Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Control Systems Engineering

Validation of robotic navigation strategies in unstructured environments: from autonomous to reactive

Supervisor: Prof. Angelo Cenedese

Co-supervisor: Dott. Nicola Lissandrini

Master Candidate: Giorgia Masin

Academic Year 2022/2023

Abstract

The main topic of this master thesis is the validation of a navigation algorithm designed to perform autonomously in unstructured environments. Computer simulations and experimental tests with a mobile robot have allowed reaching the established objective. The presented approach is effective, consistent, and able to attain safe navigation with static and dynamic configurations.
This work contains a survey of the principal navigation strategies and components. Afterwards, a recap of the history of robotics is briefly illustrated, emphasizing the description of mobile robotics and locomotion. Subsequently, it presents the development of an algorithm for autonomous navigation through an unknown environment for mobile robots. The algorithm seeks to compute trajectories that lead to a target unknown position without falling into a recurrent loop. The code has been entirely written and tested in MATLAB, using randomly generated obstacles of different sizes. The developed algorithm is used as a benchmark to analyze different predictive strategies for the navigation of mobile robots in the presence of environments not known a priori and overpopulated with obstacles. Then, an innovative algorithm for navigation, called NAPVIG, is described and analyzed. The algorithm has been built using ROS and tested in Gazebo real-time simulator. In order to achieve high performances, optimal parameters have been found tuning and simulating the algorithm in different environmental configurations. Finally, an experimental campaign in the SPARCS laboratory of the University of Padua enabled the validation of the chosen parameters.

Abstract

L'argomento principale di questa tesi è la validazione di un algoritmo di navigazione progettato per la navigazione autonoma in ambienti non strutturati. Simulazioni al computer e prove sperimentali con un robot mobile hanno permesso di raggiungere l'obiettivo stabilito. L'approccio presentato è efficace, coerente e in grado di raggiungere una navigazione sicura sia per configurazioni statiche che dinamiche.
Questo lavoro contiene un'indagine sulle principali strategie e componenti di navigazione. In seguito, viene brevemente illustrato un riassunto della storia della robotica, con enfasi sulla descrizione dei robot mobili e della locomozione. Successivamente, presenta lo sviluppo di un algoritmo per la navigazione autonoma attraverso un ambiente non noto per i robot mobili. L'algoritmo cerca di calcolare traiettorie che portano alla sconosciuta posizione di target senza ripetere continuamente gli stessi movimenti. Il codice è stato interamente scritto e testato in MATLAB, utilizzando ostacoli generati casualmente di diverse dimensioni. Tale algoritmo sviluppato, viene impiegato come benchmark per analizzare diverse strategie predittive per la navigazione di robot mobili in presenza di ambienti non noti a priori e sovrappopolati di ostacoli. In seguito, viene introdotto e analizzato un algoritmo innovativo per la navigazione, chiamato NAPVIG. L'algoritmo è stato sviluppato utilizzando ROS e testato nel simulatore a tempo reale Gazebo. Al fine di ottenere prestazioni elevate sono stati ricavati dei parametri ottimali tramite la simulazione dell'algoritmo in diverse mappe. Infine, una campagna sperimentale nel laboratorio SPARCS dell'Università di Padova ha permesso l'implementazione di NAPVIG in un robot mobile e la successiva validazione dei parametri ricavati dalle simulazioni.

Contents

List of Figures

1 Introduction

Nowadays, autonomous navigation topic is a big issue in research groups.
The direction that the industry is taking is heading towards the development of vehicles able to autonomously and safely navigate through different types of environments, such as urban roads or industrial plants. Autonomous navigating robots are transforming the way we live, work, and play, helping people with their daily tasks. They are employed in almost all fields of our life and they represent a huge resource to lean on.
In particular, in this thesis, we focus on the navigation of a ground robot in unknown and dynamic environments in the presence of static and mobile obstacles and crossroads. The peculiarity of this study is that autonomous navigation is set to take place in unknown environments and hence in the absence of a general map to guide the choices.
The lack of a mapped surrounding is a complication to overcome. Indeed, the environment must be sensed and analyzed through different types of sensors and the map needs to be created step by step. At the same time, the robot has to proceed in its journey and obstacles must be detected and avoided.
Moreover, navigation in unknown environments does not permit the computation of an optimal route and it is necessary to define a general rule to select which direction should be taken without getting stuck in dead-ends or in repetitive loops.

The thesis has a dual purpose. The first one concerns the development of a basic algorithm for autonomous navigation in unknown environments, using MATLAB software.
The second goal of the thesis is to simulate and tune a reliable and safe algorithm, named NAPVIG, designed by PhD student Nicola Lissandrini. After a campaign of numerical simulations performed on Gazebo simulator, several experimental tests were carried out on University's robotic laboratory, employing a differential drive robot (DDR).
To complete the first part of this work, a self-developed algorithm is designed to be capable of autonomously and safely exploring the assigned environment.
The idea behind the algorithm is to define a set of nodes, and to construct a link between reachable points in the map, from the robot's position. Points in the map that are visible from the robot's sensors can be interpreted as nodes of a graph. As the robot navigates through the environment, the graph is updated with new nodes and arcs connecting them. Arcs represent possible routes indeed if an arc between

nodes is not computed then does not exist a safe path from one to the other.
In the second part of the work, the simulation phase involves the tuning of some significant parameters of the NAPVIG algorithm, and the analysis of different types of environments to highlight the peculiarity of the algorithm and the situations in which its behaviour should be improved.

The thesis structure is organised as described below.

Chapter 2 firstly defines the term *Autonomy*, then gives a rapid introduction about the motivations behind the use of robotics and finally gives an overview of the main robotics evolution steps from the second half of '900 until now.

Different types of mobile robots and their components, such as controllers, actuators, sensors and power systems, are presented in Chapter 3. Then, it is introduced the theme of locomotion with particular emphasis on legged and wheeled locomotion. Subsequently, unmanned ground vehicles and kinematics are described. In the last part of the chapter is illustrated the DDR.

Chapter 4 gives an overview of the navigation task for autonomous mobile robots, both indoor and outdoor. Finally, the four phases of navigation are introduced and explained. In particular, perception, localization, cognition and motion control.

Chapter 5 concerns the explanation of the self-developed autonomous navigation algorithm in MATLAB and the relative numerical results. The last section of the chapter contains hints about further developments, such as introducing moving obstacles and testing the algorithm in a real robot.

The last chapter is dedicated to the NAPVIG algorithm. As mentioned before, firstly a campaign of numerical simulations has been performed. Afterwards, experimental tests have been carried out with the aforementioned algorithm applied to a real DDR robot. Finally, there are presented some future developments that could broaden the present work.

2 Overview of robotics

2.1 Introduction

There is not a commonly approved definition for the term robot indeed, there exist multiple different definitions of what precisely a robot is.
The term robot has evolved over time based on technological progress, and the evolution of robotic artefacts. Nowadays, this word is referred to more sophisticated and intelligent devices compared to the middle of the previous century.
The Robot Institute of America (RIA) in 1979 has defined robot as *a reprogrammable, multi-functional manipulator designed to move materials, parts, tools or specialized devices through various programmed motions for the performance of a variety of tasks.*

The International Organisation for Standardisation proposes a definition of robot in ISO 8373, intended to standardize and make universally accepted the characterization. ISO defines a robot as *an automatically controlled, reprogrammable, multipurpose, manipulator programmable in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.*
As defined in ISO, a certain degree of autonomy is required. Autonomy is the ability to perform intended tasks by its own reasoning process, without human intervention even with unforeseen situations and changing environments. The degree of autonomy differentiates what is considered a robot from other devices; the more a robot is independent in performing its tasks the more it can be thought to be intelligent and sophisticated.

According to the Japanese Industrial Robot Association (JIRA), robots can be divided regarding their level of self-sufficiency into six different classes in order of increasing autonomy:

- Class 1. Manual handling device: device characterized by several degrees of freedom and directly controlled by an operator. Some of them are also referred to as co-bots.
- Class 2. Fixed sequence robot: handling device which performs a fixed sequence of predefined actions using the same method, without an operator manoeuvring it.

- Class 3. Variable sequence robot: robot similar to the ones of class 2 except the fact that the sequence of actions can be reprogrammed easily. This property allows it to be quickly adapted to execute new assignments.
- Class 4. Playback robot: robot able to replicate a task. The operator performs the task manually, leading or controlling the robot, which is then able to repeat the task autonomously.
- Class 5. Numerical control robot: this type of robot moves through a sequence of actions received in the form of numerical data.
- Class 6. Intelligent robot: a robot capable of sensing the environment and able to complete a task despite changes around it.

The different institutions have different opinions on which of the previous classes should be considered robots or not. For example, the Robotics Institution of America (RIA) considers robots only from class 3.

Another robot classification system has been developed by the Association Francaise de Robotique (AFR):

- Type A: manually controlled handling devices and telerobotics.
- Type B: automatic handling devices with predetermined cycles.
- Type C: servo-controlled robots with programmable trajectories.
- Type D: same as type C but able to respond to their environment.

Robots are mainly divided into two broad categories: manipulators and mobile robots. The two classes differentiate in what they move: manipulators are fixed in space and move objects around them instead mobile robots move themselves leaving the surrounding world unchanged.
A robotic manipulator is a reprogrammable and multifunctional mechanical device used to manipulate materials without direct physical contact with the operator. Usually, it is a mechanism similar to an arm that consists of a series of components, sliding or jointed, which grasp and move objects in a repeated manner, through programmed motions. Manipulators are robots treated as tools, namely an entity that is physically situated in the world but which does not adapt to changes in the world. These types of robots were born to deal with radioactive materials, dangerous for humans. They have been developed to perform specific, limited and specialized functions in a restricted amount of time with high precision. The design process is based on the optimization of the performance of the robot executing a specific action. For this reason, artificial intelligence is often considered superfluous. Manipulators are present especially in industries in applications like welding, lifting, and automation (SCARA robot).

On the other hand, mobile robots are devices capable of locomotion and hence they are not fixed to one single position. These robots are considered agents, entities that can sense and induce changes in the world. Mobile robots were initially designed to move materials in different zones but one can find them in the military, security, medical and other fields.

The difference between these two classes relies both on the distinction between an agent and a tool, and also on the differences between automation and autonomy.

2.2 Automation vs. autonomy

Automation refers to a wide range of technologies performed automatically from the device, with the result of a reduction of human intervention in processes. Automation assumes that the operator performs any requirements before or after the automated sequence to complete the assignment. Multiple automation sequences are required to enable supplies to work semi-autonomously or autonomously. Automation has been obtained through mechanical, hydraulic, pneumatic, electrical, electronic devices, and computers. Complicated systems usually use all of them in combination. Automation provides benefits in almost all industries, from manufacturing to transportation but also utilities, defence and facility operations.
Instead, autonomy refers to a property of a system capable of performing the programmed operations, under defined conditions, without human input or guidance. This type of system is said to be autonomous and has some requirements to satisfy. It should have the capability of self-maintenance, to sense the environment and perform a physical task.
The distinction between automation and autonomy affects the style of programming, the hardware of designing and the kinds of failures.

Robot capability can also be interpreted through the spectrum of techniques representing four key aspects: plans, actions, models, and knowledge representations [1]. One can visualize a capability as a set of sliders, as shown in Fig.2.2.1. Each of the aspects associated with autonomy and automation contributes to the overall capability.

An autonomous robot does not necessarily have all the characteristics on the right side, as well as automation does not have all the attributes on the left side. Actually, most robot systems programmed with artificial intelligence methods have a mixture of both.

Plans - In automation applications, a system most of the time has to execute a previously generated plan. The ability of the robot is about executing the same actions exactly in the same way as fast as possible, with high precision and accuracy. In contrast, autonomous applications allow the robot to construct the plan and adapt the plan itself as the surroundings change. The adjustment of the plan is possible

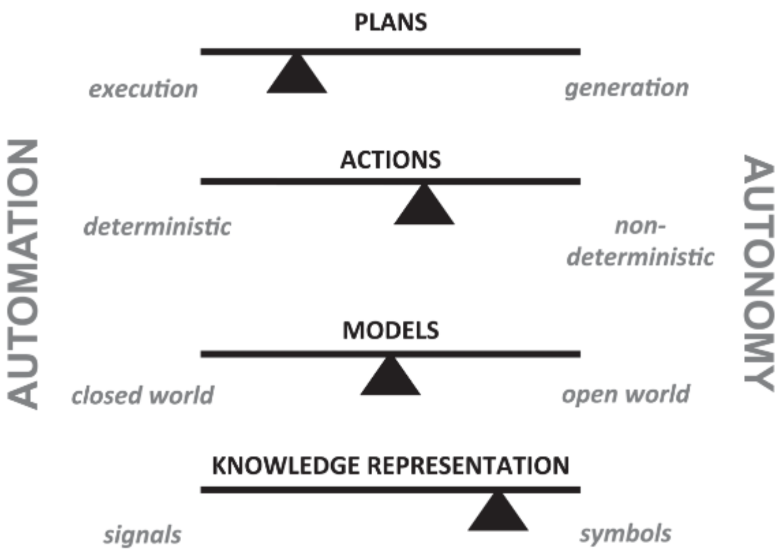


Figure 2.2.1: *How four aspects of automation and autonomy combine to create an intelligent system.*

through the perception of the surrounding world through sensors that the robot is equipped with.

Actions - Actions can be deterministic or non-deterministic. An action is said to be deterministic if, when a robot is in a given state and receives a specific set of inputs, there is only one possible output. In contrast, a non-deterministic algorithm has multiple possible outputs and the choice of one of them depends on other factors or events. Autonomous capabilities are non-deterministic and performances are measured as an average or statistical probability.

Models - A world model is an abstract representation of the spatial or temporal dimensions of our world. World modelling enables the use of robotics systems to operate in the workspace, by providing knowledge of the environment to the robot. It can be preprogrammed into a robot, may be learned by the robot or some combination of them. This is a critical component of automated robotics because the robot must sense and interpret the world using sensors, which might be not as accurate as needed.
World models are classified as being closed-world or open-world. The closed-world assumption, typical of automation, states that everything possible is known a priori; any object, condition or event that is not specified in the database is false. On the other hand, an autonomous robot operates under the open-world assumption that assumes that the list of possible states, objects or conditions cannot be completely specified.

Knowledge representations - A distinction can be made between automated and autonomous capabilities based on the form of information that the robot processes, either signals or symbols. Automation implies that the robot responds to signals or raw data, while autonomy operates in processed information or symbols.

Today, different levels and varying degrees of autonomy can be found across most industries. [2] defines autonomy as *the extent to which a robot can sense its environment, plan based on that environment, and act upon that environment with the intent of reaching some task-specific goal (either given to or created by the robot) without external control.*

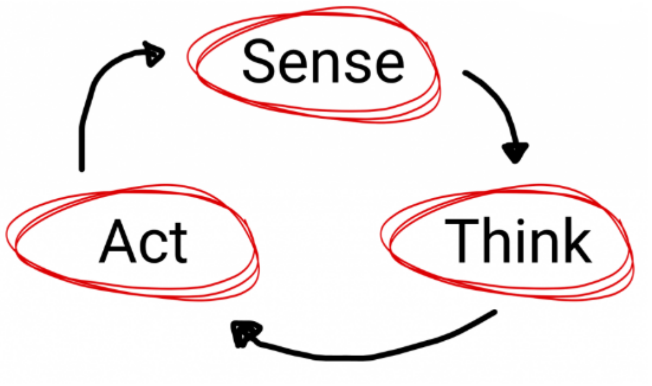


Figure 2.2.2: *Sense, plan, act for autonomy.*

There exist several levels of autonomy, reported in Table 2.2.1, that can be applied to most vehicles and robots.

2.3 Applications and motivations of robots

Robots find applications in almost any field of our lives. They can substitute humans in dangerous and unsafe environments and in boring and monotonous works. There exist several reasons to prefer a robot instead of a human. First of all, robots, as well as automation, can increase productivity, efficiency, safety, quality and consistency of products, since they are inherently fast, highly repeatable and reliable, and will perform each and every operation in exactly the same manner with a mm or even μm accuracy. Moreover, they perform tasks with precision and consistency without getting tired or bored. As regards productivity, robots can increase it exponentially since once the robot is programmed, it performs the operation always in the same amount of time, as in food preparation and manufacturing. The predictable performance combined with high levels of reliability and no need to stop makes it possible to consistently achieve and maintain productivity levels 24/7. As robots do not get injured, using them in high-risk environments can increase the safety of humans in remote, hostile or hazardous environments, for instance in the nuclear or chemical

Table 2.2.1: *The five (six) levels of autonomy in robots.*

Level	Description
0	Full manual teleoperation: simple mechanical devices operated manually.
1	Robot within line of sight (hands off): the device is able to follow predefined paths or actions but the human is required to be there for supervision. For example, a car operating under simple cruise control.
2	Operator on site or nearby (eyes off): acceleration, deceleration and steering are automated. Information from the environment is taken by sensors and used as sensory input to take decisions. The device performs autonomously some tasks but the human, being the remote supervisor, is still ultimately responsible for the safe operation.
3	One operator oversees many robots (mind off): all safety functions are automated, but the driver is still needed to take the control in emergencies.
4	Supervisor not on site (monitoring off): the device handles all decisions with no input from a human in specific scenarios. The robots are capable of finding their base stations, getting a new battery, performing minor repairs, and getting out of difficult cases. This level of autonomy needs not only the on robot software to mature, but the on-field infrastructure to automate and typically a reliable connection with remote users.
5	Robots adapt and improve execution (development off): the robot begins to learn from experience and to improve operation beyond what the human designer has programmed in. They learn from each other, on site and from robot teams from other sites. They learn to predict how events affect their capabilities and plan proactively.

industries. An example is the employment of robots in Fukushima's reactor to provide information about the damage. Moreover, underwater and space exploration would not have been possible without the use of robots. NASA developed Robonaut 2 to collect data from particular parts of the deep ocean, and Voyagers 1 and 2 were launched to study the outer solar system. But they can also be used for rescue, security and military purposes. In addition, robots do not need environmental comfort like lighting and heating, noise protection, reducing drastically costs for factories.

Other fields in which one can employ a robot are agriculture, entertainment and customer service. Robots can process multiple stimuli or tasks simultaneously, making robots faster than humans. Robots also bring benefits in applications that require high levels of cleanliness such as those found within the medical, pharmaceutical and food sectors. To reduce the risk of bacterial contamination in high-care areas, numerous robot types have been designed specifically for operation within these environments. Maintaining the high standards required for operators within these often-challenging environments requires strictly controlled operating procedures and the ongoing provision and cost of personal protection equipment.

However, robots have some disadvantages linked to costs and limited capabilities. One can encounter limited capabilities in degrees of freedom, dexterity, sensors, vision systems and real-time response. This can cause inappropriate and wrong responses, lack of decision-making power, loss of power, human injuries and damage to the robot itself or other devices.
High costs are related to maintenance, installation, and cost of equipment and programming. Robots need a lot of power to function and to buy and develop software you would invest a huge amount of money.
The last thing to remember is that if robotics comes into trend, then many workers would also lose their jobs, there would be a revolution of which worker one needs, a period of transition to substitute all those replaced workers with very specialized ones.

2.4 History of robotics

The origin of robotics comes from the need to lighten the work to man, as in the case of industrial automation, and to use means to treat hazardous materials and environments. The first use of modern robots was in factories, as industrial robots, while digitally programmed ones with artificial intelligence have been built since the 2000s.
In the following, the major milestones of robotics are presented.

The word *robotics* first appeared in Isaac Asimov's science fiction story Runaround in 1942. He also formulated the Three Laws of Robotics:

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

2.4.1 1940-1960

In 1942, during World War II, scientists and engineers created the first robotic arm, designed to handle and process radioactive materials. The telemanipulator was a sophisticated mechanical linkage which translated motions on one end of the mechanism to motions at the other end.
The first robots were constructed between 1948 and 1949. **Elmer and Elsie** (ELectroMEchanical Robot, Light-Sensitive) were two electronic robots built by William Grey Walter, often labelled as tortoises because of their shapes and the manner in which they moved. The tortoise robots had three wheels and were capable of phototaxis, namely a movement that occurs in response to light stimulus, see Fig.2.4.1a. These robots were able to manoeuvre around objects in a room, guide themselves toward a source of light and find their way back to a charging station when their battery power is low. To manage these actions they used sensor technology, a responsive feedback loop, and logical reasoning.

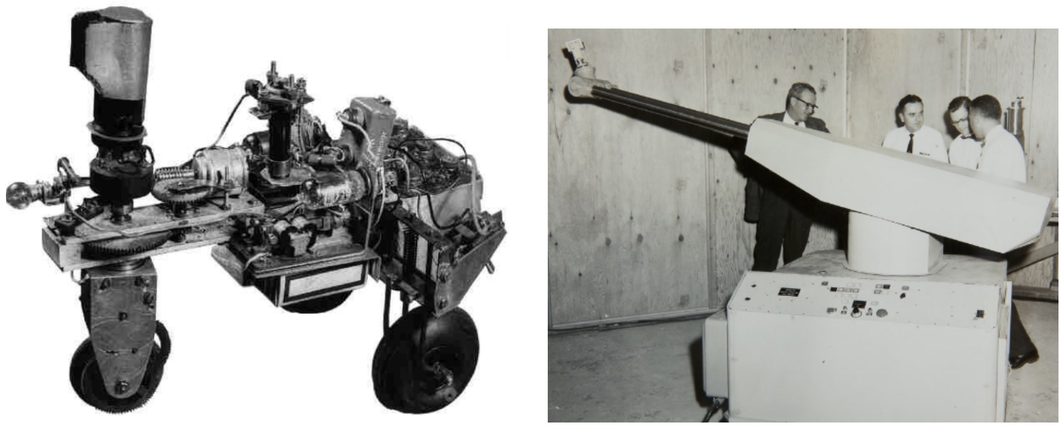
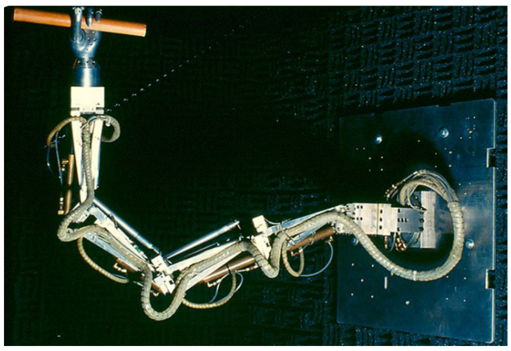

(a) Elmer and Elsie (b) Unimate

Figure 2.4.1: *Examples of robots from the period 1940-1960.*

The first stationary industrial robot was the programmable **Unimate**, invented in 1954 by the American engineer George Devol. It was a hydraulic heavy-lifting arm electronically controlled which could repeat arbitrary sequences of motions, see Fig.2.4.1b.
The introduction of transistors into computers in the mid-1950s reduced the size of the robots and increased their performance. Therefore, computing and programming could be incorporated into a range of applications, including automation. In 1954, Barrett Electronics Corporation designed the first electric vehicle that did not require a human driver, what is known to be the first AGV. In 1959, a prototype of the Unimate was introduced in a General Motors Corporation die-casting factory. The hydraulic manipulator arm possessed a rotatable, pincer-like gripper, and could follow a program of up to 200 movements stored in its memory.

2.4.2 1960-1970

More advanced computer-controlled electric arms guided by sensors were developed in the late 1960s and 1970s at the Massachusetts Institute of Technology (MIT) and Stanford University.
Marvin Minsky created the **Tentacle Arm**, visible in Fig.2.4.2a, in 1968; the arm was controlled by a computer. It was provided with 12 joints powered by hydraulics. In 1969 Victor Scheinman created the **Stanford Arm**, see Fig.2.4.2b, recognized as the first electronic computer-controlled robotic arm because the Unimate's instructions were stored on a magnetic drum.


(a) Tentacle arm

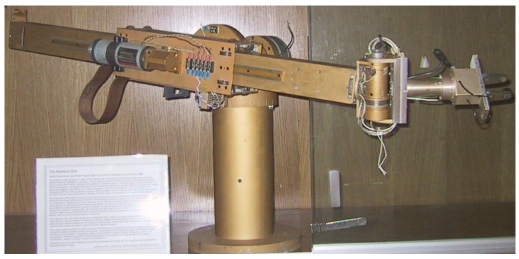

(b) Stanford arm

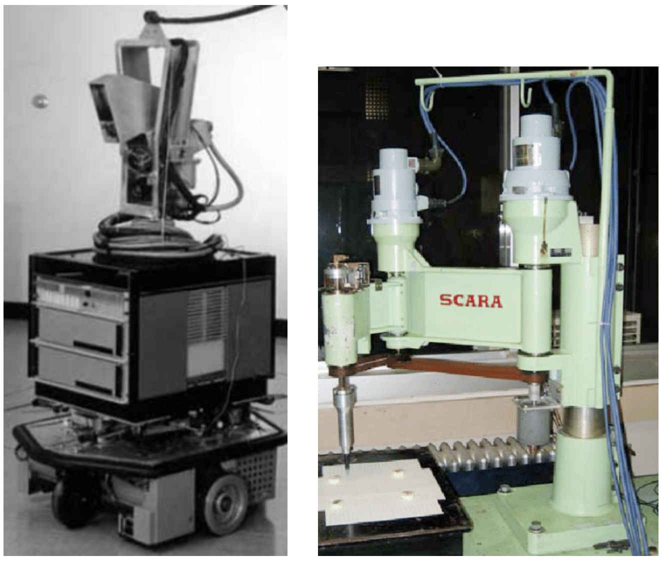
Figure 2.4.2: *Examples of robots from the period 1960-1970.*

In 1969 Victor Scheinman designed a small robot arm with joints powered by electric motors embedded in the arm itself. The arm could move much more quickly than previous robots and without complex hydraulic systems. It also had six axes of movement allowing it to closely approximate the range of a human arm. It was the first robotic arm to be controlled by software in a computer, performing calculations in real-time and, in later enhancements, reacting to its environment.

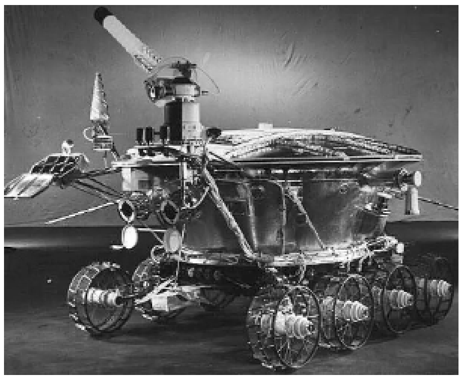
2.4.3 1970-1980

The first mobile robot capable of reasoning about its surroundings was **Shakey** in Fig.2.4.3a, built in 1970 by the Stanford Research Institute. The name derives from the stuttering way it moved around. Shakey was able to accomplish a task by observing the world around it, creating a plan, and executing it. Shakey was equipped with multiple sensor inputs, including vision cameras, tactile sensors, laser rangefinders, and bump sensors to navigate. It could move on the ground thanks to two computers (one on board and one remote) that were connected by radio. This idea of a separate planning layer was such a crucial innovation that it is still central to many robotic systems today.

In the early 1970s, precision munitions and smart weapons were developed. Weapons became robotic by implementing terminal guidance. In these years there has been a considerable advance in the development of humanoid robots by Japanese robotics scientists.
In 1978, the Japanese automation researcher Hiroshi Makino designed the efficient four-axis SCARA (Selective Compliance Assembly Robot Arm) engineered simply to pick something up, swivel around, and place it in another location with precision, all in one smooth motion. SCARA arms reported in Fig.2.4.3b are generally less flexible and not as strong as six-axis arms, but they are much faster, and able to rapidly insert small electronic components into place.
Always in the 70s, NASA developed **Mars Rover** to explore hostile or unknown terrain, see Fig.2.4.3c. It was a platform that integrated a mechanical arm, proximity sensors, a laser telemetry device and stereo cameras.



(a) Shakey **(b)** SCARA



(c) Mars Rover

Figure 2.4.3: *Examples of robots from the period 1970-1980.*

2.4.4 1980-2000

In 1986, Honda began its humanoid research and development program to create robots capable of interacting successfully with humans. Meanwhile, MIT in 1989 revealed a hexapodal robot named **Genghis** and shown in Fig.2.4.4a.
In 1994 one of the most successful robot-assisted surgery appliances was cleared by the FDA. The **Cyberknife**, visible in Fig.2.4.4b, had been invented by John R. Adler and the first system was installed at Stanford University in 1991. This radiosurgery system integrated image-guided surgery with robotic positioning. The biomimetic robot **RoboTuna** in Fig.2.4.4c was built by doctoral student David Barrett at the Massachusetts Institute of Technology in 1996 to study how fish swim in the water.

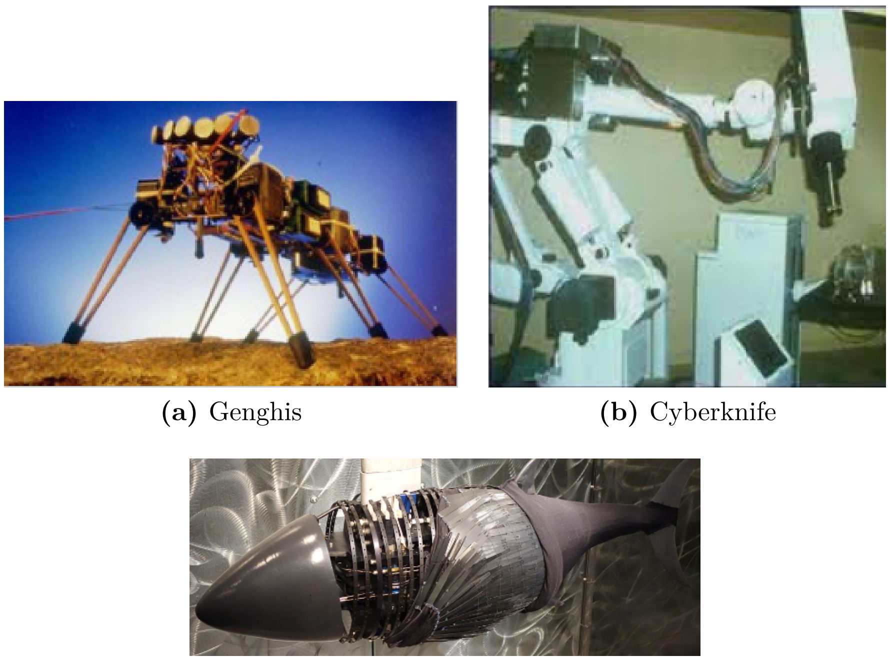
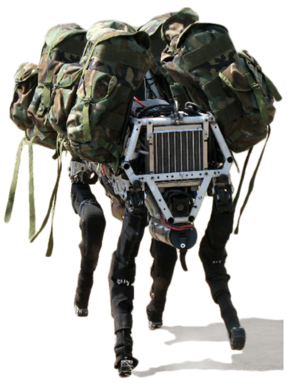

(a) Genghis **(b)** Cyberknife **(c)** RoboTuna

Figure 2.4.4: *Examples of robots from the period 1980-2000.*

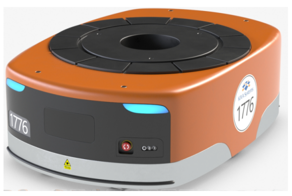
2.4.5 2000-nowadays

In April 2001, the **Canadarm2**, shown in Fig.2.4.5b, was launched into orbit and attached to the International Space Station. The Canadarm2 is a larger, more capable version of the arm used by the Space Shuttle, and is hailed as smarter.
In 2002 the company iRobot releases **Roomba**, an example is reported in Fig.2.4.5d, a robotic vacuum cleaner, still widespread today, in its updated versions.
In 2003 was created the **Kiva** robot, visible in Fig.2.4.5c. It was a squarish, close-to-the-ground orange bot that can glide around warehouses, moving racks of goods.

Kiva used some inexpensive components, which could make the robots less precise in how it moved about, compensated with software that course-corrected on the fly. The result was an autonomous machine that was far more flexible at automating a warehouse than a traditional conveyor-belt system, and relatively easy to use. Kiva's system revolutionized the efficiency of warehouse and shipping.
In 2004 Boston Dynamics ideated **BigDog**, see Fig.2.4.5a. Over the course of several years, the four-legged robot can be seen tramping through rough terrain, leafy forests, 60-degree hills, knee-deep snow and piles of bricks. It was not fully autonomous; a human controller piloted it, so it does not need a sophisticated planning and vision system. But it has 50 sensors and an onboard computer that manages the gait and keeps it stable. Most notably, BigDog was able to bounce along with only two feet touching the ground at a time, making it far more nimble than rolling robots, which are generally limited to areas that have flat surfaces, like warehouses and pavement. BigDog's mobility points to a time when everyday mass-produced robots could readily navigate front lawns, curbs or stairs, opening up new possibilities for everything from package delivery to in-home personal care.


(a) BigDog

(b) Canadarm2

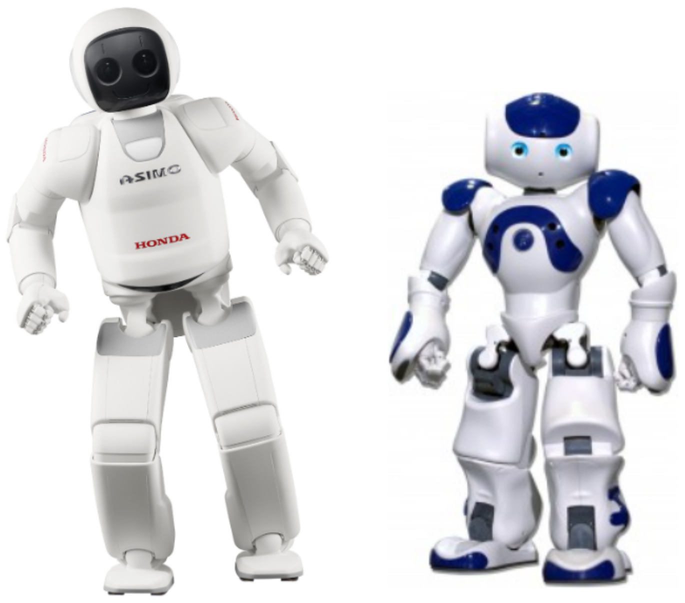

(c) Kiva

(d) Roomba

Figure 2.4.5: *Examples of robots from the period 2000-nowadays.*

The modern age of self-driving cars was launched in 2005 when a Volkswagen Touareg named **Stanley** won the second DARPA Grand Challenge. What fueled Stanley's victory was a constellation of improvements, including AI trained on the driving habits of real-world humans and five Lidar (light detection and ranging) laser sensors, a technology that enabled the car to identify objects within a 25-meter range in front of the vehicle. Lidar has since become a key component of robotic vision systems in cars and even some Kiva-style warehouse robots.

In 2000, Honda revealed the most advanced result of their humanoid project, named **ASIMO**, visible in Fig.2.4.6a. ASIMO was a mobile assistant that can run, walk, communicate with humans, recognize faces, environment, voices and posture, and interact with its environment. It was 130cm for 48kg. ASIMO was the most recent model of a series became in 1986 with model E0.
Fig.2.4.6b shows **NAO**, which was an autonomous and programmable humanoid robot of medium size designed by Aldebaran Robotics in 2004.



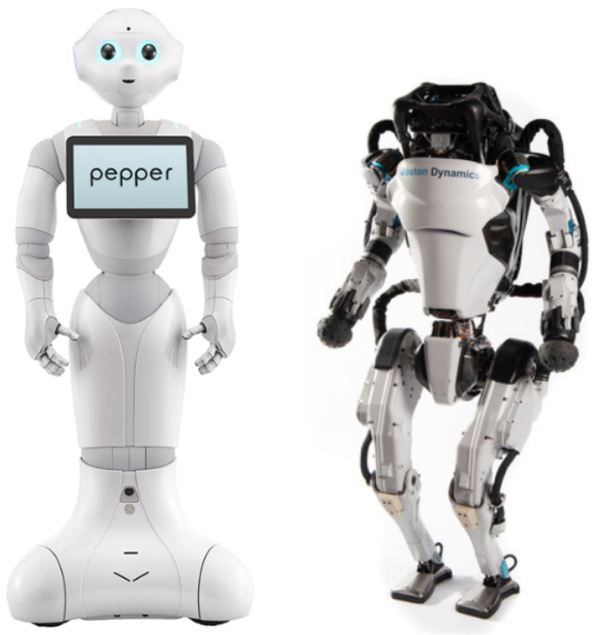
(a) ASIMO **(b)** Nao

Figure 2.4.6: *Examples of robots from the period 2000-nowadays.*

The 2010s were defined by large-scale improvements in the availability, power and versatility of commonly available robotic components, as well as the mass proliferation of robots into everyday life. The vast majority of robotic developments in the 2010s saw smaller, more specialized non-humanoid robots become cheaper, more capable, and more ubiquitous. The cost and weight reductions of these components have resulted in a proliferation of new kinds of special-purpose robots. The decade also saw a boom in the capabilities of artificial intelligence. The capacity of onboard computers used within robots increased to the point that robots could perform increasingly complex actions without human guidance, as well as independently process data in more complex ways. The 2010s also saw the growth of new software paradigms, which allowed robots and their AI systems to take advantage of this increased computing power. Neural networks became increasingly well-developed in the 2010s. The growth of robots in these years also coincided with the increasing power of the open-source software movement, with many companies offering free

access to their artificial intelligence software.
In 2014, new Tesla vehicles were fitted with the computer hardware necessary to eventually support a full autopilot software system, with increasingly autonomous software systems arriving as updates over later years. By the end of the decade, autonomous driving was possible on large highways but still required human supervision.

The first humanoid able to read emotions and react to them was **Pepper**. Pepper, in Fig.2.4.7a, was a semi-humanoid robot manufactured by SoftBank Robotics, introduced on June 2014. Pepper's ability to recognize emotion is based on the detection and analysis of gestures, facial expressions, and voice tones. Production of Pepper was paused in June 2021, due to weak demand.
Atlas, shown in Fig.2.4.7b is a bipedal humanoid robot primarily developed by the American robotics company Boston Dynamics with funding and oversight from the U.S. Defense Advanced Research Projects Agency (DARPA). The robot was initially designed for a variety of search and rescue tasks and was unveiled to the public in 2013. Atlas has become very famous for its floor routine in gymnastics, running parkour and dancing. It had the ability to perform a handstand, somersaults, and rotations all in fluid succession, backflips, jumps, balance beams, and vaults. Atlas is 1.5m tall and weighs 85kg. Atlas was battery-powered and hydraulically actuated with 20 degrees of freedom with RGB cameras and depth sensors that provide input to its control system. All the computations required for control perception and estimation happen in three onboard computers.



(a) Pepper **(b)** Atlas

Figure 2.4.7: *Examples of robots from the period 2000-nowadays.*

On October 25, 2017, at the Future Investment Summit in Riyadh, a robot called **Sophia** was granted Saudi Arabian citizenship, becoming the first robot ever to have a nationality.

In 2019, engineers at the University of Pennsylvania created millions of nanobots in just a few weeks using technology borrowed from the mature semiconductor industry. These microscopic robots, an example of which is represented in Fig.2.4.8, small enough to be injected into the human body and controlled wirelessly, could one day deliver medications and perform surgeries, revolutionizing medicine and health.

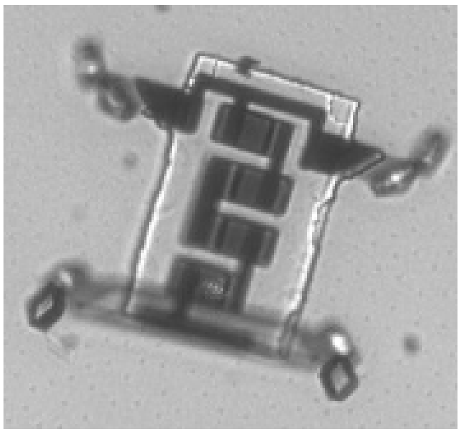


Figure 2.4.8: *Example of a nanorobot.*

3 Mobile robots

3.1 Introduction

A mobile robot is a special type of software-controlled machine that utilizes sensors and other technologies to recognize its environment and pursue its predefined assignment. This type of robot is called *mobile* because it is characterized by the capacity of locomotion, hence it can move around in the environment and it is not fixed at a single location.

Mobile robots can be classified in different manners like by the environment in which they work or by the device they use to move.
Using the latter, there are several varieties of locomotion. Different types of robots can walk, roll, jump, slide, swim, skate, and fly and most of these locomotion mechanisms have been inspired by their biological counterpart, see Sec3.2 for details.

As concerns the different operational environments, mobile robotics includes [3]:

Polar robots: devices designed to work in polar environments and to traverse icy, uneven environments.

Aerial robots, also known as unmanned aerial vehicles (UAVs) or drones: machines that perform tasks flying through the air. Initially, they were mostly used in military applications but they extended rapidly to other applications such as scientific, agricultural, commercial, recreational, policing, surveillance, product deliveries, distribution and logistics, and aerial photography.

Land or home robots, or unmanned ground vehicles (UGVs): vehicles that operate while in contact with the ground. UGVs can be used for many applications for both civilian and military use to perform a variety of dull, dirty, and dangerous activities.

Underwater robots, or autonomous underwater vehicles (AUVs): machines that explore the oceans and underwater areas that are inaccessible to humans. Different types of underwater robots have been developed, like humanoids, snakes, worms, and nanorobots.

Delivery and transportation mobile robots: devices designed to move materials and supplies around a work environment.

Mobile robots are important in several fields, from teleoperations to environmental explorations or activities dangerous for humans, or even applications in which a robot performs better and in less time. Autonomous systems can play a vital role in assisting humans in a variety of problem areas. This could potentially be in a wide range of applications like driverless cars, humanoid robots, assistive systems, domestic systems, and manipulator systems. Relevant fields are also personal assistance, military surveillance and security.
One can find mobile robots for space and ocean exploration, healthcare, distribution, rescue and search, education and research.

3.1.1 Mobile robots components

A mobile robot has mainly four components that are controllers, actuators, sensors, and power systems.

Controllers are a microprocessor, computers, or embedded microcontrollers. The robot controller is a computer system that connects to the robot in order to control its movements, often it is referred to be the brain of the robot. The control system manages, commands, directs or regulates the behavior of other devices or systems using control loops. The controller deciphers the code into instructions that the robot can use in order to complete the steps of the applications.

Actuators are motors that enable the movement of the robot. They convert the energy to mechanical form. There are many types of actuators available depending on the load involved. The term load is associated with many factors including force, torque, speed of operation, accuracy, precision and power consumption. One can find electric motors, hydraulic actuators, pneumatic actuators, shape memory metal actuators and magnetostrictive actuators.

Sensors are devices used to gather information from the environment and perceive the world, see Sec.4.5.

The components of the power system are meant to supply energy to the robots for their functioning.

3.2 Locomotion of mobile robots

The locomotion system is an important aspect of mobile robot design. The design of the locomotion system does not only rely on the medium in which the robot moves but also on other factors such as maneuvrability, controllability, terrain conditions,

efficiency, and stability.
Robot locomotion refers to the mechanism that enables a robot to move unbounded throughout its environment. There are several modes of moving and hence the selection of a robot's approach to locomotion is crucial. As mentioned before, most of the ways of moving a mobile robot are inspired by biological systems, although a perfect copy of nature is extremely difficult, or even impossible. The reasons are mechanical complexity, achieved by cell division and specialization, miniaturization, and energy storage. While in biology the mechanism of specialization permits a high complexity, in mobile robots, every single part must be fabricated individually and assembled. Moreover, very small sizes and weights, for example those of insects, are very difficult to achieve during the production part. Finally, a mechanically designed object has in rare cases the same capacity as biological systems.

In locomotion, the environment is fixed and the robot moves by imparting forces to the environment. Movements are possible thanks to the study of actuators that generate interaction forces, and mechanisms that implement desired kinematic and dynamic properties.

In locomotion, the three main issues that one has to consider are stability, characteristics of contact and type of environment.

Stability

Stability is the ability of not to fall. Stability depends on the number and geometry of contact points, the position of the centre of gravity, and the inclination of the terrain. Stability can be classified as static and dynamic stability.
Static stability means that the robot is stable at every moment of time with no need of motion. It is achieved through the mechanical design of the robot and it is maintained as long as the centre of gravity is inside the polygon of support of the robot and the polygon's area is greater than zero. The polygon of support is the convex hull formed by the projection of its points of contact onto the surface. Static stability requires at least three points of contact with the ground.
Dynamic stability is the ability of being stable while moving. It is achieved through control action, the robot must actively balance itself to prevent overturning.

Characteristic of contact

The characteristics of ground contact depend on the size of the contact point/patch, the angle of contact to the ground and the friction between the robot and the surface. For example, a large patch allows to generate larger lateral force, allowing to maintain stability in curves at high velocities. The contact patch depends on the diameter of the wheel and on the material of the tyre. In the case of inflatable wheels, depending on the internal pressure, the contact patch's size changes and hence the rolling resistance.

Type of environment

The attributes of the type of environment are the structure of the medium and the medium itself.

The different types of locomotion are:

Walking: is based on legs, whose number depends on the specific robot. One can find robots with 1, 2, 4, 6...legs. Legged motion allows to travel through uneven surfaces, and steps and overcome easily obstacles. Multiple legs allow several different gaits, even if a leg is damaged, making their movements more useful in robots transporting objects.

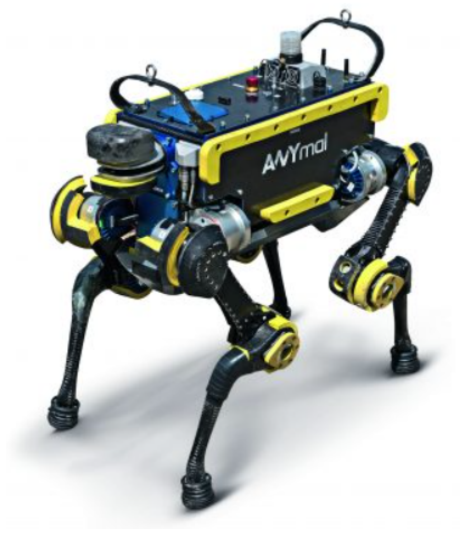


Figure 3.2.1: *Example of a walking robot.*

Swimming: refers to the way underwater vehicles move.

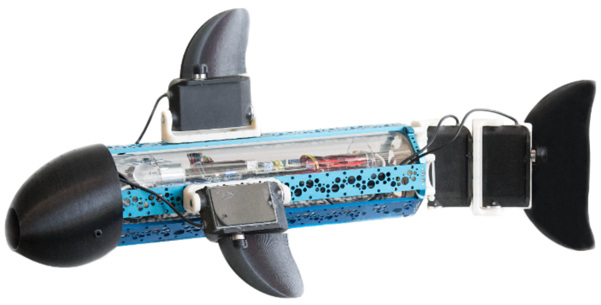


Figure 3.2.2: *Example of a swimming robot.*

Rolling: can be achieved with the usage of several types of wheels.

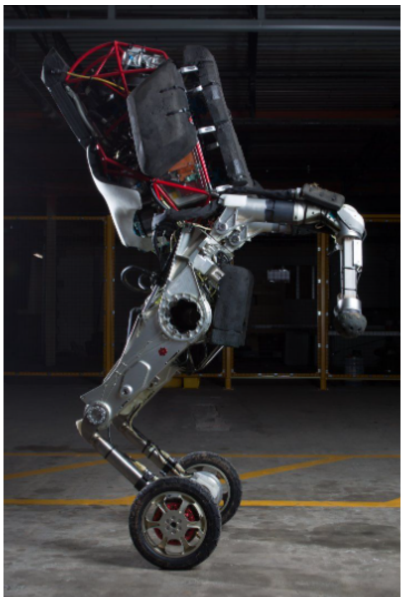


Figure 3.2.3: *Example of a rolling robot.*

Sliding: mimics the way of moving of snakes. They could be used in confined spaces, such as collapsed buildings.

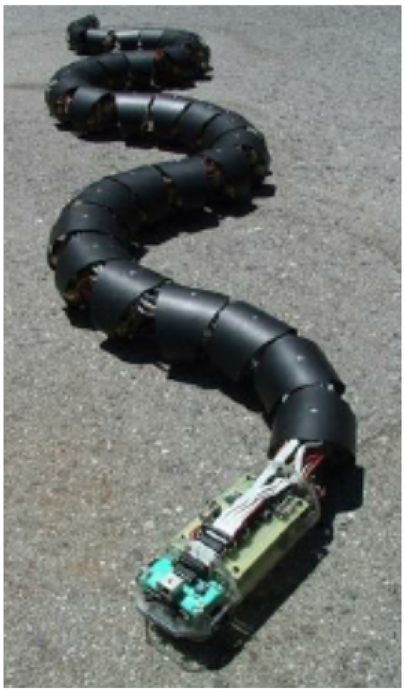


Figure 3.2.4: *Example of a sliding robot.*

Hopping: is less used. It consists of a robot provided with legs and small feet. The movement is achieved by jumping in the direction in which the robot is falling.

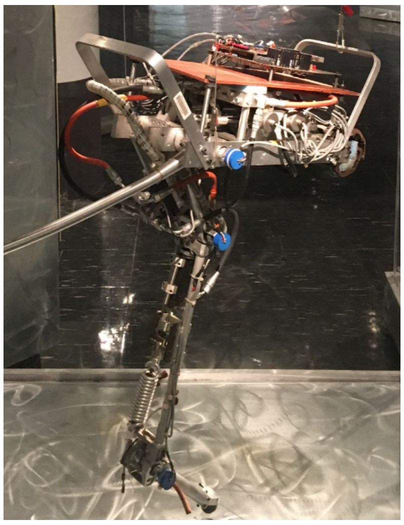


Figure 3.2.5: *Example of a hopping robot.*

Climbing: is achieved by moving over vertical or steeply inclined surfaces. It faces many challenges, including overcoming gravity, maintaining balance and negotiating obstacles.

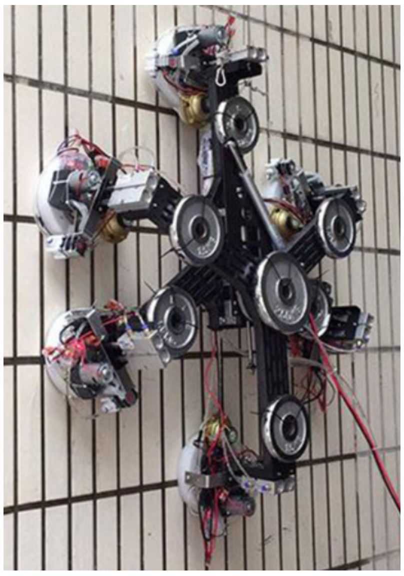


Figure 3.2.6: *Example of a climbing robot.*

Due to the several difficulties exposed before, most mobile robots are legged or wheeled. In the following, these two are analyzed more in detail.

3.2.1 Legged locomotion

Legged locomotion is characterized by a series of contact points between the robot and the ground. The key advantages include adaptability and maneuverability in rough terrain, and it is able to climb steps, and cross gaps of relative size. A final advantage of legged locomotion is the potential to manipulate objects in the environment.
One example is the StarlETH designed by the Robotic Systems Lab of ETH reported in Fig3.2.7.

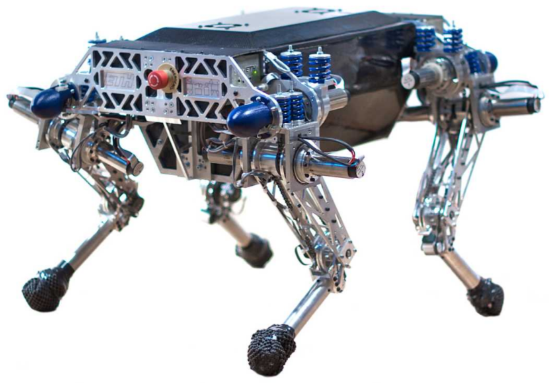


Figure 3.2.7: *StarlETH of the Robotic Systems Lab of ETH.*

The main disadvantages include power consumption and mechanical complexity. Additionally, high maneuverability will only be achieved if the legs have a sufficient number of degrees of freedom to impart forces in several different directions. To move a legged robot, each leg must have at least 2 DOF, each one requires a joint, usually powered by one servo. Lastly, the leg must be capable of sustaining part of the robot's weight and usually lifting or lowering the robot.

Regarding stability, to achieve static one a robot must have at least three legs. To be dynamically stable a robot can have even one leg. To achieve static walking you need six legs. In general, adding degrees of freedom increases maneuverability but requires energy, mass and control.

3.2.2 Wheeled locomotion

Wheeled locomotion is the most popular mechanism in mobile robotics and vehicles in general. It can achieve very good efficiencies with a relatively simple mechanical implementation. For confrontation, on hard, flat surfaces wheeled locomotion is two orders more efficient than legged locomotion. Moreover, these systems are designed to have all wheels in ground contact at all times. A two-wheeled mobile robot is considered stable, as the centre of mass is below the wheel axle however, it requires

impractically large diameters. Most of the time at least three wheels are used to assure static stability.
The focus of research in wheeled robotics is on traction and stability in rough terrain, maneuverability, and control.
Maneuverability is the combination of the mobility available based on the sliding constraint plus additional freedom given by the steering. When a robot is able to move in any direction of the ground plane it is omnidirectional. Clearly, swedish and spherical wheels have a higher maneuverability as compared to the other two classes.
Controllability is the ability of a mobile robot to be driven between positions by manipulating the velocity control inputs. There is generally an inverse correlation between controllability and maneuverability.

Wheel design

There are four major wheel types, shown in Fig.3.2.8 [4]: standard wheel, castor wheel, swedish wheel and spherical wheel. These classes differ in their kinematics and therefore the choice of the wheel type has a large effect on the overall kinematics of the mobile robot.

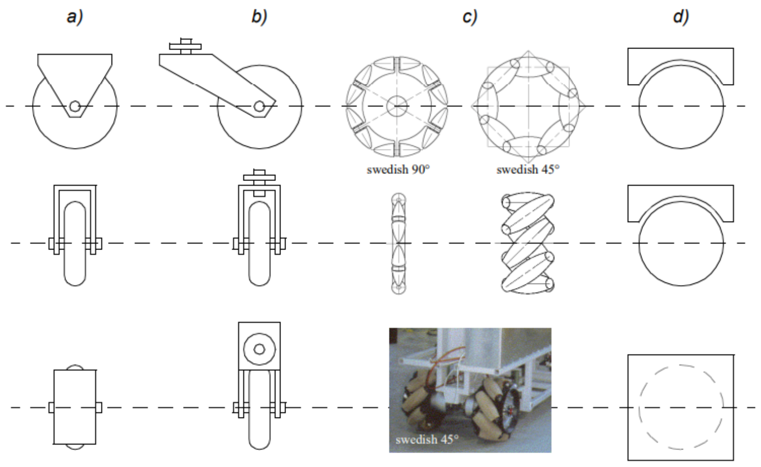


Figure 3.2.8: *The four different classes of wheel design [4].*

Both standard and castor wheels have two degrees of freedom, one around the wheel axle and one around the contact point, for the former, while around the offset steering joint for the latter. Both of them have a primary axis of rotation and are thus highly directional, to move in a different direction, the wheel must be steered first along a vertical axis. The key difference between these two wheels is that the standard wheel can accomplish this steering motion with no side effects, as the centre of rotation passes through the contact point with the ground, while the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering.

The swedish wheel can be of 45° or 90° or omni wheel direction that represents the second direction of low resistance. It has three degrees of freedom, one around the wheel axle, one around the contact point and the last one around the rollers. The third degree of freedom is less constrained by directionality, as well as the spherical wheel, which is an omnidirectional wheel. The small rollers attached to the swedish wheel around the circumference of the wheel are passive and the wheel's primary axis serves as the only actively powered joint. The key advantage of this design is that, although the wheel rotation is powered only along the one principal axis (through the axle), the wheel can kinematically move with very little friction along many possible trajectories, not just forward and backward.

3.3 Unmanned ground vehicles

In this section will be described the kinematics of an unmanned ground vehicle with particular attention on differential drive robots (DDR).
Kinematics describes the motion of points, bodies, and systems of bodies without considering the forces that cause them to move. Kinematics is a significant field of mechanics, necessary to understand the behavior of the robot both in order to design appropriate mobile robots for tasks and to understand how to create control software for an instance of mobile robot hardware.
Kinematics aims to provide a description of the spatial position of bodies or systems of material particles, the rate at which the particles are moving (velocity), and the rate at which their velocity is changing (acceleration).

By understanding the concepts involved in kinematics, one can predict objects' motion. In fact, a kinematics problem starts with the description of the geometry of the involved system and the statement of the initial conditions of any known position, velocity and acceleration of points belonging to the system. Then, using geometry and the laws of motion, the position, velocity and acceleration of any parts of the system can be calculated. Notice that there exists no direct way of measuring the robot's position, for example measuring the travelled distance of each wheel, instead, the position must be integrated over time, leading to inaccuracies of the motion estimate.

In any system, there are two types of constraints that impose limitations on the robot's movements. The first is a geometrical constraint, which imposes restrictions on the achievable configurations of the robot. The range of possible poses that the mobile robot can achieve in its environment is called the workspace. The second is a kinematic constraint, which imposes restrictions on the achievable velocities of the robot.

A robot as a single entity moves as a function of its geometry and individual wheel behavior. Each individual wheel contributes to the robot's motion and, at the same time, imposes constraints on robot motion. Wheels are assembled on robot chassis, and therefore their constraints combine to form constraints on the overall motion of the system.

Geometrical constraints are functions of positional variables and are said holonomic (e.g., limiting the system's motion to a manifold of the configuration space, depending on the initial conditions).

A kinematic constraint can be integrable, and hence it can be expressed in the form:

$$f(\xi, t) = 0, \tag{3.3.1}$$

where ξ is a vector of configuration variables.

Alternatively, they can be not integrable. In this case, they are said non-holonomic, and they are expressed through derivatives of positional variables. It is important to underline that this type of constraint does not limit the accessible configurations but only the path that can be followed to reach them.

As concerns wheels, several hypotheses are assumed in order to simplify the model. In the following they are listed:

- Movements concern only the horizontal plane
- Vertical wheel plane and point of contact of the wheels
- Wheels are not deformable
- Pure rolling
- No slipping, skidding or sliding
- Absence of friction for rotation around the contact point
- Steering axes orthogonal to the surface in which the robot lies
- Wheels connected by a rigid frame

Before explaining more in detail the sliding constraint and the pure rolling one, it is important to underline the difference between the robot reference frame ($\mathcal{F}_R$) and the world reference frame ($\mathcal{F}_I$) and how to transform variables between the two, see Fig.3.3.1.

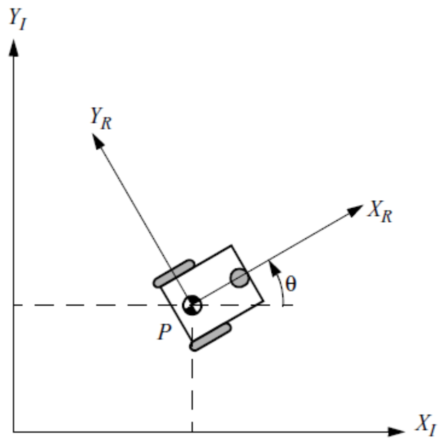


Figure 3.3.1: *Robot reference frames [4].*

The axes X_I and Y_I define an arbitrary inertial basis on the plane as the global reference frame from some origin. The basis X_R, Y_R defines the robot's local reference frame. The position of the robot is specified by choosing a point P on the robot chassis, representing the origin of $\mathcal{F}_R$, as its position reference point, usually its centre of mass.
The position in the global reference frame is expressed in x and y coordinates, while the relative orientation between the two frames is given by θ.

To describe robot motion in the plane, it is essential to map motion along the axes of the global reference frame to motion along the axes of the robot's local reference frame, through the orthogonal rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3.2}$$

The matrix $R(\theta)$ is used to relate $\mathcal{F}_R$ and $\mathcal{F}_I$, in particular the relations are:

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I \tag{3.3.3}$$
$$\dot{\xi}_I = R(\theta)^T\dot{\xi}_R \tag{3.3.4}$$

3.3.1 Sliding and rolling constraints

The rolling constraint also called pure rolling at the contact point, imposes zero translational velocity at the contact point, point O in Fig.3.3.2. This means that all motion along the direction of the wheel plane is determined by wheels spin.

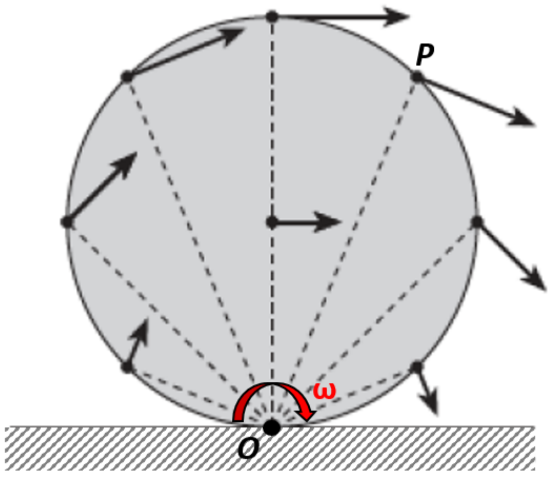


Figure 3.3.2: *Representation of pure rolling constraint [5].*

The other constraint, called sliding constraint, is the impossibility of slipping along the orthogonal to the sagital axis. In other words, the component of the wheel's motion orthogonal to the wheel plane must be zero.

The sliding constraint and the rolling constraint have different formulations for each wheel type, as reported below. Anyway, notice that only the fixed and steerable standard wheels impose limitations on the motion since $\dot{\xi}_I$ in the other types can range freely, due to internal degrees of freedom of the wheels. Therefore, only standard wheels have an impact on robot chassis kinematics.

Fixed standard wheel

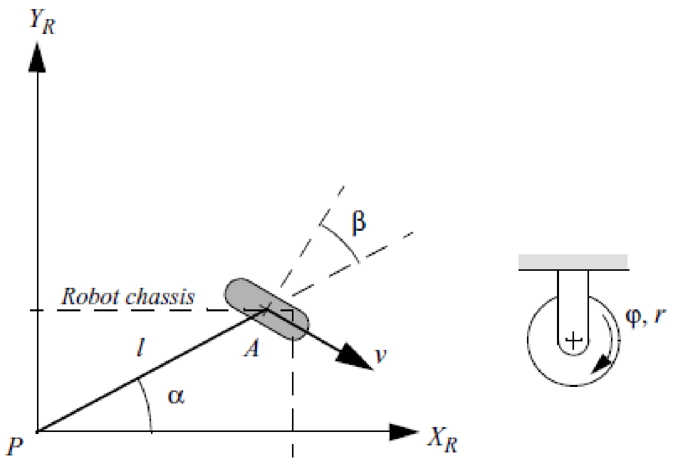


Figure 3.3.3: *Fixed standard wheel and parameters [4].*

The position of A is expressed in polar coordinates by distance l and angle α, see Fig.3.3.3. The angle of the wheel plane relative to the chassis is denoted by β, which is fixed.
The wheel has radius r and the rotational position around its horizontal axle $\varphi(t)$ is a function of time.

Rolling constraint:

$$\Big[\sin(\alpha+\beta) - \cos(\alpha+\beta)(-l)\cos\beta\Big] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.3.5}$$

Sliding constraint:

$$\Big[\sin(\alpha+\beta)\cos(\alpha+\beta) l \sin\beta\Big] R(\theta)\dot{\xi}_I = 0 \tag{3.3.6}$$

Steered standard wheel

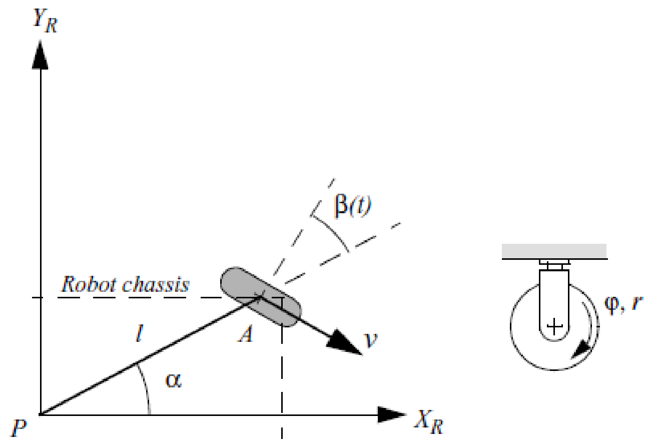


Figure 3.3.4: *Steerable standard wheel and parameters [4].*

The parameters of see Fig.3.3.4 are the same as Fig.3.3.3, except for the fact that the orientation of the wheel to the robot chassis can change over time, $\beta(t)$.

Rolling constraint:

$$\Big[\sin(\alpha+\beta) - \cos(\alpha+\beta)(-l)\cos\beta\Big] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.3.7}$$

Sliding constraint:

$$\Big[\sin(\alpha+\beta)\cos(\alpha+\beta) l \sin\beta\Big] R(\theta)\dot{\xi}_I = 0 \tag{3.3.8}$$

Castor wheel

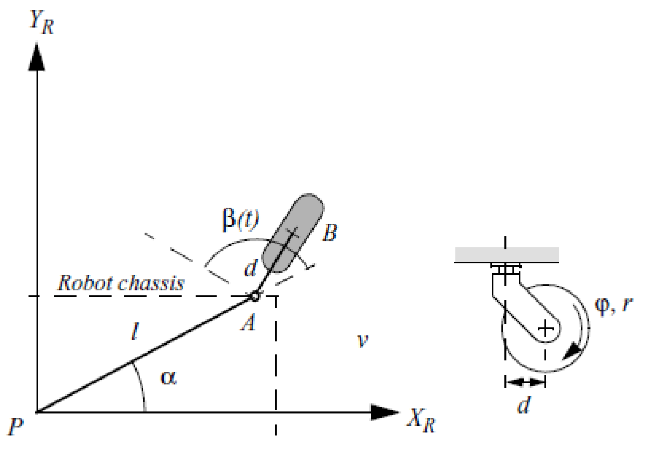


Figure 3.3.5: *Castor wheel and parameters [4].*

The wheel contact point id B, connected by a rigid body of fixed length d to A, see Fig.3.3.5. There are two parameters that vary over time: $\varphi(t)$, which represents the wheel spin over time, and $\beta(t)$, which denotes the steering angle and orientation of the rigid body AB.

Rolling constraint:

$$\left[\sin(\alpha+\beta) - \cos(\alpha+\beta)(-l)\cos\beta\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.3.9}$$

Sliding constraint:

$$\left[\sin(\alpha+\beta)\cos(\alpha+\beta)d + l\sin\beta\right] R(\theta)\dot{\xi}_I + d\dot{\beta} = 0 \tag{3.3.10}$$

Swedish wheel

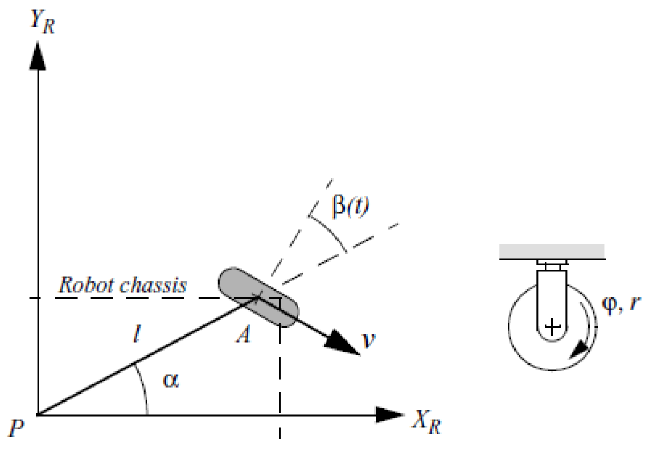


Figure 3.3.6: *Swedish wheel and parameters [4].*

The parameters are the same as Fig.3.3.3, with the addition of the variable γ, which is the angle between the main wheel plane and the axis of rotation of the rollers, see see Fig.3.3.6.

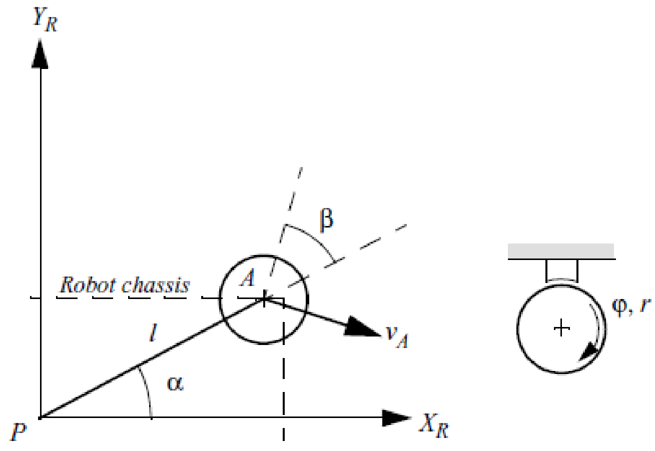
Rolling constraint:

$$\left[\sin(\alpha+\beta+\gamma) - \cos(\alpha+\beta+\gamma)(-l)\cos(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi}\cos\gamma = 0 \tag{3.3.11}$$

Sliding constraint:

$$\left[\sin(\alpha+\beta+\gamma)\cos(\alpha+\beta+\gamma) l \sin(\beta+\gamma)\right] R(\theta)\dot{\xi}_I - r\varphi\sin\gamma - r_{sw}\varphi_{sw} = 0 \quad (3.3.12)$$

Spherical wheel



In Fig.3.3.7, the difference consists of the absence of direct constraints on motion, and there are no principal axis of rotation.

Figure 3.3.7: *Spherical wheel and parameters [4].*

Rolling constraint:

$$\left[\sin(\alpha+\beta) - \cos(\alpha+\beta)(-l)\cos\beta\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \quad (3.3.13)$$

Sliding constraint:

$$\left[\sin(\alpha+\beta)\cos(\alpha+\beta) l \sin\beta\right] R(\theta)\dot{\xi}_I = 0 \quad (3.3.14)$$

The total dimensionality of the robot's chassis on the plane is three, two for the position and one for orientation along the vertical axis, which is orthogonal to the plane. Additional degrees of freedom and flexibility, internal to the robot and wheels, are ignored since not relevant to the study of kinematics.

3.3.2 DDR

A differential drive robot can be modelled as a unicycle, which is a planar vehicle with two orientable wheels, controlled in driving and steering velocity, $v, \omega \in \mathbb{R}$.
Generally, a DDR is characterized by the presence of two standard wheels, connected through a chassis, and a spherical wheel in the front for stability purposes. The latter allows to define a third contact point on the plane ensuring static stability. Fixed standard wheels imply that the Instantaneous Centre of Rotation (ICR) of the robot

lies along the extension of the rotation axis of the two wheels. If the wheels have the same velocity then the ICR is placed at infinite. Instead, if the velocities differ the robot follows a circular path, as shown in Fig.3.3.8.

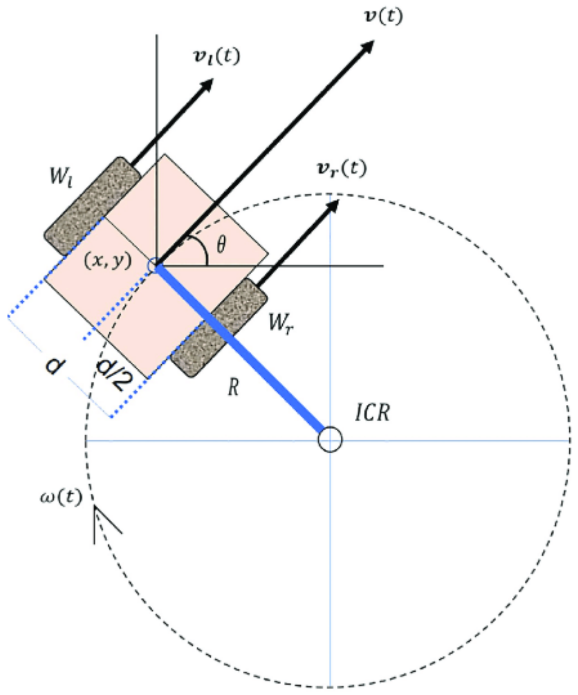


Figure 3.3.8: *Scheme of involved variables.*

The state is described by the triplet $\boldsymbol{q} = [x \quad y \quad \theta]^T$ and the kinematic model results:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \sin\theta & 0 \\ \cos\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.3.15}$$

In a differential drive model, the driving velocity and the steering velocity are obtained from the angular speed ω_R and ω_L of its wheels:

$$v = \frac{r(\omega_R + \omega_L)}{2} \tag{3.3.16}$$

$$\omega = \frac{r(\omega_R - \omega_L)}{d} \tag{3.3.17}$$

where r is the radius of the wheels and d is the distance between their centres.

4 Navigation of autonomous mobile robots

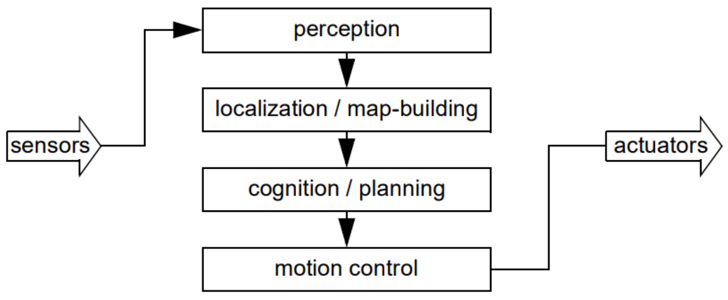
4.1 Introduction

An autonomous mobile robot has the ability to move in the environment, perform different tasks, adapt to changing environments, to learn, plan and act accordingly. Moreover, it is able to build an internal representation of the world that can be used for reasoning processes like navigation.
The objective of an autonomous mobile robot is to move from one place to another in a known or unknown environment, based on sensors. Autonomous mobile robots are equipped with sensors which are considered the senses of a robot: they allow a robot to see, feel and touch the surrounding. The combination of sensors and reasoning permits an autonomous exploration of the world and the construction of a map. If they detect an unexpected obstacle while maneuvering inside an environment they will apply a navigation technique, such as collisions avoidance to stop, slow, or divert their path around the object and then continue with their predefined mission.

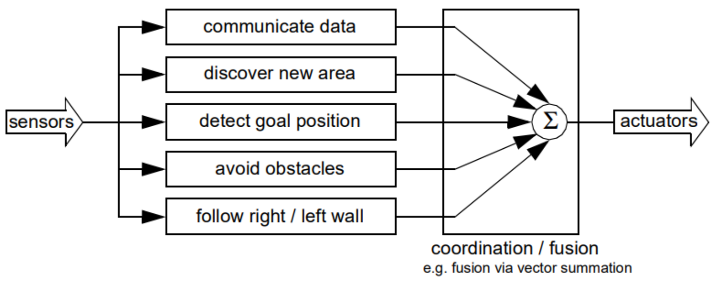
Navigation is defined as the ability to determine your location within an environment and to be able to create a path that will take you to a goal. Whereas, autonomous navigation is the capability of a vehicle to learn and execute movements without any human intervention to reach that predefined goal. An accurate collision-free path planning is mandatory for the motion of mobile robots to perform tasks autonomously with greater accuracy.

Mobile robot navigation systems depend on the level of abstraction of the environment representation. One can differentiate two categories of navigation approaches: map-based navigation and behavior-based navigation [4].
The two main navigation problems are different for distances, scales, obstacle avoidance techniques and the knowledge of the goal position.

To perform successful map-based navigation, the robot passes through different stages such as perception, localization, cognition and motion control, described later on in this chapter. Using this approach, the robot explicitly attempts to localize itself with respect to a map of the environment, by collecting sensor data and updating the belief about its position. The three mentioned processes may rely on two distinct sources of information. The first one is the information concerning the



(a) Map-based navigation architecture



(b) Behavior-based navigation architecture

Figure 4.1.1: *Navigation approaches.*

internal state of the robot, like speed and acceleration. The integration of these measurements results in a position estimate of robot motion. The second source of information provides external details about the environment, that may be used to directly recognize a place or may be combined with the other one to correct position estimate error.
Due to the explicit use of the map-based location concept, model-based navigation systems are suitable for any simple or complex environment. Moreover, the available information about its location can be exploited by a human operator to command the robot. However, model-based navigation systems depend on the internally stored map and historical information of the environment. If the model of the environment diverges from reality, then the robot will get lost due to being incapable of estimating its own position and failing to accomplish its navigation mission. The more accurate the map is and the more information is collected by the sensors, the more there is a chance that the robot will operate constantly without fail. However, higher costs for implementing the map and sensors are required.

Whereas, behavior-based navigation approaches are based on a layered set of task-achieving modules implementing a specific behavior, using sensors to observe and extract relevant information from the surroundings, which is completely unknown. Thus, each module is designed to solve only a portion of the navigation problem, and a set of modules can cooperate to mimic more complex behaviors. Due to this network structure, behavior-based navigation does not require the localization

process. Furthermore, behavior-based navigation systems are based on a belief that sensors and actuators are noisy and information-limited, so they avoid creating a global map of the environment.
Behavior-based navigation systems may be implemented very quickly for a single environment with a small number of goal positions and they have good flexibility because they do not need precise localization. However, the difficulty is higher since the absence of a map does not permit the calculation of an optimal path from the starting point to the target goal. Moreover, it can be time-consuming and heavily dependent on specific hardware and environmental characteristics. Finally, the method does not directly scale to other environments or to larger ones.

To achieve autonomous navigation and intelligent behaviour in robots, it is necessary to solve specific problems. Among all, it is essential that the robot can establish its actual and previous locations, and consequently determine the unexplored places and the possible paths to reach them. In order to accomplish these tasks, autonomous mobile robots must perform planning, localization, and mapping consecutively to operate successfully in the environment [6]. If any of these three activities is absent, then a robot can not safely and autonomously walk in real-life deployment scenarios. Below these three activities will be briefly presented.

Planning: is the most important aspect of an autonomous robot. During path planning, the robot is required to design a path to follow by employing an obstacle avoidance algorithm. The route is computed almost in real-time and continuously updated, taking into consideration the kinematics of the vehicle.

Localization: is the procedure of estimating the position of the robot in space. This stage requires the use of sensors to acquire information and the interpretation of the latter.

Mapping: is the process of designing a map of the environment. It may consist of modifying the already present map or even the creation of it from scratch, as the robot perceives information about the surroundings.

Usually, autonomous robots use a technique called simultaneous localization and mapping (SLAM) to start in an unknown location in an unknown environment, and then incrementally build up a map while using it to compute the current position. So, the sensors of the robot should be accurate enough to create a precise map.

There are several approaches to solve navigation problems, each of them related to a specific situation and they can be divided into two main types, depending on the environment in which the robot operates: indoor and outdoor navigation [7].

4.2 Indoor navigation

Indoor navigation is meant mainly for buildings, and hence it enables the exploration of indoor environments. The most common applications are in cleaning services and rescue activities but also in warehouses to move and classify items.
One of the most famous indoor robots in the cleaning industry is *Roomba*, produced and sold by the company IRobot. These robots are capable of cleaning a house in complete autonomy, thanks to the use of advanced sensors, which enables them to detect obstacles. Moreover, most of the time they are provided with a technology that permits the robot to construct a detailed map of the house, allowing a more efficient way of cleaning. As the robot explores the house, it stores in memory obstacles, doors, walls and whatever allows the navigation algorithm to determine their position. The evolution of the last years has made these robots very intelligent; actually, updated models are able to sense particles in the air and report which parts of the house are dirtier.

4.3 Outdoor navigation

Like indoor navigation, outdoor navigation requires the use of landmarks, obstacle avoidance and position estimate. Differently, a complete a priori map of the environment is impossible to achieve, due to the dynamic environment. It can be divided into outdoor navigation in structured environments and unstructured environments.

Structured environments present models that are simple, and contain roads, lane widths, etc. The navigation relies mostly on obstacle avoidance and the ability of the robot consists of road following. Road-following is the ability to recognize the lines that separate the lanes or separate the road from the berm, the texture of the road surface, and the adjoining surfaces. Road-following is characterized by problems caused by shadows, changing illumination conditions, and changing colors.

Unstructured environments are characterized by the absence of regular properties that could be perceived and tracked. In such cases, the vision system can make use of at most a generic characterization of the possible obstacles in the environment.

4.4 Obstacle avoidance

Obstacle avoidance is a crucial task in the field of robotics. Obstacle avoidance is defined to be *the ability to avoid collisions with objects while navigating in an environment*. The design of a collision-free algorithm is a prerequisite for any autonomous mobile robot since it guarantees a safe trajectory to pursue the goal.

Obstacle avoidance is highly related to path planning, studied in Sec.4.7 but, during navigation, the robot can deviate from its path to avoid obstacles on the basis of reactive navigation strategies. This is possible thanks to the use of sensors and specific algorithms to elaborate information and give instructions to the robot.

4.5 Perception

During the perception phase, the robot constantly extracts meaningful information from the environment, ensuring accurate localization by interpreting data acquired by sensors. The use of sensors makes it possible to perform robot positioning and localization tasks, but they are also used for mapping, representation, and other robotic applications, such as object recognition.
The main challenges during the perception stage are seeing, feeling, and understanding the environment, the uncertainty present in the measurements and the partially available information.

The capability of understanding the world is based on *sensing*. Robotic sensing gives robots the ability to see, touch, hear and move and employs algorithms that require environmental feedback or sensory data. In other words, sensing is the combination of algorithms and sensors that produces a percept.
In order to autonomously navigate, a robot needs data about the environment, coming from sensors like cameras, lidar and ultrasonic sensors, that must be analyzed to produce a resulting action.

4.5.1 Sensor classification and performance

Sensors are classified based on two different scales: proprioceptive/exteroceptive and passive/active.

Proprioceptive sensors measure the internal state of the robot, such as motor speed, wheel loads, joint angles, and battery voltage, hence they are used for the robot's self-control. Whereas, exteroceptive ones acquire information from the robot's surroundings, such as distances, light intensity, and sound amplitude. They are exploited to extract meaningful data from the environment and are utilised for navigation and object recognition.

Passive sensors, such as microphones, temperature probes, and Charge Coupled Devices (CCD) or Complementary Metal Oxide Semiconductor (CMOS) cameras, measure ambient environmental energy entering the sensor. Active sensors instead radiate energy into the surroundings measuring the reaction. They can deal with more controlled interactions with the work environment, often achieving higher performance. However, active sensing introduces several risks: the outbound energy may affect the very characteristics that the sensor is attempting to measure. Furthermore, an active sensor may suffer from interference between its signal and those beyond its control. For example, signals emitted by other nearby robots, or similar sensors on the same robot, may influence the resulting measurements. Examples of active sensors include wheel quadrature encoders, ultrasonic sensors, and laser rangefinders.

It is important to characterize sensor's performance using basic variables such as the dynamic range, power, resolution, linearity, bandwidth or frequency, sensitivity, errors, accuracy and precision. In the following, these parameters are explained.

Dynamic range is the ratio of the maximum input value to the minimum measurable input value. It is used to measure the spread between the lower and upper limits of input values to the sensor while maintaining normal sensor operation. Robot sensors often operate in environments where they are frequently exposed to input values beyond their working range. In such cases, it is critical to understand how the sensor will respond.

Resolution is the minimum difference between two values that can be detected by a sensor. Usually, the lower limit of the dynamic range of a sensor is equal to its resolution.

Linearity is an important measure governing the behavior of the sensor's output signal as the input signal varies.

Bandwidth or frequency is used to measure the speed with which a sensor can provide a stream of readings. Formally, the number of measurements per second is defined as the sensor's frequency in Hertz [Hz]. Due to the movement through their environment, mobile robots are often limited in maximum speed by the bandwidth of their obstacle detection sensors.

Errors are divided into random errors and systematic errors. Random errors are caused by variability in measurements due to fluctuations in the environment or the instrument used for interpretation and they affect measurements in unpredictable ways. While systematic errors are consistent or proportional differences between the observed and true values. Measurements of the same thing will vary in predictable ways.

Accuracy describes how close a given set of measurements are to their true value and represents the description of systematic errors.

Precision is how measurements are close to each other and represents the description of random errors.

Sensitivity is the ratio of output change to input change. It measures the degree to which an incremental change in the target input signal changes the output signal. Cross-sensitivity is the technical term for sensitivity to environmental parameters that are orthogonal to the target parameters for the sensor.

Repeatability is the capability of reproducing as output similar measured values over consecutive measurements of the same constant input quantity.

Stability is the capability of keeping the same measuring characteristics over time and/or temperature.

4.5.2 Wheel/motor sensors

Wheel/motor sensors are devices used to measure the internal state and dynamics of a mobile robot. They measure the wheel's speed and position. These sensors are brush encoders, potentiometers, synchros, resolvers, optical encoders, magnetic encoders, inductive encoders and capacitive encoders.

4.5.3 Tactile sensors

Tactile sensors are divided into two main categories based on their functioning. The first one contains devices like bumpers and contact switches that measure forces in response to the physical interaction with the environment, allowing the robot to control the position and the grasping force of the end-effector. The sensors belonging to the second group (optical barriers and non-contact proximity sensors) are designed to sense objects at a small distance in order to detect closeness without direct contact, reporting the exact position of an object. Moreover, tactile sensors can also reveal variations in heat.

4.5.4 Heading sensors

Heading sensors are employed to determine the robot's orientation and inclination. They allow us, together with appropriate velocity information, to integrate the movement into a position estimate. They include compasses, gyroscopes and inclinometers.

4.5.5 Ground-based beacons

Beacons are exploited to solve the problem of localization in mobile robotics. Using the interaction of onboard sensors and the environmental beacons, the robot can identify precisely its position. They include GPS, active optical or RF beacons, active ultrasonic beacons and reflective beacons.

4.5.6 Active ranging

Ranging sensors provide direct measurements of the distance from the robot to objects in its vicinity. For obstacle detection and avoidance, most mobile robots rely heavily on active-ranging sensors. They include reflectivity sensors, ultrasonic sensors, laser rangefinders, optical triangulation and structured light.

4.5.7 Motion/speed sensors

Motion/speed sensors measure directly the relative motion between the robot and its environment. Their functioning is based on the detection of a moving object relative to the robot's reference frame and the estimation of its relative speed. There are a number of sensors that inherently measure some aspect of motion such as doppler radar and doppler sound.

4.5.8 Vision-based sensors

Vision-based sensors, as the name suggests, capture the same raw information as the human vision system. Specifically, they elaborate an image based on the incoming light, even if they have specific limitations in performance when compared to the human eye. The two technologies that enable this property are CCD/CMOS sensors.

4.6 Localization

Mobile robot localization is defined as *the procedure of estimating the current pose of a robot in the employed environment, using data extracted by external sensors.*

Based on the information about the initial position, the localization problem is classified into position tracking, global positioning/localization, and the kidnapped robot problem.

The objective of position tracking is to track the robot's position at each instance of time during its navigation in the environment, by exploiting odometry and sensor data. The initial location of the robot must be known and its current location is updated using the robot's prior position, by continually monitoring the route of the robot. In position tracking, the uncertainty about the actual position of the robot is required to be small otherwise the robot might not be localized.

On the other hand, in global localization robot's position is tracked without any information on its initial pose. Solving the problem depends on locating itself globally within the environment (relocation). Robots that have been kidnapped have been

taken to an unidentified place. If the robot is aware that it has been abducted, the issue is comparable to the global localization problem. But, in the other case, the robot still believes it knows where it is and problems start to occur.
Current sensor data and already-known data are combined to estimate the robot's position and hence solve the issue of relocation. An autonomous robot should be able to handle pose monitoring and relocation simultaneously. It should be able to recognize that it is being kidnapped and it should recover its pose by applying relocation method.

Localization includes building a map, and identifying and representing the robot's position relative to that map, both for indoor and outdoor navigation tasks. The existing GPS network provides accuracy within several meters, which is unacceptable for almost every application. Furthermore, GPS does not function indoors or in obstructed areas, thus making localization an essential strategy for map-based navigation.

Robot localization techniques need to be able to deal with noisy observations and generate not only an estimate of the robot's location but also a measure of the uncertainty of the location estimate.
To avoid position uncertainty unbounded growth, the robot must localize itself in relation to its environment. A mobile robot can keep track of its motion using odometry, which is the usage of data coming from motion sensors to estimate changes in position over time. Due to odometry estimation error, the robot will become uncertain about its position after some movements. To allow a more accurate localization, the information provided by the robot's odometry can be combined with the one furnished by exteroceptive observations, thus correcting the error of the estimate.
Localization is a challenging issue from several points of view since it requires the model of the robot, sensors and response to the environment, which are subject to uncertainty. Moreover, the robot can lie in a dynamic environment where nearby objects move and hence having an a priori map is not feasible.
Sensor and effector uncertainty is one of the reasons for the difficulties in localization tasks.

Sensor noise is random (generally unknown) and unwanted variations of sensor output unrelated to variations in sensor input. In other words, it accounts for modifications that a signal may experience during capture, storage, transmission, processing, or conversion. Sensor noise induces a limitation on the consistency of sensor readings in the same environmental state and, therefore, on the number of useful bits available from each sensor reading. Often, the origin of sensor noise problems is that some environmental features are not captured by the robot's representation and are thus neglected. One example is illumination dependence, present in vision-based sensors. Picture jitter, signal gain, blooming, and blurring are all additional sources of noise, that can potentially reduce the useful content of a color video image.
The solution is to take multiple readings into account and combine multiple sensors, employing temporal fusion or multisensor fusion to increase the overall information

content of the robot's inputs.

Another source of uncertainty is sensor aliasing. Aliasing is an effect that causes different signals to become indistinguishable when sampled, hence the same place can look different or even different places can look the same. In robots, the nonuniqueness of sensor readings, or sensor aliasing, is always present. Formally, there is a many-to-one mapping from environmental states to the robot's perceptual inputs and hence the robot is not able to distinguish these multiple states. Even if a sensor is noise-free, the amount of data is generally insufficient to identify the robot's position from a single-percept reading.

Also, mobile robot effectors introduce incertitude about future states, and even motion tends to increase the uncertainty of a mobile robot. This error in motion is viewed as an error in the robot's ability to estimate its position over time using knowledge of its kinematics and dynamics (odometry). The true source of errors generally lies in an incomplete model of the environment. All unmodeled sources of error result in inaccuracy between the physical motion of the robot, the intended motion of the robot, and the proprioceptive sensor estimates of motion. In odometry (wheel sensors only) and dead reckoning (also heading sensors), the position update is based on proprioceptive sensors. The movement of the robot sensed with wheel encoders or heading sensors or both, is integrated to compute the position. Because the sensor measurement errors are integrated, the position error accumulates over time and thus the position has to be updated from time to time by other localization mechanisms. Otherwise, the robot is not able to maintain a meaningful position estimate during navigation.

Localization can also be influenced by the type of environment in which the robot operates. For example, static environments are considered simple because the robot is the only moving obstacle. Instead, dynamic ones lead to a more complex localization approach since any object in the scene could move. Usually in the latter, the robot creates the map as it explores the environment.

Map-based localization systems differ by the type of representation available as well as in the characteristics of the sensors used to observe the environment. A robot must represent the environment (map representation) and its belief about its position on the map (belief representation).

4.6.1 Belief representation

A belief is a hypothesis about the location of the robot in the world. Belief representation refers to the method used to describe the estimate of the robot's state. There exist two types of representation:

Single-hypothesis belief: the robot's position in the environment is represented as a single unique point on a map, thus avoiding any ambiguity about its position. As a consequence, the robot can assume that its belief about its location on the map is correct and update its future state based only on this unique belief, simplifying the decision-making process. The disadvantage regards sensors and effector noise, which induce uncertainty in robot motion. Since the update is based on only one belief and there is no correction about it, the error accumulates making this process challenging and, often, impossible.

Multiple-hypothesis belief: the robot tracks a possibly infinite set of positions. A convex polygon that lies in a 2D map of the environment describes the robot's possible positions without any order. Nevertheless, it is possible to introduce a mathematical distribution representing an order over the possible robot's locations, based on the fact that some positions are likelier than others. As a consequence, the robot can explicitly maintain uncertainty regarding its position and measure its own degree of uncertainty regarding its position. During the decision-making process, some of the robot's possible positions imply a motion trajectory that is inconsistent with some of its other possible positions, thus making the process computationally expensive.

4.6.2 Map representation

Map representation depends on the aspects of the environment that one would describe as well as on the level of fidelity the map represents the environment.
Decisions made regarding environmental representation can have an impact on the choices available for robot position representation. Often the fidelity of the position representation is bounded by the fidelity of the map.
Three fundamental relationships must be understood when choosing a particular map representation:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
3. The complexity of the map representation has a direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

The map can be represented in a continuous or discretized way.
In a continuous map, the position of salient environmental features can be annotated precisely in a 2D continuous space. These elements may be approximated using very straightforward convex polygons, sacrifying exact map reproduction for the sake of computational speed. Together with the use of the closed-world assumption, these techniques can enable a continuous-valued map to be of equal cost, and sometimes

even less costly, than a standard discrete representation. Geometric maps can capably represent the physical locations of objects without referring to their texture, color, elasticity, or any other secondary features that do not relate directly to position and space.
A meaningful way of representing obstacles is to approximate the world with a set of infinite lines. Basically, this transformation from the real world to the map representation is a filter that removes all nonstraight data and extends line segment data into infinite lines that require fewer parameters.
The key advantage of a continuous map representation is the potential for high accuracy and expressiveness with respect to the environmental configuration as well as the robot's position within that environment.

As concern decomposition and abstraction, the representation of the map can potentially be minimized, capturing only the relevant, useful features of the world. On the contrary, there could be a loss of fidelity between the map and the real world, both qualitatively, in terms of the overall structure, and quantitatively, in terms of geometric precision.

Among all decomposition methods, should be mentioned for importance the occupancy grid representation and the topological decomposition.
The basic idea of the occupancy grid is to generate a map of an environment using an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. An example is reported in Fig.4.6.1.

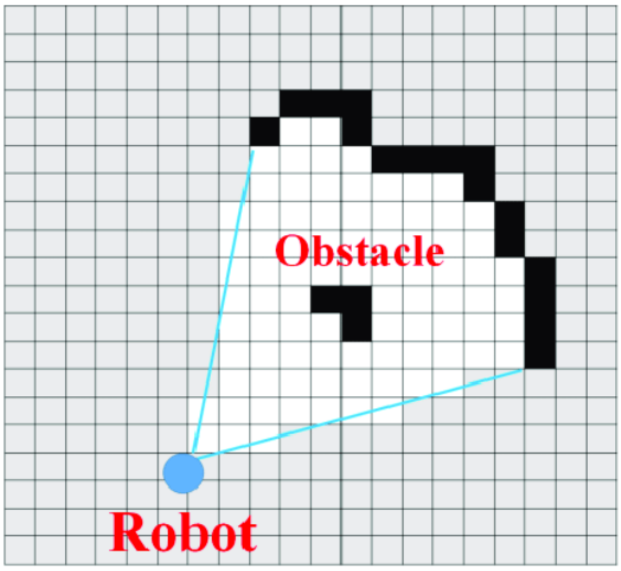


Figure 4.6.1: *Occupancy grid.*

The are two main disadvantages correlated to the implementation of this decomposition method. First, the size of the map in robot memory grows with the dimension of the environment and if a small cell size is used, the computational power can quickly become unsustainable. Moreover, this approach is not compatible with the closed-world assumption, which enabled continuous representations to have potentially very small memory requirements in large, sparse environments. Finally, the a priori geometric grid imposed on the world does not take into account details of the environment, modifying their real dimensions and shapes. This can be appropriate in cases where geometry is not the most salient feature of the environment.

Topological approaches avoid direct measurement of geometric environmental qualities, instead, the method focuses on visualizing the most relevant characteristics of the environment that permit robot localization. An example is shown in Fig.4.6.2.

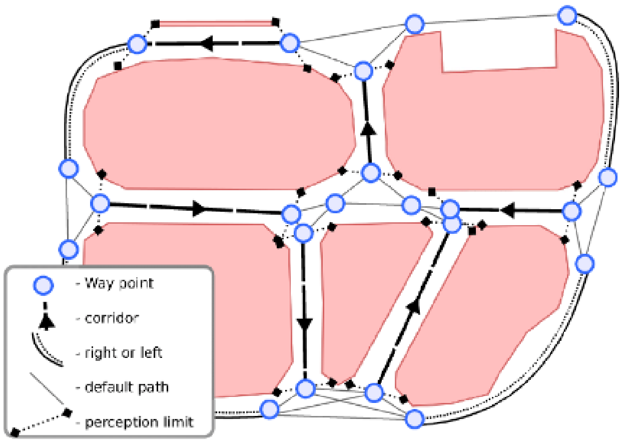


Figure 4.6.2: *Topological representation.*

A topological representation is a graph that specifies nodes and the connectivity between those nodes. Nodes are used to denote areas in the world and arcs are used to denote adjacency of pairs of nodes. When two nodes are connected by an arc, then the robot can directly pass from one to the other without requiring going through any other intermediary node. First, the robot must have the capacity to represent its current position in terms of the nodes of the topological graph. Second, it must have a means for travelling between nodes using robot motion. The node sizes and particular dimensions must be optimized to match the sensory discrimination of the mobile robot hardware.

4.6.3 Localization approaches

Mobile robot localization approaches can be classified into two main categories: probabilistic and autonomous map building [8].

The former identifies the probabilities of the robot being in specific positions, and the probability of a robot in a specific configuration. Since measurement errors

affect sensor data only the probability of a robot in a specific configuration can be computed. The absolute localization problem obtains the absolute position using beacons, landmarks or satellite-based signals (e.g. GPS).
A few nodes (called anchors) need to know their absolute positions, and all the other nodes are absolutely localized in the coordinate system of the anchors. Absolute localization uses the following:

- Active beacons, where the absolute position of a mobile robot is computed by measuring the direction of incidence of three or more transmitted beacons. The transmitters use light or radio frequencies and are placed at known positions in the environment.

- Recognition of artificial landmarks, which are placed at known locations in the environment and are designed to provide maximal detectability even under bad environmental conditions.

- Recognition of natural landmarks, which must be known in advance. The use of natural landmarks, which are distinctive features of the environment, provides lower reliability than the artificial landmarks method.

- Model matching, that is the comparison of the information received from on-board sensors and a map of the environment. The absolute location of the robot can be estimated if the sensor-based features match the world model map.

There are two main strategies:

Markov localization: is an application of Bayesian filter algorithm. It uses an arbitrary probability density function across all possible robot positions in the state space. These systems implement the generic belief representation by dividing the robot configuration space into a finite, discrete number of possible robot poses in the map, usually using grid or topological maps. This representation allows updating the probability of each state within the entire state space at each iteration, as required by the algorithm.
Localization is a two-step process. The prediction step evaluates the belief of the robot's state at time *t* using the belief at time *t-1* and control input. The robot's belief state is usually represented as separate probability assignments for every possible robot pose in its map. In the measurement step, the algorithm predicts the sensor value from the actual state. The probability that the measurement may have been observed is evaluated given the state at time *t*.
Localization employing the Markov method is possible starting from any unknown position and, since the robot can track multiple, completely disparate

potential positions, it is feasible to recover from ambiguous situations. However, the required memory and computational power can thus limit precision and map size.

Kalman filter localization: uses a single, well-defined Gaussian probability density representation to describe robot's belief state and scan matching. The estimates about the next state are updated using a weighted average, where estimates with higher certainty weigh more with respect to the others.
In order to decrease the overall uncertainty of the robot, Kalman Filtering allows the combination of the uncertainties regarding the current state of the robot and its sensor measurements. Differently from Markov localization, Kalman filter localization does not independently consider each possible pose in the robot's configuration space and the initial position has to be known, moreover, it can be used in continuous world representations.
Kalman filter localization is inherently both precise and efficient. It can run in real-time using only the present input measurements and the previously calculated state. However, if the uncertainty of the robot becomes too large the algorithm can fail to capture the multitude of possible robot positions and can become irrevocably lost.
Extended Kalman Filtering is an extension of Normal Kalman Filtering. It removes the restriction of linear state transition and measurement models allowing the use of any kind of nonlinear function to model the state transition and the measurements, being at the same time not computationally expensive. The models are linearized around the current robot state consequently, the measurement model and the state transition model are approximately linear around the actual state. After every time step, the linearization is updated around the new state estimate.

Other important techniques to mention are landmark-based navigation, globally unique localization, positioning beacon systems and route-based localization.
Landmark-based navigation is based on the creation of artificial landmarks that the robot frequently and precisely localizes. If it does not see a landmark it relies only on a prediction about the future state which can lead to getting lost. Therefore, the placement of the landmarks and the map's layout are crucial, thus making the technique dependent on a specific environment and difficult to adapt to others.
In globally unique localization, the robot locates itself with the use of its own sensors that provide data coming from the environment. The technique needs a lot of data coming from environments characterized by several distinctive features.
Positioning beacon systems work with beacons positioned throughout the environment that communicate with the robot and provide localization information. If the information is received from at least three beacons, the localization is very precise but, the system is not adaptable to changing environments.
Route-based localization is based on a set of predefined paths that the robot can take. As the previous technique, it is very precise and not adaptable.

Relative localization is used when the map is constructed step by step by the robot itself. Starting from an arbitrary initial point, a mobile robot should be able to autonomously explore the environment with its onboard sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map. The robot starts navigating from a random initial point and begins to explore the environment through both proprioceptive and exteroceptive sensors. As it acquires the needed knowledge, it creates a map with landmarks as walls and obstacles. This procedure is known also as SLAM. The collected data are based on the displacement of the robot estimated from the odometry and features such as lines, corners and planes. Generally, SLAM is formulated in a probabilistic way, where the present position and robot map are estimated as a probability distribution.

4.7 Cognition

In the cognition phase, the robot plans the necessary steps to reach the target. The cognitive architecture of the robot has the role of planning the path that the robot has to follow to attain its objectives, based on the information from the sensors and the robot's goals. Therefore, the cognitive level of the robot is the decision-making phase and the execution of movements that the robot utilizes to achieve high-level objectives.
Cognitive models intend to represent the robot, the environment, and the manner in which they interact. Mapping algorithms are exploited to build maps of the environment and, eventually, motion planning and other artificial intelligence algorithms might be used to determine how the robot should interact with it.

Path planning deals with finding the best path in order for the mobile robot to reach the target without collision, thus allowing a mobile robot to navigate through obstacles from an initial configuration to another one. Path planning cannot always be designed in advance as the global environment information is not always available a priori. Nevertheless, with the help of a suitable algorithm, it can be successfully applied in fully or partially known environments, as well as in unknown structured ones, where sensors provide raw data to update the map and inform the robot. The complexity of the problem increases with an increase in degrees of freedom of the system.
Every decision in path planning algorithms is chosen accordingly to the available information based on the current state, constraints, and conditions. An appropriate trajectory is generated as a sequence of actions established in order to maintain the correct direction from the starting position to the target point through several intermediate states. The selected trajectory must be smooth without extreme turns as a robot may have several motion constraints, such as the non-holonomic condition in underactuated systems.

Path planning is one of the most crucial research problems in robotics since it needs:

Environment model: it is the environment in which the robot operates. The model is provided with geometrical characteristics and constraints.

Robot model: it contains robot characteristics, as dimensions, physics.

Trajectory choice: it is provided of parameters to choose the trajectory.

The first step of any path-planning system is to transform the possibly continuous environmental model, constructed during localization, into a discrete map suitable for the chosen path-planning algorithm. Different path-planning algorithms affect differently the discrete decomposition. There exist three general strategies for decomposition, which are described in the following:

1. **Road map:** is a network of 1D curves or lines, capturing the connectivity of the robot's free space. It is a decomposition of the robot's configuration space based on obstacle geometry that identifies a set of routes within the free space and that can be used for robot motion planning. Path planning is thus reduced to connecting the initial and goal positions of the robot to the road network, then searching for a series of roads from the initial robot position to its goal position.

   The critical task is to construct a set of routes that together enable the robot to go anywhere in its free space while minimizing the number of total roads. Generally, completeness is preserved in such decompositions as long as the degrees of freedom of the robot have been properly captured. Two road map approaches are visibility graph and Voronoi diagram.

   - For a polygonal configuration space a visibility graph, visible in Fig.4.7.1a, consists of edges connecting all pairs of vertices with an unobstructed path. The method is extremely fast and efficient in sparse environments since the size of the representation and the number of edges and nodes increase with the number of obstacles. However, it can be slow and inefficient compared to other techniques when used in densely populated environments. Visibility graph planning tends to conduct the robot as close as possible to obstacles on the way to the destination, causing problems related to collisions. The visibility graph method can be designed to keep the robot as close as desired to objects in the map.
   - Voronoi diagram method attempts to maximize the distance between the robot and obstacles in the map, see Fig.4.7.1b. The method is based on calculating the distance of each point in the free space to the nearest obstacle. At points that are equidistant from two or more obstacles, such a distance plot has sharp ridges. The Voronoi diagram consists of the edges formed by these sharp ridge points. When the configuration space obstacles are polygons, the Voronoi diagram consists of straight and parabolic segments. In localization problems, the path will be quite

poor if short-range sensors are employed since the robot will be in danger of failing to sense its surroundings.

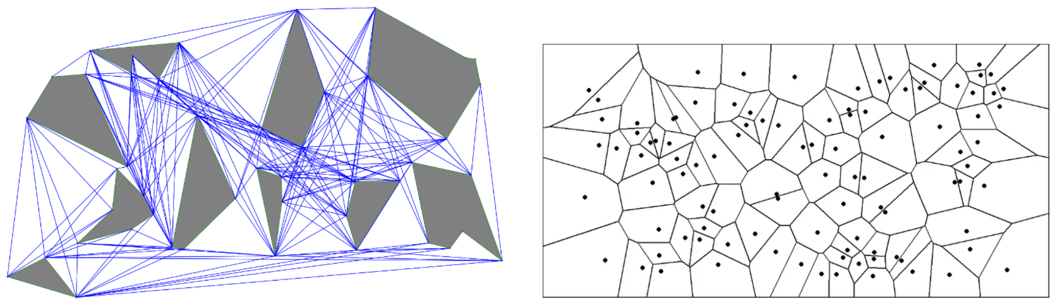
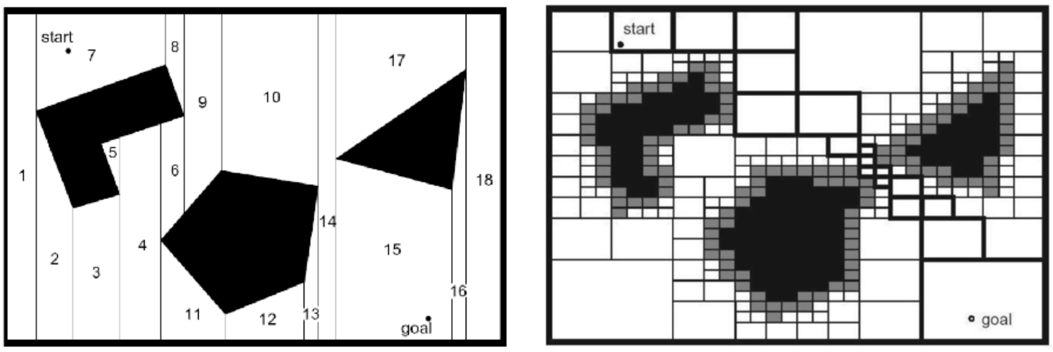

(a) Visibility graph (b) Voronoi diagram

Figure 4.7.1: *Road map representations.*

2. **Cell decomposition:** discriminate between free and occupied cells, see Fig.4.7.2. There are two categories of cell decomposition based on the placement of the boundaries between cells. If the boundaries are placed as a function of the structure of the environment, such that the decomposition is lossless, then the method is defined as exact cell decomposition. Instead, if the decomposition results in an approximation of the actual map, the system is called approximate cell decomposition.

    - Exact cell decomposition divides the space into non-overlapping cells in the shape of triangles and trapezoids, which can be accomplished by adding vertical line segments at every obstacle's vertex. The representation can be extremely compact because each area is stored as a single node, thus making irrelevant the precise position of the robot. What matters is the robot's ability to cross adjacent areas of free space. If this information is expensive to collect or even unknown, then such an approach is not feasible.

    - The most popular approach of approximate cell decomposition is fixed decomposition, in which the world is tessellated, transforming the continuous real environment into a discrete approximation for the map. It is possible for narrow passageways to be lost during such a transformation. The key disadvantage of this approach stems from its inexact nature. Yet another approach is adaptive cell decomposition.



(a) Exact decomposition (b) adaptive cell decomposition

Figure 4.7.2: *Cell decomposition representations.*

3. **Potential field:** imposes a mathematical function over the space, see Fig.4.7.3. The potential field method treats the robot as a point under the influence of an artificial potential field. It creates a field, or gradient, across the robot's map that smoothly directs the robot to the goal position from multiple prior positions while simultaneously avoiding known obstacles.

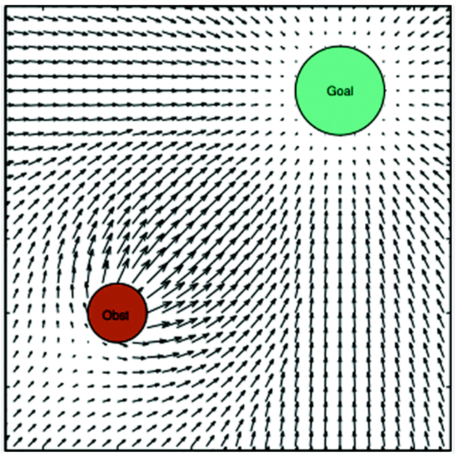


Figure 4.7.3: *Potential field representation.*

Generally, path planning algorithms are divided into two categories based on available information about the environment [9].
The first is global path planning. Under this situation, it is given largely complete environmental information, the environment is static, and its global information is known a priori in the control design. The algorithm produces a complete path from the starting position to the goal configuration, before the robot starts following the planned trajectory. This approach is expensive in implementation and relatively well-studied in the existing literature.
The second is local path planning, where the path is generated by taking data from

the sensors during the movement of the robot. In opposition, generally the environment is unknown or dynamic therefore the robot generates a new path to respond to changes of the environment. This method is more complicated in design but more applicable in practice.

There are four essential predominant trade-off criteria that must be considered in a path planning algorithm. The first is optimization, which ensures that the chosen solution is the best path in realistic static environments. Second, it must be applicable to dynamic environments. Third, it must remain compatible with and enhance the chosen self-referencing approach. Fourth, it must minimize the complexity, data storage, and computation time.

Over the years, numerous methodologies have arisen that attempt to solve the motion planning applied to mobile robots [9][10].
The most popular ones are Dijkstra's algorithm and A* algorithm. To support path planning in dynamic environments, D* is discussed as an efficient tool for quick re-planning in cluttered environments. As D* does not guarantee solution quality in large dynamic environments, the Rapidly-exploring Random Trees method (RRTs) is also explored.

4.7.1 Dijkstra's algorithm

Dijkstra's algorithm is one of the first algorithms for finding the shortest paths among nodes in a graph in an acyclic environment. The algorithm uses a greedy approach, it finds the next local optimal solution hoping that the end result is the best solution for the whole problem. The subsequent closest vertex is chosen by maintaining the new vertices in a priority-min queue and storing only one intermediate node so that only one shortest path can be found.

Four steps are repeated until the shortest distance between the origin and destination is found:

Step 1: convert the road network to a graph, and distances between nodes in the graph are expected to be found by exploration.

Step 2: pick the unvisited node with the lowest distance from the source node.

Step 3: calculate the distance from the picked node to each unvisited neighbour and update the distance of all neighbour nodes if the distance to the picked node is smaller than the previous distance.

Step 4: mark the visited node when the calculation of distances to all neighbours is accomplished.

Dijkstra is a reliable algorithm for path planning conceived by computer scientist Edsger W. Dijkstra in 1956. Computations require a considerable quantity of memory as the algorithm has to compute all the possible outcomes in order to determine the shortest path, and it cannot handle negative edges. Dijkstra is best suited for a static environment and/or global path planning as most of the required data is predefined for the computation of the shortest path. There exist many variants of the algorithm that can handle, for example, dynamic environments. In this case, the environment is partially or completely unknown, and thus the node information with respect to obstacles is computed in real-time.

4.7.2 A* algorithm

The A* Algorithm is a popular graph traversal path planning algorithm, based on the best-first search. A* operates similarly to Dijkstra's algorithm but differently it only finds the shortest path from a specified source to a specified goal, and not the shortest path tree from a specified source to all possible goals. It is formulated in terms of a weighted graph, where it aims at finding a path of minimal cost from a starting node to a final goal position. A* uses a priority queue to perform the repeated selection of the nodes of minimum cost to expand. At each step of the algorithm, the node with the lowest value is removed from the queue, the values of its neighbours are updated accordingly, and these neighbours are added to the queue. The algorithm continues until a removed node is a goal node.
A* is the most widely used for approaching a near-optimal solution with the available dataset and in static environments. It is simpler and less computationally heavy than many other path planning algorithms, with its efficiency lending itself to operation on constrained and embedded systems.

4.7.3 D* algorithm

The D* (or Dynamic A*) is an incremental search algorithm used to generate a collision-free path among moving obstacles. D* partially repairs the cost map and the previously calculated cost map. The D* algorithm processes a robot's state until the path cost from the current state to the goal is less than a minimum threshold. The states sequence is computed with back pointers to either direct the robot to the goal position or update the cost due to a detected obstacle and place the affected states on the open list. The cost changes are propagated to the next state, and the robot continues to follow back pointers in the new sequence towards the goal.
D* algorithm is suitable for partially known and dynamic environments, solving the path planning problem in an efficient way with high memory consumption. It can be employed for any path cost optimization problem where the path cost changes during the search for the optimal path to the goal. D* is most efficient when these changes

are detected closer to the current node in the search space. The D* algorithm has a wide range of applications, including planetary rover mission planning.

4.7.4 RRT algorithm

RRT is a dynamic and online algorithm designed to efficiently search in non-convex, high-dimensional spaces by randomly building a space-filling tree without requiring a predefined path. The branches expand in all regions, and, based on weights assigned to each node, the algorithm creates a path from the starting position to the goal. It constructs a tree that attempts to explore the workspace rapidly and uniformly via a random search.
It is composed of two phases:

Phase 1: a collision-free probabilistic road map is constructed and stored as a graph.

Phase 2: a path that connects original and targeted nodes is searched from the probabilistic road map.

RRTs can be viewed as a technique to generate open-loop trajectories for nonlinear systems with state constraints. An RRT can also be considered as a Monte-Carlo method to bias search into the largest Voronoi regions of a graph in a configuration space. They were specifically designed to handle non-holonomic constraints and are widely used for commercial and industrial purposes, also because of the possibility of being applied to almost any wheeled system.

4.7.5 Ant Colony Algorithm

The ant colony optimization (ACO) algorithm is based on a heuristic multi-agent approach inspired by the collective behavior of trail-laying ants to find the shortest and collision-free path, locating optimal solutions by moving through a parameter space representing all possible solutions.
When a colony of ants is confronted with the choice of reaching their food via two different routes of which one is much shorter than the other, their choice is entirely random. However, those who use the shorter route reach the food faster and therefore go back and forth more often between the anthill and the food.

4.8 Motion control

The motion control phase allows the robot to achieve a desired trajectory by modifying its motor outputs. Motion control is the process that decides robot's movement to complete tasks that have already been defined using rotary and linear actuators. Robot arms move through the action of rotating and sliding joints, while mobile robots move through locomotion and steering.

Whenever more than one way to do something with a robot is present, the chosen way should have special qualities that are not present in the other ones. A path can maximize distance from a collision, improve strength, minimize time, avoid workspace limits, reduce power consumption, and improve accuracy. In practice, the best motion will usually be a combination of these qualities.
Motion control must also incorporate constraints like speed and acceleration limits for robot joints. Actuators have maximum torque or force. Physical parts of the robot cannot overlap in space, and joint limits cannot be exceeded. These are constraints imposed by the physical reality of the robot and the world. The desired tasks, constraints, and optimizations combine to make robot motion control a challenge.

An automated motion control system consists of three main components: a motion controller, a motor driver or amplifier, and a motion device.

The primary purpose of a motion controller is to control the dynamics of the motion device. Motion controllers calculate the path and generate the commands for the rest of the system, that guide the trajectory and velocity of the machine to the predefined goal. Motion controllers use multiple algorithms to calculate the precise movements of the device and adjust their commands to meet the requirements of the system on the basis of the received feedback information. Common motion systems use three types of control methods: position control, velocity control, and torque control. Each control method is based on a feedback device whose basic function is to transform a physical parameter into an electrical signal for use by the controller.
The motor driver converts the low-voltage command signals from the motion controller into power signals required to move the motor. They use these signals to convey to the rest of the motion control system the amount of high-power current and voltage needed.
The motion device, also called actuator, is any mechanical device that provides motion once actuated by a motor. Such motion devices typically contain feedback devices to provide information such as position and velocity to the motion controller. Motors power the motion control system by converting electrical energy into mechanical energy. Feedback sensors, used in closed-loop systems, provide feedback that motion controllers use to make necessary adjustments to the system, ensuring that the proper commands are given at all times. The most popular feedback device is an encoder, which is an electromagnetic device that provides information on position, velocity and direction. There are also absolute encoders, which directly track positions using many unique values.

5 Design of an algorithm for autonomous navigation

5.1 Introduction

In this chapter, we will present an algorithm for autonomous navigation in an unknown environment where the target is not known.
In this case study, we consider a ground robot that safely navigates in an unstructured and unknown environment without human supervision.
The scene to explore is thought to be an office building or an unstructured working area characterized by static obstacles, narrow passages, and bifurcations of corridors. Although this work is meant for an office building, it can be adapted to a variety of different contexts with few modifications.

The central focus of this chapter is the mathematical definition and implementation of a decision algorithm designed to autonomously reach a target position, also in the presence of bifurcations or obstacles. The main difficulty in these cases is deciding which corridor or road to take to achieve a predefined goal of unknown position. This scenario is challenging to handle because of the option of repeating the same choices, due to the fact that the map of the environment is not given to the robot. One must find a way to avoid repeating the same step over and over since this would result in not reaching the target. For example, if you take as decision rule the maximum distance that you can reach, you may get stuck between the same two points if no other points are more distant.

A way to address the problem of taking decisions is to define a graph-based formulation. It involves delineating a graph in $\mathbb{R}^2$ whose nodes represent the robot's possible positions, and arcs between two nodes provide possible paths. The creation of a map, as the building is explored, permits an efficient way of reaching the target. The algorithm has been completely designed and tested using MATLAB R2020b.

In the following sections, the mathematical formulation of the problem and its parts will be illustrated and explained.
Sec. 5.2 will present the general problem and the space in which the variables live. In Sec. 5.3 will be defined the graph and any variable that is used in the algorithm. The obstacles are introduced in Sec. 5.4 and in Sec. 5.5 will be explained the implemented algorithm and the step that it requires. Finally, Sec. 5.6 reports the results of some

simulations obtained changing the number of keypoints and dimensions of obstacles in the environment.

5.2 Mathematical formulation of the problem

Navigation is the ability to determine your location and plan a path to some goal. The task of an autonomous robot is to find and execute a continuous sequence of actions that leads it to a given goal, avoiding collisions with obstacles without a human, using a set of sensors.

Consider the space $\mathbb{R}^2$ as composed of two complementary sets:

$$\mathbb{R}^2 = \mathcal{X}_{free} \cup \mathcal{X}_{coll} \tag{5.2.1}$$

Assume the collision space $\mathcal{X}_{coll} \subset \mathbb{R}^2$ a subset of the plane that includes portions of the space in which are present exclusively obstacles. On the other hand, $\mathcal{X}_{free} \subseteq \mathbb{R}^2$ consists of obstacle-free plane parts in which a robot can freely move.

For now, suppose that the set of points between which navigate is given and call it $\mathcal{P}$:

$$\mathcal{P} = \{\boldsymbol{p}_i \in \mathbb{R}^2,\ i = 1, \dots, T\},\ T \in \mathbb{N}, \tag{5.2.2}$$

T being the number of provided keypoints.

The robot is assumed to move in an unknown environment, along a trajectory described by a sequence of points $\{\boldsymbol{p}_1, \dots \boldsymbol{p}_F\} \subseteq \mathcal{P}$ with $F \in \mathbb{N}$ being the index of used keypoints.

The objective of the algorithm is the achievement of the position $x_{goal} = p_k,\ p_k \in \mathcal{P}$.

5.3 State

An undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a relational structure composed of nodes and arcs that connect them in pairs, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of arcs. Mathematically,

$$\mathcal{V} = \mathcal{P} \tag{5.3.1}$$

$$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}, \quad e \in \mathcal{E}, \iff \exists i, j \in \mathcal{V} \text{ s.t. } e = (i, j) \tag{5.3.2}$$

Every single node of $\mathcal{V}$, the cardinality of which could be infinite, represents a position in $\mathbb{R}^2$. The third dimension is not modelled, since the robot can move only on the plane and can not move along the z-direction.

For the given problem, suppose that the set $\mathcal{V}$ of nodes is drawn according to a uniform distribution in the map, while $\mathcal{E}$ needs to be found.

The set of nodes is comprised of several keypoints, some of which could lead to collisions. The scope of the algorithm will be to decide which nodes can be reached and which nodes lead to collisions.

As concern arcs, they are constructed only if it is possible to travel across them, passing from one node to the neighbour. Otherwise, the arc is not built since there is an obstacle.

Consider the set $\mathcal{N}(x_t) \subseteq \mathbb{R}^2$, called *neighbour*, defined to be a collection of nodes at a distance smaller than a predefined value $\mathcal{N}(x_t) = \{\boldsymbol{p}_i \in \mathcal{V} \ s.t. \ \|x_t - \boldsymbol{p}_i\| \leq d_{neighbour}, \ d_{neighbour} > 0\}$.

At time t, the state x_t is represented by a node $\boldsymbol{p}_i \in \mathcal{N}(x_t)$:

$$x_t = k \tag{5.3.3}$$

The two possible outcomes can be synthesized as follow:

- if the segment connecting x_t and $\boldsymbol{p}_i \in \mathcal{N}(x_t)$ intersects an obstacle, an edge does not exist
- the segment does not intersect any obstacle, and the edge that links x_t and $\boldsymbol{p}_i$ can be included in the list of possible paths

Define the segment that connects two points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ as

$$e_{i,j} = \boldsymbol{p}_i + \lambda(\boldsymbol{p}_j - \boldsymbol{p}_i), \ with \ \lambda \in [0, 1] \tag{5.3.4}$$

$e_{i,j} \in \mathcal{E}$ if $\forall f \in e_{i,j}, \ f \in \mathcal{X}_{free}$.

The initial position x_0 defines the position on the world reference frame and can be chosen arbitrarily. For sake of simplicity, suppose the starting point to be in position [0, 0] with respect to the world reference system.

5.4 Obstacles

In this section we define and describe two sets of obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$, a set of circular obstacles, and $\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_W\}$, a collection of linear obstacles. A linear obstacle $\mathcal{L}_i$ is part of a straight line bordered by two points $a_i, b_j \in \mathbb{R}^2$:

$$\mathcal{L}_i = \{a_i + \lambda(b_j - a_i), \ with \ \lambda \in [0, 1], \quad i = 1, \ldots W \in \mathbb{R}^2\} \tag{5.4.1}$$

A circular obstacle $\mathcal{O}_i$ is a set of points equally distant from a predefined centre $o \in \mathbb{R}^2$. The distance of each point from its centre is $r_i, \;\; i = 1, \dots N$, where $N \in \mathbb{N}$ is the number of obstacles:

$$\mathcal{O}_i = \{(x - o_x)^2 + (y - o_y)^2 = r_i^2, \quad i = 1, \dots N\} \tag{5.4.2}$$

The centres and the dimensions of the obstacles are randomly generated in the given square space of $\mathbb{R}^2$.

5.5 Update rule

In order to accomplish the task, it is necessary to differentiate keypoints that the robot can reach without hitting obstacles from those that imply an accident.

The first order of business is to define the set $\mathcal{E}$ of arcs between reachable places. Notice that, while $\mathcal{V}$ includes all the points that the robot perceives, $\mathcal{E}$ contains only the paths that it can safely travel, discarding all the ones that could lead to an obstruction.

Once you have defined all the feasible corridors, the update rule leads the robot to the next node. The strategy just explained is repeated until the achievement of the objective.

5.5.1 Step 1: intersections

The first step consists in verifying whether a potential arc intersects impediments. This results in searching for intersections between straight lines constructed among x_t and nodes of the neighbourhood and the given obstacles.

At time $t + 1$, all the nodes inside a certain predefined distance $d_{neighbour}$ are saved in the set $\mathcal{N}(x_t)$. We define a line passing through the actual position x_t and the next possible state $\boldsymbol{p}_j, \; \forall \boldsymbol{p}_j \in \mathcal{N}(x_t)$.

Given $a, b \in \mathcal{V}$, the line passing through them can be found with the formula:

$$l = a + \lambda(b - a), \; with \; \lambda \in [0, 1] \tag{5.5.1}$$

Once all the elements are defined, a possible intersection is easily found by checking the presence or absence of common points.

All edges that do not present intersections with obstacles are considered feasible and hence they are added to $\mathcal{E}$.

The pseudocode for this part is reported in the following.

Algorithm 1: Straight lines computation

```
$X_a \leftarrow x_t$
for $i \in \mathcal{N}(x_t)$ do
    $X_b \leftarrow \boldsymbol{p}_i$
    $\mathrm{X}_a + \lambda(X_b - X_a),\ with\ \lambda \in [0,1]$
    for $o \in \mathcal{O}$ do
        $xyr \leftarrow [x_{circleCentres}(o)\ y_{circleCentres}(o)\ radii(o)]$
        $x_o \leftarrow xyr \cdot cos(ang)$
        $y_o \leftarrow xyr \cdot sin(ang)$
        $eliminations \leftarrow polyxpoly(X_a, X_b, x_o, y_o)$
    end
    for $l \in \mathcal{L}$ do
        $x_l \leftarrow [x_{in}(l)\ x_{end}(l)]$
        $y_l \leftarrow [y_{in}(l)\ y_{end}(l)]$
        $eliminations \leftarrow polyxpoly(X_a, X_b, x_l, y_l)$
    end
end
$\mathcal{N}(x_t) \leftarrow !eliminations(\mathcal{N}(x_t))$
```

5.5.2 Step 2: decision

The second step consists in deciding which edge to follow and then which node will represent the state at time $t+1$. A way to choose which node to visit is to define the following cost function for each node $i \in \mathcal{N}(x_t)$:

$$J_{fit}(i) = d_i(x_t, i) + \frac{w_i}{n} \tag{5.5.2}$$

n provides the number of nodes already travelled inside a predefined radius R.
In detail, $d_i(x_t, i)$ is the distance between the state x_t and a node i:

$$\forall i \in \mathcal{N}(x_t), \quad d_i(x_t, i) = \sqrt{(x_{t_x} - i_x)^2 + (x_{t_y} - i_y)^2} \tag{5.5.3}$$

Define the set $\mathcal{Z}$ to be the set of nodes $\boldsymbol{p}_i \in \mathcal{V}$ that are already visited. Define the set $\mathcal{T} = \{j \notin \mathcal{Z}\ \&\ d_{ij}(i,j) \leq R,\ i \in \mathcal{N}(x_t)\}$. Lastly, the function w represents the cost attributed to the proximity of nodes to nodes already travelled:

$$w_i = w_i - \frac{g}{d_{ij}(i,j)},\ i \in \mathcal{N}(x_t),\ j \in \mathcal{T} \tag{5.5.4}$$

In this case the variable g is set to the value 100.

The next point to reach will be:

$$x_{t+1} = c, \quad c \in \arg\max_{i \in \mathcal{N}(x_t)} \{d_i(x_t, i) + weight_i\} \tag{5.5.5}$$

In case of multiple value of c, choose one value at random.

x_{t+1} is the farthest node from the previous position and also from the already travelled nodes. The idea is to choose the most distant node if it is far away from nodes previously selected, since a greater travelled distance at each time step improves the algorithm's exploration efficiency. But, if this node is near to some $j \in \mathcal{T}$ it is better to pick a closer $i \in \mathcal{N}(x_t)$ that is located distant from all $j \in \mathcal{T}$.

Thereby, we keep track of the sequence of movements around the building, that helps avoiding revisiting the same paths, leading to an endless-loop. By saving the path, first will be visited the unseen and far away nodes and after, you can retrace some edges if needed. The selection of points distant from nodes in $\mathcal{T}$ reduces the zig-zag exploration of the environment.

The pseudocode for this part is reported in the following.

Algorithm 2: Next state computation

```
for $i \in \mathcal{N}(x_t)$ do
    $weight \leftarrow 100$
    $D \leftarrow d_i(x_t, i)$
    $n \leftarrow 1$
    for $j \in \mathcal{T}$ do
        $d \leftarrow d_{ij}(i, j)$
        if $d == 0$ then
            $weight \leftarrow 0$
        else
            if $d \leq R$ then
                $weight \leftarrow (weight - \frac{100}{d})$
                $n \leftarrow n + 1$
            end
        end
    end
    $fitness(i) = D + \frac{weight}{n}$
end
$j \leftarrow max(fitness)$
$choice \leftarrow \mathcal{N}(x_t)(j)$
```

5.6 Results

In this section will be presented some cases regarding the functioning of the implemented algorithm.
Most of the time, the underlying algorithm does not lead to the fastest route, achievable with global path planning, because the robot chooses the next state based on a trial-and-error approach. The algorithm does not calculate a direct route to the target, since it is unknown, but only the next state based on the provided local information. As the robot reaches a point, new information about it and its surroundings is added to the map. The map is not completely known until any point of the environment has been explored and the target position is visible only if the point belongs to the neighbour of the actual state.
Nevertheless in the considered experimental cases, we observe that the algorithm always arrives at the final goal. The reason relies on the trial-and-error approach of the algorithm which makes it run until the target is finally reached, even if it takes a long time.

The simulations are not intended to represent any real case. Actually, the objective is to develop an idea about an autonomous algorithm for navigation and implement it using MATLAB then, test the correctness of the self-developed algorithm in a simplified case. In this regard, the robot is treated as a point in $\mathbb{R}^2$ and the dimensions employed in simulations do not reflect the true dimensions of a real environment. Nevertheless, the sizes of obstacles and action radius of a hypothetical sensor are consistent with each other and scaled with respect to the real world.

For the experiments, $d_{neighbour}$, the maximum distance in which keypoints are observable, is set at the value of 15cm, mainly because of the randomly generated points. Specifically, if the distance is set to a smaller value it could happen that any point is inside the radius and hence the algorithm stops. The dimensions of the map vary in the two reported cases, as described below.

5.6.1 100 keypoints

Firstly, the algorithm has been tested with 100 keypoints uniformly distributed in a window of dimension $50cm \times 50cm$. In the same manner, have been generated 9 circular obstacles with a maximum radius of 7cm and 9 linear obstacles with a maximum length of 10cm.

Due to the random process of generating the obstacles and the keypoints, the navigation task difficulty changes on each case shifting from a simple scenario to an impossible one in which the target can not be reached for geometrical reasons.

In the following, three different situations that summarize this behavior will be reported. The first reported case regards a simple layout of the environment where obstacles permit the completion of the navigation task. At any step there exists at least one reachable point, obstacles do not block all the roads so there is at least one feasible road avoiding the interruption of the algorithm. Finally, the target point is not contained in any circular obstacle or obstructed by linear ones, and hence it is reachable.

Thanks to the weights imposed on the keypoints (eq. 5.5.4) we can avoid retracing the same locations and points in their proximity. If the target had been placed near a visited keypoint, the algorithm would have detected it and it would have been reached. If this is not the case, it makes sense to search in an area as distant as possible coherently with the imposed constraints, in order to reduce the necessary steps.

As Fig.5.6.1 shows, in seven steps the target is reached. We can note that the state is chosen far away from the actual position and no obstacles are hit. As we can see, the direction for reaching the second state abruptly deviates from the previous trajectory due to the presence of many obstacles obstructing the path.

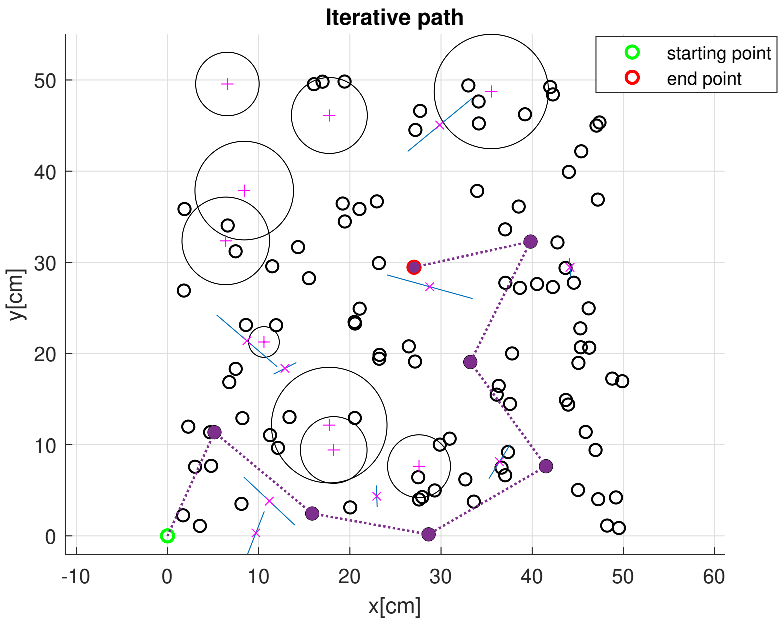


Figure 5.6.1: *Iterative easy path with 100 keypoints (black markers) and a reachable target. Purple markers are the states of the algorithm, blue lines represent linear obstacles and black circles circular obstacles. Finally, magenta crosses are the obstacles' centres, from which the obstacles are generated.*

The second case, Fig.5.6.2, reports a difficult circumstance. The target position is hidden from above by obstacles and hence the robot can not follow a direct path. The algorithm works as expected; obeying to the rule previously explained once the algorithm changes direction to avoid getting closer to a visited area. In the bottom left of the graph, a point is reached two times because all the other positions were inaccessible due to the presence of obstacles.

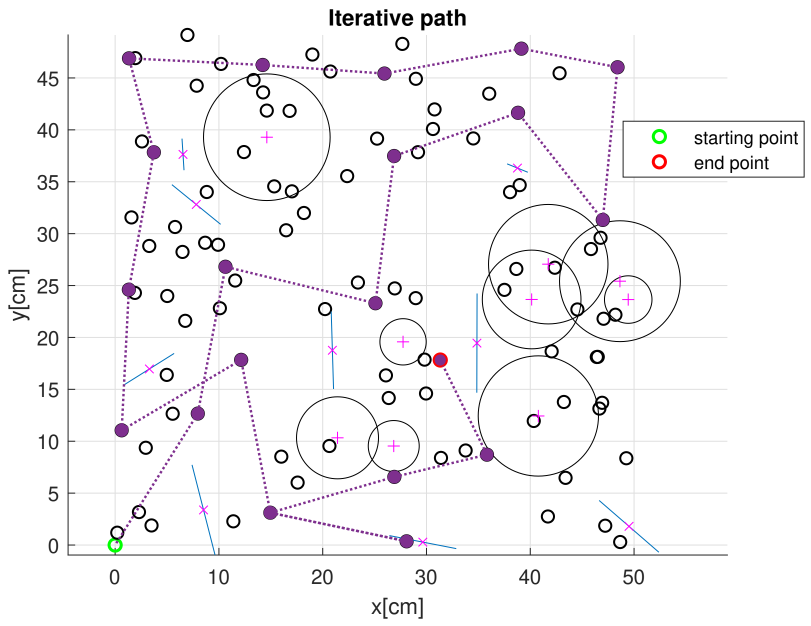


Figure 5.6.2: *Iterative path with 100 keypoints (black markers) and a reachable target in a more complex situation. Purple markers are the states of the algorithm, blue lines represent linear obstacles and black circles circular obstacles. Finally, magenta crosses are the obstacles' centres, from which the obstacles are generated.*

Lastly, Fig.5.6.3 illustrates an unsolvable situation in which the target is not reachable, since it is encapsulated into a circular obstacle. The simulation is manually stopped, as there is no stop condition in addition to the one regarding the reaching of the target.

5.6.2 200 keypoints

In a second moment, the number of keypoints has been raised to 200 on a window of dimension $100 \times 100cm$. Also, the number of obstacles has been increased to 15 for each type. The maximum radius has been set to 10cm while the maximum length to 15cm.

The approach that the algorithm applies and the obtained results are the same as the previously reported case. Fig.5.6.4 and Fig.5.6.5 display as said.

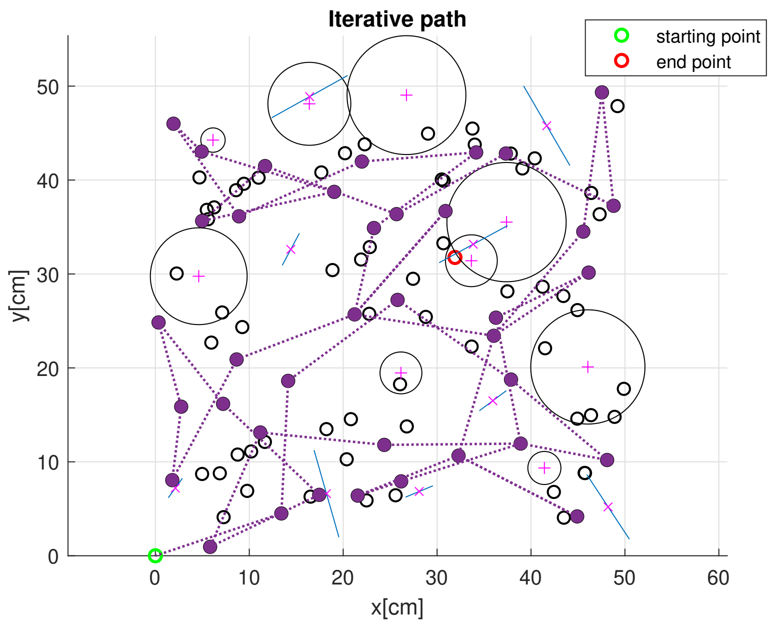


Figure 5.6.3: *Iterative path with 100 keypoints (black markers) and an unreachable target. Purple markers are the states of the algorithm, blue lines represent linear obstacles and black circles circular obstacles. Finally, magenta crosses are the obstacles' centres, from which the obstacles are generated.*

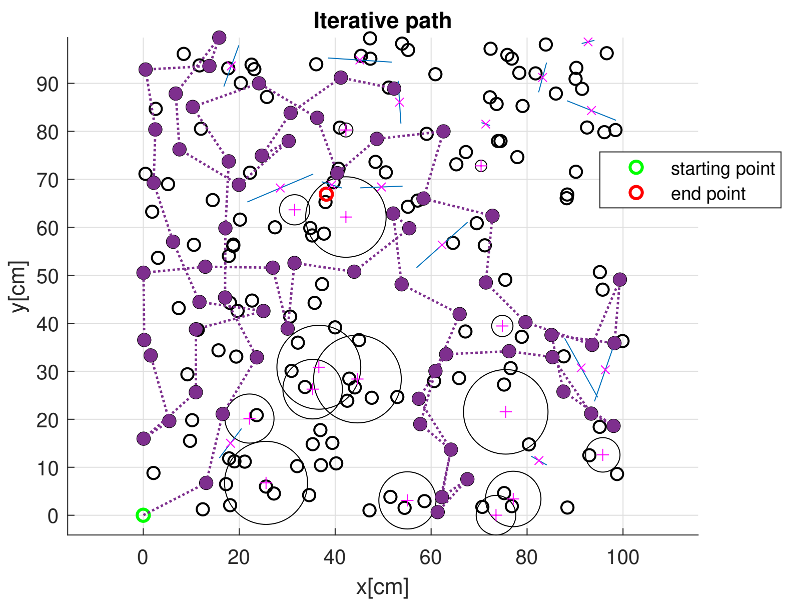


Figure 5.6.5: *Iterative path with 200 keypoints (black markers) and an unreachable target. Purple markers are the states of the algorithm, blue lines represent linear obstacles and black circles circular obstacles. Finally, magenta crosses are the obstacles' centres, from which the obstacles are generated.*

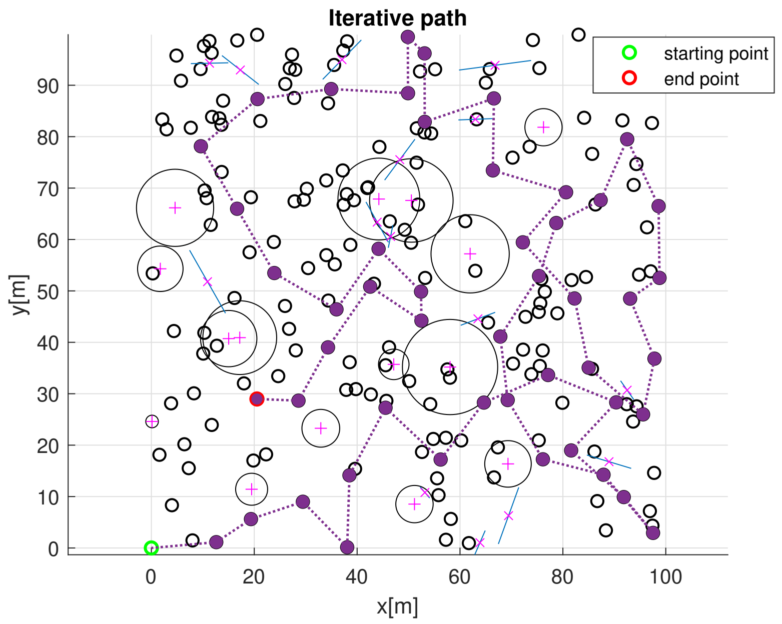


Figure 5.6.4: *Iterative path with 200 keypoints (black markers) and a reachable target. Purple markers are the states of the algorithm, blue lines represent linear obstacles and black circles circular obstacles. Finally, magenta crosses are the obstacles' centres, from which the obstacles are generated.*

5.7 Future works

The algorithm can be enhanced regarding some control actions that can be added. Since points are randomly generated, it could happen that there are no points reachable by the actual state, due to a greater distance than $d_{neighbour}$ or to the presence of obstacles in the middle of the trajectory. To overcome this problem, one can decide to add a point at a distance less than $d_{neighbour}$. This point can be randomly generated with the only constraint of the distance to the explored point. Another interesting solution can be maintaining the previous direction until the detection of at least one keypoint.
A further improvement is the one concerning a condition of stoppage of the algorithm in the case in which the target point is randomly positioned inside an obstacle, leading to an unsolvable situation.

Moreover, obstacles can be dynamic and not static as in this work.

In the end, the algorithm can be implemented and tested in a real robot in which new trajectory points are detected by onboard sensors.

6 NAPVIG algorithm

6.1 Introduction

In this chapter we will present and describe an innovative algorithm for safe autonomous navigation.
The developed algorithm is called NAPVIG (Narrow Passage Navigation) and it has been proposed in [11] in order to safely and reactively navigate through an environment that can change over time.

Reactive navigation deals with navigation in unknown, cluttered, and dynamic environments where the algorithm has to adapt to respond to the variations around the vehicle, using any prior global map information.
The problem addressed by reactive approaches is the computation of only the next command of the robot. Usually, the approach is based on the definition of a function that specifies the cost of the possible movements in terms of surrounding obstacles and the direction of the desired goal if provided. At each time step, the algorithm looks for the minimum cost and the correspondent control is executed. For the computation to be fast enough and allow the robot to be reactive to the changes in the environment, only the local information and a short-time period are considered.

The NAPVIG algorithm implements local and reliable autonomous navigation with the aim of reaching an unknown target position in an unknown environment, with possibly moving obstacles.

For an autonomous robot, handling a non-static scenario is a very important requirement. For example, consider an office building where a package or a chair can be suddenly moved and placed in the middle of the corridor when the robot is passing by. The robot must avoid obstacles to protect itself and the people around it.
With this simple example we can understand that handling unknown environments is a very important task, hence developing a completely autonomous navigation algorithm, that needs no map to safely navigate and reach a goal position by exploring the area, is a step forward for autonomous driving in all areas.

The NAPVIG algorithm proposes an innovative online approach for reactive navigation in environments characterized by narrow passages and moving obstacles. This algorithm presents high accuracy in computation to ensure treating challenging sce-

narios and presents an efficient way of continuously computing precise trajectories according to sensor data coming from onboard sensors.
It has been implemented in Python and C++ languages and implemented using ROS (Robot Operating System). ROS is a set of software libraries and tools that allows testing the algorithm in simulation, using a large variety of robots, also with one similar to the one present in the laboratory.

6.2 Algorithm description

The method applied in the algorithm enables the computation of the desired trajectory without a grid map approximation. High reactivity is achieved thanks to the low computational costs allowing the algorithm to be used in particular scenarios, where high reactivity is fundamental.
The first thing to take into consideration is that the robot is not aware of its own global position in the world frame and can only sense the target with onboard sensors. The algorithm does not predict a direct trajectory to the target, since it could result in dead-ends. Moreover, the algorithm is not intended for global navigation problems.

To cope with dynamic environments, the algorithm iteratively computes a trajectory to follow, based on the value of the landscape, the value of the landmarks and the distance from the target. The target is a position $x_f(t) \in \mathcal{C}_{free}$, where $\mathcal{C}_{free}$ is the complementary set of $\mathcal{C}_{coll}(t) \subset \mathbb{R}^2$, representing the possible time-variant collision space. The chosen trajectory is the one with the maximum possible distance from every obstacle. Indeed, the best trajectory is the one that solves the optimization problem:

$$\xi* = \arg \max_{\xi^{(h)}, h=0,\cdots,H} \{J(\xi^{(h)}, c_\xi^{(h)})\} \tag{6.2.1}$$

where $c_\xi^{(h)}$ is the associated exit status that causes the prediction to terminate and $H \in \mathbb{N}$ is the total number of predicted trajectories to each of them it is associated with the cost value $J(\xi^{(h)}, c_\xi^{(h)})$. In cases in which the trajectory has to be suddenly discarded (e.g. collisions) the cost is set to be infinite.

The trajectories computed by the NAPVIG algorithm always follow the GVD (Generalized Voronoi Diagram) of the map. The generalized Voronoi diagram is a structure that divides space into a complex of generalized Voronoi cells (GVCs) around objects. Each GVC contains exactly one object or site, and every point in the GVC is closer to its contained object than to any other object. The generalized Voronoi diagram is the boundary of the cell complex, and thus every point on the GVD is equidistant from two or more closest objects.

As said before, the trajectory depends on three parameters.
The landscape is a function that maps each point of $\mathbb{R}^2$ to a value related to the dis-

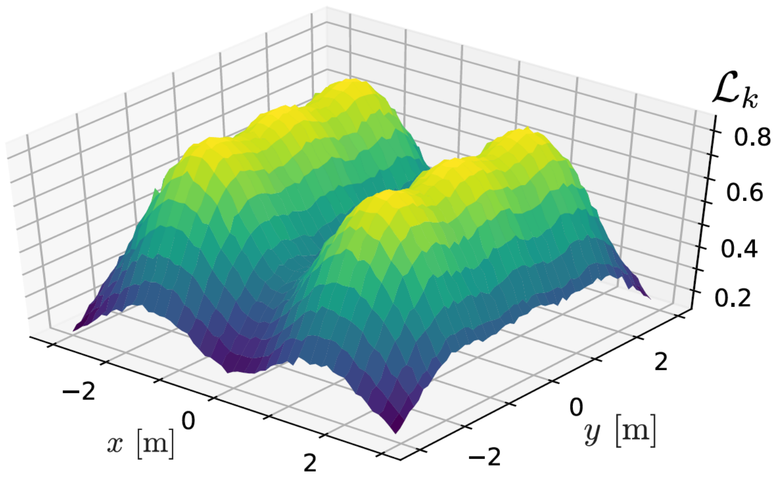


Figure 6.2.1: *Smooth landscape function obtained. The yellow part represents two obstacles (where the measurements come from) while the blue areas are the safe regions, where the trajectories rely.*

tance of the nearest measurement. It is the superposition of Gaussian-like functions convoluted with a Gaussian kernel to obtain a smooth function, visible in Fig.6.2.1. The value of this function is proportional to the distance to the closest obstacle. The robot has to travel along the minimum of the landscape.

The algorithm is provided with the relative distance of the target with respect to the robot but, the robot does not know where it is in a world frame. The total lack of knowledge of the target is too restrictive for the cases of interest and the relative distance helps in tricky situations. This assumption implies that the robot tends to point towards the target, avoiding getting away from it if unnecessary. If there were no obstacles the robot would head directly to the target without exploring the surrounding area saving time and resources. If there are obstacles, depending on the weight associated with this parameter, the trajectory may move away.

On the other hand, also landmarks have a very important role in the computation of the next step since they allow to keep track of the movements of the robot by saving areas already explored. A landmark represents the exploration factor and it is defined by the triplet $l = (\mathcal{F}_l, t_l, \mathbf{x}_l) \in \mathbb{SE}(2) \times \mathbb{R}^2 \times \mathbb{R}^2$, where:

- $\mathcal{F}_l \in \mathbb{SE}(2)$ is the last measurement frame when the landmark is created
- $t_l \in \mathbb{R}^2$ is the timestamp of the moment the landmark is created
- $\mathbf{x}_l \in \mathbb{R}^2$ is the position of the robot in the moment of the landmark creation, expressed in the frame $\mathcal{F}_l$

Landmarks are saved in a batch of limited size so that old and distant landmarks are substituted by new ones.

These points are associated with a cost, to penalize the repetition of the same steps and steps close to them. Landmarks are used both to penalize routes already seen and to highlight roads that lead to dead ends in order to explore new ways to reach the goal and avoid endless loops since the map is unknown. It could also be needed that older landmarks are weighted less than new ones to allow the exploration of the same area after a desired amount of time, with the scope of handling non-static scenarios. In fact, old and not practicable routes characterized by some obstacles, may be accessible at a different instant due to the temporal variation of the environment. At the generic time t, the exponential decay term is $e^{-\lambda(t-t_{il})}$, where $\lambda \geq 0$ is the tuning parameter acting as a time constant in the exponential decay.

Changing the weights associated with the two parameters, as will be analyzed in the next section, can give more importance to reaching the target or avoiding the repetition of the passages, leading to different behaviors of the robot during navigation. In addition, choosing the wrong weights can determine the failure of the algorithm, since it is not able to reach the target. The correct weights allow both reaching the target and achieving low computational cost. The latter can be obtained because the employment of the right values for these parameters shortens the time and distance that the robot needs to reach the target position.

The algorithm is based on a policy-switching method. NAPVIG allows to access the GVD, so the policies can be defined algorithmically, with a method based on the optimization of the cost function Eq.6.2.1. The method is based on six policies categorized into three classes: predictive (fully-exploitative, fully-explorative, partly explorative), reactive (legacy), and auxiliary (free-space, halt). Each policy plays a role in a specific moment of the computation, that depends on the current status of the robot and generates a trajectory to follow. We will not enter in finer details but it is important to note that the main navigation task is performed by the predictive policies.

The **fully-exploitative** policy is the one that generates a trajectory that points towards the target. The switch to other policies occurs when the target is approached or when all the generated trajectories are discarded (e.g. dead-ends). Actually, despite this policy could result in a very efficient navigation, the computed trajectory can be not safe or valid.

If any direct trajectory towards the target is not feasible, and hence the fully exploitative policy is rejected, the algorithm switches to the explorative policies. Both of them search the trajectory in 8 directions spanning the entire round angle.
The **fully-explorative** policy is used when the target is not in sight and the goal of the navigation task is to explore the map as much as possible. The cost function associated with each predicted trajectory penalizes those being close to visited areas.

For each point in the map, the penalty term is:

$$J_i \ : \ \mathbb{R}^2 \to \mathbb{R} \ : \ \mathbf{x} \mapsto \ J_i(x) := w_l e^{-\frac{\|x-x_{il}\|^2}{2\rho_l}} \tag{6.2.2}$$

where $w_l \in \mathbb{R}$ is the weight associated to the landmark and $\rho_l \in \mathbb{R}$ is a value tuning the peak radius of the Gaussian.
The total cost associated with each point in $\mathbb{R}^2$ is then the sum of the contributions of the penalty terms of each landmark, and the total cost associated with the predicted trajectory, is the sum of the cost of each sample, provided that the prediction ended without faults (TNF) or collisions (TCO):

$$J(\xi) = \begin{cases} +\infty, \quad if \ c_\xi = TCO \ \vee \ TNF \\ \sum_{K=0}^{|\xi|} \sum_{i=1}^{N_L} e^{-\lambda(t-t_{il})} J_i(\xi_k), \quad otherwise \end{cases} \tag{6.2.3}$$

If every predicted trajectory results in a collision/fault, the robot could not move if an external action does not happen.

The **partly-explorative** policy is used when the target is in sight and the goal is reaching the target in absence of a direct trajectory. It adds a term to the cost 6.2.3 that penalizes the distance to the target:

$$penalty(\xi) = \sum_{k=0}^{K_{max}} w_{target} \left\| \xi_k - \mathbf{x}_f \right\|^2, \tag{6.2.4}$$

where $w_{target} \in \mathbb{R}$ is a weight balancing the trade-off between exploration and exploitation. Essentially, the exploration term allows overcoming local minima-like situations by locally increasing the cost function.

The last three policies are used to extend navigation tasks or improve computational efficiency.
The legacy policy is used to follow the same direction in absence of bifurcations.

The free-space policy is needed because all the other policies output a point on the GVD, while the target can be any point in the free space. The algorithm switches to that policy when the robot is sufficiently close to the target so that the space in between could be considered safe. The robot can navigate in the free space without considering obstacles at all, for the short space that separates the robot from the target. The trajectory in this case is then directly the final point itself.

Finally, the halt policy is used when the robot needs to stop, for example when all the other policies have been rejected.

6.3 Simulation and improvements

The algorithm is tested in the Gazebo simulator, where different environments are presented to the robot. Gazebo is an open-source 3D robotics simulator that can model sensors that capture the simulated environment.
To satisfy the requirements the robot must avoid collisions and reach the target, which can be positioned at any point in the free space. The environment presents several difficult situations to overcome such as the presence of walls, obstacles and bifurcations. The objective is to reach the target by exploring the proposed area.

6.3.1 Simulated environments description

Simulations have been performed by changing the characteristics of the environment in which the robot navigates in order to explore different situations and behaviors of the algorithm. For this reason, four different maps have been created using different sizes and types of obstacles. Fig.6.3.1-6.3.4 report them.

Map 1

The first map is the simplest one. The main block of obstacles is concentrated in the middle of the available space, surrounded by a viable corridor. It is characterized by linear obstacles, representing walls and some simple dead-ends, easily recognizable by the robot's sensors and avoided in a short time.
Moreover, the target would be reachable regardless of where it would be placed.

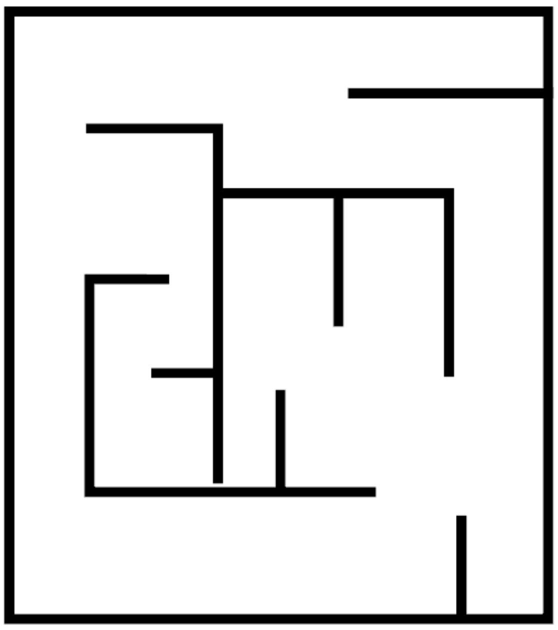


Figure 6.3.1: *First map.*

Map 2

The second map features linear obstacles and walls as the first map, with a more complicated configuration. Obstacles are organized in two main zones on the sides, connected by a central area with several separators.
On the right side of the map there is a long, dead-end spiral-shaped corridor. Finally, there are several tricky crossroads.

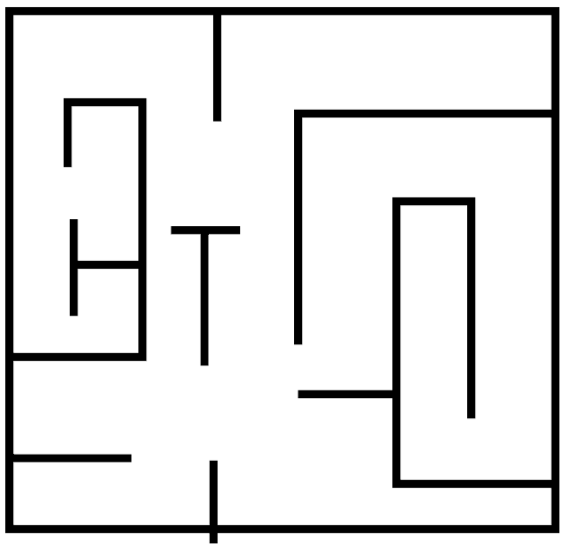


Figure 6.3.2: *Second map.*

Map 3

The third map presents both circular and linear obstacles. Circular obstacles are positioned in the central part of the map to divide the initial lower zone with the final upper one.
It is a simple configuration due to the presence of few obstacles. It has primarily designed to test the direction taken by the robot in the first steps.

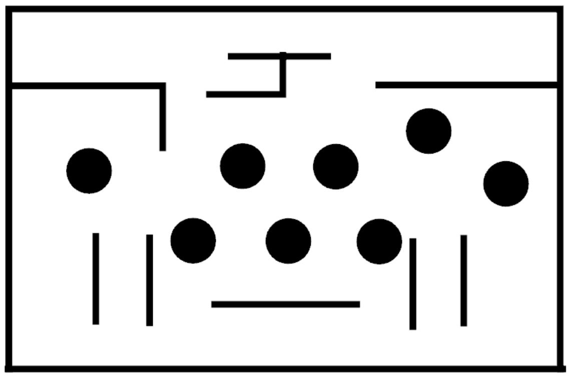


Figure 6.3.3: *Third map.*

Map 4

The fourth map is completely made of circular obstacles of different sizes, positioned in a regular pattern in which obstructions are staggered from row to row.
It simulated an environment populated by only pillars.

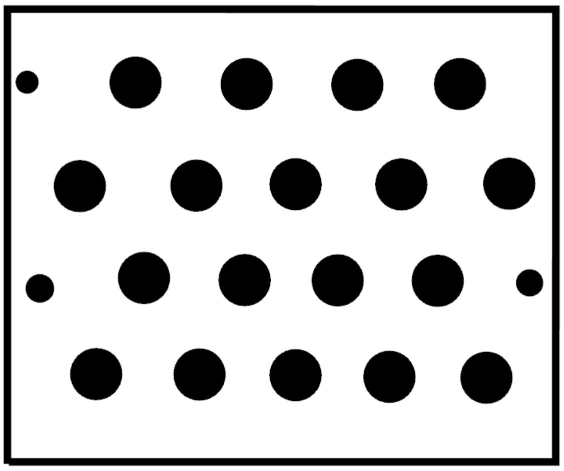


Figure 6.3.4: *Fourth map.*

6.3.2 Parameters tuning and results

To improve results, the following parameters have been modified over the course of several different simulations:

- weight of the landmarks
- weight of the target
- decay of the landmarks
- batch size

The first two in the list represent how much the decision of the next step is influenced by the landmarks, namely the already travelled points, and the target. This is because one would like to explore as much space as possible and not get stuck in the same area. Moreover, the target's weight is needed to avoid getting away from it if unnecessary.
The third one represents the exponent of the decay of the landmarks. More in detail, the first points of the queue are deleted since old, probably distant and limited space of the queue.
The last parameter describes the size of the queue of the saved landmarks.

Starting with the last parameters, batch size has been increased by 50 points with respect to the default value of 500. The reason relies on the fact that sometimes the robot repeats the same path even if there are other streets to visit. This happens because the first points saved in the queue are replaced by newer ones and the robot does not recognize them as already travelled. Additional saved points help in exploring different parts of the map.

As concerns weights and decay factors, several combinations of them have been tried. In the beginning, the three parameters has been changed one at a time, with the aim of discovering how a single parameter influences the overall simulation. What has been proved is that the weight of the landmarks must be at least two orders of magnitude greater than the one of the target. The reason is that with a lower ratio, the robot always tries to reach directly the target position. In this case, if this position is not directly accessible the robot ends in a loop, repeating the same steps all over again. In the cases in which the ratio is lower, lowering the decay factor has no effect on the simulation. Increasing this factor means reducing the time in which the landmarks are saved in the batch, counterproductive with the aim of keeping track of the travelled path for a period long enough.
Starting from the default values of the parameters, have been chosen six values for the weight of the landmarks, six for the one associated with the target, and five for the decay factor.
After having found the minimal ratio that leads to efficient navigation, several combinations of these three parameters have been discarded, since already knew that would lead to inefficient behaviors of the robot, such as failing to reach the target.

Different combinations of values have been tried on four different maps, previously described, to study the performance of the algorithm in different situations.
To obtain an objective evaluation of the performance of the algorithm, measures like time elapsed and travelled distance to reach the target have been taken into consideration with the use of two counters. By comparing the obtained values of the time and distance, for different input parameters and varying the initial and final positions, the best values for the before mentioned parameters have been found. The lower the values returned by the two counters, the better is the combination of the parameters.

Two different starting points and targets have been selected to evaluate if any change in these values brings changes in the simulation performance.

The major result that has been found is that a combination of values optimal for every environment and position does not exist. Instead, the rule of the order of magnitude has been proved to be general and valid for all types of environments and each chosen position both for target and initial condition. Moreover, also the decay factor has a very important role in fact, in the majority of the cases only the value of 0.001 and in some cases 0.01 are found to be valid. Other values lead to longer computational time to reach convergence.

Since the presence of disturbs both in sensing and tracking, simulations performed with the same settings may give different results, whenever tricky situations appear such as bifurcations or crossroads, as shown in the graphs below.
The landscape is computed through the Monte Carlo method, which introduces variability and approximation. Even a small approximation in the computations can lead to a change in trajectories at different times.
A slight movement from the Voronoi also can lead to a different choice of the next step.

Given that the algorithm has been previously tested and it has shown to be fully functioning, in the following are reported only limit case situations in which it has encountered some minor difficulties and suboptimal solutions.
The goal of these tests is to analyze in detail what happens when the algorithm encounters a situation in which it has to choose between more feasible trajectories.

Fig.6.3.5 illustrates the situation previously described encountered in the first map. For these simulations the parameters are chosen to be:

- landmarks' weight = 100
- target's weight = 0.1
- decay constant = 0.001

while the initial position is set to (x,y)=(1,1) and target to (x,y)=(-2,-2).

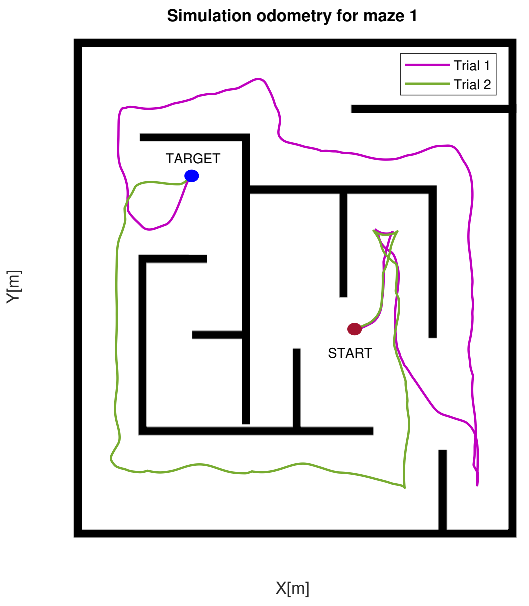


Figure 6.3.5: *Trajectories executed during a simulation in the first map performed with the same configuration.*

From the starting point, marked in red on the right side of the figure, the robot moves up encountering a blockage. Then it heads downwards finding a bifurcation that leads to different trajectories in the two simulations.
Nevertheless the application of the same values of the parameters, the algorithm performs different choices in the same situation on the basis of the cost function, leading to a shorter path for the green trajectory. Hence the performance of the algorithm to solve a determined navigation problem does not only depend on the chosen parameters but also on the small variations due to approximations as described earlier.

Fig.6.3.6 illustrates the situation previously described encountered in the second map.
For these simulations the parameters are chosen to be:

- landmarks' weight = 1
- target's weight = 0.01
- decay constant = 0.001

while the initial position is set to (x,y)=(1,1) and target to (x,y)=(-3.5,-1).

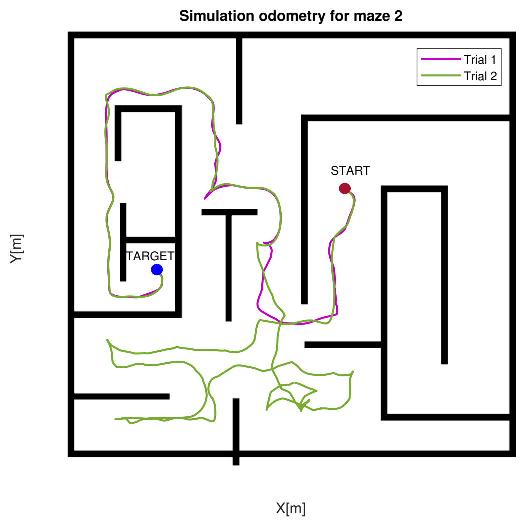


Figure 6.3.6: *Trajectories executed during a simulation in the second map performed with the same configuration.*

As concern maze number 3, two simulations with the same parameters lead to completely different results, even from the beginning. A possible reason is a slightly different value of the cost of the initial trajectories. Since the map is almost symmetric, the two trajectories lead to an almost symmetrical pattern.

Fig.6.3.7 illustrates the situation previously described encountered in the third map. For these simulations the parameters are chosen to be:

- landmarks' weight = 10
- target's weight = 0.04
- decay constant = 0.001

while the initial position is set to (x,y)=(0,-3) and target to (x,y)=(0,3.5).

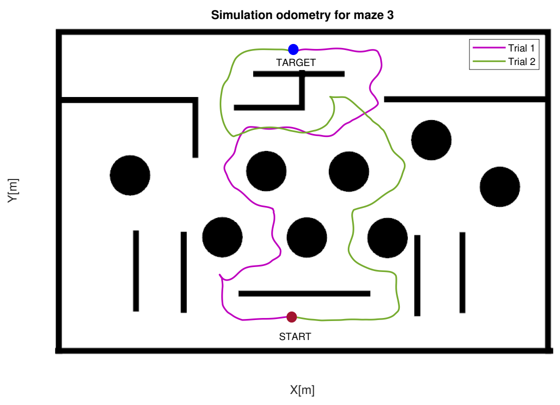


Figure 6.3.7: *Trajectories executed during a simulation in the third map performed with the same configuration.*

As concerns green trajectory, notice that in the final part of the simulation, the algorithm leads the robot to reach directly the target, finding an obstruction ahead. The shortest path could be the one represented by the pink trajectory. The choice of heading to the left side of the map is due to the cost associated with the landmarks which penalizes the possibility of repeating the same steps to overcoming the obstacle.

Due to the particular configuration of the fourth environment, the simulations give always the same results. Anyway, it is a particular case.
Fig.6.3.8 illustrates the situation previously described encountered in the fourth map.
For these simulations the parameters are chosen to be:

- landmarks' weight = 10
- target's weight = 0.4
- decay constant = 0.001

while the initial position is set to (x,y)=(-1,-4.5) and target to (x,y)=(4,3).

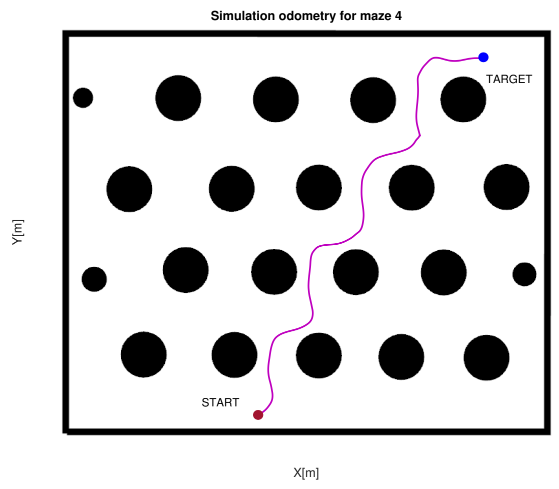
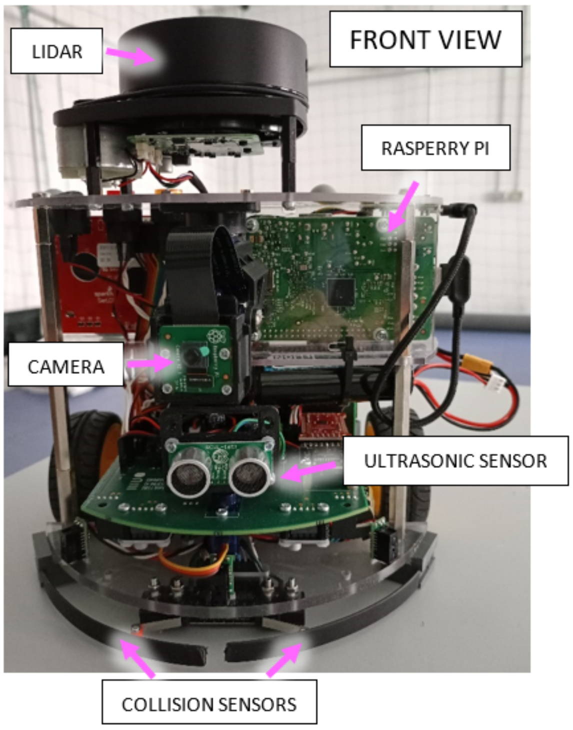


Figure 6.3.8: *Trajectory executed during a simulation in the fourth map.*

Notice that the light-blue graph assumes values close to zero at points in which the algorithm makes the robot turn to the left. This situation in the previous environments led to different choices of paths in distinct simulations. In this case, instead, the particular configuration of the obstacles and the slight offset of the starting position make the algorithm prefer to stay on the left side of the obstacle in every simulation.

6.4 Real environment tests

After numerous simulations performed on Gazebo simulator, NAPVIG algorithm is tested on a real customizable unmanned ground vehicle (UGV), as described in Chapter 3, provided with several sensors. Between all, should be noted the presence of a Lidar, an RGB camera, ultrasonic, infrared, collisions and proximity sensors. Finally, VICON markers are mounted to connect the robot with the laboratory.
In particular, the robot we take into consideration is a differential drive robot (DDR) characterized by right and left motors, a stabilizer, and the IMU to measure and report force, angular rate, and the body's orientation. Fig.6.4.1 and Fig.6.4.2 report the involved robot from four different perspectives.



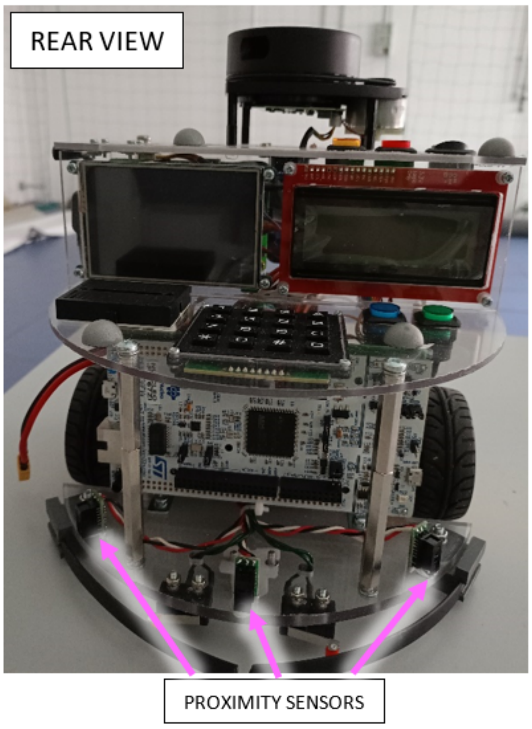
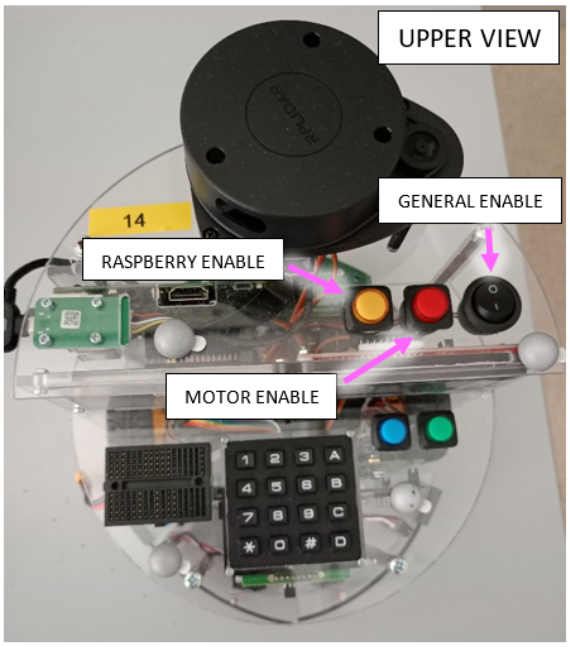


Figure 6.4.1: *Real robot front and rear views.*



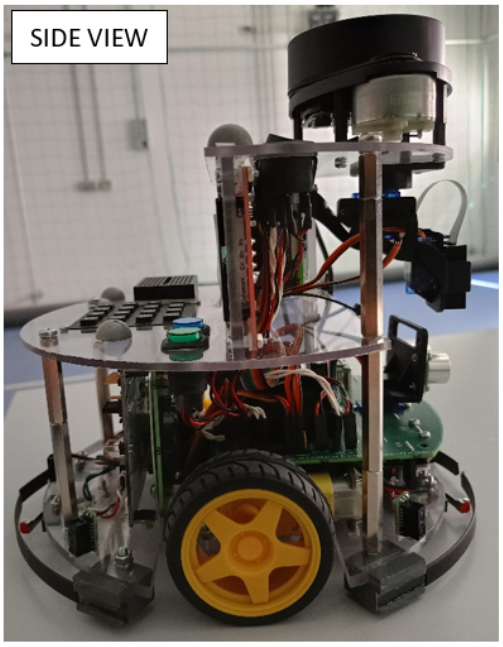


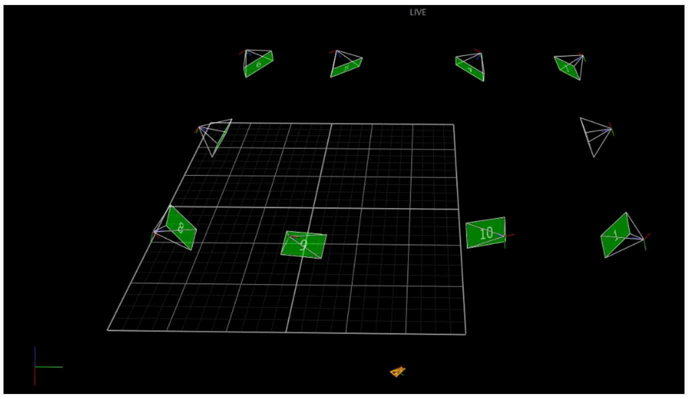
Figure 6.4.2: *Real robot upper and lateral views.*

6.4.1 Experiments

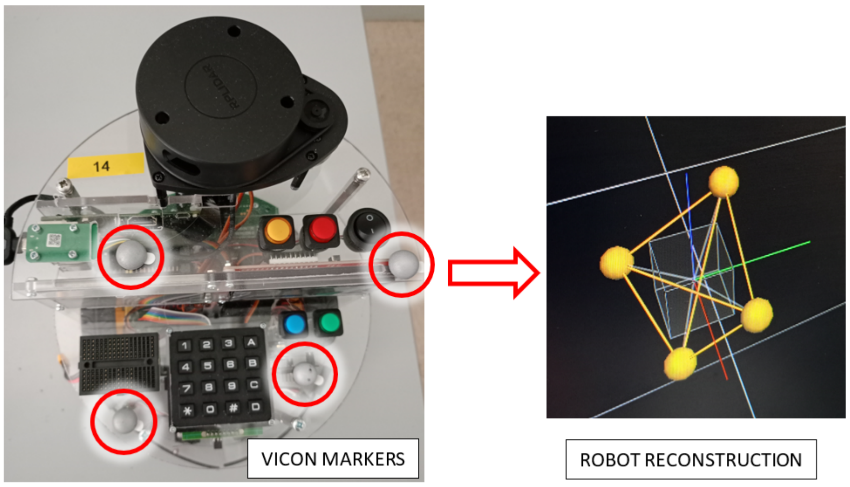
To keep track of the movements of the robot it has been employed the Vicon motion capture system. Motion capture is the process of recording the movements of objects or people, sampled many times per second. The purpose of motion capture is to record only an object's movements and not its visual appearance.

The Vicon system is composed of ten cameras that work together to follow the robot during the simulations and record its position as a function of time. The placement of these cameras is shown in Fig.6.4.3a.

To follow the robot during navigation, the Vicon system needs four reflector points (half-spheres visible from Fig.6.4.3b) positioned in clearly visible spots of the robot's bodywork without any particular pattern. If these markers are set in a symmetrical way the Vicon system may misunderstand the orientation of the robot failing the reconstruction.



(a) Vicon environment



(b) Robot in the Vicon system

Figure 6.4.3: *Vicon system environment.*

If these markers are positioned out of sight, Vicon cameras fail to recognize the robot, and then the construction of a virtual object, correspondent to the robot, is not possible. The same can also occur if robot markers are shielded by obstacles.

6.4.2 First test

The first test case has been ideated to verify the correctness of the algorithm and to ensure that the robot avoids positioned obstacles and can reach the target goal, as proved in simulations. To further test the algorithm, obstacles have been moved during navigation to simulate a dynamic environment. Some of them have been also suddenly placed in front of the robot.

To make the first test simple configurations of obstacles have been used as shown in Fig.6.4.4, where some cardboard boxes were set to form a *1* geometry.

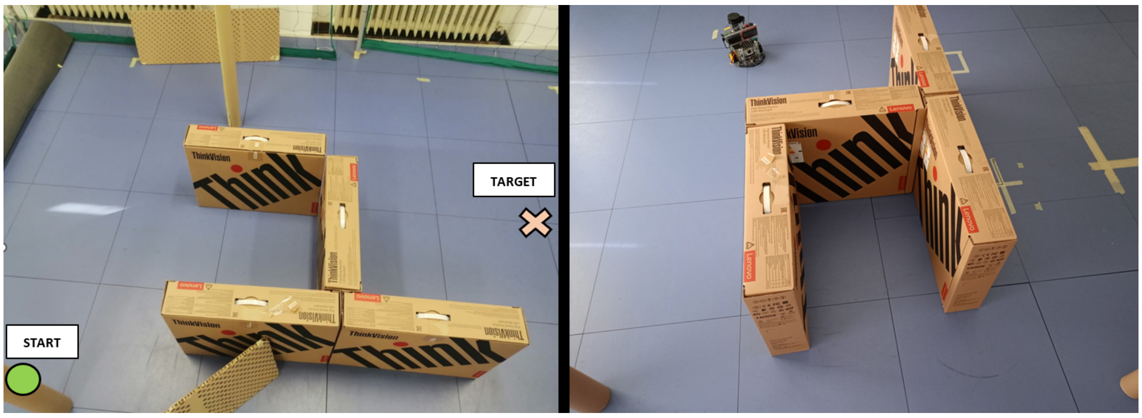


Figure 6.4.4: *First environment for the experiments.*

In Fig.6.4.5 is plotted the odometry recorded by the Vicon system during the first experiment. As expected, the Napvig algorithm tries to reach directly the target as reported on the upper part of the odometry graph. As soon as the robot recognizes an obstruction, goes back and takes a deviation around the boxes.

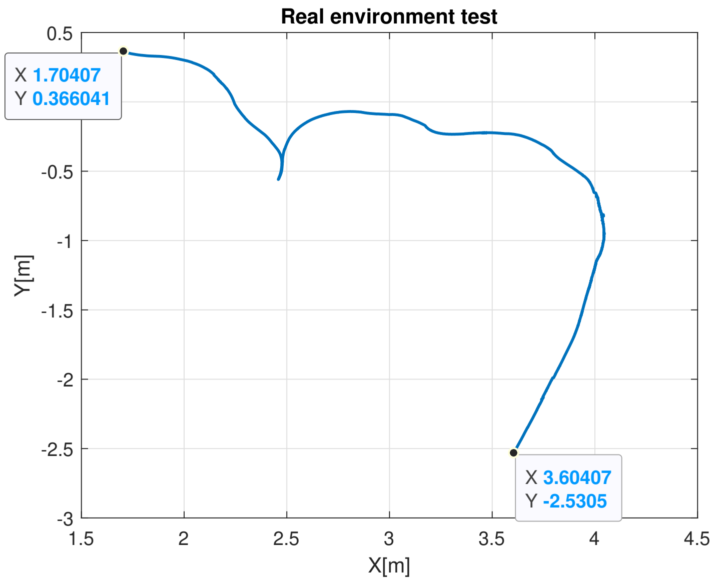


Figure 6.4.5: *Odometry graph for the first environment.*

6.4.3 Second test

The second test (Fig.6.4.6) is intended to verify the functioning of the algorithm along narrow passages. Differently from the first test, the boxes have been positioned at a distance that permits the robot to pass between them. A cylindrical obstacle has been placed inside the created corridor, to simulate a column (see Sec.6.3 for reference).
The starting point has been set in proximity to the entrance of the corridor, while the target lies outside on the right side.

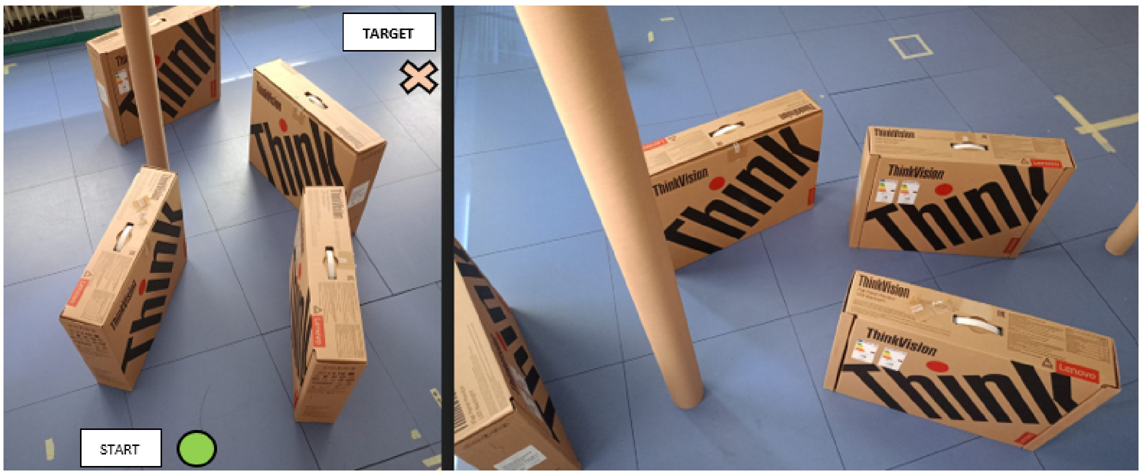


Figure 6.4.6: *Second environment for the experiments.*

Fig.6.4.7 shows the odometry, registered by the Vicon system, during the second experiment. As expected, the Napvig algorithm is capable of calculating a trajectory even in narrow passages as long as the space perceived by the Lidar sensor is enough.

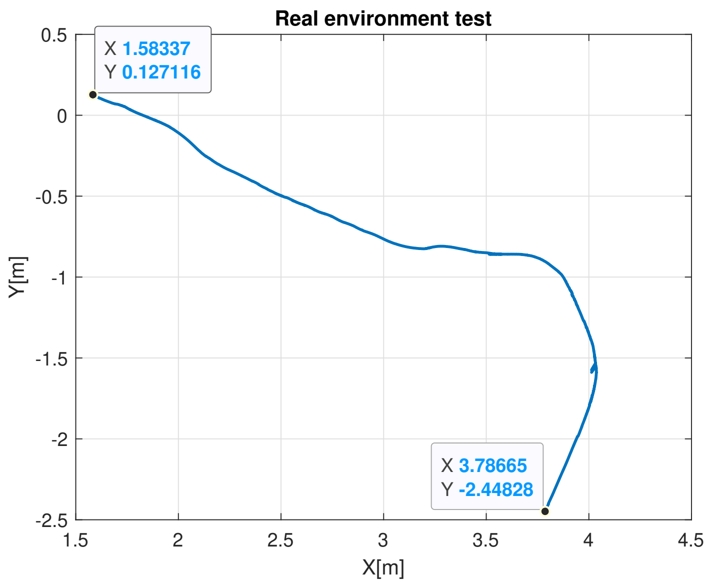


Figure 6.4.7: *Odometry graph for the second environment.*

6.4.4 Third test

The third test case has been developed to recreate a difficult situation for the algorithm. In the underlying situation, reported in Fig.6.4.8, the robot is in front of a central obstacle and it has to decide by which side to circumvent it.

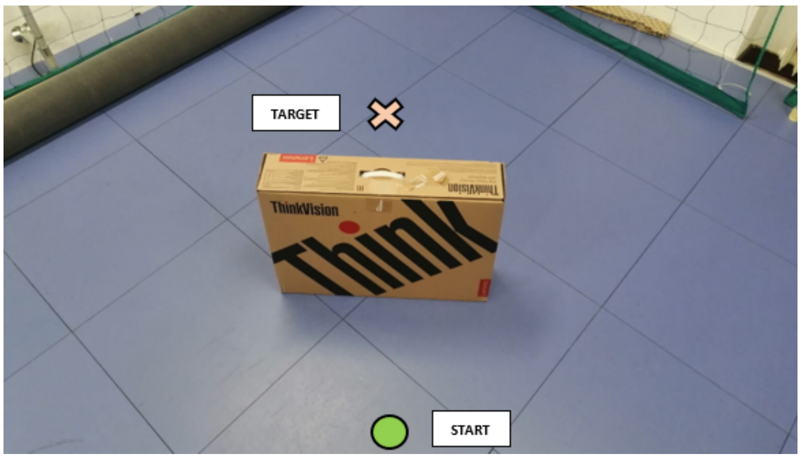
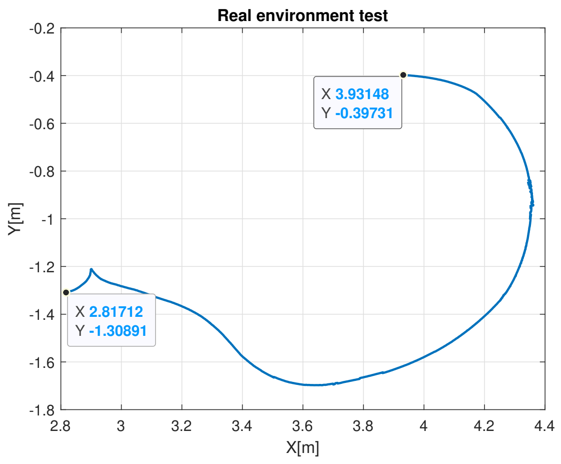
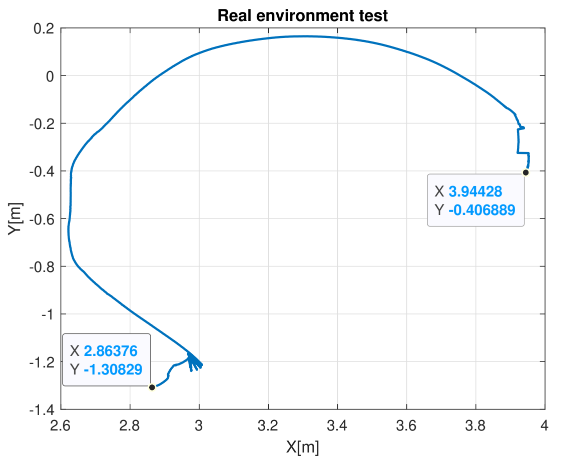


Figure 6.4.8: *Third environment for experiments.*

As expected (see Sec.6.3), since the obstacle and the environment are almost symmetric the algorithm sometimes takes the left path while other times takes the right path. A possible reason for this behavior could be due to an almost similar cost for each of the two trajectories, at least in the first part of the navigation. Fig.6.4.9 shows the odometry of two different experiments in the same environment.



(a) Right path



(b) Left path

Figure 6.4.9: *Odometry graph for the two different paths of the third environment.*

Fig.6.4.10 and Fig.6.4.11 show the behavior of the difference between the angles and costs of the two cheapest computed trajectories. In particular, the pink one represents the difference of angle, in radians, between the two cheapest calculated trajectories. If the value is small the two trajectories lead to the same next step on the Voronoi diagram. Instead, if the value is close to π then the two trajectories lead to opposite directions, such as in front of a wall.
As concerned with the light-blue graphs, they illustrate the trend of the difference

between the costs of the two cheapest calculated trajectories. When the values are small the trajectories have a similar cost value, and hence the choice of one with respect to the other is more affected by approximations.

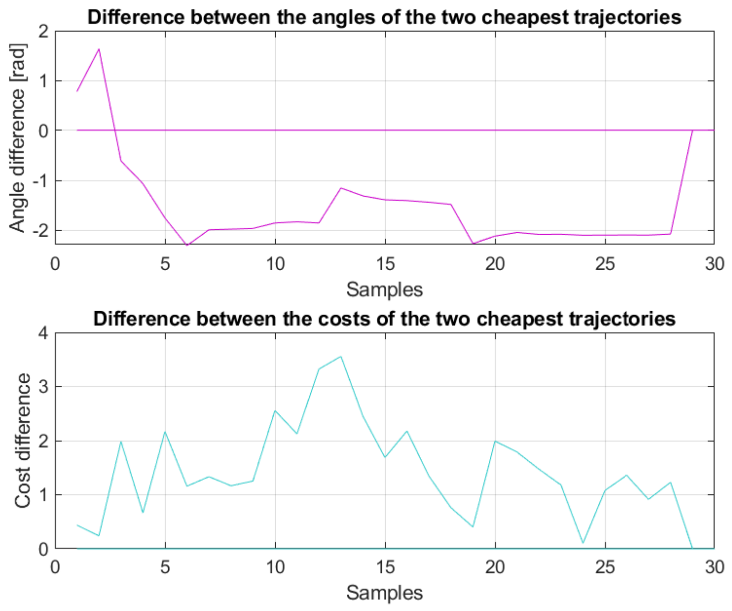


Figure 6.4.10: *Difference between the angles and costs of the two cheapest trajectories for the right path of the third environment.*

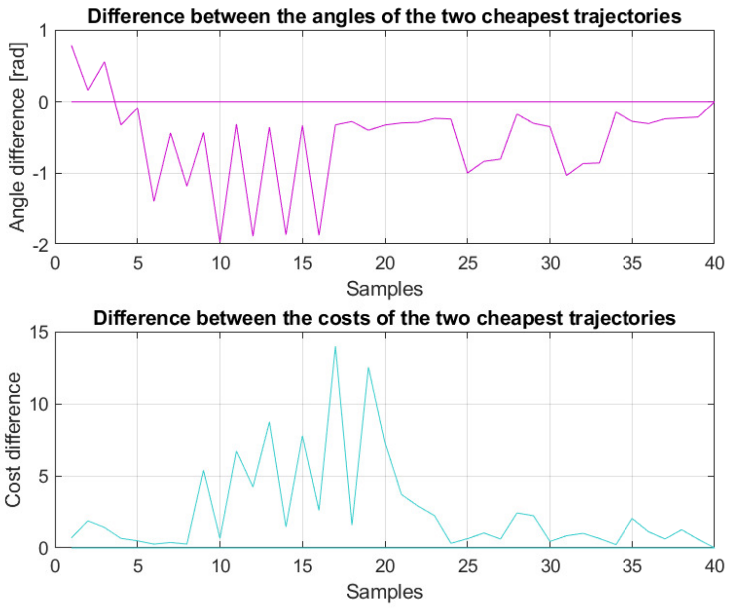


Figure 6.4.11: *Difference between the angles and costs of the two cheapest trajectories for the left path of the third environment.*

If the difference in cost is close to zero and the difference in the direction is significant then it can happen, as in this case, that during various simulations the algorithm may take different decisions.

Regarding Fig.6.4.11, notice the oscillations in the sample interval between 5 and 20. The rapid change in angle difference is related to the rapid change in costs difference, indeed the negative peaks of the pink graph correspond to points in which the cost difference is small. This means that the two most probable trajectories head in very different directions (2rad) and at the same time have a similar cost. This situation probably corresponds in Fig.6.4.9b to the point in which are visible several small oscillations (0.05m). Here the robot takes a route but after small travel it changes completely its path. After several indecisions, the algorithm chooses to overcome the obstacle on the left side.
Finally, it can be noticed that between the first and the second attempts, there is a small difference in the location of the starting point that could have an impact on the decision-making process of the algorithm.

6.5 Results and future works

In this section are presented some suggestions about potential future developments. Future improvements rely on the possibility of computing trajectories in advance with respect to the actual state. There is the possibility of calculating the costs of the valid trajectories two or three steps ahead, improving path selection and avoiding getting stuck in dead-ends and hence revisiting the same points backwards.

Another important aspect could be the optimization of self-localization, with the implementation of internal odometry with dedicated compensation of relative errors removing the dependency from the Vicon system. This refinement could allow the execution of outdoor or indoor experimental tests where motion capture systems are not installable.

Finally, it is necessary to perform several tests with the goal of collecting numerical data to deeply understand the phenomena involved in the functioning of the NAPVIG algorithm, which are described only in a qualitative manner in this thesis because of the lack of data.

Bibliography

[1] Robin R. Murphy. *Introduction to AI Robotics*. 2019.

[2] Jenay M Beer, Arthur D Fisk, and Wendy A Rogers. “Toward a framework for levels of robot autonomy in human-robot interaction”. In: *Journal of human-robot interaction* 3.2 (2014), p. 74.

[3] 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).

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

[5] Halliday and Resnick. *Fundamentals of Physics Extended, 10th Edition*. Wiley, 2014.

[6] Ravi Raj and Andrzej Kos. “A Comprehensive Study of Mobile Robot: History, Developments, Applications, and Future Research Perspectives”. In: *Applied Sciences* 12 (2022). DOI: 10.3390/app12146951.

[7] G.N. Desouza and A.C. Kak. “Vision for mobile robot navigation: a survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.2 (2002).

[8] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. “Localization strategies for autonomous mobile robots: A review”. In: *Journal of King Saud University - Computer and Information Sciences* 34.8, Part B (2022).

[9] Karthik Karur et al. “A survey of path planning algorithms for mobile robots”. In: *Vehicles* 3.3 (2021), pp. 448–468.

[10] Chengmin Zhou, Bingding Huang, and Pasi Fränti. “A review of motion planning algorithms for intelligent robots”. In: *Journal of Intelligent Manufacturing* (2021), pp. 1–38.

[11] Luca Battistella Nicola Lissandrini, Giulia Michieletto Markus Ryll, and Angelo Cenedese. “NAPVIG: Local Generalized Voronoi Approximation for Reactive Navigation in Unknown and Dynamic Environments”. In: *American Control Conference* (2023).