low.

However, in the perspective of its usage alongside the NAPVIG algorithm, the results obtained are satisfying and both the error-grow rate and the error in the final positioning of the robot are acceptable, while for other applications, it could be necessary to adopt other position estimations methods, such as the usage of an IMU or a gyroscope, that working together with the odometric estimation algorithm could provide a better estimated pose of the robot.

4

Automatic target-locking

For an autonomous mobile robot, the knowledge of the final position it has to reach inside and environment is crucial. As possible solution, the target position could be specified directly by hand. This however would limit the autonomous capabilities of an autonomous mobile robot, requiring an external system which specifies the target position every time (for example the position could be provided by an human operator). To overcome this problem and extend the autonomous capabilities of a robot, it was decided to develop an algorithm that, by making use of a camera mounted on a robotic arm installed on an UGV, is able to detect and track a target during the entire motion process of the robot.

4.1 Robotic Manipulator

A robotic manipulator, or robotic arm, is a type of mechanical arm which is generally programmable which has similar functions to a human arm. As already mentioned in section 2.1.1, a robotic arm can be found in an industrial environment, where it is used to perform repetitive tasks with extreme precision and high speed. Nonetheless, a robotic arm can be used in other circumstances or can be installed in autonomous robots.

From the mechanical point of view, a robotic arm is composed by a sequence of rigid bodies (denoted as *links*) which are connected one to another by mean of *joints* (q_i). Each joint guarantees to the manipulator a Degree of Freedom (DoF) and can be essentially of two types:

- Revolute: allows relative rotation between two consecutive links ($q_i = \vartheta_i$);
- Prismatic: allows relative translation between two consecutive links ($q_i = d_i$).

The sequence of links and joints is denominated as *kinematic chain*. A robotic arm is installed on a *base*, which is typically fixed to the world, but in recent applications it is possible to find robotic arm installed even in mobile agents. The last link of the robotic arm, denominated as *end-effector*, usually is equipped with a specific tool which is used to perform a task (eg. solderer, gripper). As for the other types of autonomous robots, to ensure the correct functioning, a robotic arm is not only composed of its mechanical structure but also requires sensors, actuators as well as controllers to work correctly.

4.1.1 Direct Kinematics

As described in section 4.1, from the geometric point of view a robotic manipulator can be seen as a sequence of links interconnected by joints. It is of particular interest to determine the pose of the end-effector (with respect to the base reference frame $\mathcal{F}_b$ $(O_b - x_b y_b z_b)$ as function of its joint values. This process is called *Direct Kinematics* (or also *Forward Kinematics*).

The pose of a rigid body with respect to a reference frame can be described using the position vector that connects the origin of the reference frame with the rigid body and the unit vectors of a frame attached to the body itself. The frame of the end-effector $\mathcal{F}_e$ is typically assigned accordingly to the particular task geometry, and its pose with respect to the base reference frame $\mathcal{F}_b$, can be expressed by means of an homogeneous transformation matrix:

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} \mathbf{n}_e^b(\mathbf{q}) & \mathbf{s}_e^b(\mathbf{q}) & \mathbf{a}_e^b(\mathbf{q}) & \mathbf{p}_e^b(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Where:

- $\mathbf{q} \in \mathbb{R}^{n \times 1}$ is the vector of joint variables;
- $\mathbf{n}_e^b$ (*normal*), $\mathbf{s}_e^b$ (*sliding*), $\mathbf{a}_e^b$ (*approach*) are the unit vectors of the frame $\mathcal{F}_e$ attached to the end-effector;

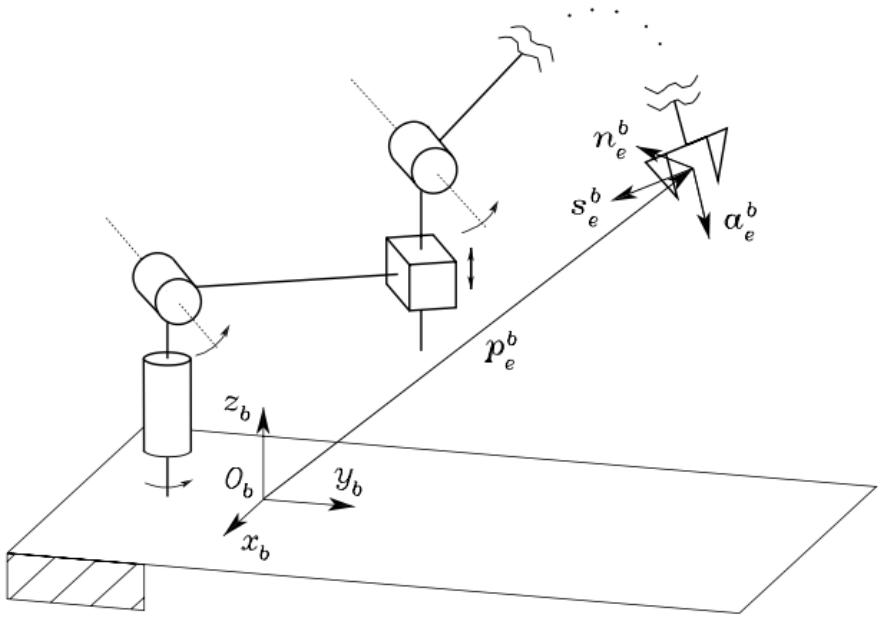


Figure 4.1: Description of the pose of the end-effector frame

- $\mathbf{p}_e^b$ is the vector representing the position of the end-effector expressed with respect to $\mathcal{F}_b$.

To determine the direct kinematics of the robotic arm, it is necessary to determine a closed form expression for $\mathbf{n}_e^b$, $\mathbf{s}_e^b$, $\mathbf{a}_e^b$ and $\mathbf{p}_e^b$. This can be done principally in two ways. The first one resorts to a geometric approach, where the expressions for $\mathbf{n}_e^b$, $\mathbf{s}_e^b$, $\mathbf{a}_e^b$ and $\mathbf{p}_e^b$ are derived from inspection of the robot and are based on its geometric characteristics. While this method can be fast for manipulators with low number of joints, for a robot with high number of joints this operation can become difficult to solve. In these cases, it is preferable to adopt a less direct solution which is however systematic and general and exploits the open-chain structure of the robotic arm.

The typical structure of a robotic manipulator is composed of $n + 1$ links connected by n joints, with Link 0 fixed conventionally to the ground (or a base of a mobile robot). As mentioned before, each joint provides to the structure a single DoF and connects two consecutive links. By assigning to each link a reference frame, it is possible to express the coordinate transformation from frame $\mathcal{F}_{i-1}$ to frame $\mathcal{F}_i$ by means of an homogeneous transformation matrix $\mathbf{A}_i^{i-1}(q_i)$, which is function of a single joint variable. Then, it is possible to determine recursively the coordinate transformation matrix from frame 0 to frame n (after the appropriate frame assignation from Link 0 to Link n) using

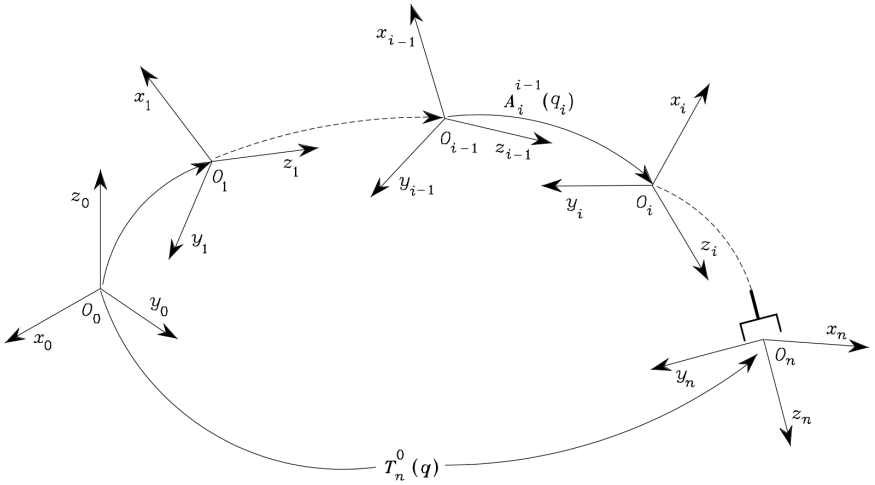


Figure 4.2: Coordinate transformation in an open kinematic chain

the consecutive product of each homogeneous transformation matrix between consecutive frames as:

$$\mathbf{T}_n^0(q) = \mathbf{A}_1^0(q_1)\mathbf{A}_2^1(q_2)...\mathbf{A}_n^{n-1}(q_n) \tag{4.2}$$

At last, the homogeneous transformation matrix describing the end-effector pose with respect to the base reference frame can be expressed as:

$$\mathbf{T}_e^b(q) = \mathbf{T}_0^b\mathbf{T}_n^0(q)\mathbf{T}_e^n \tag{4.3}$$

The assignment of the reference frames to each link can be done arbitrarily or can be done by resorting to systematic procedures that simplify the entire process (e.g. the usage of Denavit-Hartenberg [8] convention).

Joint space and operational space

Direct kinematics equations allow to determine and express the end-effector position and orientation with respect to the base reference frame $\mathcal{F}_b$. While for the position this operation is quite easy, for the orientation part it could be quite difficult, since the representation through the triplet $\mathbf{n}_e^b$, $\mathbf{s}_e^b$, $\mathbf{a}_e^b$ always requires to satisfy the rotation matrix constraint $R^T R = I$. Instead of resorting to a rotation matrix, the orientation of the end-effector can be specified through a minimal set of angles (for example the Euler angles), describing therefore the pose of the

end-effector as:

$$\mathbf{x}_e = \begin{bmatrix} \mathbf{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix} \tag{4.4}$$

where $\mathbf{p}_e$ accounts for the position and $\boldsymbol{\phi}_e$ regards the orientation of the end-effector. The vector $\mathbf{x}_e$ is defined in the space in which the manipulator task is specified and is typically called *operational space*, while for a robot the *joint space* denotes the space in which the joint variables vector $\mathbf{q} = [q_1 \dots q_n], \quad \mathbf{q} \in \mathbb{R}^n$ is defined. By taking into account the dependencies of position and orientation from the joint variables, equation (4.2) can be rewritten as:

$$\mathbf{x}_e = \kappa(\mathbf{q}) \tag{4.5}$$

where $\kappa(\mathbf{q}) \in \mathbb{R}^{m \times 1}$ is a vector of function (typically non linear) that allows the computation of operational space variables starting from joint variables. Again, the determination of the functions of $\kappa(\mathbf{q})$ is easy for simple cases, but in general cases with a six-dimensional operational space ($m = 6$) this operation is quite difficult and requires again the calculation of $\mathbf{n}_e$, $\mathbf{s}_e$, $\mathbf{a}_e$ (which then can be converted to a Euler angle representation.

Robot workspace

Referring to the operational space, the *workspace* of a robot is defined as the region described by the origin of the end-effector when all the manipulator joints execute all possible motions. For a robotic manipulator, it is possible to define two types of workspaces:

- *Reachable workspace*: is the region that the origin of the end-effector frame can reach with at least one orientation;
- *Dextereous workspace*: is the region that the origin of the end-effector frame can describe with different orientations.

Kinematic redundancy

A robotic manipulator is said to be *kinematically redundant* when it has a number of DoFs which is greater than the number of variables necessary to

express a task (namely the dimension of the operational space). Relatively to the spaces expressed above, the redundancy is obtained when the dimension of the operational space is smaller than the dimension of the joint space ($m < n$).

4.1.2 Inverse Kinematics

Direct kinematics allowed to determine a set of equations that describes the end-effector pose in terms of joint variables. It is now of interest the solution of the *inverse kinematics* problem, which consists in the determination of the appropriate joint variables given an end-effector pose. The solution of this problem is fundamental since it allows to transform the motion specifications assigned to the end-effector in the operational space into the corresponding joint space motions, allowing therefore the execution of the desired motion of the robot. Differently from direct kinematics, where the determination of a solution is straightforward, for inverse kinematics finding a solution is quite difficult for the following reasons:

- Generally, the equations that have to be solved are non linear and not always it is possible to determine a closed form solution;
- Multiple solutions may exist;
- Infinite solution may exist (for example with a kinematically redundant robot);
- No solution may exist (given end-effector position does not belong to the reachable workspace of the manipulator).

Analytical inverse kinematics

As for direct kinematics, one possible solution to the inverse kinematics problem can be found by geometric inspection of the robotic manipulator. This approach provides a set of algebraic equations in closed form which solve the inverse kinematics problem. On the other hand, the application of this approach is quite difficult in cases where the manipulator presents a high number of joints (high number of DoFs), therefore it is usage is preferable in cases where the number of joints is limited (typically a two or three DoF manipulator).

Numerical inverse kinematics

Another possible solution to inverse kinematics resorts to the usage of a numerical method. The final goal is to solve the equation:

$$\mathbf{x}_e^d = \kappa(\mathbf{q}) \tag{4.6}$$

where $\mathbf{x}_e^d$ is the desired position of the end-effector and $\kappa(\mathbf{q})$ are the direct kinematics equations of the robotic arm where $\mathbf{q} \in \mathbb{R}^n$ is the vector of n-unknowns. In this thesis there are presented two solution methods for the inverse kinematics problem.

Numerical IK - Newton method

The first numerical solution method is the *Newton method*. This method, starting from an initial guess of the joint variables $\mathbf{q}^0$, generates a sequence of values for $\mathbf{q}$ that hopefully converges to a solution $\mathbf{q}^*$: The solution method starts by considering the first-order Taylor expansion of (4.6) around $\mathbf{q}^k$, which is the point reached by the joint variable at the k-th iteration.

$$\mathbf{x}_e^d = \kappa(\mathbf{q}) \approx \kappa(\mathbf{q}^k) + J_A(\mathbf{q}^k)(\mathbf{q} - \mathbf{q}^k) \tag{4.7}$$

where $J_A = \frac{\delta\kappa(\mathbf{q}^k)}{\delta\mathbf{q}}$ is a quantity called *Analytical Jacobian*. The next iteration value, namely $\mathbf{q}^{k+1}$, can be obtained solving the equality:

$$\mathbf{x}_e^d = \kappa(\mathbf{q}^k) + J_A(\mathbf{q}^k)(\mathbf{q}^{k+1} - \mathbf{q}^k) \tag{4.8}$$

which leads to:

$$\mathbf{q}^{k+1} = \mathbf{q}^k + J_A^{-1}(\mathbf{q}^k)(\mathbf{x}_e^d - \kappa(\mathbf{q}^k)) \tag{4.9}$$

While Newton method exhibits quadratic convergence when near to a solution $\mathbf{q}^*$, it is not always possible to guarantee convergence. The choice of the initial value $\mathbf{q}^0$ plays a lead role in the convergence of the algorithm. Moreover, the solution via Newton method requires the evaluation of the inverse of the Analytical Jacobian matrix, which can be computed only if $n = m$, therefore in presence of a redundant manipulator the method needs to be modified accordingly.

Numerical IK - Gradient descent method

The second numerical solution method is the *gradient descent method*. Instead of considering the first-order Taylor expansion, this method considers the following error function:

$$H(\mathbf{q}) = \frac{1}{2}\|\mathbf{x}_e^d - \kappa(\mathbf{q})\|^2 \tag{4.10}$$

and, as updating rule, it considers the direction of the negative gradient as searching direction. From:

$$\nabla_{\mathbf{q}} H(\mathbf{q}) = -J_A^T(\mathbf{q})(\mathbf{x}_e^d - \kappa(\mathbf{q})) \tag{4.11}$$

the updating rule obtained is:

$$\mathbf{q}^{k+1} = \mathbf{q}^k + \alpha J_A^T(\mathbf{q}^k)(\mathbf{x}_e^d - \kappa(\mathbf{q})) \tag{4.12}$$

where $\alpha > 0$ is the stepsize that should be properly tuned in order to guarantee the condition $H(\mathbf{q}^{k+1}) < H(\mathbf{q}^k)$. The usage of the transpose of the Analytical Jacobian instead of its inverse makes the gradient descent method computationally simpler with respect to the Newton method. Also, it can perform when the manipulator is not redundant. On the other side, it could happen that gradient method gets stuck in a point $\mathbf{q} \in ker(J_A^T(\mathbf{q}))$ where the error $e = \mathbf{x}_e^d - \kappa(\mathbf{q})$ is different from zero.

4.1.3 Controller

In section 2.1.3 it was given a description of the role of the control system in a robotic setup, indicating how the *control unit* is responsible for all planning and control tasks of the robot. The goal of the control unit is to provide to the actuators of the robot an appropriate command signal $u(t)$ accordingly to a reference signal $r(t)$, which can be assigned both in terms of end-effector position $\mathbf{x}_e^d$ or as joint configuration $\mathbf{q}_d$. In the present thesis work there is analyzed and used a simple *feedback* control strategy by means of a PID controller, however in literature there exists also different control strategies:

- *Feedforward strategies*: the reference signal is computed offline and guarantees perfect tracking in the ideal case (namely without uncertainties,

external disturbances and initial errors);

- *Hybrid feedforward-feedback*.

In a feedback control system, the input signal to the robot is computed by taking as input the error signal $e(t) = r(t) - y(t)$, where $y(t)$ is the state of the variables at time instant t (typically for a robotic manipulator the joint variables vector **q** is considered as state of the plant).

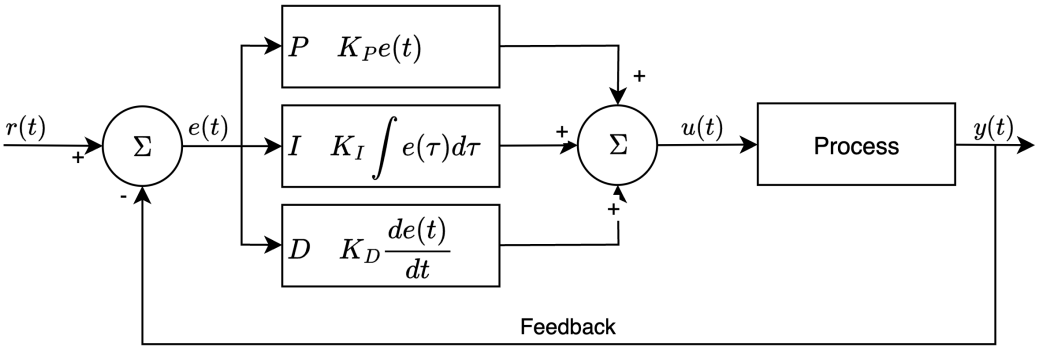


Figure 4.3: Generic scheme of a PID controller

PID CONTROLLER

PID controller is a control structure employed in feedback control systems. The mathematical structure of a PID controller is reported in equation (4.13):

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \tag{4.13}$$

where the three terms K_P, K_I and K_D (called respectively *proportional*, *integral* and *derivative* gain) are parameters that, with an appropriate tuning, allow an efficient working of the controller. Each one of the three terms accounts for a specific action over the control process (in table 4.1 is reported the effect obtained on different parameters of the system as the control gains increase):

- **Proportional gain**: depends directly on the error signal $e(t)$. The increase of the proportional gain causes the augmentation of the reaction speed of the entire control loop. However, with too high value, this can produce oscillations and even make the system unstable;

- **Integral gain**: sums error over time. Its main contribution is to eliminate the steady-state error caused by the proportional gain;
- **Derivative gain**: causes the system to react strongly to the error and increases the overall response speed. As drawback, an high derivative gain makes the system oversensitive to noise and could cause instability of the overall system.

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_P	Decrease	Increase	Small change	Decrease	Degrade
K_I	Decrease	Increase	Increase	Eliminate	Degrade
K_D	Minor change	Decrease	Decrease	No effect	Small change

Table 4.1: Effects of PID gains on system

4.2 Target-locking algorithm

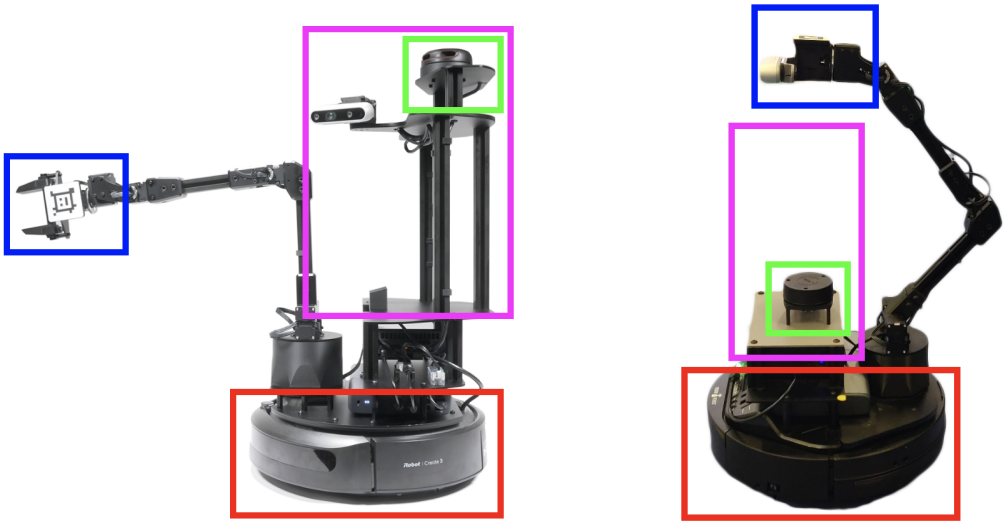
In this section there is presented the procedure that lead to the development and implementation of the algorithm that allows the localization and following of a target during the navigation process of a robot, which will be called from now on as *target-locking algorithm*. As mentioned in the introduction of the present chapter, the aim of this algorithm is to extend the autonomous capabilities of a mobile robot by making it able to detect on its own the destination it as to reach inside an environment. The target-locking algorithm has been developed based on the mechanical structure of the robot that will be used later in the experiments, nonetheless this does not restrict the field of application of it with other robots. In the perspective of the navigation of the robot, and in particular of an usage of it alongside the NAPVIG algorithm presented in chapter 3, the target-locking algorithm will play a key role in the estimation of the position of the target as well as its following during the navigation of the robot.

4.2.1 Robot description

The robot that had been used for the development of the target-locking algorithm is a modified version of the "Locobot_wx200" built by *Trossen Robotics* (which will be denoted as *Locobot* from now on). As can be seen from picture 4.4a, the original robot is constituted of a mobile base (the one highlighted in

red) which can be modeled as a DDR, and a robotic arm, mounted on the top of the mobile base with a gripper as end-effector. For the purposes of this thesis, the original robot has been modified in the following parts (for reference in figure 4.4b are reported the changes made):

- removal of the camera and LiDAR tower (highlighted in magenta);
- positioning of the LiDAR on top of the mobile base (highlighted in green);
- substitution of the arm's end-effector with the camera (highlighted in blue).



(a) Original *Locobot* (b) Modified *Locobot*

Figure 4.4: Modifications made in *Locobot*

The interest of the target-locking algorithm focuses on the robotic arm of the *Locobot*, which is a 5 DoF arm with all revolute joints. The scheme reported in figure 4.5 shows a simple logical scheme of the arm, where the revolute joints are indicated as cylinders. There are also reported the reference frames assigned to all the joints of the arm as well as the reference frame assigned to the camera $\mathcal{F}_c$, where the red axis accounts for the x-axis, the green one for the y-axis and the blue one for the z-axis of each frame[1].

[1]Each frame, except for the camera one, rotates together with the link moved by the joint
[2]The scheme is only representative of the joints and frames of the robotic arm, it does not take into account the original proportions and dimensions of the arm

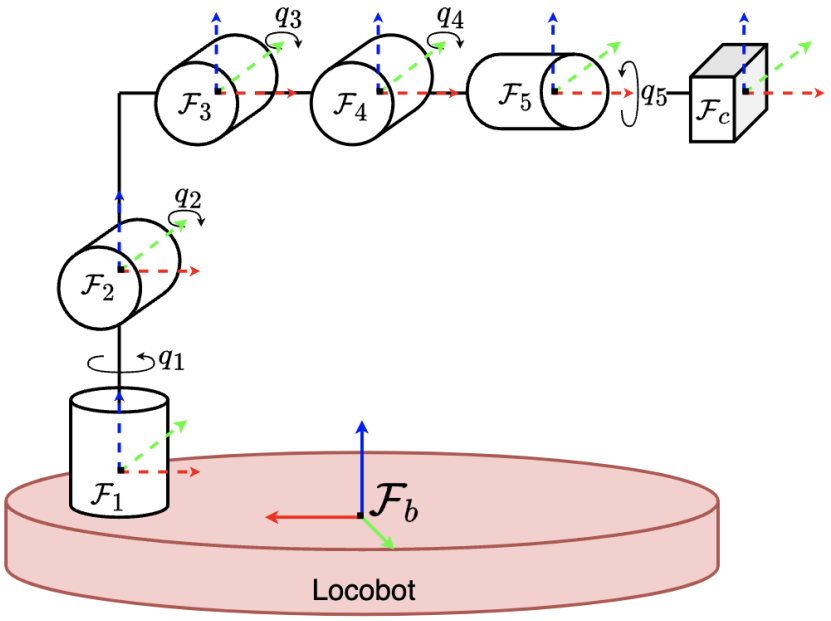


Figure 4.5: *Locobot*'s arm structure[2]

At last, to use the same notation adopted in section 4.1.1 for the vector of joint variables $\mathbf{q} \in \mathbb{R}^n, n = 5$, there is reported in table 4.2 the appropriate conversion from the name of each joint to the corresponding joint variable.

Joint name	Joint variable	Associated reference frame
shoulder_joint	q_1	$\mathcal{F}_1$
upper_arm_joint	q_2	$\mathcal{F}_2$
forearm_joint	q_3	$\mathcal{F}_3$
wrist_angle_joint	q_4	$\mathcal{F}_4$
wrist_rotate_joint	q_5	$\mathcal{F}_5$

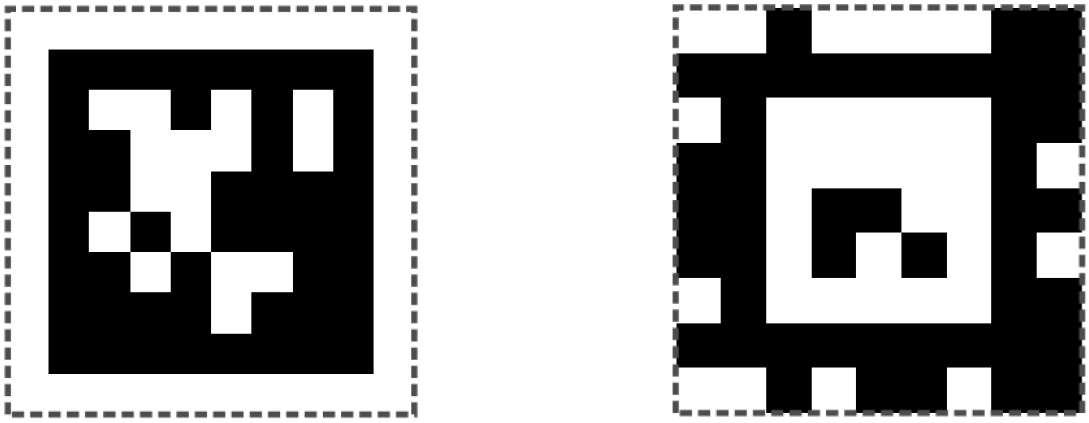
Table 4.2: Joint names, variables and frames

4.2.2 AprilTag

To work correctly, the target-locking algorithm requires an object that acts as a target for the entire system. For this thesis, it was decided to use an AprilTag as target. AprilTag is a visual fiducial system which is widely used in robotics applications and others, such as augmented reality or camera calibration. Targets can be created with an ordinary printer and using the AprilTag detection

software it is possible to determine the 3D pose of the tag with respect to the camera used for the detection [25]. The choice of AprilTag as target was made for the following reasons:

- since the entire algorithm will be used in the ROS environment, the usage of AprilTag was convenient since it is possible to use the ROS wrapper of the AprilTag detection system to easily obtain the estimate the pose of the tag directly from the camera of the robot;
- the estimation of the tag is fast and it does not affect the speed of the robot.



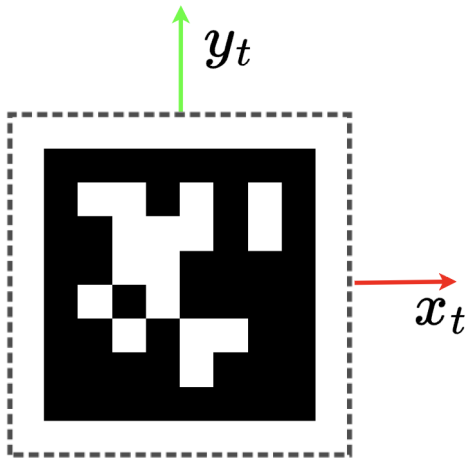
(a) Tag36h11 (b) TagStandard41h12

Figure 4.6: Examples of AprilTag. The tag format in figure 4.6a will be used as target

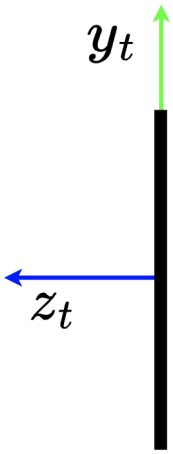
4.2.3 Algorithm concept

The target-locking algorithm makes use of the modified robotic arm mounted on the the top of the *Locobot*'s mobile base to obtain the "lock" effect on the target during the navigation of the robot. The main idea behind the algorithm is to maintain, when possible, the origin of the target reference frame $\mathcal{F}_t$, $(O_t - x_t y_t z_t)$ centered inside the Field of View (FoV) of the camera mounted on the robotic arm, by positioning it accordingly in the cases where:

1. the robot is moving and the target is fixed in the world;
2. both the robot and the target are moving in the environment.



(a) Front view of the target's reference frame



(b) Lateral view of the target's reference frame

Figure 4.7: Representation of the target's reference frame

From a geometrical point of view, the condition of being centered can be interpreted as the x-axis of the camera reference frame $\mathcal{F}_c$, $(O_c - x_c y_c z_c)$[3]pointing directly towards the origin of the target reference frame $\mathcal{F}_t$.

As said in section 4.2.1, the robotic arm is composed of 6 links interconnected by means of 5 revolute joints which gives to the arm $n = 5$ DoFs. For the target-locking algorithm the interest lies only in the correct positioning of the camera, while for the orientation the interest is only in the pitch of the camera. In this sense, the operational space reduces down to m=4 dimensions, with the *Locobot*'s arm being redundant for the scopes of the algorithm ($m < n$, $m = 4$, $n = 5$). The correct pose has to assume to follow the target depends on the values assumed by the joints of the arm, therefore it is necessary to evaluate the inverse kinematics of the robotic arm, which will be examined in depth in the next section. At last, it is necessary to put down some working assumptions that will simplify the determination of the inverse kinematics and the final algorithm:

1. the fifth joint of the robotic arm, namely the *wrist_rotate_joint*, is always keep fixed to the position $q_5 = 0$ since the rotation of this joint is meaningless for the purposes of the algorithm;

[3]This frame corresponds to the end-effector frame and is not to be intended as the frame of the optics of the camera. However, these frames shares the same origin and differs only in the orientation, therefore the choice of one of them is irrelevant for the algorithm

2. the camera is always maintained parallel to the ground;

3. the distance of the target from the camera is ignored in the target-locking algorithm;

4. the orientation of the target is irrelevant, however it is reasonable to keep it positioned in such a way it is recognizable from the camera.

4.2.4 Inverse kinematics of the *Locobot*'s robotic arm

In this section there is be presented the procedure that lead to the determination of the inverse kinematics of the *Locobot*'s arm. Given the simple mechanical structure of the arm, it was decided to proceed using an analytical approach instead of a numerical one. In this way, other than the advantages reported in section 4.1.2 about using an analytical approach, it had been possible to apply some simplifications in the determination process by using the assumption made in the previous section as well as the correct exploitation of the structure of the arm.

The first simplification is done taking into account the first working assumption made in section 4.2.3. By keeping the *wrist_rotate_joint* fixed, it is possible to consider the link between the fourth and fifth joint and the link between the fifth joint and the camera as an unique link, other than reducing the overall arm's structure to a 4 DoF arm. The second and most important simplification is done considering the movement of the target with respect to the position of the camera and the relative movements that the robotic arm has to do in order to follow it. Without loss of generality, the determination of the inverse kinematics is done considering the special motion case where the robot is fixed and the target can move freely in the world. This however is not restrictive for the motion cases indicated in section 4.2.3, since the inverse kinematics determined will be applicable for all the motion cases. Also, in the analysis there is only considered the translation of the target along the axis of its origin frame, neglecting completely the orientation it can assume. In this perspective, the target can move:

- along its x-axis: in this case the robotic arm, needs to rotate to follow the target (the movement is reported in figure 4.8);

- along its y-axis: this movement requires a variation of the height of the position of the camera to keep the lock on the target (the movement is reported in figure 4.9).

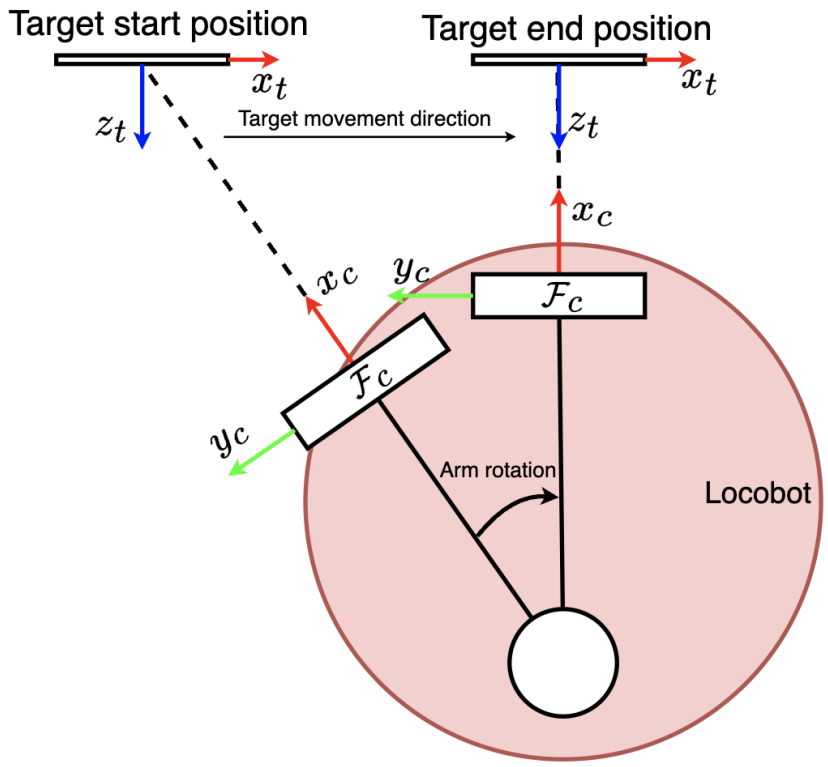


Figure 4.8: Rotation of the arm

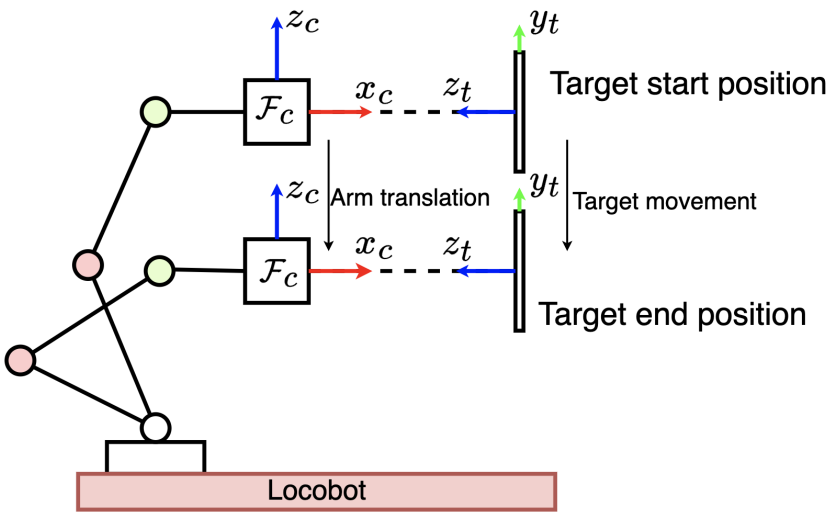


Figure 4.9: Height variation of camera

This division of the movements accomplished by the robotic arm, united with its mechanical structure, allows the simplification of the determination of its inverse kinematics by assigning each movement to specific portions of the arm. In particular:

- the variation of the height of the camera can be obtained by the combined movement of the upper_arm, the forearm and the wrist_angle joints. In this way the inverse kinematics of this block can be determined as the inverse kinematics of a generic 3R planar arm (*height-variation movement*);
- the rotation of the camera instead can be dealt with an appropriate rotation of the shoulder_joint of the *Locobot*'s arm (*rotation movement*).

Since the rotation of the arm consist simply in the rotation of the shoulder_joint, the determination of the inverse kinematics concentrates only on the 3R planar arm constituted by the upper_arm, forearm and wrist_angle joints. The inverse kinematics of this portion of the arm is determined starting from a generic 3R planar arm, shown in figure 4.10, and then, the formulas obtained are changed suitably for the *Locobot*'s arm case.

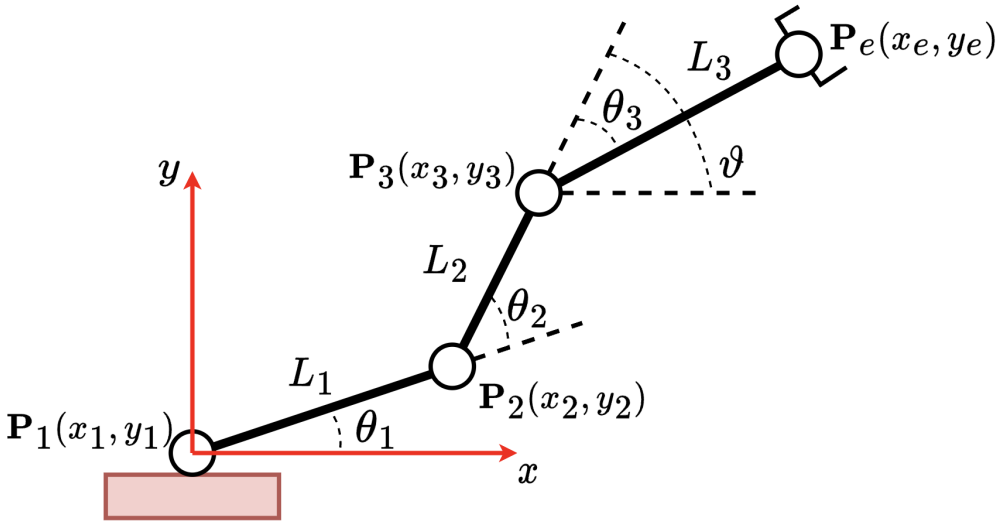


Figure 4.10: Generic 3R planar arm

Direct kinematics of the generic 3R planar arm

The determination of the inverse kinematics of the generic 3R planar arm starts from the evaluation of its direct kinematics. The arm end-effector's posi-

tion is expressed through the triplet $\mathbf{P}_e = [x_e, y_e, \vartheta]$, where $[x_e, y_e]$ represents the position of the end-effector and ϑ accounts for the orientation of the end-effector relatively to the x-axis of the base frame. The joint positions of the robotic arm can be determined as:

$$\mathbf{P}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{P}_2 = \begin{bmatrix} L_1 \cos(\theta_1) \\ L_1 \sin(\theta_1) \end{bmatrix} \tag{4.14}$$

$$\mathbf{P}_3 = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \tag{4.15}$$

where L_1, L_2 are the length of the first and second link of the arm respectively and θ_1, θ_2 are the joint variables for the first and second joint. The direct kinematics of the 3R planar arm then are determined as sum of the joint positions, obtaining:

$$\mathbf{P}_e = \begin{bmatrix} x_e \\ y_e \\ \vartheta \end{bmatrix} = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) + L_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) + L_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix} \tag{4.16}$$

where L_3 is the length of the last link of the robotic arm and θ_3 is the third joint variable.

Inverse kinematics of the generic 3R planar arm

As mentioned in the introduction of the section, the determination of the inverse kinematics of the arm is done following an analytical approach instead of a numerical one. Considering again the generic 3R planar arm, the solution to the problem starts from the determination of the position of the last joint of the general 3R planar arm P_3, which is obtained as:

$$\mathbf{P}_3 = \begin{bmatrix} x_e - L_3 \cos(\vartheta) \\ y_e - L_3 \sin(\vartheta) \end{bmatrix} \tag{4.17}$$

P3 can also be obtained from equation (4.14), while the joint angles θ_1 and θ_2 can be obtained from the solution of the inverse kinematics of a 2R planar arm

with $\mathbf{P}_3$ as end-effector. The angles α and β in figure 4.11 are determined as:

$$\alpha = \arccos\left(\frac{x_3^2 + y_3^2 - L_1^2 - L_2^2}{2L_1L_2}\right) \tag{4.18}$$

$$\beta = \arcsin\left(\frac{L_2 \sin\alpha}{\sqrt{x_3^2 + y_3^2}}\right) \tag{4.19}$$

For a 2R planar arm there are two set of possible solutions. Considering the second joint of the arm as the elbow joint, the solutions found are found in the elbow-up or elbow-down configuration. The triplet of joint angles $\theta = [\theta_1, \theta_2, \theta_3]^T$ then can be found as:

$$\theta_1^a = \arctan\frac{y_3}{x_3} - \beta; \qquad \theta_1^b = \arctan\frac{y_3}{x_3} + \beta \tag{4.20}$$

$$\theta_2^a = \pi - \alpha; \qquad \theta_2^b = -(\pi - \alpha) \tag{4.21}$$

$$\theta_3^a = \vartheta - \theta_1^a - \theta_2^a; \qquad \theta_3^b = \vartheta - \theta_1^b - \theta_2^b \tag{4.22}$$

where (a) represents the elbow-down condition while (b) is the elbow-up condition.

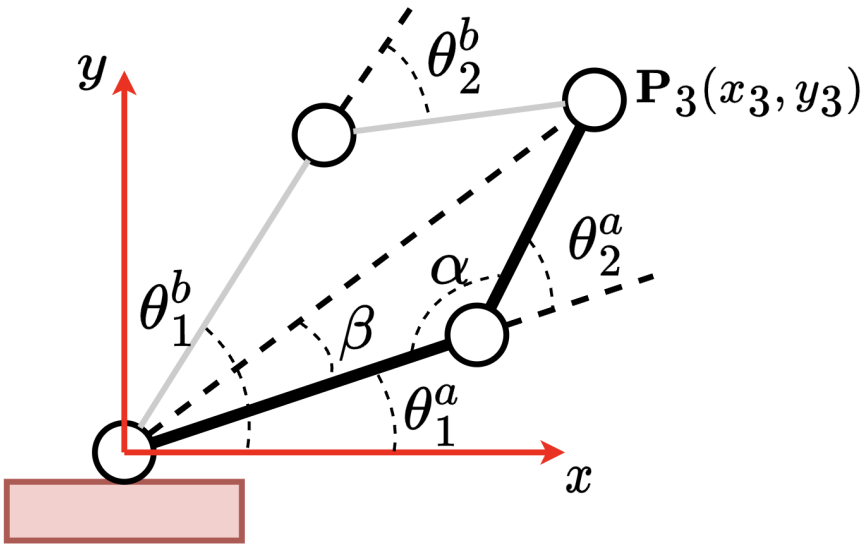


Figure 4.11: Generic 2R planar arm

Inverse kinematics for the *Locobot*'s 3R planar arm

The equations determined previously for the inverse kinematics of the generic 3R planar arm needs to be adjusted properly for the 3R planar arm section of the *Locobot*'s arm. This has to be done for two main reasons:

1. the model of the *Locobot*'s arm is slightly different with respect to the generic 3R planar arm (in particular the difference lies between the forearm_joint of the *Locobot*'s arm and the joint $\mathbf{P}_2$ of the generic 3R planar arm)

2. in the definition of the inverse kinematics of the generic 3R planar arm there were omitted the range of the joints as well as their zero position. For the *Locobot*'s arm these parameters are taken into account to define the correct inverse kinematics equations.

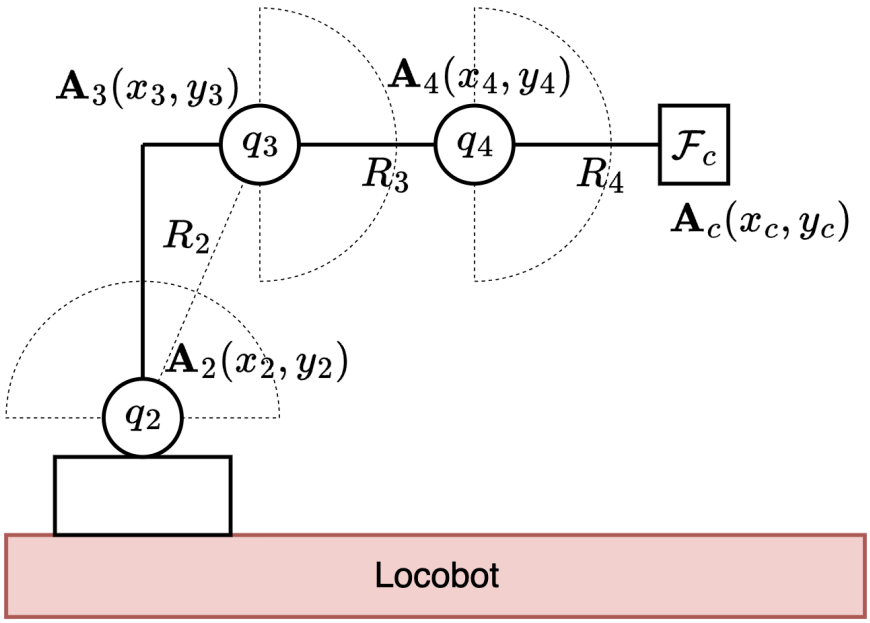


Figure 4.12: Representation of the joint limits of the 3R planar arm section of the *Locobot*'s arm

In this perspective, in figure 4.12 is reported a lateral view of the *Locobot* with the joint limits for the joints[4] $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_4$ (the fifth joint is not reported since, given that is always fixed, is useless in the evaluation of the inverse kinematics) as well

as their position expressed with respect to a fixed reference frame coincident with the reference frame of the second joint $\mathcal{F}_2$ [5].

Joint name	Joint variable	Joint range
upper_arm_joint	q_2	$[-90°, 90°]$
forearm_joint	q_3	$[-90°, 90°]$
wrist_angle_joint	q_4	$[-90°, 90°]$

Table 4.3: Joint variables and limits

Before defining the appropriate inverse kinematics equations of the 3R planar arm section of the *Locobot*'s arm, it is necessary to redefine some quantities of interest that will be used in the equations. For the generic 3R planar arm, the definition of its inverse kinematics started from the position of its last joint $\mathbf{P}_3$. The same reasoning can be applied for the *Locobot*'s arm, where the last joint is identified in the wrist_angle_joint $\mathbf{A}_4(x_4, z_4)$. With this, it is possible to redefine the angles α and β of equations (4.18) and (4.19) as:

$$\alpha = \arccos\left(\frac{x_4^2 + z_4^2 - R_2^2 - R_3^2}{2R_2R_3}\right) \tag{4.23}$$

$$\beta = \arcsin\left(\frac{R_3 \sin\alpha}{\sqrt{x_4^2 + z_4^2}}\right) \tag{4.24}$$

where the lengths of the links of the generic 3R planar arm have been changed appropriately with the lengths of the *Locobot*'s arm (the changes made are reported in table 4.4).

Values in generic 3R arm	Values for *Locobot*'s arm
L_1	R_2
L_2	R_3
L_3	R_4

Table 4.4: Parameter conversion for inverse kinematics equations

It is also necessary to redefine equation (4.17) suitably for the *Locobot*'s arm,

[4]They are not the precise joint limits but for the target-locking algorithm it was decided to set them with the values in table 4.3

[5]The pose of the reference frame $\mathcal{F}_2$ is shown in figure4.5

where the end-effector pose is represented by the camera pose $\mathbf{A}_c(x_c, z_c, \vartheta)$, as:

$$\mathbf{A}_4 = \begin{bmatrix} x_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_c - R_4 \cos(\vartheta) \\ z_c - R_4 \sin(\vartheta) \end{bmatrix} \tag{4.25}$$

With all the necessary quantities redefined, it is finally possible to give the inverse kinematics equations of the 3R planar arm section of the *Locobot*'s arm. For the target-locking algorithm it was decided to use the elbow-up configuration of the robotic arm, and, by taking into account also the joint limits of table 4.3, the final equations of the inverse kinematics of the arm become:

$$q_2 = \arctan \frac{z_4}{x_4} - \beta - \pi/2 - \pi/13 \tag{4.26}$$

$$q_3 = -(\pi/2 - \alpha) + \pi/13 \tag{4.27}$$

$$q_4 = \vartheta - q_2 - q_3 \tag{4.28}$$

where the joint variables of the standard 3R planar arm have been substituted as:

$$\theta_1 = q_2, \quad \theta_2 = q_3, \quad \theta_3 = q_4 \tag{4.29}$$

and $\pi/13$ accounts for a correction factor used to solve the problem of the different arm model.

4.2.5 Algorithm description

In the previous sections there have been given all the tools necessary to define the target-locking algorithm. The last remaining passage to do in order to obtain the final algorithm is to put in relation the estimated position of the target obtained by the camera with the movements the arm needs to do to follow it. The position of the target is obtained through the camera mounted on the arm. Recalling that the target is represented by an AprilTag, it is possible to use the AprilTag detection algorithm to get an estimate of the pose of the target, which is expressed with respect to the camera frame $\mathcal{F}_c$. For the target-locking algorithm, the position of the target needs to be expressed into the reference frame of the shoulder_joint $\mathcal{F}_1$ and is denoted as $\mathbf{P}_{target} = [x_{target}, y_{target}, z_{target}]^T$.

Starting from the rotation movement of the arm, the relation with the position of the target $\mathbf{P}_{target}$ can be found by taking a look at the geometric relation between the target and the origin of the frame $\mathcal{F}_1$ reported in figure 4.13.

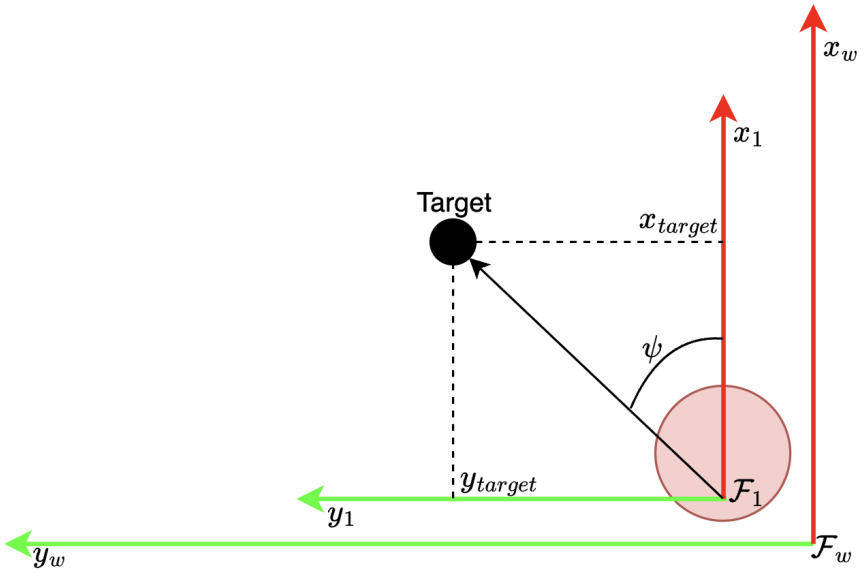


Figure 4.13: Top view of the target-following problem

Considering the vector that connects the origin of the shoulder_frame to the target, it describes an angle (ψ) between itself and the x-axis of the shoulder_frame and (therefore with the x-axis of the camera), which can be evaluated as:

$$\psi = \arctan(y_{target}, x_{target}) \tag{4.30}$$

To align the camera to the target it is sufficient to sum the angle ψ to the actual joint value q_1 of the shoulder_joint, leading to the new joint value $q_1' = q_1 + \psi$.

On the other hand, the solution to the height-variation movement of the arm is obtained using the inverse kinematics for the 3R planar arm portion of the *Locobot*'s arm determined in the previous section.

To apply correctly the inverse kinematics equations of the 3R planar arm section it is necessary to define the values of the pose of the camera $\mathbf{A}_c = [x_c, z_c, \vartheta]$ accordingly to the position of the target $\mathbf{P}_{target}$, so that the "lock" effect is obtained. The angle ϑ is simply put to zero using the second working assumption presented in section 4.2.3. Instead, for the position of the camera, the coordinate z_c is put equal to the coordinate z_{target} of the target position minus an offset given by the length of the first link (R_1) of the arm, while for the coordinate x_c the choice is done so that the coordinate x_4 of the last joint of the arm $\mathbf{A}_4(x_4, z_4)$ is equal to zero. Taking into account equation (4.17) and the assumptions made beforehand, the position of the fourth joint of the *Locobot*'s

arm is expressed as:

$$\mathbf{A}_4 = \begin{bmatrix} x_c - R_4 \\ z_c \end{bmatrix} = \begin{bmatrix} x_c - R_4\cos(\vartheta) \\ z_c - R_4\sin(\vartheta) \end{bmatrix} = \begin{bmatrix} x_c - R_4 \\ z_{target} - R_1 \end{bmatrix} = \begin{bmatrix} 0 \\ z_{target} - R_1 \end{bmatrix} \tag{4.31}$$

In this way, making use of the inverse kinematics equations defined in (4.23) and (4.26), it is possible to determine the joint values of the *Locobot*'s arm. The iterative repetition of the procedure described in this section allows the arm to follow correctly the target over time, as will be proven with the results shown in chapter 5.

Algorithm 1 Target-locking algorithm

Input: joint states, target position
Output: joint reference
while true do
  $q = [q_1, q_2, q_3, q_4] \leftarrow$joint states
  $p = [p_x, p_y, p_z] \leftarrow$target position
  {inverse kinematics for rotation}
  $q_1' \leftarrow q_1 + \sin(p_y, p_x)$
  {inverse kinematics for height-variation}
  $q_2' \leftarrow \arctan\frac{p_z}{p_x} - \beta - \pi/2 - \pi/13$
  $q_3' \leftarrow -(\pi/2 - \alpha) + \pi/13$
  $q_4' \leftarrow \vartheta - q_2 - q_3$
  $[q_1', q_2', q_3', q_4'] \rightarrow$ joint reference
end while

Boundaries

Before proceeding with the tests it is necessary to put down some limitations to the movements of the arm. This has to be done first of all to ensure the correct functioning of the algorithm in all the possible situations, but also it is necessary to avoid collisions between the arm and the rest of the robot. Considering again the movements decomposition of the arm, for the rotation of the arm, inside the simulated environment there are no limits on this movement, while for the real-world setup it is necessary to put down some restraints. These are discussed together with other mechanisms adopted for the real-world experiments in appendix A. For what concerns the height variation of the arm instead, the limits are put considering the position of the fourth joint $\mathbf{A}_4$ and in particular

only in its z coordinate since, as reported in equation (4.31), the x coordinate has been put equal to zero. In particular, there are identified two possible scenarios:

- **Fully-lowered arm**: the lower height limit of the arm ($z_{down} = 0.2cm$) has been chosen so that the camera never collides with *Locobot*'s base and in particular with the LiDAR;
- **Fully-extended arm**: on the other hand, the higher limit ($z_{top} = 0.4cm$) has been chosen so that the height reached by the fourth joint of the arm is always below the sum of the lengths of the links R_2 and R_3.

Hypothesis relaxation

With this setup of the algorithm, where the camera is maintained always parallel to the ground, the perfect follow of the target cannot be always guaranteed, especially in the case where the arm is fully extended and the target is far above the sight line of the camera. Given that the arm is at it maximum extension possible, the solution to this problem can only be found modifying the orientation of the camera without maintaining it parallel to the ground, relaxing therefore the second working assumption made in section 4.2.3. In this perspective and with the arm in the fully-extended configuration, the orientation of the camera depends directly on the position of the fourth joint of the arm q_4, therefore with the appropriate modifications at the inverse kinematics equation of this joint it is possible to align again the camera with the target.

The solution of the problem can be found applying the same reasoning made for the rotation movement of the arm. In this case, the interest is in the position of the target with respect to the frame of the fourth joint $\mathcal{F}_4$, which, given the fully-extended arm setup, is simply found starting from the position of the target in the reference frame of the first joint $\mathcal{F}_1$ as:

$$P'_{target} = P_{target} + [0, 0, z_{top}] = [x_{target}, y_{target}, z_{target} - z_{top}] \tag{4.32}$$

Considering the vector that connects the origin of the reference frame $\mathcal{F}_4$ with the target position, the interest lies in the angle φ formed by this vector and the x-axis of the fourth joint's reference frame, which can simply be determined as:

$$\varphi = \text{atan2}(z_{target} - z_{top}, x_{target}) \tag{4.33}$$

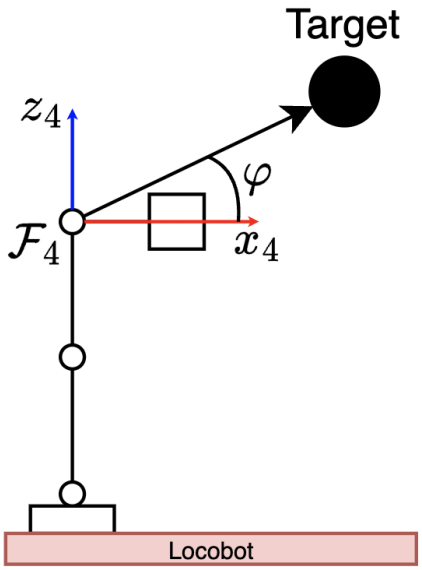


Figure 4.14: Representation of the φ angle

and therefore, the inverse kinematics equation of the fourth joint reported in (4.26) can be modified taking into account the angle φ as:

$$q_4 = \vartheta - q_2 - q_3 - \varphi \tag{4.34}$$

5 Results

In this chapter there are presented and discussed the results obtained from a campaign of simulations and real-world experiments done to validate the functioning of the target-locking algorithm presented in chapter 4. The major part of the tests is done with the base of the robot in motion, where the navigation is obtained using the NAPVIG algorithm presented in chapter 3.

5.1 Simulative validation

In the first section of this chapter there are presented and discussed the results obtained in a simulated environment provided by Gazebo robot simulator. The implementation of the target-locking algorithm is done in Python and works inside a ROS environment in order to meet real time constraints. Also, this choice has been done so that the target-locking algorithm can work finely alongside the NAPVIG algorithm.

Parameter	Value
Resolution	640*480
Minimum detection distance	28$[cm]$
Horizontal angular Field of View	70 deg
Vertical angular Field of View	43 deg
Frame rate	30 fps

Table 5.1: Simulated camera settings

The target that has been used in the simulations is represented by an AprilTag

of size $10[cm] \times 10[cm]$. The robot that will be used is the modified *Locobot* presented in section 4.2.1. For what concerns the camera, it has been simulated using a Gazebo plugin that allows the simulation of the camera with user-definable settings. For the purposes of the target-lock algorithm and to meet the parameters of the camera that will be used in the real world experiments, it was decided to set the camera using the parameters reported in table 5.1. At last, considering the robotic arm, the gains of the controllers driving each motor of it have been tuned using a trial and error approach.The values obtained for the tuning of the controller gains are reported in table 5.2[1].

Joint	Proportional gain	Integrative gain	Derivative gain
shoulder_joint	60.0	25.0	3.5
upper_arm_joint	300.0	70.0	1.7
forearm_joint	270.0	65.5	1.5
wrist_angle_joint	250.0	50.0	1.2

Table 5.2: Simulated gains of the PID controllers of the robotic arm

5.1.1 Scenario #1: Free space

For the first simulation scenario the robot is required to reach the target in a completely free space (for reference consider the snapshot of the experiment reported in figure 5.1). Considering the global reference frame of the simulation, the robot starting position is placed at $\mathbf{x}_i = [-0.5, 0.5]$, while the target, which is fixed in the environment, is placed at $\mathbf{x}_f = [3.0, 0]$, with an height of $0.5[m]$ from the ground. The target has been placed in such a way it is recognizable from the camera of the robot directly from the start of the simulation. To avoid the collision of the robot (and in particular of the camera) with the target, it was decided to apply an offset to the x coordinate of the target of $x_{offset} = 0.4[m]$ so that the robot stops its motion before an eventual collision. Moreover, this slight modification of the target position was necessary since in this way the minimum detection distance of the camera is respected.

The evaluation of the performances of the target-locking algorithm is done using the graphs reported in figure 5.4, where there is reported the Cartesian decomposition of the position of the target $P_{target,c} = [x_{target,c}, y_{target,c}, z_{target,c}]$

[1]The gains of the fifth joint are not reported since this joint is always locked

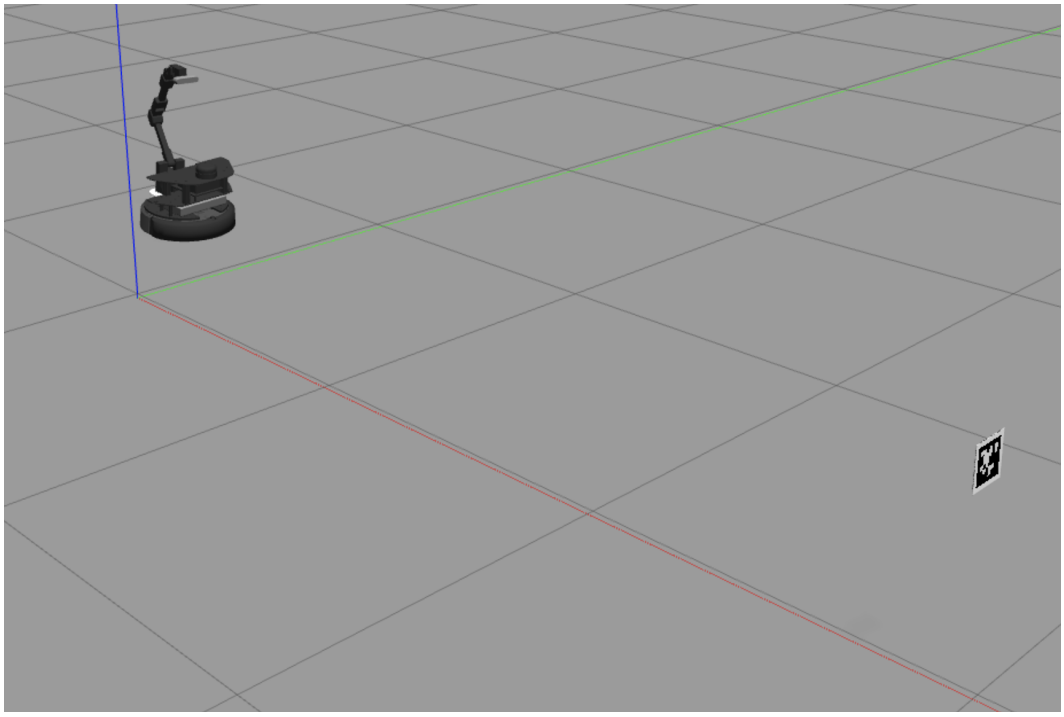


Figure 5.1: Gazebo rendering of scenario #1

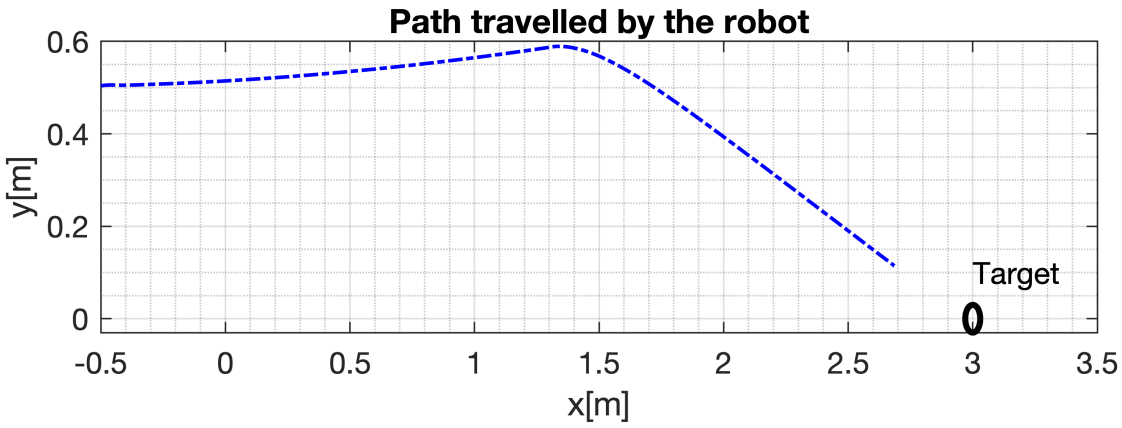


Figure 5.2: Representation of the path travelled by the robot in simulation scenario #1

expressed with respect to the camera frame $\mathcal{F}_c$, $(O_c - x_c y_c z_c)$. Considering the graphs in figures 5.4b and 5.4c, it is possible to notice that the position of the target along that axis have been put in comparison with the corresponding linear FoV, and in particular:

- the graph in figure 5.4b accounts for the position of the target along the y-axis of the camera reference frame $\mathcal{F}_c$, and it has been put in comparison with the horizontal linear FoV of the camera. Recalling the movement

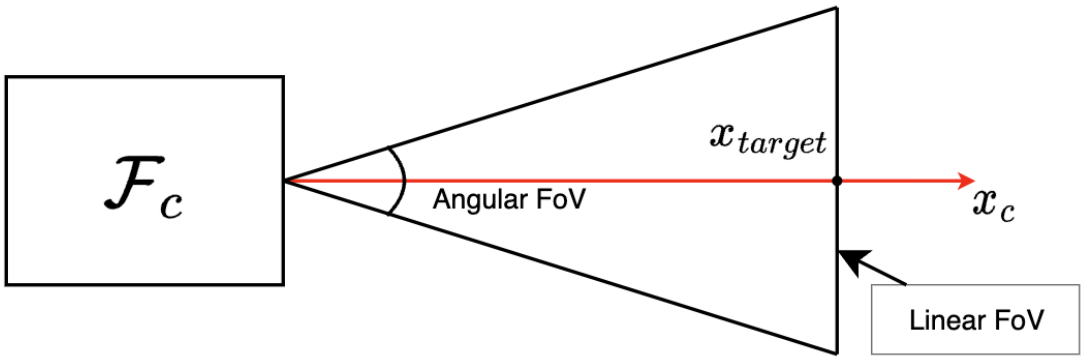


Figure 5.3: Scheme representing the linear FoV of the camera

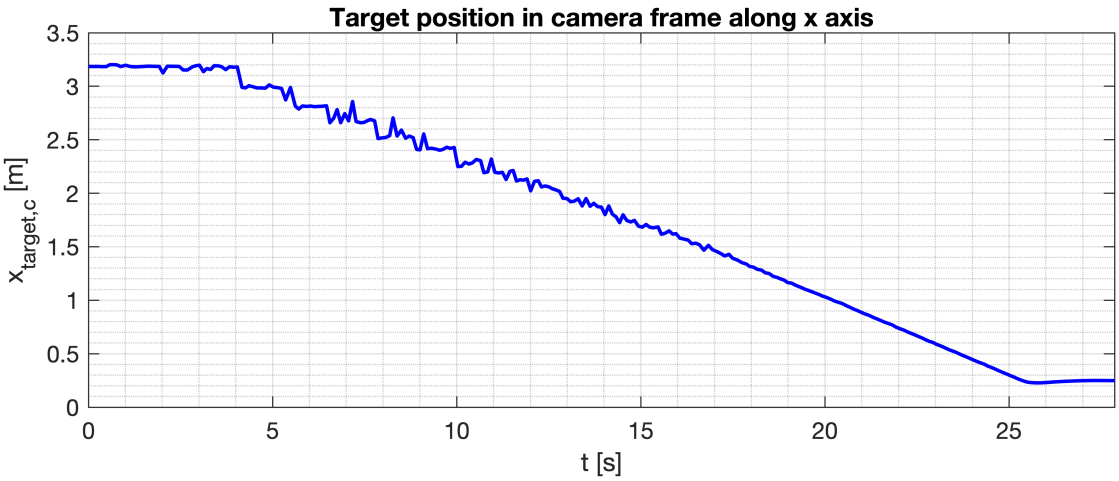
decomposition of the robotic arm presented in section 4.2.4, the positioning along this axis is obtained through the rotation movement of the arm;

- the graph in figure 5.4c instead accounts for the position of the target along the z-axis of the camera reference frame $\mathcal{F}_c$, which is obtained through the height variation movement of the arm. The position along the z-axis has been put in comparison with the vertical linear FoV of the camera.

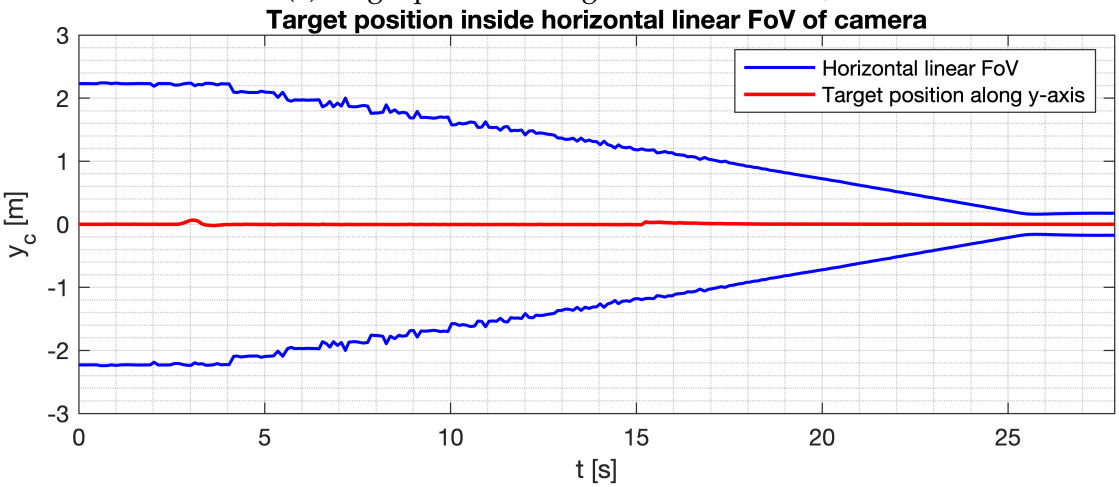
Both horizontal and vertical linear FoVs are obtained considering the position of the target along the x-axis of the camera, which is reported in figure 5.4a. This coordinate lies on the optical axis of the camera, and by putting it into relation with the angular FoV of the camera, it is possible to determine the corresponding linear FoV at that coordinate (for reference take a look at the scheme in figure 5.3).

As can be noticed, in both cases the position of the target is centered inside the linear FoV of the camera, which indicates that the target-locking algorithm is working properly. To better understand what happens to the position of the target along the y and z-axis and to see the action of the target-locking algorithm, there is reported in figures 5.5a and 5.5b a zoom out of the graphs in figures 5.4b and 5.4c.

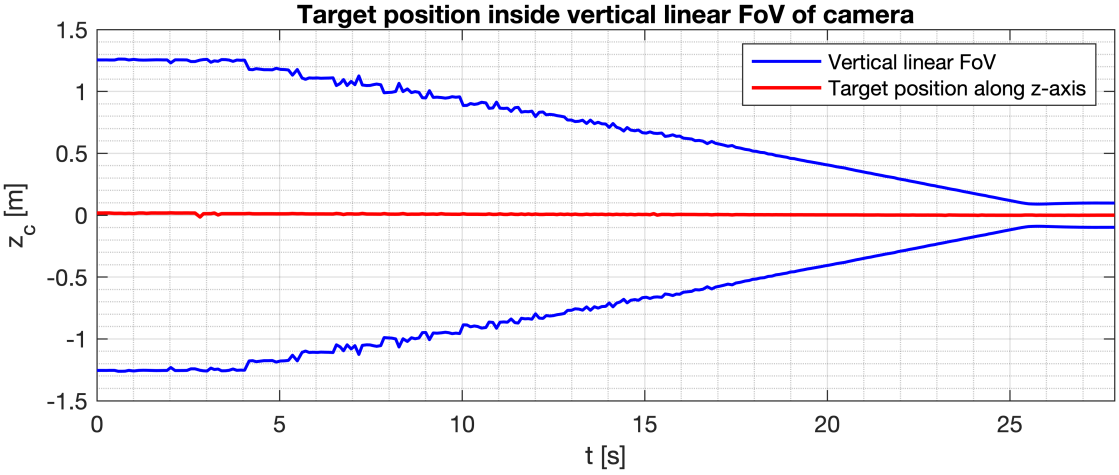
Considering the position of the target along the y-axis of the camera frame, reported in figure 5.5a, it can be noticed the presence of some spikes in the graph. These can be justified by taking a look at the path followed by the mobile base of the robot during the navigation, reported in figure 5.2, as well as the graph of the joint state of the first joint of the robotic arm presented in figure 5.6a. During the navigation, the robot does not follow a perfect linear trajectory but



(a) Target position along x-axis of frame $\mathcal{F}_c$



(b) Target position along y-axis of frame $\mathcal{F}_c$ inside horizontal linear FoV of the camera

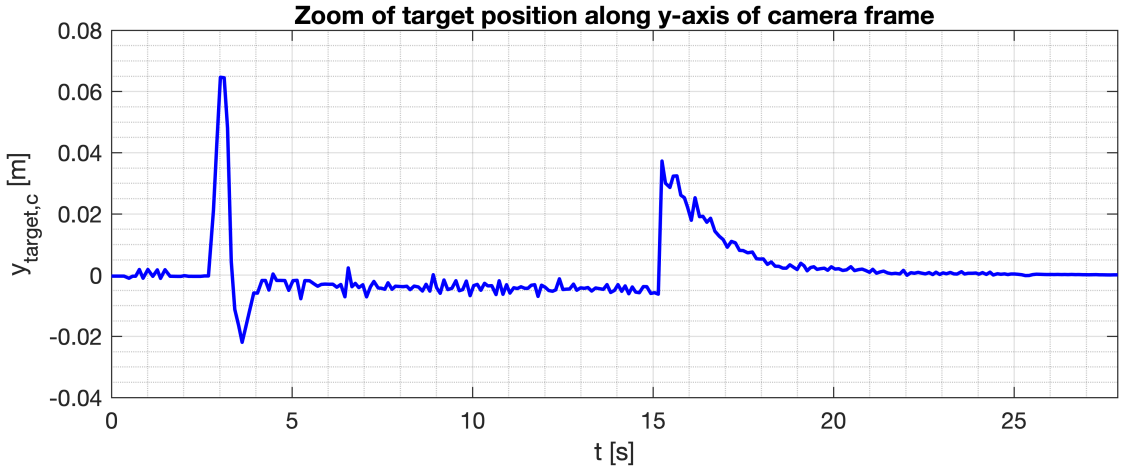


(c) Target position along z-axis of frame $\mathcal{F}_c$ inside vertical linear FoV of the camera

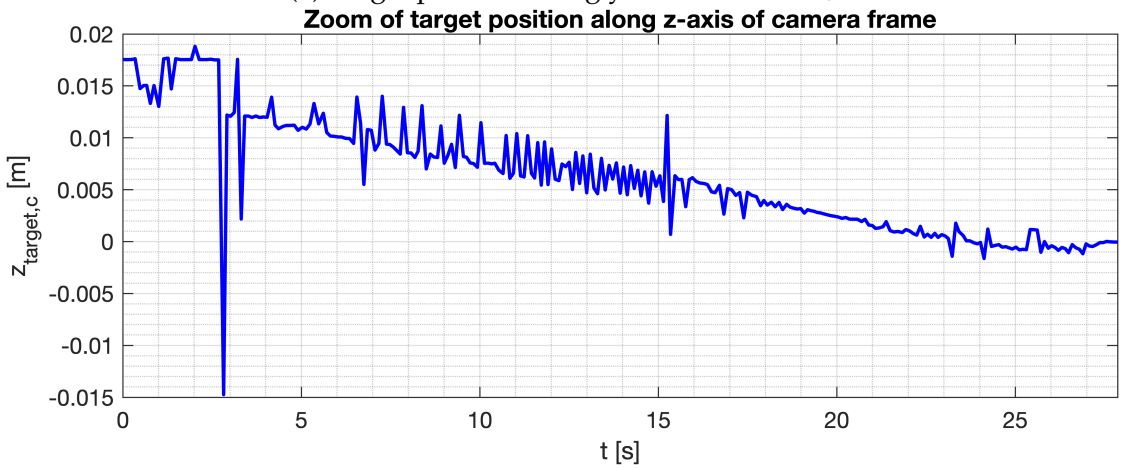
Figure 5.4: Position of the target expressed in the camera frame $\mathcal{F}_c$ for simulation scenario #1

steers in order to reach the target. This steering maneuver causes the slippage of the target from the center of the horizontal linear FoV of the camera. To solve this issue, the shoulder joint of the arm rotates properly following the reference signal provided by the target-locking algorithm, restoring the perfect alignment of the target inside the horizontal linear FoV of the camera.

On the other hand, the height variation movement of the arm maintains the arm fixed during most of the simulation. The only movement performed is near to the end of the simulation, where the perspective of the target with respect to the camera changes. The presence here of little spikes has to be imputed to an imperfection of the simulated model of the robot, which causes some oscillations of the mobile base during the movement and therefore makes its



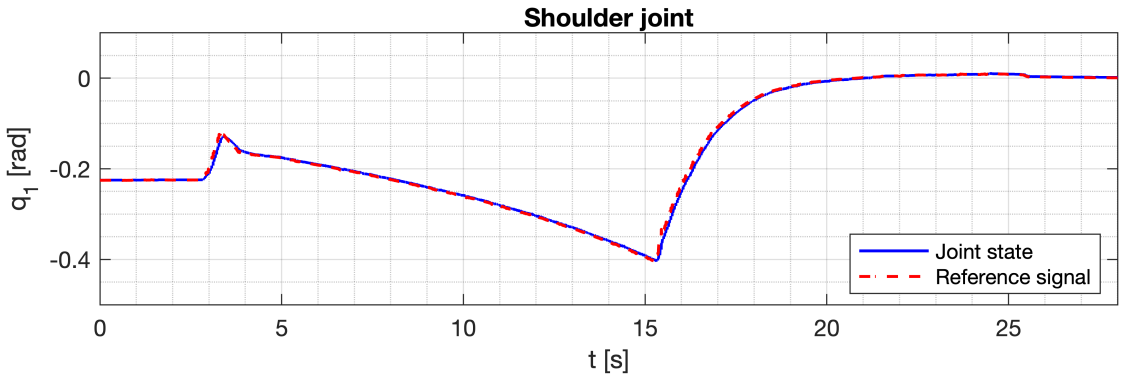
(a) Target position along y-axis of frame $\mathcal{F}_c$



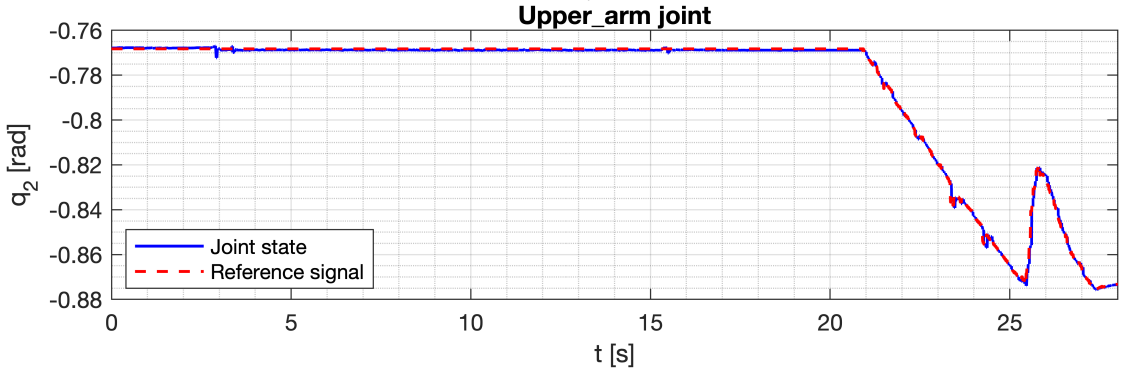
(b) Target position along z-axis of frame $\mathcal{F}_c$

Figure 5.5: Zoom out of the position of the target expressed in the camera frame $\mathcal{F}_c$ for for simulation scenario #1

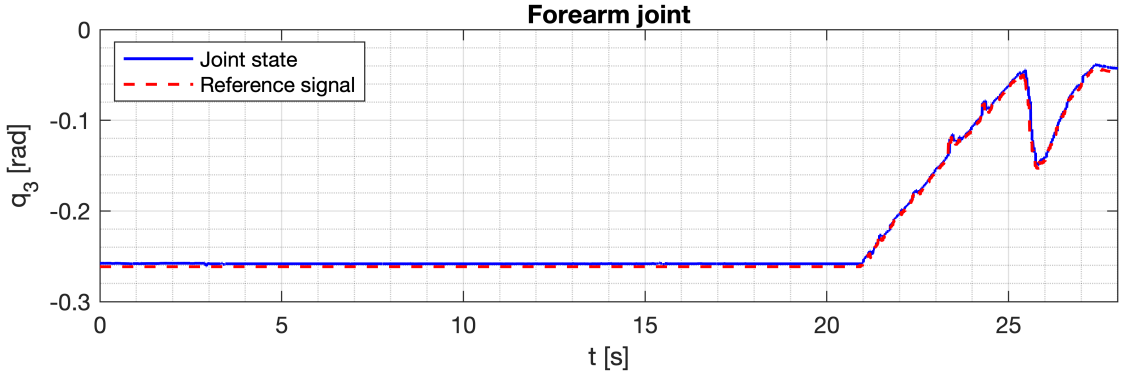
robotic arm oscillate too. Considering instead the offset of the position of the target with the center of the vertical linear FoV of the camera, it can be justified considering the graphs reporting the joint states of the 3R planar arm portion of the robotic arm, with particular attention to the ones in figures 5.6c and 5.6d. From what can be noticed, the third and fourth joint of the arm do not follow perfectly the reference signal provided by the target locking algorithm with the presence of a little steady-state error which persists over time. This causes a misplacement of the arm and therefore of the camera, which produces the error of the target position along the z-axis. The solution to this problem could be found by enhancing the action of the PID controllers of these joints (in particular acting on the integral gain of the controller), however it was decided to leave the gains at the values reported in table 5.2 since with higher gains the arm started to move in an uncontrolled manner, symptom of an unstable controller. Moreover, this misplacement of the camera happens at high distances from the target while at closer distance the arm tends to align perfectly.



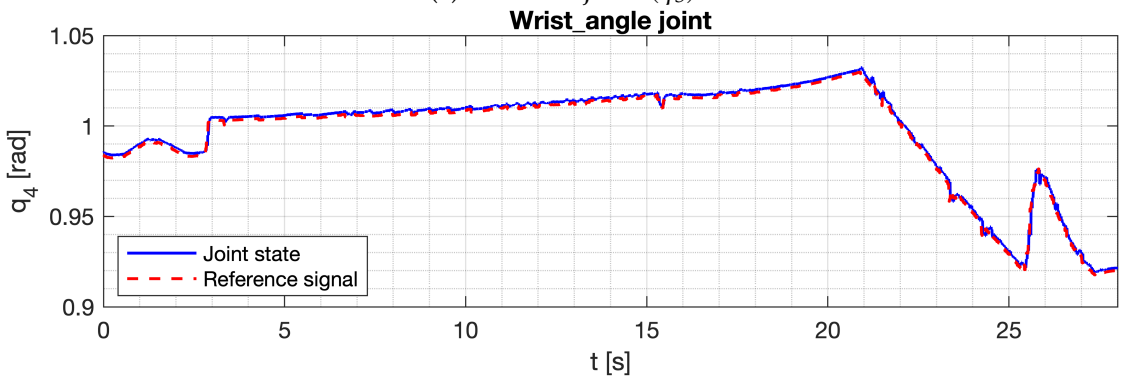
(a) Shoulder joint (q_1)



(b) Upper_arm joint (q_2)



(c) Forearm joint (q_3)



(d) Wrist_angle joint (q_4)

Figure 5.6: Detail of reference signals and joint states for simulation scenario #1

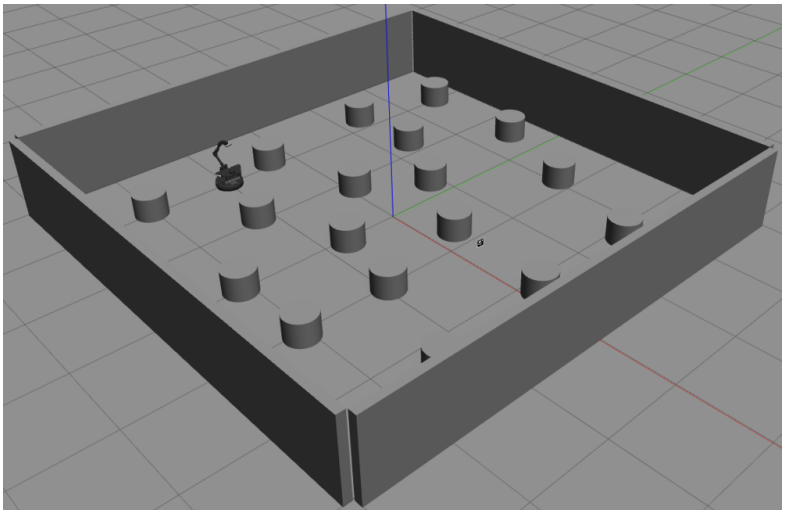


Figure 5.7: Gazebo rendering of scenario #2

5.1.2 Scenario #2: Cluttered environment

In the second simulation scenario, the idea is to put the robot in a more complicated environment, presented in figure 5.7, constituted by a closed box filled with columns of radius $0.2[m]$ and height $0.3[m]$. With respect to the global reference of the simulated environment, the target is placed at the final position $\mathbf{x}_f = [1.6, -0.16]^T$ with an height from the ground of $0.5[m]$, so that the columns do not obstruct the sight line with the target. As happened in the first simulation scenario, the target is always visible from the camera of the robot. Also, it has been considered again the offset $x_{offset} = 0.4[m]$ to avoid the collision of the camera with the target. On the other hand, the robot in this scenario starts from the position $\mathbf{x}_i = [-2, -1]^T$ expressed with respect to the global reference frame of the simulation.

The evaluation of the performances of the target-locking algorithm is done using the same approach adopted for the first simulation scenario and takes into account the Cartesian decomposition of the position of the target expressed in the camera reference frame $\mathcal{F}_c$. Starting from the graphs presented in figures 5.10b and 5.10c, it can be noticed that the position of the target along both y and z-axis is always inside of the horizontal and vertical linear FoVs of the camera respectively, with a nearly perfect alignment inside of them. As in 5.1.1, the linear FoV in both direction has been obtained starting from the position of the target along the x-axis of the camera frame, whose evolution is reported in figure 5.10a. In this case however, differently from the first simulation scenario, the position

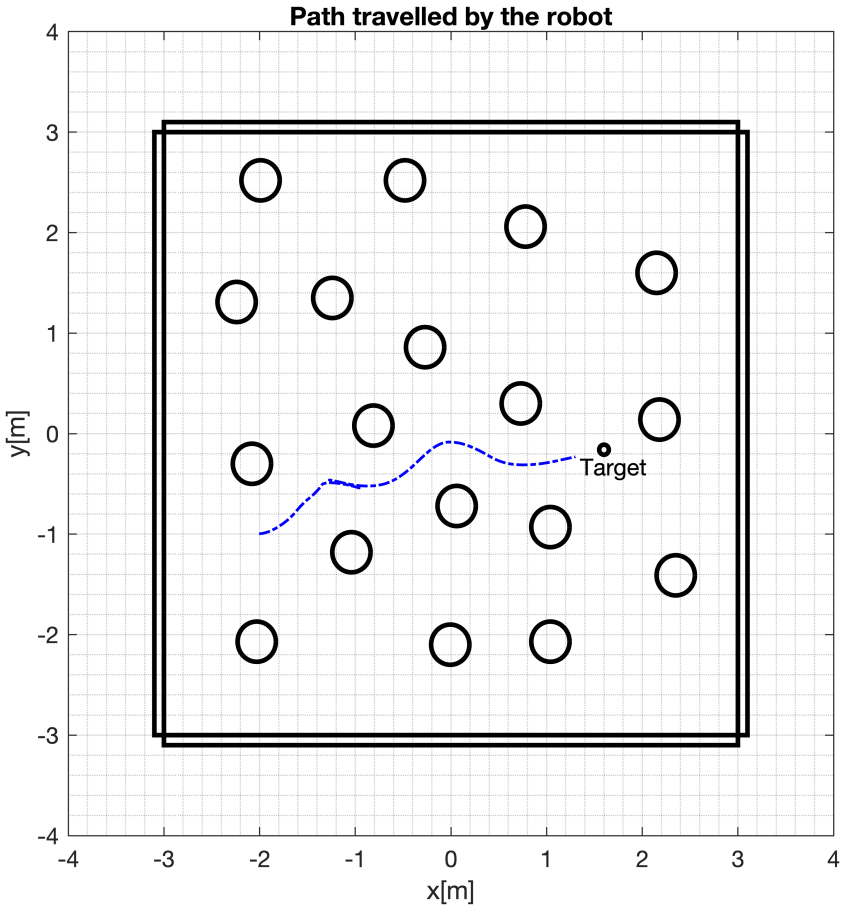
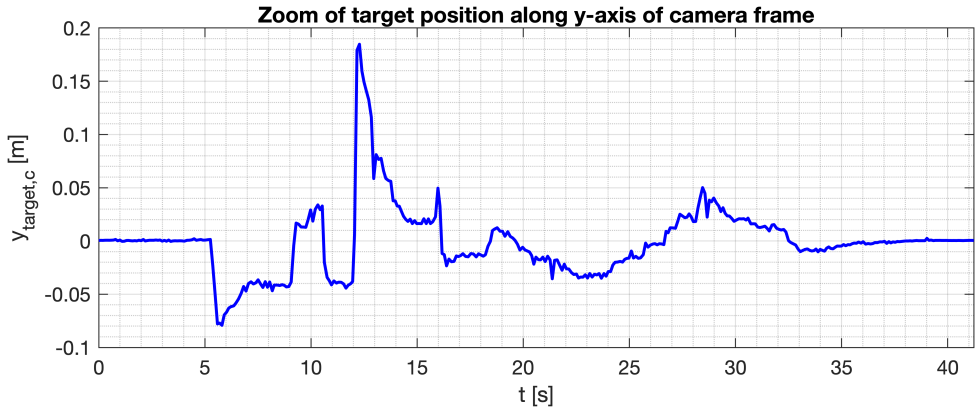


Figure 5.8: Representation of the path travelled by the robot in simulation scenario #2

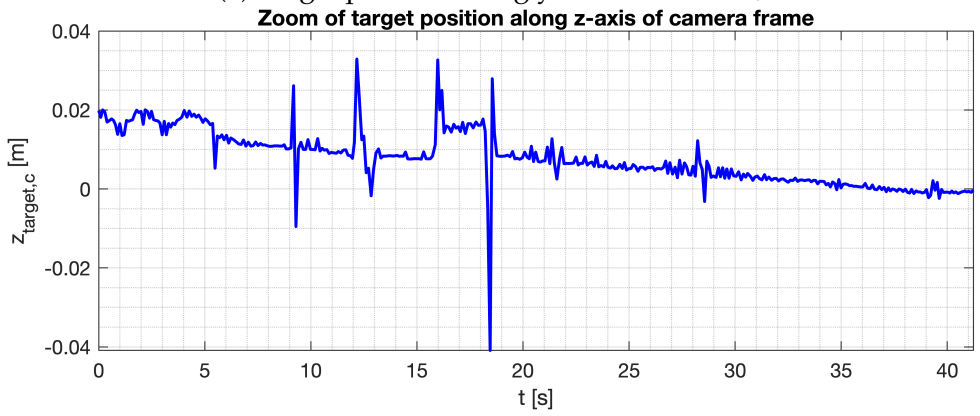
of the target, and in particular the one along the y-axis of the camera reference frame, presents more perturbations in its evolution. This can be appreciated by taking a look at the zoom out of the position of the target along the y and z-axis of the camera reference frame presented in figures 5.9a and 5.9b.

During the navigation of the robot, as shown in figure 5.8, it can be noticed how the mobile base needs to turn multiple times in order to avoid the obstacles and reach the target. With the same considerations made in the first simulation, during the rotation of the mobile base of the *Locobot*, the target starts slipping from the center of the horizontal linear FoV of the camera. The action of the target-locking algorithm corrects this misalignment, bringing again the target inside the center of the horizontal linear FoV of the camera. As proof of this, in the graph reported in figure 5.11a it is possible to notice how the shoulder joint rotates properly following the reference signal provided by the target-locking algorithm.

Instead, as in the first simulation, the height of the target is fixed, therefore the height-variation movement is done only in relation to a perspective change of the target with respect to the camera. The target-locking algorithm works correctly when keeping the arm fixed and when dealing with the perspective change during the motion. As in the first scenario, at the start of the simulation the alignment of the target with the camera is not perfect. This is caused again by the presence of a little steady-state error between the reference signal provided to each joint and the actual joint states (for reference take a look at the graphs reported in figure 5.11 and in particular the ones for the third and fourth joint in figures 5.11c and 5.11d), but, as for the first simulation scenario, it was decided to leave unaffected the gains of each controller to avoid uncontrolled movements of the arm, considering the results obtained acceptable for the target-locking algorithm purpose.

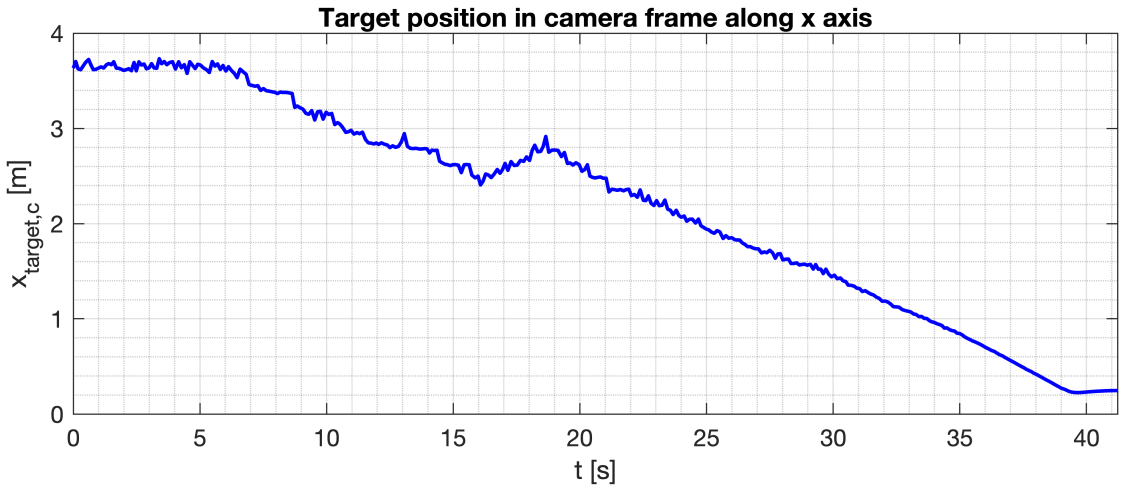


(a) Target position along y-axis of frame $\mathcal{F}_c$

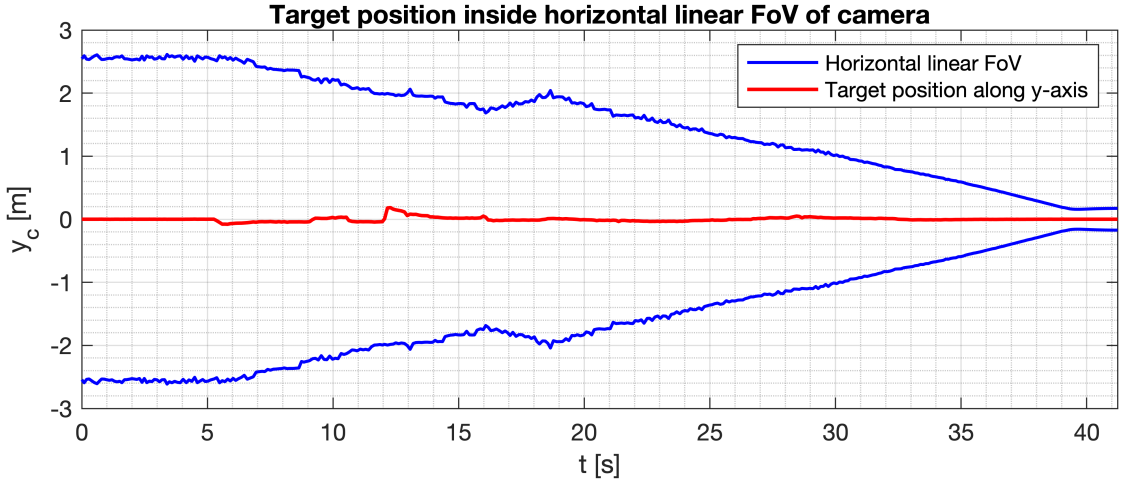


(b) Target position along z-axis of frame $\mathcal{F}_c$

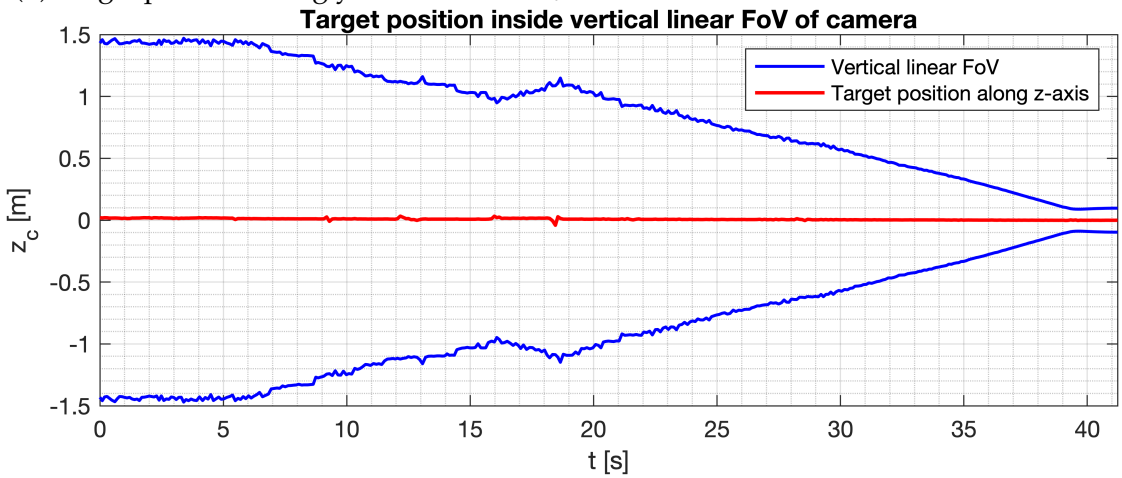
Figure 5.9: Zoom out of the position of the target expressed in the camera frame $\mathcal{F}_c$ for simulation scenario #2



(a) Target position along x-axis of frame $\mathcal{F}_c$

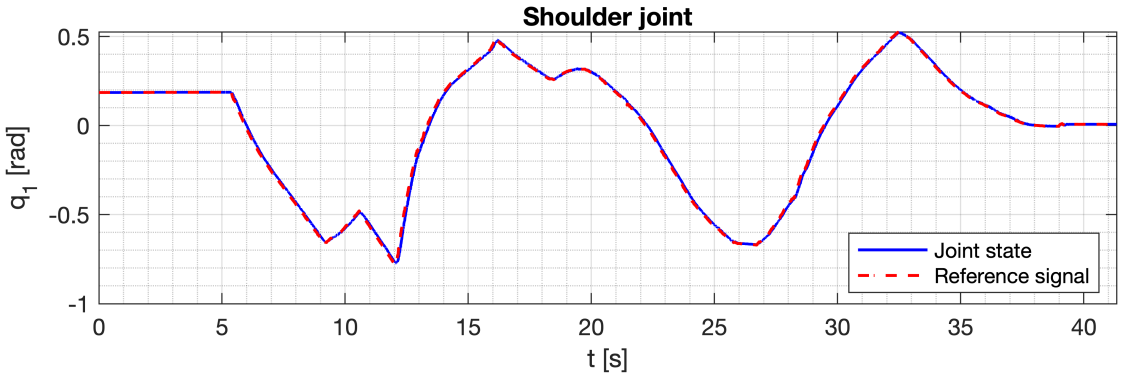


(b) Target position along y-axis of frame $\mathcal{F}_c$ inside horizontal linear FoV of the camera

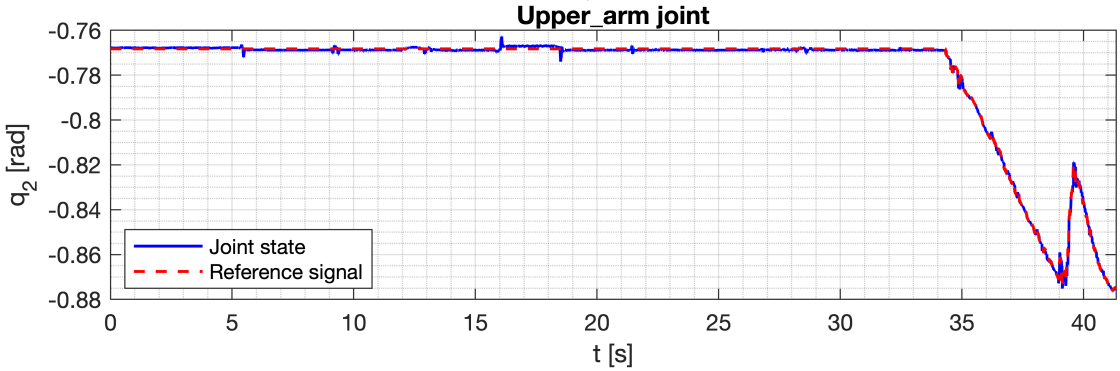


(c) Target position along z-axis of frame $\mathcal{F}_c$ inside vertical linear FoV of the camera

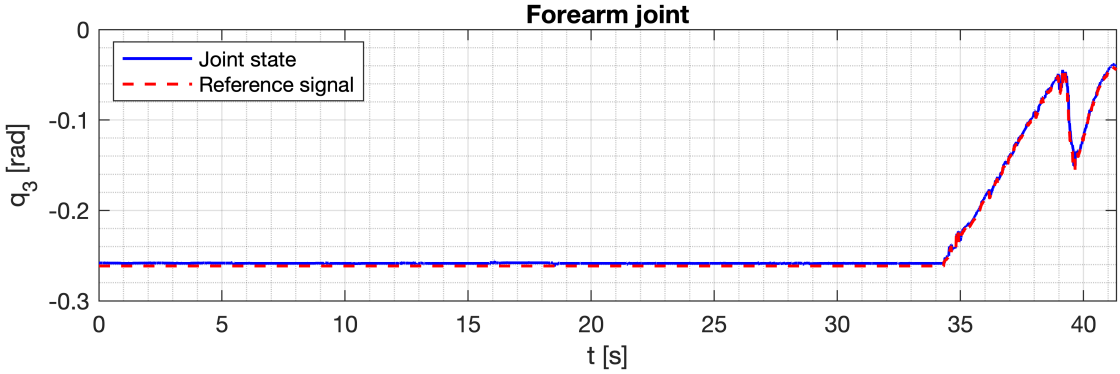
Figure 5.10: Position of the target expressed in the camera frame $\mathcal{F}_c$ for simulation scenario #2



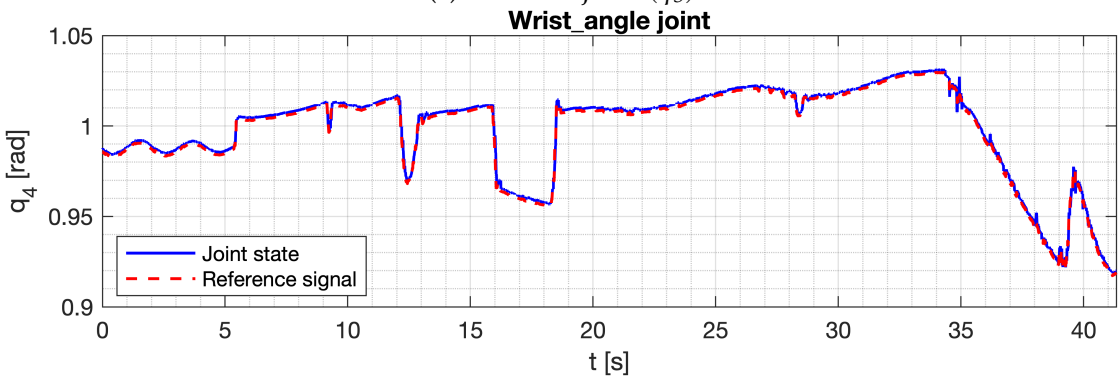
(a) Shoulder joint (q_1)



(b) Upper_arm joint (q_2)



(c) Forearm joint (q_3)



(d) Wrist_angle joint (q_4)

Figure 5.11: Detail of reference signals and joint states for simulation scenario #2

5.1.3 Conclusions

In this section there were presented the results obtained in a simulated environment to test the target-locking algorithm. These results are quite satisfactory and prove that the algorithm works correctly maintaining when possible the target centered inside both horizontal and vertical linear FoV of the camera. With these tests it was only possible to evaluate the performances of the algorithm in relation to a restrict number of motion cases, in particular when considering the height-variation movement of the arm. Instead, the rotation movement of the arm was tested properly and the results obtained prove the efficiency of the target-locking algorithm.

5.2 Experimental validation

The results obtained in the simulations are quite satisfactory and constitute a starting point for the real-world experiments. However, as mentioned in section 5.1.3, in the simulated environment it was only possible to perform only a part of the motion cases introduced in section 4.2.3, while in a real-world scenario as will be shown it was possible to perform experiments that included all of these motion cases.

In the simulated experiments, the implementation of the algorithm was quite straight-forward and did not required any precaution or special gimmick in order for it to work properly. The same thing cannot be said for the real-world experiments, where it was necessary to adopt some measures in order for the target-locking algorithm to work properly avoiding any damaging situation to the robot structure. All the measures adopted are presented in the appendix A.

As in the simulations, for the navigation inside the environment it was decided to use the NAPVIG algorithm of chapter 3, while the target is again

Parameter	Value
Resolution	640*480
Minimum detection distance	28[cm]
Horizontal angular Field of View	70 deg
Vertical angular Field of View	43 deg
Frame rate	60 fps

Table 5.3: Real-world camera settings

Joint	Proportional gain	Integrative gain	Derivative gain
shoulder_joint	700.0	55.0	0.0
upper_arm_joint	400.0	55.0	5.0
forearm_joint	550.5	55.5	3.0
wrist_angle_joint	459.0	70.0	3.0

Table 5.4: Gains of the PID controllers of the robotic arm used during the experiments

represented by an AprilTag of size $10[cm] \times 10[cm]$. The camera that has been used in the experiments mounted as end-effector of the *Locobot*'s robotic arm is an Intel®RealSense™ D435, whose main working characteristics are reported in table 5.3. Moreover, it was decided to reduce at the minimum possible the exposure of the camera in order to increase the performance of the target-locking algorithm while maintaining the target visible from the target.

Instead, for what concerns the controllers of each motor of the robotic arm, their gains have been tuned again using a trial and error process which led to the values indicated in table 5.4.

5.2.1 Scenario #1: Standing robot

In the simulated experiments there were only possible to test the performances of the target-locking algorithm about only the rotation movement of the arm, while for the other movement regarding its variation in height it was only possible to prove that the target-locking algorithm keeps always aligned the camera with the target along this axis.

Figure 5.12: Snapshot of the experiment #1

In the real-world setup, it is possible to change the position of the target while keeping the robot fixed, and with that it is possible to test the efficiency of the target-locking algorithm in both the movement direction of the target. The first real-world scenario is identical to the one used in the determination of the inverse kinematics of the robotic arm and will be used as proof of the effectiveness of the target-locking algorithm in the real-world setup. The robot is kept fixed in its startup position $\mathbf{x}_i = [0, 0]^T$ which is expressed with respect to its body reference frame $\mathcal{F}_b$. For what concerns the target, it has been placed at a distance of about $2[m]$ from the camera of the robot. The initial and the final position of the target are the same, while the movements done by it during the experiments are completely random and do not follow a specific path. The detail of the path followed by the target is reported in figure 5.13, while the Cartesian decomposition of the path followed is reported in figure 5.14. The path travelled by the target is expressed with respect to the base frame of the robot $\mathcal{F}_b$. In this test, the target is always visible from the camera and the movements that have been done are gentle, without any rapid change in the movement direction of the target.

Considering the position of the target inside both horizontal and vertical linear FoV of the camera, reported respectively in figures 5.15b and 5.15c, it can be noticed that the target never leaves the FoV of the camera, indicating that the target-locking algorithm is working properly in the following action of the target.

For what concerns the rotation movement of the arm, the action of the shoulder joint, reported in figure 5.16a, allows the correct following of the target. With respect to the horizontal linear FoV of the camera, the target is nearly centered inside it during the overall simulation, and never reaches the horizontal linear FoV limit.

Instead, in the height variation movement of the arm there has to be done different considerations. Inside the time slot around $t_1 = 39[s]$ and $t_2 = 45[s]$, the target reaches dangerously the limit of the vertical linear FoV, however the action of the target-locking algorithm prevents the losing of it. Recalling that the height variation movement is obtained using the 3R planar arm portion of the *Locobot*'s arm, to evaluate why this is happening it is necessary to take a look at the graphs in figures 5.16b, 5.16c and 5.16d. As can be noticed, the reaction of the motors driving the upper_arm and the forearm joint, with respect to the reference signal provided by the target-locking algorithm, is slightly slow when

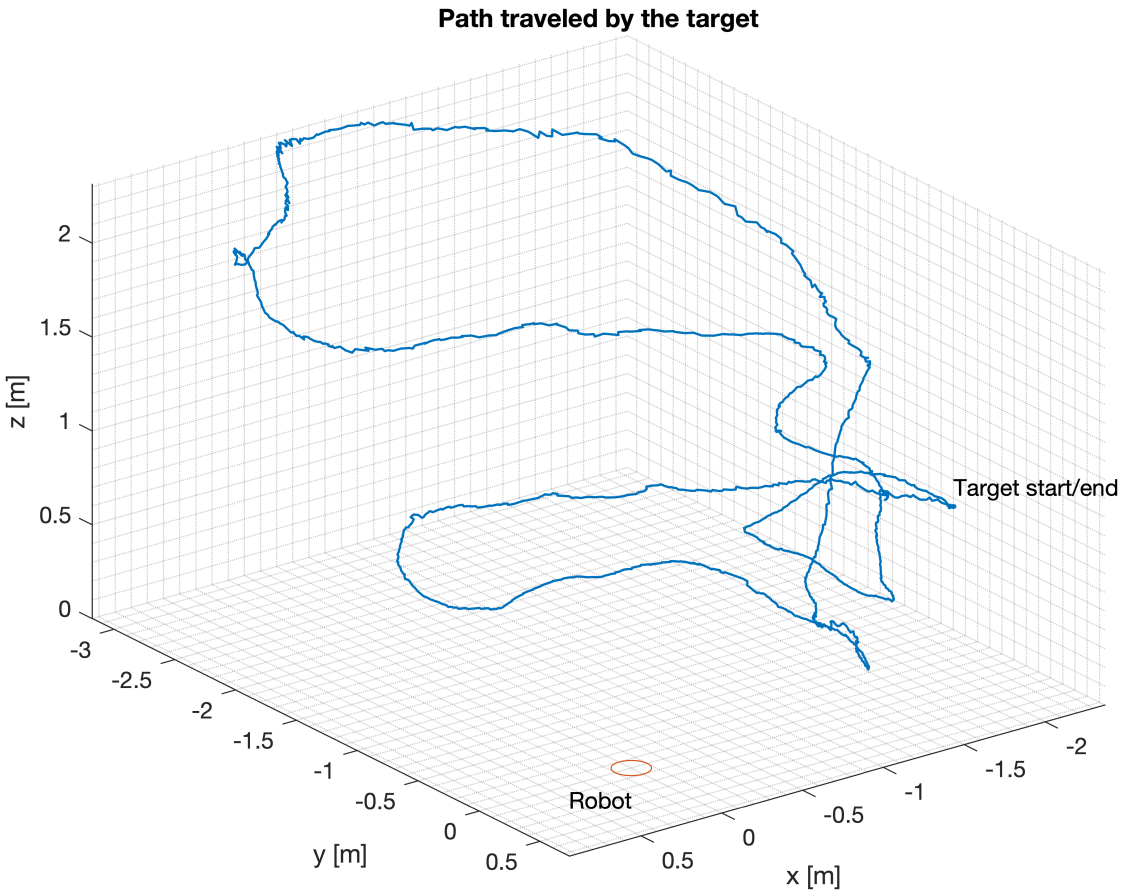
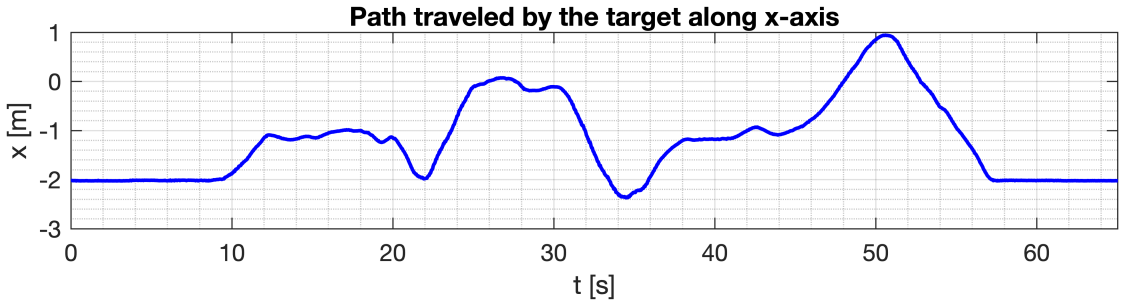


Figure 5.13: Detail of the path traveled by the target during the experiment #1

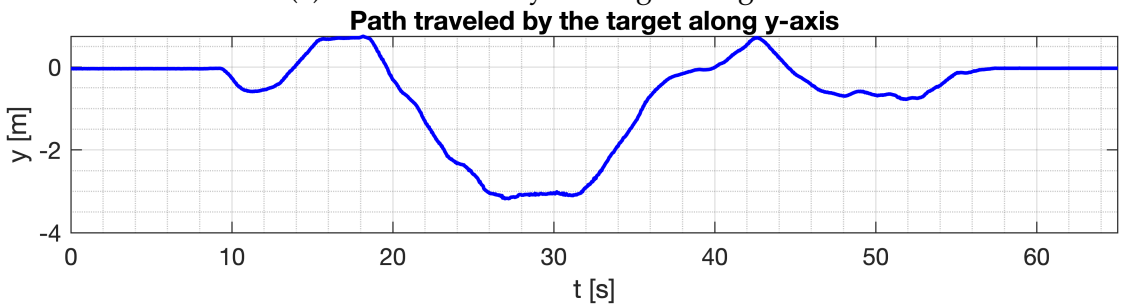
compared to the other graphs. This effect, united to the way the arm moves when performing the height variation movement, causes the target to reach the border of the vertical linear FoV. Increasing the effect of the control action could fix this problem by making the arm faster, however since during the experiment the target never left the vertical linear FoV of the camera, it was decided to leave the gains of the controllers unmodified.

On the other hand, considering the rest of the experiment, it is possible to notice one thing. Both the upper_arm joint and the forearm joint are fixed during the major part of it, which indicates that the arm is positioned in the **fully-extended** configuration defined in section 4.2.5. During these time slots however the target is still followed by the camera, and this result is possible by the motion of the wrist_angle joint, which in fact continues to move even when the arm is fully extended. This result can be used to prove the hypothesis relaxation made at the end of section 4.2.5.

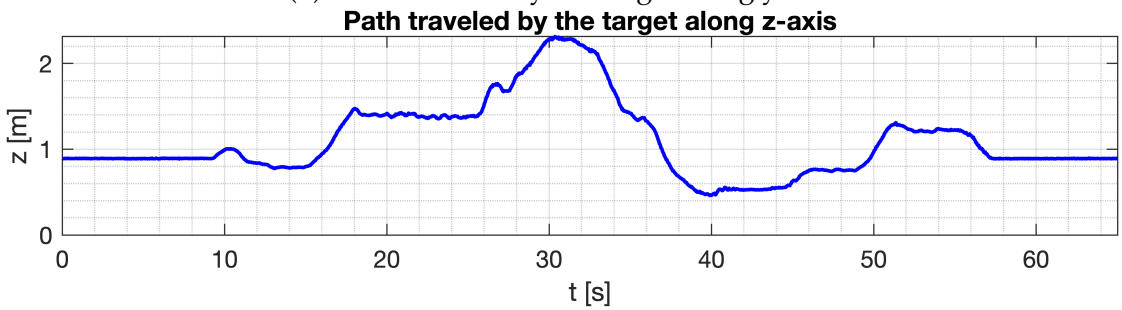
At last, as can be noticed in figure 5.15c, at the end of the experiment the



(a) Path travelled by the target along x-axis



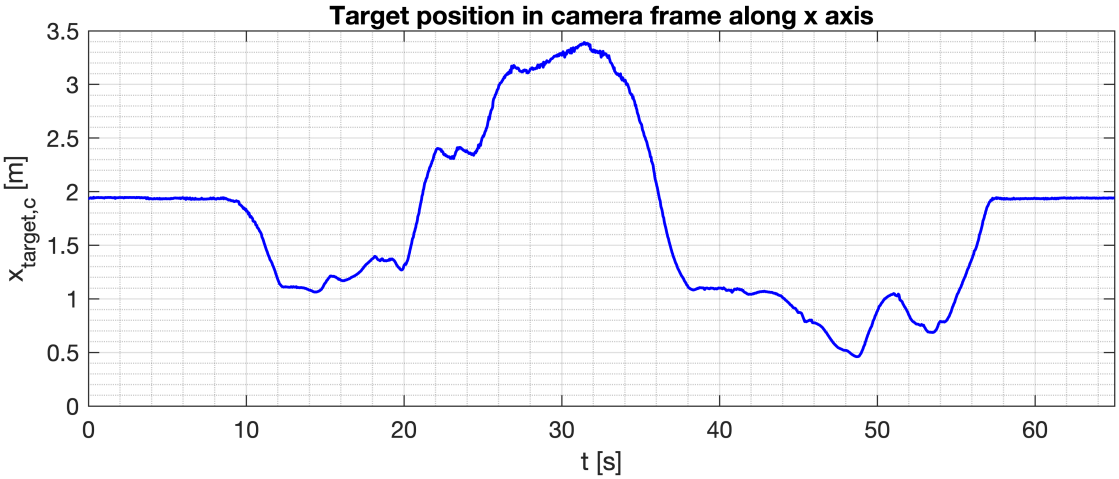
(b) Path travelled by the target along y-axis



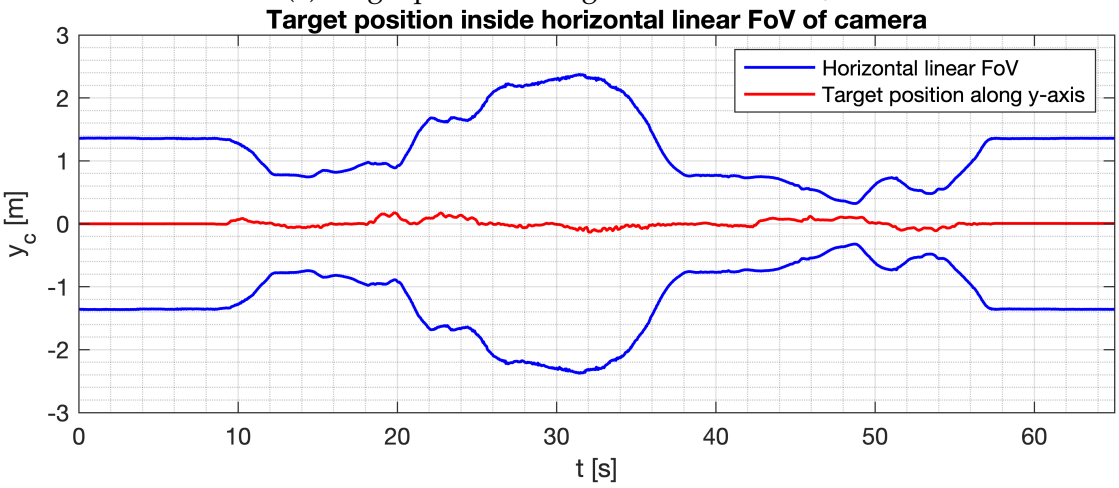
(c) Path travelled by the target along z-axis

Figure 5.14: Cartesian decomposition of the path travelled by the target in the experiment #1

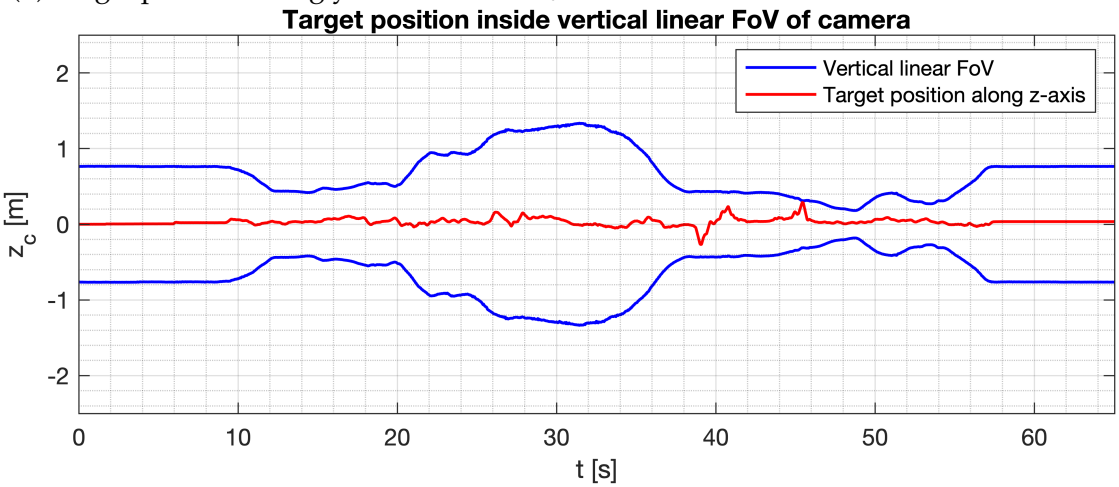
position of the target along the z-axis is not perfectly zero. This problem is not due to an issue of the target-locking algorithm but it has to be imputed to the mechanical drift of the motor, which may cause a little misplacement in the joint position. However, for the purposes of the target-locking algorithm, this error can be considered acceptable.



(a) Target position along x-axis of frame $\mathcal{F}_c$

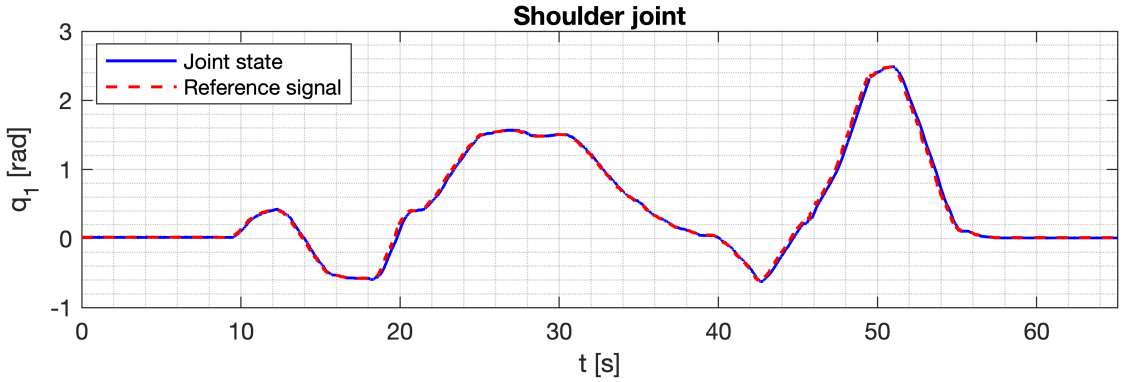


(b) Target position along y-axis of frame $\mathcal{F}_c$ inside horizontal linear FoV of the camera

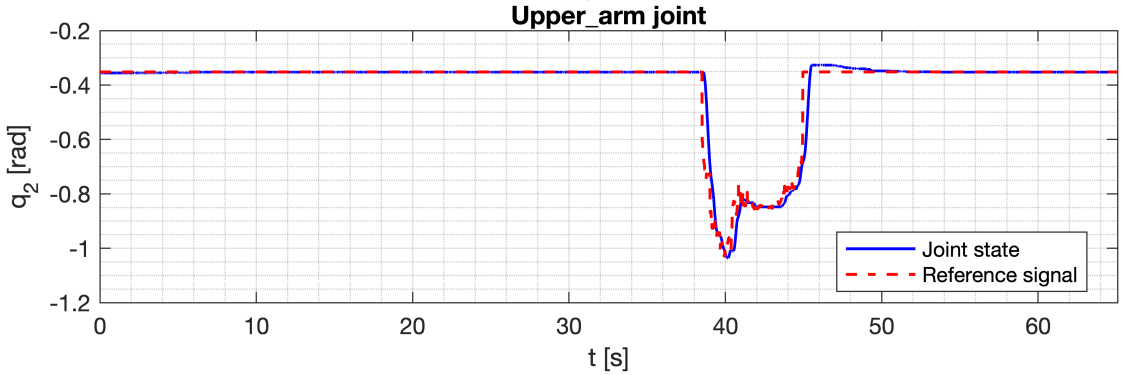


(c) Target position along z-axis of frame $\mathcal{F}_c$ inside vertical linear FoV of the camera

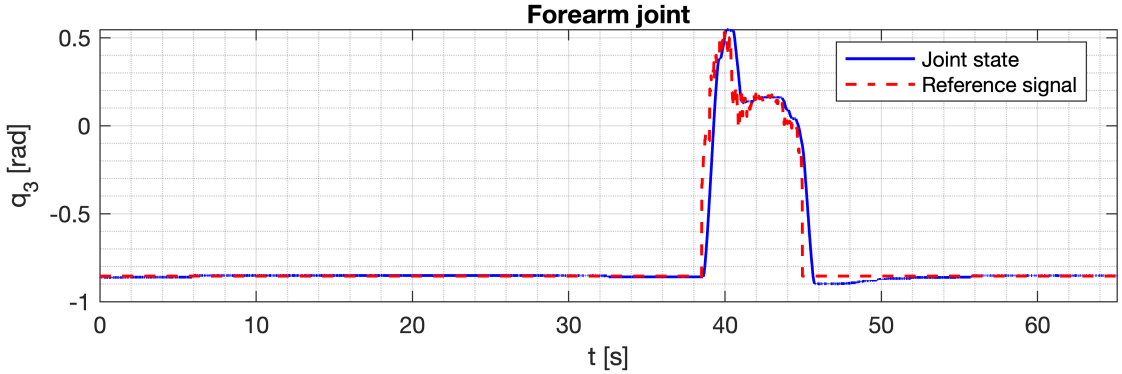
Figure 5.15: Position of the target expressed in the camera frame $\mathcal{F}_c$ for experiment #1



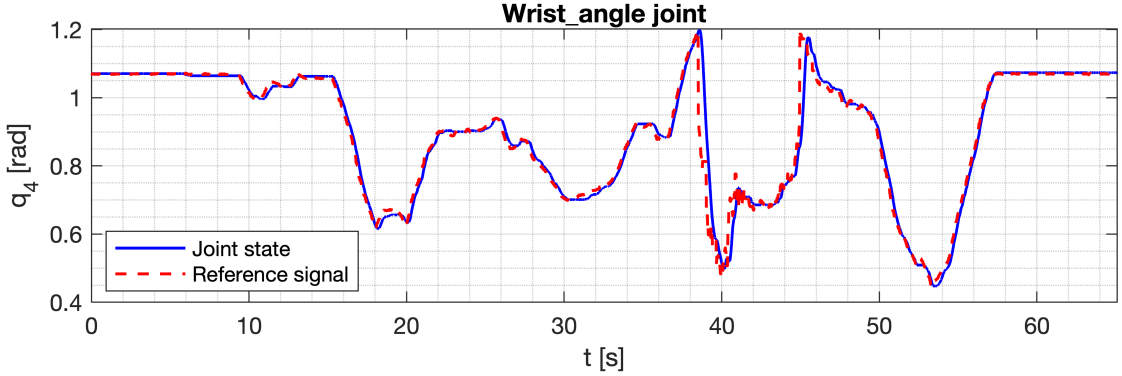
(a) Shoulder joint (q_1)



(b) Upper_arm joint (q_2)



(c) Forearm joint (q_3)



(d) Wrist_angle joint (q_4)

Figure 5.16: Detail of reference signals and joint states for experiment #1

5.2.2 Scenario #2: Target and robot moving in cluttered environment

The last experiment performed in the real world was done considering the case where both the target and the robot are moving in the environment. With respect to the experiments done so far, this one is more sophisticated and, to simplify the analysis of the results obtained, it has been divided into two phases:

- in the first phase the robot is required to reach the target while navigating inside a cluttered dynamic environment using the NAPVIG algorithm. During this phase the target is fixed and it is positioned in such a way it is always detectable by the camera of the robot. The target is mounted below a quadcopter;
- in the second phase of the experiment the quadcopter, and therefore the target, starts moving inside the environment following a user-defined trajectory. This requires the *Locobot* to move again inside the cluttered environment in order to follow it.

The detail of what happens during each phase will be described later when analyzing the results obtained. In figure 5.19 there are reported some snapshots

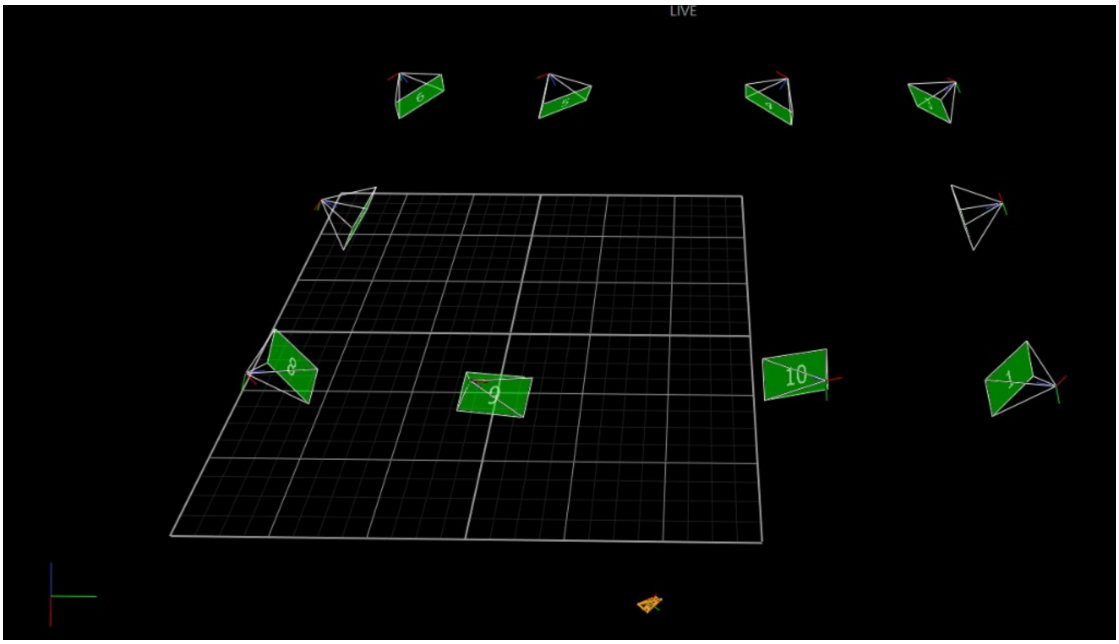


Figure 5.17: Representation of the Vicon environment

of the laboratory environment in which the experiment took place. To get a precise estimate of the path travelled by the *Locobot* and the quadcopter, it was decided to adopt the Vicon motion capture system installed in the laboratory.

The Vicon system is made of ten infrared cameras that, working together, allows the tracking of the robot. Their displacement inside the laboratory is shown in figure 5.17. The detection of the movement is made by making use of four reflector points placed on the robot in an asymmetrical configuration. In figure 5.18 there is reported the detail of the path followed by the quadcopter in the second phase of the experiment, expressed in the reference frame of the Vicon, where there have been put in evidence the positions where it was decided to make it stop. Instead, the snapshots of the path followed by the *Locobot* during the experiments, expressed in the reference frame of the Vicon, are presented in figures 5.20 and 5.22.

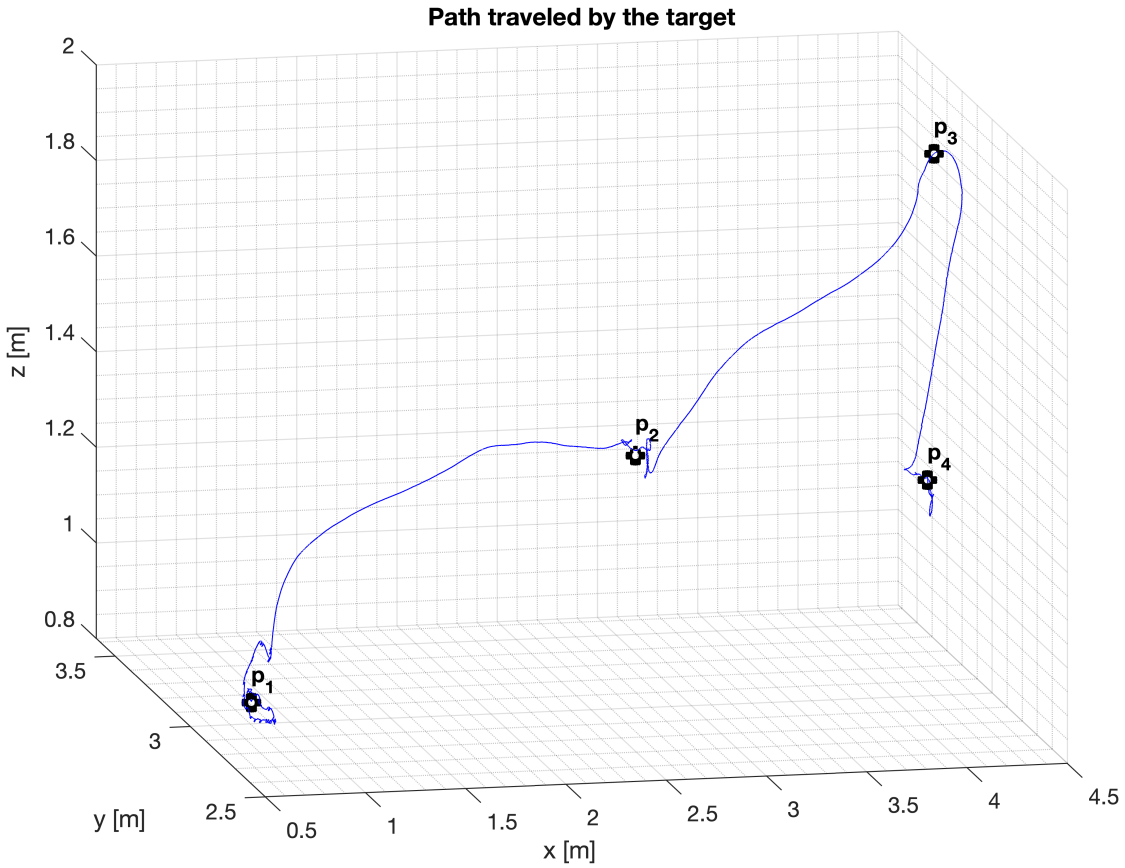


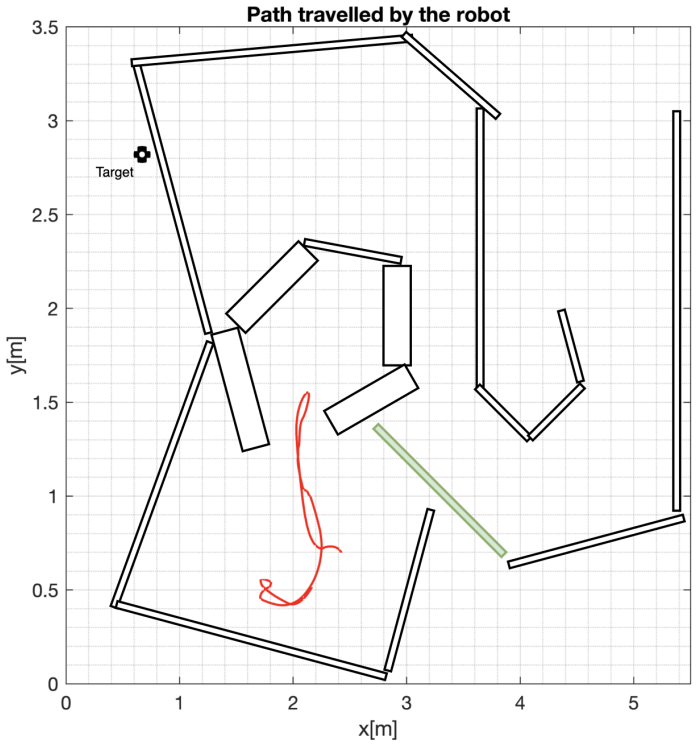
Figure 5.18: Path travelled by the quadcopter during the second phase of the experiment #2

Figure 5.19: Snapshots of real world experiment #2. In red there are reported the positions in which the quadcopter will stand during the second phase of the experiment. The circle in red on the first picture instead represents the dynamic obstacle

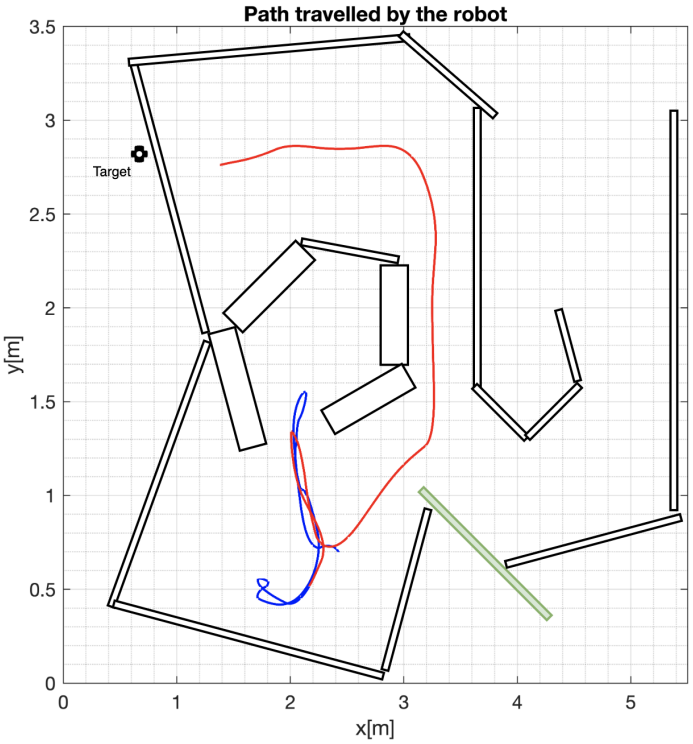
Experiment #2 - phase #1

The first phase of the experiment lasts for about 110[s] from the start of it, which corresponds approximately with the time instant in which the robot reaches the target. During this phase, the target is fixed inside the environment and the robot needs to find a suitable path to reach it inside a cluttered dynamic environment. Considering the snapshots of the experiment in figure 5.19, the area highlighted with the red circle is closed at the start of the experiment and opens during the first phase. This provides to the robot a suitable path that it can travel to reach the target. The path followed by the robot during this phase is presented in figure 5.20.

In this phase, the target-locking algorithm is able to maintain the lock on the target during the navigation process in most of the cases. Starting from the position of the target inside the vertical linear FoV of the camera, reported in figure 5.21c, it can be noticed that the target remains centered for all the duration of this phase of the experiment. On the other hand, considering the position of the target inside the horizontal linear FoV presented in figure 5.21b, what can be noticed is that the target is continuously slipping away from the center of the FoV, while the action of the target-locking algorithm tries to bring it back again to the center of the horizontal linear FoV. The slipping is due to the fast rotations performed by the mobile base of the *Locobot*, which is turning in order to find a suitable path to reach the target and to move away from situations where the navigation would lead to closed roads. In figures 5.21b and 5.21c, there can be noticed the presence of some areas highlighted in red. These correspond to the time slots in which the target is not directly visible from the camera. As mentioned before, the rotation of the mobile base requires the action of the target-locking algorithm to maintain the target centered in the FoV of the camera, and in particular for the rotation movement it requires the rotation of the shoulder joint of its robotic arm. In some cases, it may happen that this joint reaches its rotation limit, making therefore unable the following of the target. To solve this problem, the realigning mechanism presented in appendix A performs an appropriate rotation in order to remove the shoulder joint from the "stuck" condition, allowing again the following of the target during the navigation. As proof of this, by taking a look at graph 5.23a, it can be noticed that the shoulder joint, where it reaches its limit, performs a rotation in the opposite direction to remove from the stuck condition.

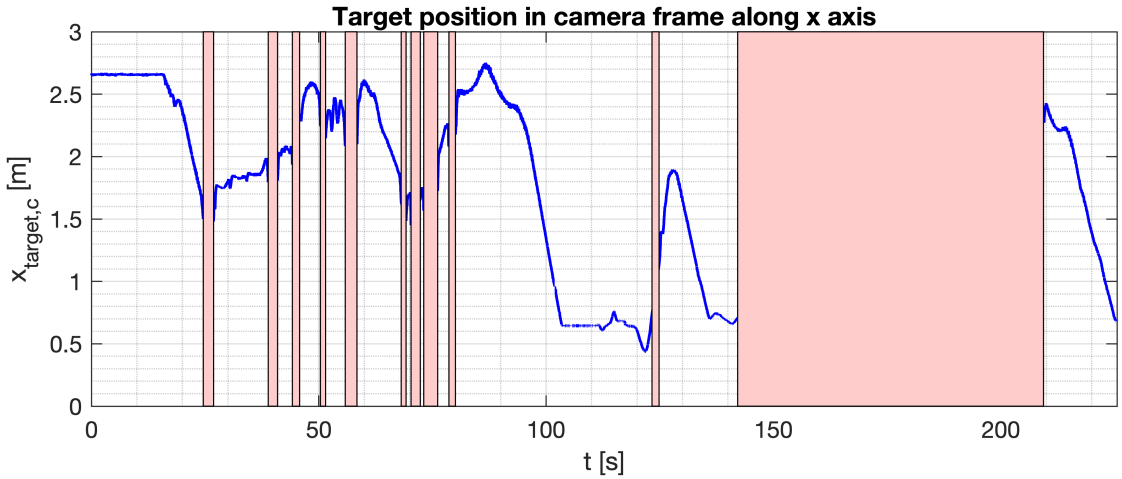


(a) Path travelled before the obstacle removal, which is represented in green

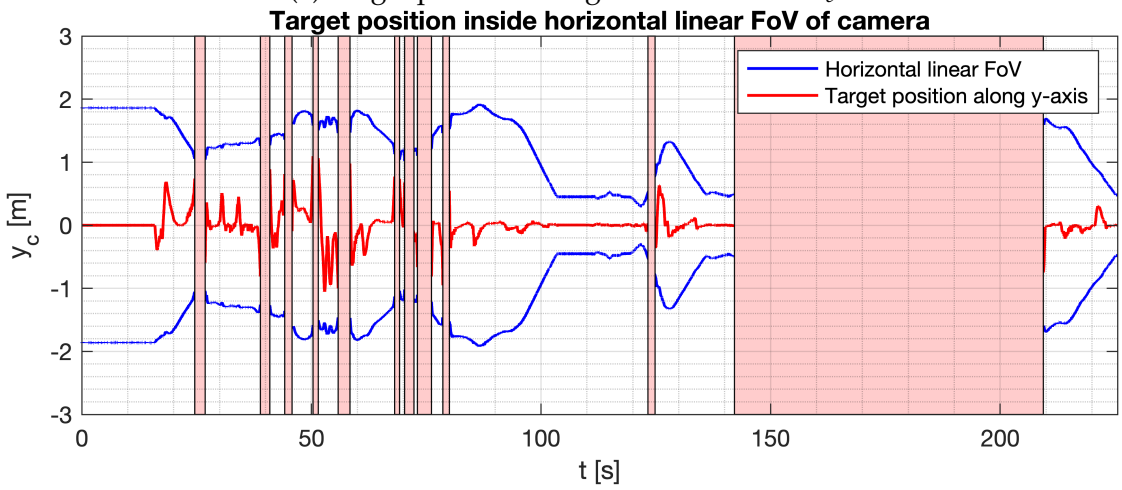


(b) Path travelled after the obstacle removal. The path in blue represents the path travelled before the obstacle removal

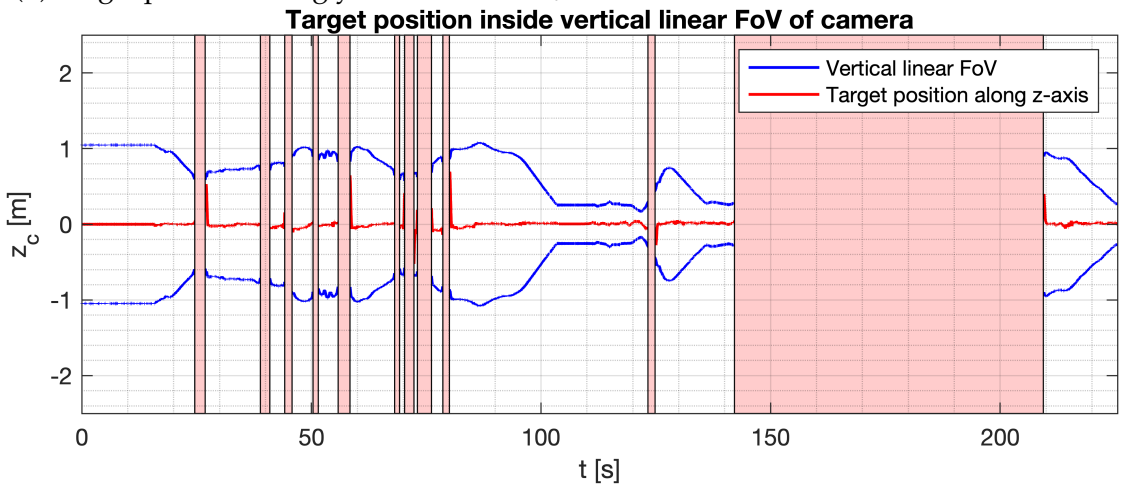
Figure 5.20: Path travelled by the *Locobot* during the first phase of the experiment #2



(a) Target position along x-axis of frame $\mathcal{F}_c$



(b) Target position along y-axis of frame $\mathcal{F}_c$ inside horizontal linear FoV of the camera



(c) Target position along z-axis of frame $\mathcal{F}_c$ inside vertical linear FoV of the camera

Figure 5.21: Position of the target expressed in the camera frame $\mathcal{F}_c$ for experiment #2

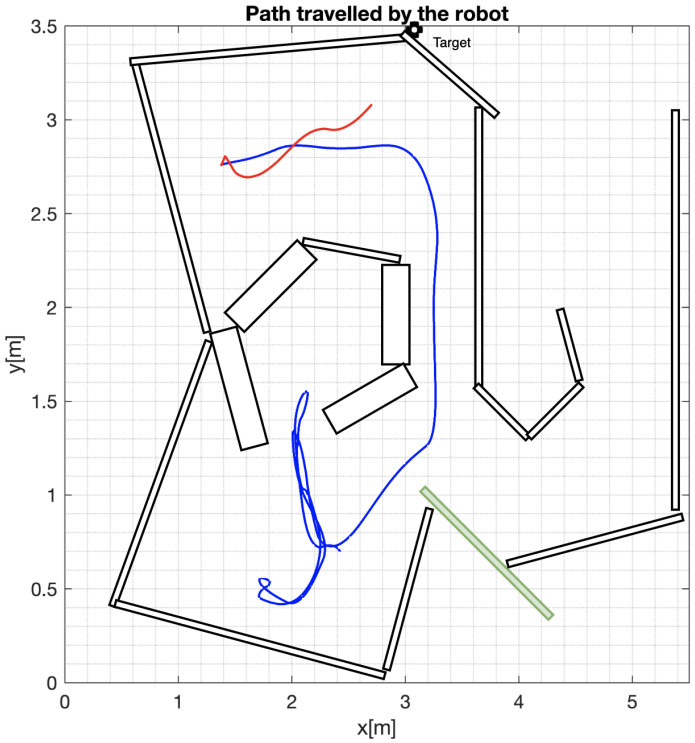
Experiment #2 - phase #2

The second phase of the experiment starts after the *Locobot* reaches the target and spans from the time instant of $t_0 = 110[s]$ until the end of the experiment. At that time instant, the quadcopter departs from its initial position p_1 and starts following the trajectory presented in figure 5.18. During the transition from the position p_1 to the position p_2, the *Locobot* continues to detect correctly the target and follows it. Instead, in the transition from the position p_2 to the position p_4, the target is occluded to the sight of the *Locobot*, requiring therefore an exploration of the map to find it again and reach it. The detail of the path followed by the robot in the second phase is reported in figure 5.22 and its a continuation of the path obtained in the first phase of the experiment.

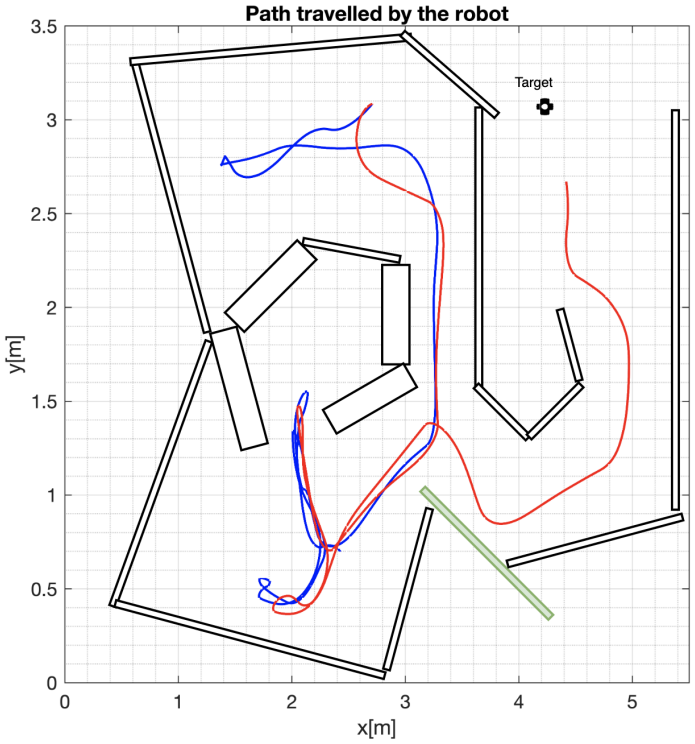
In this phase, while the target is still visible, the performances of the target-locking algorithm are still comparable to the ones obtained for the first phase of the experiment, with the target being correctly centered in both horizontal and linear FoVs of the camera. What is more interesting to analyze in this phase is the time period which spans from $t_1 = 142[s]$ until $t_2 = 210[s]$, which corresponds to the period in which the target is occluded to the camera. During this time slot, the mobile base of the *Locobot* performs the exploration of the map, searching for the target. To help this search, instead of keeping the arm fixed in the last position reached, the effect of the periscope mechanism described in appendix A makes the shoulder joint of the robot swing back and forth, increasing the possibility of the detecting the target during the navigation. At the time instant t_2, the camera gets again the lock on the target and the mobile base starts to reach it. Again, the target-locking algorithm with its action maintains the target centered in the FoV of the camera, until the end of the experiment where the *Locobot* reaches the target.

5.2.3 Conclusions

The results obtained prove that the target-locking algorithm works correctly in the real-world setup. The discussion of the results shows that the approach is effective and is able to deal even in fast-varying situations. The support provided by the mechanisms described in appendix A is fundamental, especially in the last experiment, and drives the robot even further in the direction of the fully autonomy.

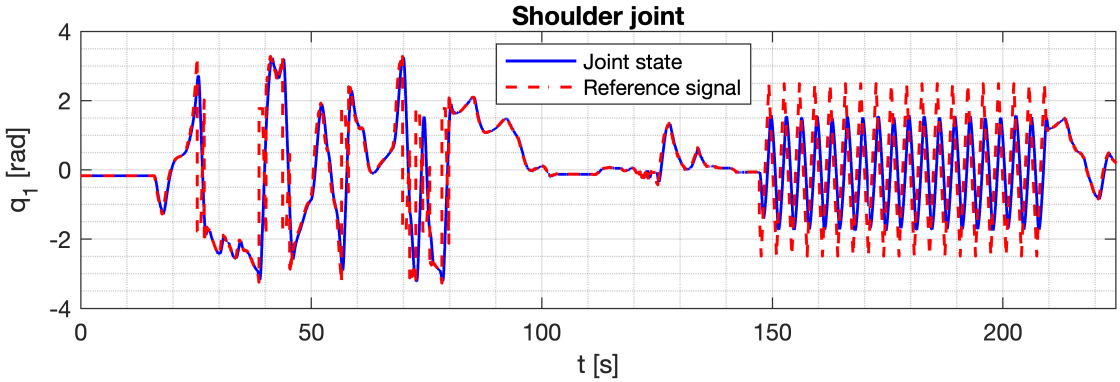


(a) Path travelled to follow the quadcopter from position p_1 to position p_2

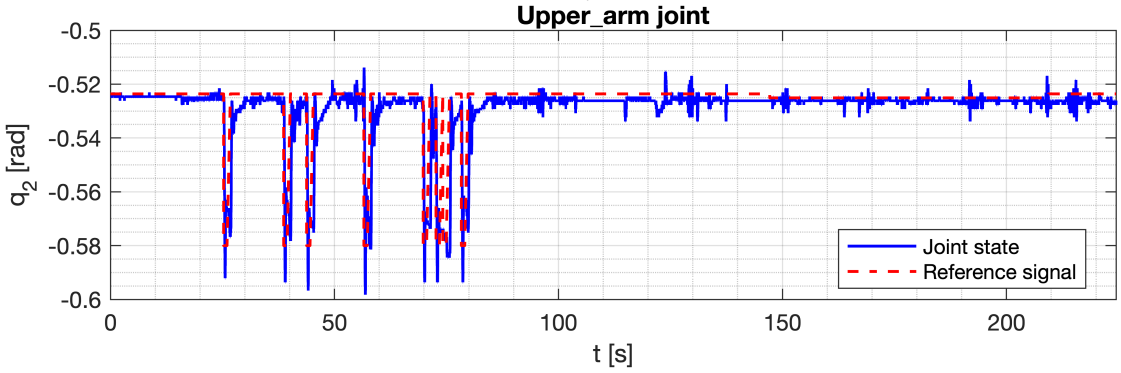


(b) Path travelled in the environment exploration between position p_2 and position p_4

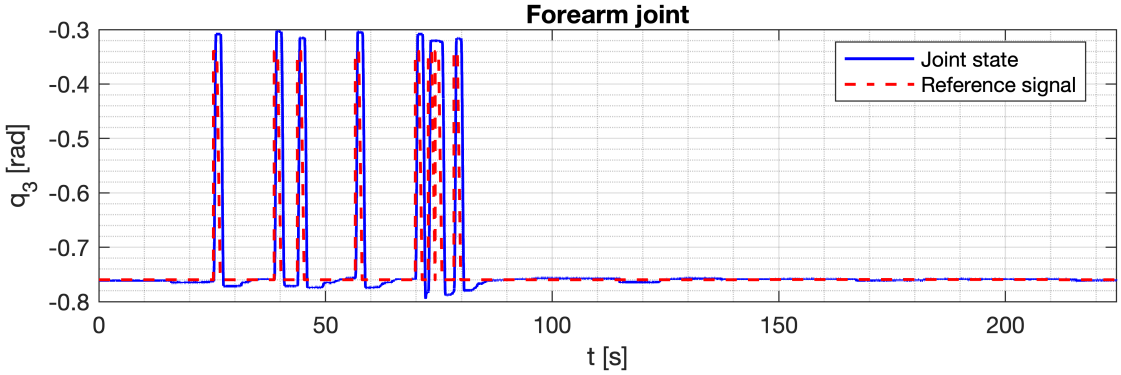
Figure 5.22: Path travelled by the *Locobot* during the second phase of the experiment #2



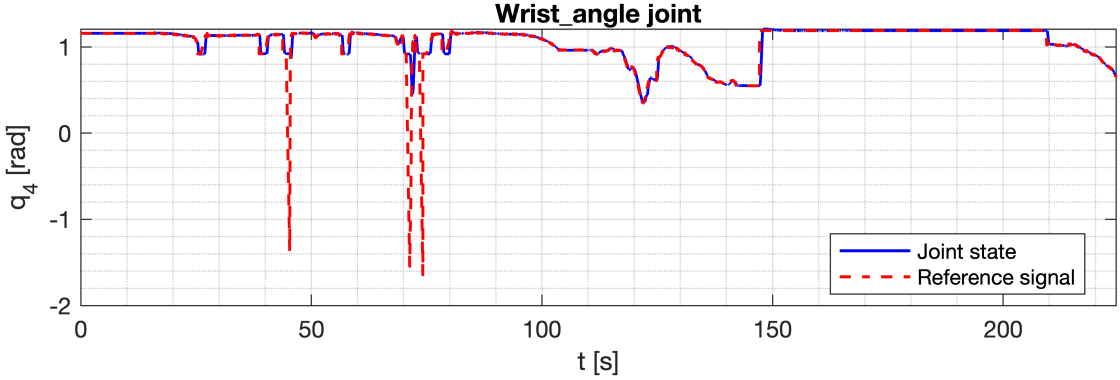
(a) Shoulder joint (q_1)



(b) Upper_arm joint (q_2)



(c) Forearm joint (q_3)



(d) Wrist_angle joint (q_4)

Figure 5.23: Detail of reference signals and joint states for experiment #2

6

Conclusions and Future Works

In this thesis, it has been presented an algorithm that allows the constant following of a target using a camera mounted on a robotic arm. The performances obtained for the target-locking algorithm in both the simulated and real-world experiments are quite satisfactory.

Starting from the simulated experiments, the target-locking algorithm responded well when dealing with the turns of the mobile robots, providing the appropriate driving signal to the shoulder joint of the arm in order to keep always in sight the target. On the other hand, considering the height variation movement of the arm, the algorithm was able to maintain it in a proper configuration during all the navigation process of the mobile base. Moreover, the problems related to the modeling allowed to prove that the algorithm works well even in situations where external disturbances are added and consequently rejected.

For what concerns the real-world experiments, the results were as satisfactory as the ones obtained in the simulations. In case of standing robot, the following of the target has been proven to be quite easy, with the robotic arm responding correctly to the inputs provided by the target-locking algorithm over time. Considering instead the results obtained in the last experiment, the following of the target during the navigation proves to be more complicated, especially when dealing with the fast rotations of the mobile base of the *Locobot*. Even in this cases however, the action of the target-locking algorithm works properly when dealing with the fast variation of the movement direction, with the major limit residing in the action of the motors driving the robotic arm. Also,

the aid provided by the support mechanisms presented in appendix A allows the correct detection and search of the target during the navigation, and in this sense, gives a huge contribute when perceiving the fully autonomy of the robot.

The target-locking algorithm has been developed for the solely use with robotic manipulators which presents the same mechanical structure as the one depicted in section 4.2.1. With the appropriate modifications however it is possible to implement it also in with different typologies of robotic manipulators, with the tricky part resting in the determination of a proper inverse kinematics procedure for the arm chosen. Nonetheless, it is still possible to exploit the structure of the different arm in order to find some expedients that could simplify the entire process. In the perspective of enhancing the autonomous capabilities of the robot, a possible future work related to this project may concern the usage of a different target instead of using an AprilTag. By making use of appropriate machine learning algorithm, it could be possible to make the robot follow a different object or even a person as final target to approach.

A

Support mechanisms for the target-locking algorithm

The determination procedure of the target-locking algorithm presented in chapter 4 has been done in a complete ideal scenario, where the joints of the arm nearly completely free to move and where it has been ignored the presence of cables connecting the motors of the arm or the camera. In the real-world scenario however, it is impossible to ignore these factors. In the following there are presented some mechanisms that have been implemented alongside the basic target-locking algorithm in order to avoid issues that may arise in the real world.

A.1 Target acquisition

The acquisition of the target is done by the camera mounted on the arm and, as mentioned in section, 4.2, the estimate of the pose of the target is obtained using the AprilTag detection algorithm. The obtained estimate is then used in the evaluation of the proper joint values that the arm has to assume in order to follow the target. To get a faster execution of the target-locking algorithm, it was decided to store the estimated pose each time a new estimate arrives, and it was done for the following reasons:

- the rate of acquisition of the target position is different from the rate at which the target-locking algorithm works, therefore by keeping always in

memory the last estimated position of the target allows to speed up the entire following mechanism;

- in the real-world scenario it may happen that a fast rotation of the base causes the exiting of the target from the camera FoV with a consequential loss of it. In these cases, the knowledge of the last estimated position allows the target-locking algorithm to realign correctly the arm with the target even in cases where the target is temporarily lost.

A.2 Rapid turn mechanism

In the determination of the target-locking algorithm there were only presented the joint limit regarding the upper_arm, forearm and wrist_angle, while it was ignored on purpose the limit about the shoulder joint. In an ideal scenario, this joint could rotate forever without any restraint, while in the real-world scenario this is impossible due to the presence of cables connecting the motors of the arm to the controller which drives them. In this sense, it was necessary to introduce an operative limit for the shoulder joint, which spans in the range $[-190°, 190°]$. This however puts a restrain on the target-locking algorithm. In the ideal case, the complete rotation of the base of the robot does not limits the rotation of the shoulder joint of the arm, with it being able to follow the target infinitely. In the real world scenario instead, when the shoulder joint is at its limit, it may happen that the rotation of the base of the robot makes the camera lose the target. In these cases, it was decided to implement a mechanism that performs a 360° rotation in the direction opposite to the limit reached by the shoulder joint. During the perform the rotation, the 3R planar arm section is put in a safe position in order to avoid collisions with the base of the robot. The idea in performing this rotation is to put the arm again in a situation where it can freely rotate to follow the target. After the rotation, it could happen that the target is still not visible from the camera, however, the knowledge of the last estimated position of the target allows the target-locking algorithm to perform the correct motion to align correctly the camera with the target.

A.3 Periscope mechanism

In some cases, it may happen that the target is not directly visible from the start or it may happen that a quick change in its position causes the arm to lose it forever. To push even further the robot towards the fully autonomy, it was decided to develop a simple mechanism that, after fixing the arm in the *fully-extended* configuration, starts rotating the shoulder joint of the arm in the range $[-90°, 90°]$, acting in a way similar to a submarine's periscope. In this way, the arm starts searching in the surroundings of the robot for the target, and if it is found, it locks on it with the effect of the target-locking algorithm.

References

[1] Haider A. F. Almurib, Haidar Fadhil Al-Qrimli, and Nandha Kumar. "A review of application industrial robotic design". In: *2011 Ninth International Conference on ICT and Knowledge Engineering*. 2012, pp. 105–112.

[2] J.P. Barreto et al. "FED-the free body diagram method. Kinematic and dynamic modeling of a six leg robot". In: *AMC'98 - Coimbra. 1998 5th International Workshop on Advanced Motion Control. Proceedings (Cat. No.98TH8354)*. 1998, pp. 423–428.

[3] Johann Borenstein, Yoram Koren, et al. "The vector field histogram-fast obstacle avoidance for mobile robots". In: *IEEE transactions on robotics and automation* 7.3 (1991), pp. 278–288.

[4] Johann Borenstein et al. "Mobile robot positioning: Sensors and techniques". In: *Journal of robotic systems* 14.4 (1997), pp. 231–249.

[5] B. Brown and G. Zeglin. "The bow leg hopping robot". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 1. 1998, 781–786 vol.1.

[6] Luca Bruzzone and Giuseppe Quaglia. "Locomotion systems for ground mobile robots in unstructured environments". In: *Mechanical sciences* 3.2 (2012), pp. 49–62.

[7] Raja Chatila. "Path Planning and Environment Learning in a Mobile Robot System". In: *Proceedings of the 5th European Conference on Artificial Intelligence*. ECAI'82. Paris, France: North-Holland, 1982, pp. 211–215.

[8] Peter I Corke. "A simple and systematic approach to assigning Denavit–Hartenberg parameters". In: *IEEE transactions on robotics* 23.3 (2007), pp. 590–594.

[9] Yuval Davidor. *Genetic Algorithms and Robotics: A heuristic strategy for optimization*. Vol. 1. World Scientific, 1991.

[10] MA Porta Garcia et al. “Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation”. In: *Applied Soft Computing* 9.3 (2009), pp. 1102–1110.

[11] Erann Gat, R Peter Bonnasso, Robin Murphy, et al. “On three-layer architectures”. In: *Artificial intelligence and mobile robots* 195 (1998), p. 210.

[12] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[13] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[14] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.

[15] O. Khatib. “A unified approach for motion and force control of robot manipulators: The operational space formulation”. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[16] Steven M LaValle, James J Kuffner, BR Donald, et al. “Rapidly-exploring random trees: Progress and prospects”. In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 293–308.

[17] Yun-Jung Lee and Zeungnam Bien. “Path planning for a quadruped robot: an artificial field approach”. In: *Advanced Robotics* 16.7 (2002), pp. 609–627.

[18] Nicola Lissandrini et al. “NAPVIG: Local Generalized Voronoi Approximation for Reactive Navigation in Unknown and Dynamic Environments”. In: *2023 American Control Conference (ACC)*. 2023, pp. 28–33.

[19] Lyndon E. Llewellyn and Scott J. Bainbridge. “Getting up close and personal: The need to immerse autonomous vehicles in coral reefs”. In: *OCEANS 2015 - MTS/IEEE Washington*. 2015, pp. 1–9.

[20] David G Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[21] Pierre Merriaux et al. “A study of vicon system positioning performance”. In: *Sensors* 17.7 (2017), p. 1591.

[22] Emam Fathy Mohamed, Khaled El-Metwally, and AR Hanafy. "An improved Tangent Bug method integrated with artificial potential field for multi-robot path planning". In: *2011 International Symposium on Innovations in Intelligent Systems and Applications*. IEEE. 2011, pp. 555–559.

[23] Illah Reza Nourbakhsh. *Interleaving planning and execution for autonomous robots*. Vol. 385. Springer Science & Business Media, 2012.

[24] Colm Ó'Dúnlaing and Chee K Yap. "A "retraction" method for planning the motion of a disc". In: *Journal of Algorithms* 6.1 (1985), pp. 104–111.

[25] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 3400–3407.

[26] Larona Pitso Ramalepa and Rodrigo S Jamisola Jr. "A review on cooperative robotic arms with mobile or drones bases". In: *International Journal of Automation and Computing* 18.4 (2021), pp. 536–555.

[27] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. "A review of mobile robots: Concepts, methods, theoretical framework, and applications". In: *International Journal of Advanced Robotic Systems* 16.2 (2019), p. 1729881419839596.

[28] Francisco Rubio et al. "Direct step-by-step method for industrial robot path planning". In: *Industrial Robot: An International Journal* 36.6 (2009), pp. 594–607.

[29] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[30] Paul J Springer. *Military robots and drones: a reference handbook*. ABC-CLIO, 2013.

[31] Anthony Stentz and Is Carnegle Mellon. "Optimal and efficient path planning for unknown and dynamic environments". In: *International Journal of Robotics and Automation* 10.3 (1995), pp. 89–100.