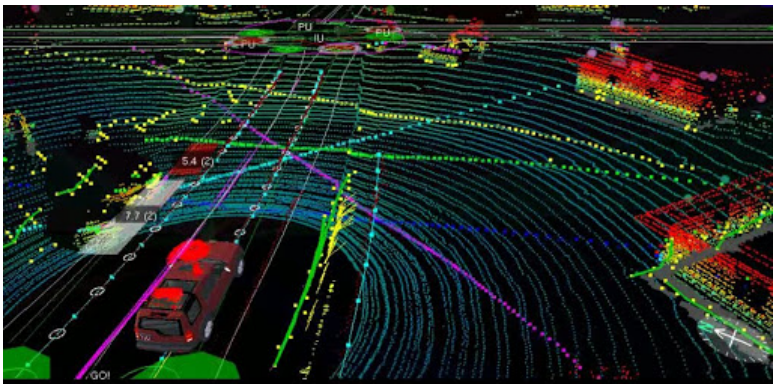
TÉCNICO LISBOA



RADAR vs LIDAR for obstacle detection and collision avoidance

Tiago Filipe Félix de Andrade

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. Alexandra Bento Moutinho

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Alexandra Bento Moutinho
Member of the Committee: Prof. Mário António da Silva Neves Ramalho

October 2020

Acknowledgments

I wish to show my gratitude to everyone directly and indirectly involved in the development of this thesis.

Firstly, I would like to express my sincere gratitude to Prof. Alexandra Moutinho for her words of encouragement and guidance in the most crucial moments of this work.

I thank my family for all the unconditional support in all the highs and lows of this work.

I also thank my lab mates João Carreira, Guilherme Kano, Rui Assis, Mário Torres, Daniel Leite, João Oliveira and Miguel Ramalho for all the support and unforgettable moments shared during this period of our lives.

I extend my gratitude to Eng. Camilo Christo and Mr. Luís Raposeiro for all the time spent and technical assistance involved in this thesis.

Resumo

A área de veículos autónomos é um campo científico dinâmico e em constante crescimento cuja popularidade tem vindo a crescer recentemente. A investigação nesta área tem-se focado no desenvolvimento de sistemas de sensores cada vez mais precisos e versáteis para integração em algoritmos robustos de navegação autónoma, culminando em veículos altamente autónomos. A diversidade de sensores é alta, havendo um elevado número de sensores capazes de adquirir diversos tipos de dados do ambiente úteis para navegação autónoma.

Esta dissertação tem como foco dois dos sensores mais utilizados na área de veículos autónomos, LIDAR e RADAR Doppler, explorando a tecnologia por trás destes sistemas. O principal objectivo deste trabalho é o desenvolvimento de um sistema de detecção de obstáculos e prevenção de colisões incorporando estes sensores e a sua implementação num robô de pequena escala capaz de navegar ambientes desconhecidos autonomamente com a finalidade de atingir um ou vários pontos de referência. É desenvolvido um protótipo real, assim como o seu correspondente virtual para simular o seu comportamento em ambientes controlados e sem riscos de danos, destinado a testes de segurança.

A modularidade é um factor significativo deste trabalho, com o intuito de facilitar a integração deste protótipo em qualquer rede Wi-Fi. As capacidades da plataforma ROS são exploradas com a sua integração no sistema.

Numa fase final, o sistema é testado num conjunto de cenários, avaliando o desempenho de cada um dos sensores em separado bem como a sua integração (através da fusão sensorial) em ambientes estáticos e dinâmicos.

Palavras-chave: Veículos autónomos; Sensores; LIDAR; RADAR Doppler; Deteção de obstáculos; Prevenção de colisão; Fusão sensorial.

Abstract

The field of autonomous vehicles is a dynamic and ever-growing field of research which has seen a rise in popularity in recent years. Research in this area has mostly shifted towards development of increasingly accurate and versatile sensing systems to be integrated in robust autonomous navigation algorithms, culminating in highly autonomous vehicles. Diversity in the field of sensing is significant, with a high number of developed sensing methods to acquire a variety of environment data used in autonomous navigation.

This thesis focuses on two of the most commonly used sensors, LIDAR and Doppler RADAR, exploring the technology behind these devices. The main goal of this work is the development of an obstacle detection and collision avoidance system incorporating these sensors and its implementation in a small-scale robot capable of autonomously navigating unknown environments to reach a single or multiple waypoints. A physical prototype is developed as well as its virtual counterpart to simulate its behaviour in a controlled, risk-free environment intended for prior testing.

Modularity is a significant factor in this work, with the intent of easing the integration of the prototype in any Wi-Fi network. The capabilities of the ROS framework are explored with its integration in the assembly.

Finally, the sensing system is tested in a set of trials, evaluating the performance of the standalone sensors as well as its coupling (through sensor fusion) in static and dynamic environments.

Keywords: Autonomous vehicles; Sensors; LIDAR; Doppler RADAR; Obstacle detection; Collision avoidance; ROS; Sensor fusion.

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

$RADAR$	Radio Detection And Ranging
$LIDAR$	Light Detection And Ranging
ROS	Robot Operating System
$MDVO$	Minimal Deviation Velocity Obstacles
SVO	Standard Velocity Obstacles

Greek symbols

λ	Wavelength.
ϕ	Orientation [rad]
ψ	Angular position [rad]
θ	Angle, angular [rad]

Roman symbols

c	Speed of light [km/s]
d	Range [m]
i	Current [A]
r	Radius [m]
t	Time [s]
f	Frequency [Hz]
d	Distance [m]
u, v, w	Velocity Cartesian components [m/s]

Subscripts

max	Maximum.

x, y, z Cartesian components.

Superscripts

c Command.

Chapter 1

Introduction

Autonomous automation is a dynamic and ever-growing field of research across several areas of industry as well as academia, which has seen a rise in popularity in recent years [1].

Autonomous cars have been around for almost a century, dating as far back as the 1900s with the creation of the first radio-controlled vehicles. The definition of an autonomous vehicle was not as restrictive as in the present days, with driverless remote-controlled cars being considered autonomous vehicles at the time, as opposed to the computer-driven driverless cars that are considered the very definition of an autonomous vehicle in the present. Despite the need for an operator to operate the remote controls of these driverless ”phantom-cars”, which were not self-driving per se, they did not fail to amaze and captivate the public, which strengthened the belief of a bright future in this area and opened the door for further improvements to these systems. When these vehicles first emerged, one of the main aspects the public and the scientific communities were focused on was the possibility of improving the safety at a time when cars were deadly, which still holds true to this day.

Although the idea of an automated vehicle that is capable of performing various tasks automatically and never making wrong decisions that could cost the lives of people involved is enticing, this topic is met with a fair share of scepticism from the general public. As these systems are rapidly improving and converging to truly automated systems, shifting the whole environment data collection and decision making processes to a computer is an unsettling idea to some, since their safety would essentially be placed in the hands of a computer, which may be susceptible to inaccuracies, errors or even failure. To tackle this problem, researchers are emphasizing the minimization of any mistakes these systems may incur. This translates into improving the methods and hardware used in acquiring information about the vehicle’s surroundings, which is of great importance in the decision making process.

Many are the organisations involved in research of new methods and algorithms to develop full-fledged automated vehicles, as evidenced, for example, by the VisLab Intercontinental Autonomous Challenge (VIAC) sponsored by the European Research Council [1] and the series of competitions sponsored by the Defense Advanced Research Project Agency (DARPA) belonging to the United States of America Department of Defense [2].

Researchers work with state-of-the-art technology to create commercial and industrial autonomous

vehicles with a very specific goal in mind: creating machines that can unequivocally emulate the behaviour of a human driver, thus eliminating the need of a person to be operating said vehicle. Such systems must be supplied with accurate readings of the vehicle's surroundings to ensure the safety of its operation. With this in mind, several methods and algorithms have been developed over the years to act as a bridge between the system's processing unit and the outside world, providing relevant real-time data and identifying any potential collision threats. These are continuously subjected to rigorous tests so as to ensure they are fool-proof and will not fail in any circumstances.

1.1 Historical Background

The first record of a driverless vehicle dates all the way back to the 1901-1904 period. During this time, Spanish inventor Leonardo Torres Quevedo started to develop the idea of remote control applied to vehicles as a way to test his prototypes without risking any passengers' lives [3]. This concept, appropriately named Telekine (from the greek *tele*, far away, and *kine*, movement), was first applied to an aquatic vehicle resembling a submarine, as shown in figure 1.1. A year later, this concept would be applied to a simple tricycle with remarkable results, being able to maneuver the vehicle from up to roughly 30 meters. This technology was later applied to airborne vehicles and, as a demonstration to the Royal Country House (*Real Casa de Campo*), on a *dinghy* boat with a crew of eight, achieving operating distances of up to roughly 2 km. However, the development of this technology ceased when Quevedo's application for a project on submarine teleoperated torpedoes was denied.

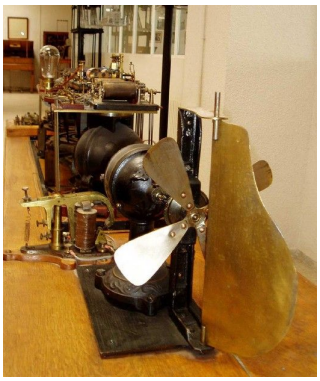


Figure 1.1: Telekine[1]

The advent of autonomous vehicles however dates all the way back to the 1920s with the appearance of the first "driverless" cars. In 1921, the world witnessed the first prototype of a driverless vehicle developed by the American military. Far from the likeness of any commercial car, the vehicle resembled a coffin, with three wheels attached for motion in a tricycle configuration, as shown in figure 1.2. Utilizing the same technology as the Telekine, the tricycle was radio-controlled but its use was limited to the transportation of light loads.

[1] http://cyberneticzoo.com/early-robot-enabling-technologies/1902-telekine-telekino-leonardo-torres-quevedo-spanish/
[2] http://blogs.discovermagazine.com/d-brief/2017/12/13/driverless-car-houdina-houdini/#.XK3lRehKhEY

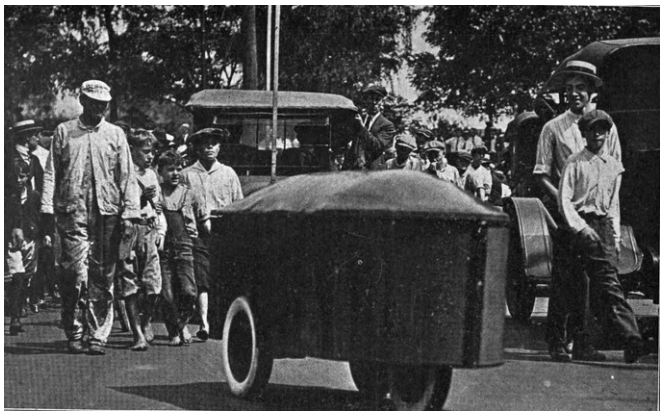


Figure 1.2: American military prototype of radio-controlled vehicle[2]

The unofficial driverless car era began in 1926 with the *Linriccan Wonder*, a radio-controlled car demonstrated in New York by Houdina Radio Control Co., a radio equipment firm [1]. The model was adapted from an already-existing car model, the *1926 Chandler*, as shown in figure 1.3, hence why it is unofficially considered the first driverless car. It consisted in the base vehicle with a receiving antennae mounted on its rear compartment connected to circuit-breakers which operated small electric motors that directed the car's movements.

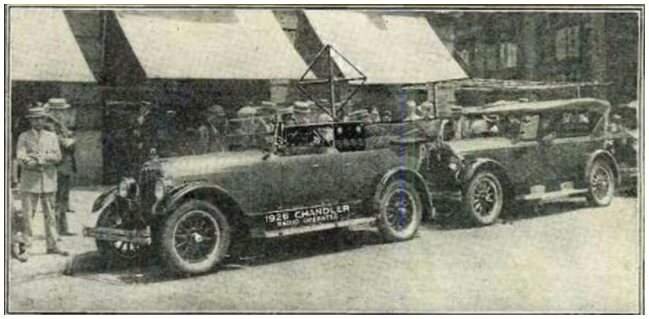


Figure 1.3: Houdina's *Linriccan Wonder*[2]

The antennae received radio impulse signals from a vehicle following the first, which would transmit the operator's inputs into the radio-controlled car. A modified version of this model was presented by Achen Motors under the name *Phantom Auto* in 1926.

An interesting new approach came in the 1950s-1960s with General Motors' collaboration with RCA Labs and later taking over the project, where the focus of the steering signals sent to the vehicle was shifted to the roadway as opposed to a second vehicle. Detector circuits buried in the roadway were able to detect the presence and velocity of any metallic cars moving on its surface. These would essentially act as signal transmitters, simulating automatic steering, accelerating and braking control on a pair of prototype vehicles equipped with radio receivers and audible and visual warning devices, represented in figure 1.4. This technology was later modified by United Kingdom's Transport and Road Research Laboratory, incorporating magnetic cables in the roadways to improve detectability. Vehicles were then able to switch between lanes (each with their own set of magnetic cables) and follow them for long

distances, achieving speeds of up to 130 km/h without deviating from their intended course.

Figure 1.4: GM's self-driving car[3]

It was not until much later, in the 1980s, that the first self-sufficient and fully autonomous vehicles were created. In a joint project with DARPA, the Environmental Research Institute of Michigan, University of Maryland, Martin Marietta and SRI International, the first autonomous vehicle equipped with sensing technology was created. The prototype, Autonomous Land Vehicle (ALV, represented in figure 1.5), utilized Computer Vision algorithms coupled with a Light Detection And Ranging sensor (LIDAR) to feed an obstacle detection algorithm. Although the first of its kind, it achieved autonomous driving with no human intervention at speeds up to 31 km/h in free roads.

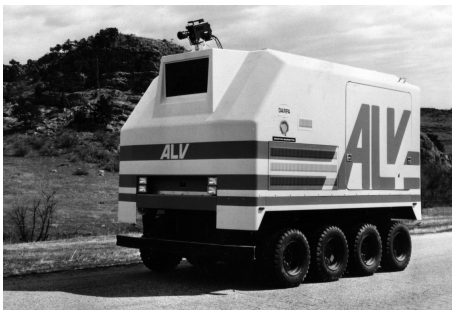


Figure 1.5: ALV project's prototype[4]

Afterwards, several joint projects between research companies and universities were created, contributing to the rapid development of the subject with prototypes that achieved remarkable levels of autonomy. Most notably, Ernst Dickmanns of Bundeswehr University Munich in conjunction with Mercedes-Benz, developed a modified version of an S-Class Mercedes-Benz model equipped with computer vision algorithms applied to a camera feed and microprocessors with integral memory to react almost instantly to external disturbances [4]. The vehicle, represented in figure 1.6, boasted a 95% autonomous driving factor in a 1590 km drive between Munich, in Germany and Copenhagen, in Denmark, reaching speeds of up to 175 km/h in the German *Autobahn* and being able to autonomously drive in free lanes, overtake

[3]https://spectrum.ieee.org/tech-history/heroic-failures/selfdriving-cars-were-just-around-the-cornerin-1960
[4]https://paleofuture.gizmodo.com/darpa-tried-to-build-skynet-in-the-1980s-1451000652

other cars and convoy driving.

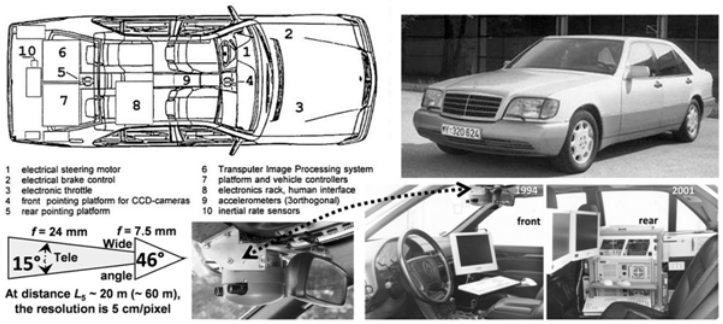


Figure 1.6: Ernst Dickmanns' self-driving Mercedes-Benz[5]

Nowadays, a few select commercial car manufacturers are in the front line of development of autonomous vehicles. Tesla Inc., an American car manufacturer known for their revolutionary take in the field of environmental-friendly cars with top-of-the-line fully electricity-powered cars, has released a series of car models with an autopilot mode. This feature relies on Computer Vision algorithms, RADAR (Radio Detection And Ranging) and ultrasonic sensors as environment data collection methods to perform obstacle detection and collision avoidance, as shown in figure 1.7.

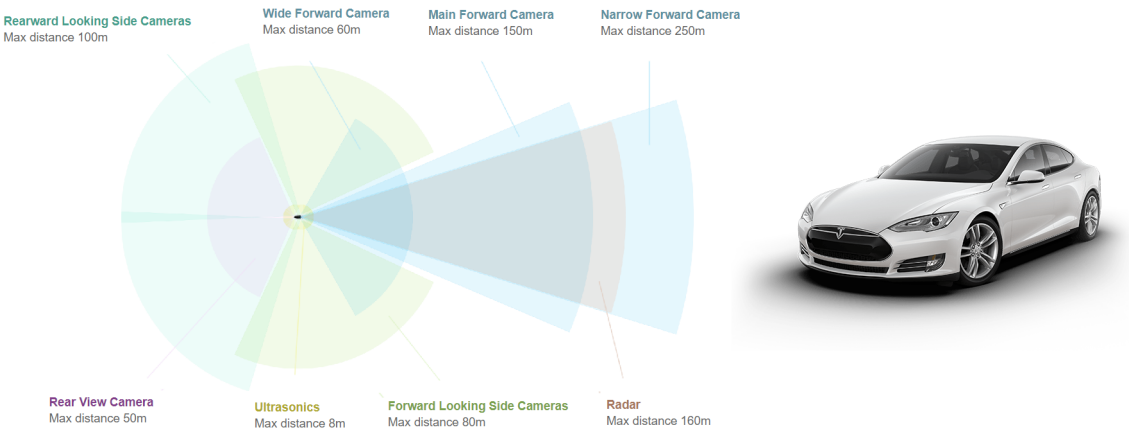


Figure 1.7: Tesla's Model X sensing system[6]

Although not as publicized as commercial cars, autonomous vehicles have played a major role in the industry sector from transportation to assembly lines. Keller und Knappich Augsburg, also known as KUKA Robotics, is one of the most world-renowned manufacturer of industrial robots and solutions for factory automation who is closely related to autonomous systems in industry. After the development of numerous collaborative automated robots such as robotic arms and AGVs (Automated Guided Vehicles) equipped with obstacle detection technology, KUKA recently released the OmniMove (Figure 1.8), their take on heavy load autonomous transportation. The OmniMove is a versatile vehicle capable

[5]https://www.lifehacker.com.au/2016/02/creator-of-the-worlds-first-self-driving-cars-ernst-dickmanns/

[6]adapted from https://www.tesla.com/en_GB/autopilot?redirect=no and http://www.stickpng.com/img/transport/cars/tesla/tesla-model-s

of performing SLAM (Simultaneous Localization And Mapping) using several sensing and positioning methods such as LIDAR, indoor GPS (Global Positioning System), magnetic grids, inductive and optical guidance.

Figure 1.8: KUKA's OmniMove[7]

1.2 Motivation

Mechatronics systems in general have been burdened with human dependence when operating, although to a limited degree, due to the unpredictable nature of the environment they are operating in.

With the current trend of lowering costs of sensory components and processing units, coupled with the rapid development of hardware and software integrated in these sensing systems, autonomous systems have seen an increase in popularity across the manufacturing industry as well as services and domestic applications. Nowadays, software implementations of obstacle detection algorithms are quite advanced and robust to a point in which their performance is mostly limited by the hardware performing the readings and computations. As a result, research has mostly shifted towards improving the hardware supporting the computations behind these complex algorithms as well as the sensing hardware needed to obtain environment data, since these are the determining factors of an algorithm's precision.

Among the many options of sensing systems available to perform this task, some of the most accurate, each with their own advantages and drawbacks, are sensors which make use of electromagnetic waves for surveying the surrounding area. Often coupled with computer vision-based algorithms to complement their readings, these are the most commonly used sensors due to their accuracy and precision.

1.3 State-of-the-art

The field of autonomous robotics is a widely researched topic with diverse approaches due to the versatility of the subject itself. Among systems solely capable of obstacle detection and collision avoidance, a number of relevant implementations are worth special mention.

Padgett and Browne [5] explore the implementation of a short-range obstacle detection and collision avoidance algorithm applied to LIDAR sensing. The robot is equipped with a set of 4 mecanum wheels,

[7] http://robotek.no/filer/dokumenter/4-Brattvag-02.11.2016.pdf

granting the ability to move in any direction without shifting its pose (omnidirectional motion). Utilizing basic strafing motions, the robot is set to follow the outline of a wall, avoiding any protrusions it may encounter. Although sensing data are collected, no mention of clustering or real time monitoring is made. Nonetheless, the robot is fully autonomous along any unknown wall-like paths with computational operations being handled by the robot's own processing unit, granting a high degree of modularity.

Peng et al. [6] on the other hand explore an implementation of long-range collision avoidance using LIDAR sensing. With an effective median clustering algorithm, the robot is able to segment and reconstruct clusters as similar basic geometries (lines, rectangles and circles). Data obtained by the sensor is imported to a *Matlab* instance where avoidance is computed. However, such an implementation increases the bulk of the algorithm, introducing the necessity of a good processing unit to achieve real-time avoidance. A simulation test is carried out in the imported scenario to validate the algorithm. No mention of omnidirectional motion is made and simulations are limited to imported scenarios.

Hutabarat et al. [7] explores an implementation of LIDAR sensing in short-range obstacle avoidance. A thorough study is made of the performance with sensed objects with various different physical traits such as color and size. Motion control is based on modifying each of the rear motors' angular speeds (tricycle configuration). Obstacle avoidance bulk is minimized, increasing or decreasing motor speed based on proximity to obstacles on either side of the robot. While basic, this approach demonstrates a good performance when navigating narrow obstacle courses. Computational operations are handled by a single Raspberry Pi 3 acting as a master to all the assembly hardware. No mention of real-time data monitoring or simulation features is made.

Shim and Kim [8] in turn opt for an approach of short-range LIDAR sensing applied to a collision avoidance and SLAM algorithm, using the ROS framework as a means of data communication and visualization. A Pioneer 3-DX robot is used for its compatibility with the ROS framework. An improvement to the expanded guide circle method for obstacle avoidance is proposed, improving stability of the avoidance trajectory. A selective decision-making approach is made, allowing the robot to navigate the environment in either entry or bypass mode. These modes allow the robot to either navigate gaps and narrow spaces between detected obstacles (if avoidance is possible) or bypass them altogether, selecting an orbit-like trajectory to avoid dense obstacle zones. This grants a high versatility to the implementation, allowing it to navigate sparse or dense obstacle courses with ease. However, such algorithms pose the need of a high resource processing unit such as a computer. No mention of virtual simulation capabilities is made.

RADAR sensing implementations in small-scale collision avoidance implementations are uncommon, denoting a slight bias towards LIDAR sensing in these conditions. Ruiz et al. [9] present a RADAR-based approach to this problem, with a focus on exploring the limitations of RADAR sensing applied to less controlled scenarios such as roads and low visibility environments. It is shown that RADAR sensing is mostly impervious to low visibility conditions, being able to navigate long sections of a gravel road with sparse vegetation in either side as obstacles. Despite the good performance, this implementation's bulk demands a good processing unit, with a laptop being used for computations.

All aforementioned research works culminate in a common conclusion, rendering either LIDAR or

RADAR to be effective sensing methods for collision avoidance implementations when complemented with a variety of decision-making algorithms, each with their own advantages and drawbacks. However, to the author's knowledge, the majority of academic work focuses on the application of each of these devices as standalone data collection platforms, not providing a means of comparison between the two or advantages of sensor fusion. Simulation capabilities are scarce, limiting the range of possible trial scenarios and providing no means of testing in controlled, risk-free scenarios prior to real world trials.

Prototypes are mostly designed with no computational resource limitations, resulting in the necessity of a bulkier processing unit. While such an option often offers the possibility of real-time data monitorization, it hinders the implementation in terms of modularity.

1.4 Objectives

This thesis focuses on the analysis of LIDAR and RADAR sensors, two of the most common sensors used in small-scale obstacle detection systems. Their capabilities are explored with their integration in an obstacle detection and collision avoidance algorithm mounted on a small-scale robot. A prototype of a small-scale autonomous robot is developed, based on [10], [11] and [12], with the ability of navigating unknown environments with a given goal while avoiding collisions with any static or dynamic obstacles. The main contribution of this work is an analysis of the performance of the developed model with different combinations of sensing capabilities, utilizing the standalone sensors as well as a sensor fusion solution as an obstacle detection system. A physical model is developed as well as a virtual simulator based on the hardware of the physical implementation to enable testing in virtual scenarios without the risk of damaging the prototype. The implementation is developed with modularity in mind, limiting computational operations to the processing unit of the robot.

1.5 Thesis Outline

After this introductory chapter, Chapter 2 provides a theoretical background of the RADAR and LIDAR sensing systems, with an in-depth analysis of the working principles of this technology. An analysis of the standalone sensors is made, discussing the physics background which led to the development of these devices as well as the intricacies of each of the specific models. A theoretical take on the expected performance of these sensors is presented, highlighting various factors which may influence the behaviour of these systems. Finally, a brief analysis of the concept of vehicle safety and obstacle detection algorithms is made, presenting relevant options to be used with these specific sensors and selecting the most suitable.

Chapter 3 begins with an overview of the assembly of the prototype, justifying the decisions behind the choice of hardware as well as a general overview of the specifications of the assembly. A brief introduction of the ROS framework is presented. This chapter concludes with an overview of the data processing algorithms implemented in this work.

Chapter 4 provides an overview of the virtual implementation, describing the simulation of the physical model in a virtual environment and its specifications.

Chapter 5 presents an overview of the experimental implementation with a detailed description of the configuration of the various components in the assembly and their interactions.

Chapter 6 then provides an analysis of the testing environment, describing the scenarios in which the prototype is tested and the specifications of each trial. The results of these trials are then presented. Chapter 7 concludes with an analysis of the results obtained in the previous chapter, commenting on the performance of the robot against the expected results. Finally, a few remarks about the contribution of this work are presented as well as general guidelines for future work.

Chapter 2

Theoretical Background

In this chapter, a theoretical approach to the sensors is made, explaining the physics behind the sensing methods and the differences between them. A take on the expectations of the performance of these sensors is presented, highlighting relevant factors that may impact their effectiveness. Then, a brief overview of obstacle detection and collision avoidance algorithms based on these systems is made, discussing their advantages and disadvantages.

2.1 Sensing Systems

In this section, the theoretical basis of the sensing methods used in each of the sensors is described, introducing the most common types of sensors and noting the mathematical models behind the physics phenomena observed in these systems as well as uncertainties associated with these sensing methods.

2.1.1 RADAR

RADAR (RAdio Detection And Ranging) is an active remote sensing method that relies on the use of electromagnetic waves in the radio range of the spectrum to calculate distances from the emitter system to any obstacles within its scanning area. Radars have had remarkable development since their early days, when these systems were used solely for target detection and ranging. Nowadays, these devices are used to perform many complex additional tasks such as tracking, identification and classification of said targets. In this section, the theoretical background of these systems is explored, identifying the processes behind their diverse capabilities.

RADAR types

RADAR systems can be classified as one of two general categories, imaging and non-imaging. Imaging RADARs are capable of generating a map-like image from the information received, while the latter have limited capabilities, providing results in pure numerical values only. These systems can be further broken down into several types, as shown in figure 2.1.

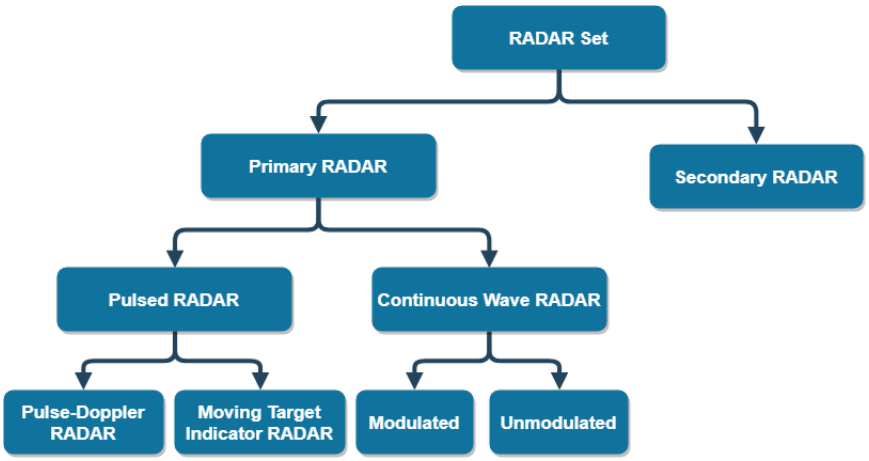


Figure 2.1: RADAR system types

A RADAR set may consist of a primary or secondary radar, represented in figure 2.2. Primary RADARs work independently of any external components, functioning both as an emitter as well as receiver to the electromagnetic waves generated by itself. A secondary RADAR works differently, relying on a separate device denominated transponder (transmitting responder) for communications. This transponder will respond to any interrogation signals (broadcasted by the secondary RADAR), providing information about the platform it is mounted on through its own generated signal. This allows for more detailed information being transmitted by the transponder (which can transmit information about the current state of the platform it is mounted on) and a stronger response signal at greater ranges, since the problem of signal attenuation present in primary RADAR configurations is greatly reduced.

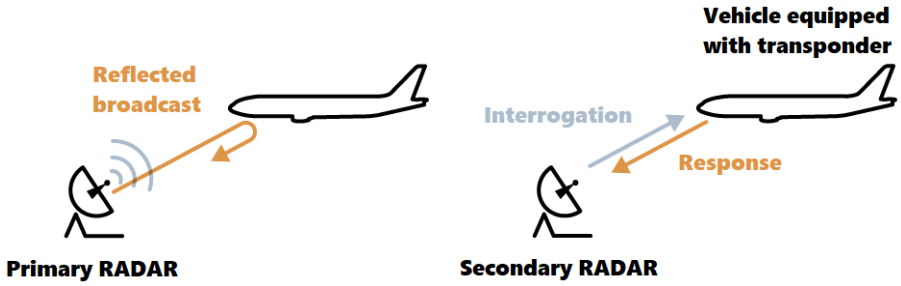


Figure 2.2: Primary and secondary RADAR[1]

Among primary RADARs, two distinct methods of wave propagation can be identified. Pulsed RADAR systems utilize electromagnetic pulses characterized by high power and frequencies. Within pulsed RADARS, Pulse-Doppler RADARs are able to provide information about the designated target's velocity, while Moving Target Indicator RADARs provide more limited information about the movement of targets but in turn avoiding distance measurement ambiguities.

Continuous-Wave (CW) RADARs, on the other hand, cast a continuous signal, rendering them unable to measure distances to any detected targets in the case of unmodulated systems. However, due to the continuous nature of the signal, these systems are capable of measuring rates of change in signal reception, which effectively translates into information pertaining the velocity of detected targets. With

[1] adapted from http://mh370photos.blogspot.com/2014/03/where-is-plane-where-is-flight-mh370.html

wave modulation, the emitted waves can be matched with the received signal, offering the ability to measure distance to targets as opposed to their unmodulated counterpart.

Working Principle

The principle behind RADAR sensors is similar to the principle of sound-wave reflection used in sensors such as SONAR (Sound Navigation And Ranging). Much like sound-waves being reflected off of sound-reflecting surfaces, such as caves and other enclosed spaces, electromagnetic waves emitted by the RADAR source will be propagated through air until they reach an object. As a result of this collision, in the case of a primary RADAR set, a small portion of the reflected energy being reflected off the obstacle in every direction (echo, similarly to its sound-wave counterpart terminology) will return to the sensor. By analysing the time elapsed between transmission and reception of these waves, t_{travel}, the distance to a target, d_{target}, can be calculated through

$$d_{target} = \frac{t_{travel}}{2} * v_{EW} \tag{2.1}$$

Since electromagnetic waves (EW) in the radio wavelength range travel at a similar speed as light does (visible light wavelength), the propagation speed, v_{EW}, can be approximated to the values in table 2.1.

Mean	Speed of Light (v_{light}) [km/s]
Air	299.704
Vacuum	299.792

Table 2.1: Speed of light in relevant means

Secondary RADAR sets work differently since these are dependent of external devices (transponders). The RADAR system will broadcast an interrogation signal similarly to primary RADARs. Objects, generally vehicles, equipped with transponders will receive this interrogation signal and broadcast a response signal containing detailed information about its current state. This signal will then be received by the RADAR and processed, extracting data about the vehicle including distance based solely on the travel time of the response signal.

Unambiguous range measurement

The non-instantaneous transmission of electromagnetic waves is the driving factor of the entire method of distance measurement in RADAR systems. However, this method is not foolproof and can lead to inaccurate measurements. Due to the travel time component in these calculations and the inability to match received pulses with the equivalent emitted waves within a certain distance threshold, a specific unambiguous measurement range can be identified in which the RADAR system will be able to function correctly. If the pulse round-trip travel time (time it takes for a pulse to be emitted, reflected and received again) is too long, it could result in incorrect measurements.

Figure 2.3 represents an ambiguous measurement scenario, in which the RADAR measurement will

not correspond to reality. Two targets, A and B, are placed inside the RADAR scanning zone, with target A being placed closer to the sensor than B. Pulse emissions are represented by taller rectangles and pulse receptions are represented by shorter rectangles. The real location of targets is represented in black, while ambiguous estimations of target locations are represented in gray.

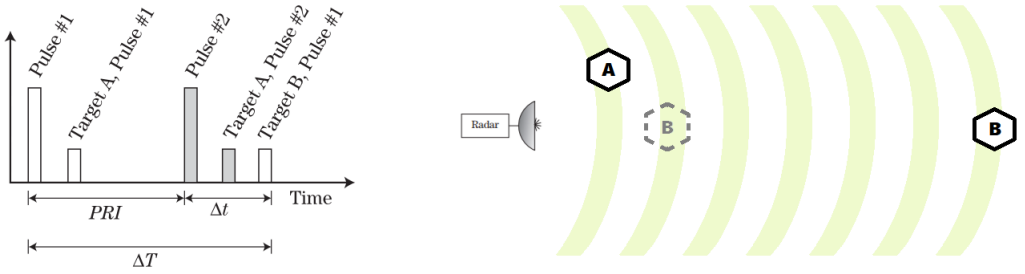


Figure 2.3: Pulse RADAR range ambiguity[2]

In this scenario, a first pulse (pulse #1) will be emitted towards targets A and B. Since target A is closest to the sensor, the pulse will be reflected and received shortly after. Then, during the round-trip time window of the pulse being emitted to target B, a second pulse (pulse #2) is broadcasted from the sensor. Again, the reflection off of target A is received shortly after and only then is the reflection of pulse #1 off of target B received. Since both these reflections are received after the emission of pulse #2, these will be interpreted as reflections of pulse #2 when, in fact, only one of the signals is a reflection of pulse #2. This results in an incorrect reading of two targets close to the sensor.

Based on this example, a mathematical condition for the unambiguous scanning zone can be deduced. From (2.1) and solving for ΔT (round-trip time of the RADAR pulse),

$$\Delta T = \frac{2 \cdot R}{c} \tag{2.2}$$

where ΔT is the inter-pulse time, R is the scanning range and c the speed of light. To avoid range ambiguities in measurements, the pulse repetition interval (inter-pulse period) must be equal or longer than the round-trip time corresponding to the maximum intended ambiguous scanning range

$$PRI \geqslant \Delta T_{max} = \frac{2 \cdot R_{max}}{c} \quad \Leftrightarrow \quad R_{max} \leqslant \frac{c \cdot PRI}{2} = \frac{c}{2 \cdot PRF} \tag{2.3}$$

where PRI is the pulse repetition interval, PRF is the pulse repetition frequency and R_{max} is the maximum ambiguous scanning range.

Although this parameter can be tuned to suit the necessary unambiguous scanning range by modifying the *PRI* (or *PRF*, as these are dependant of each other), this process will negatively impact other requirements (such as unambiguous Doppler shift measurements, explained further ahead). As such, some systems are unable to avoid an ambiguous range condition.

[2] adapted from [13]

Velocity estimation - Doppler shift

If there is relative motion between the RADAR and a target, the frequency of an electromagnetic wave being emitted by the source and the one echoed from the target will be different. This phenomenon is denominated *Doppler effect*, which is common to all wave phenomena. This concept was initially studied solely as an acoustic phenomenon due to the limited knowledge of the subject at the time. This phenomenon can be explained considering the case of an ambulance moving from point A to point B while its siren is turned on (figure 2.4).

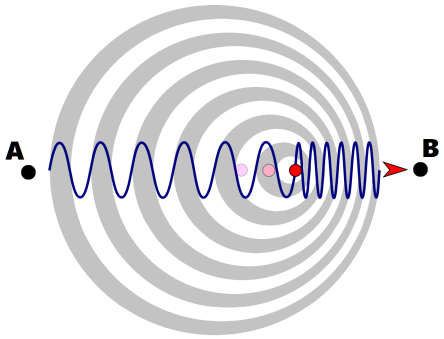


Figure 2.4: Doppler effect example[3]

As the ambulance is moving (red dot), the siren is acting as a sound-wave source, continuously emitting waves in every direction. As a result, these sound waves, despite being identically generated, will be received differently by any pair of receptors placed in different points in space. In this example, an individual standing in point A will receive these sound waves with a set frequency, depending on the speed at which the ambulance is moving. However, an individual standing in point B will receive the very same sound waves with an increased frequency due to the ambulance moving towards him. It is said that each of these distinct individuals is receiving these identical sound-waves with different ”apparent frequency”, hence why both will hear distinct sounds characterized by a higher pitch (frequency) in the case of point B and a lower pitch for point A.

This phenomenon is analogous to electromagnetic waves in the sense that relative movement between a RADAR-target pair will lead to responses with shifted frequencies when compared to the emitted wave. This frequency shift, denominated *Doppler shift* (f_d), is approximately given by

$$f_d \approx \frac{2 \cdot v_r}{\lambda} \tag{2.4}$$

where v_r is the radial component of the target's velocity vector toward the radar and λ is the wavelength of the broadcasted electromagnetic wave [13]. By convention, a closing target will yield a positive frequency shift (increased response frequency), whereas a target moving away from the source will yield a negative shift. As a result, the relative velocity between the RADAR and a target can be estimated by

$$v_r \approx \frac{f_d \cdot \lambda}{2} \tag{2.5}$$

[3]adapted from https://upload.wikimedia.org/wikipedia/commons/f/f7/Doppler_effect_diagrammatic.png

Unambiguous Doppler shift measurements

As previously stated, the pulse repetition frequency (*PRF*) of the RADAR can be tuned to increase resolution and minimize range ambiguity errors. However, this process has its drawbacks as well. From equation (2.3) it is clear that a decrease in *PRF* will increase the RADAR's unambiguous range. However, since a pulsed RADAR measures the Doppler shift at the pulse repetition frequency, lowering the *PRF* too much can lead to Doppler frequency ambiguities due to a very low sampling rate.

Nyquist's sampling theorem states that to ensure unambiguous measurement, the sampling rate should be at least double the maximum signal frequency. This statement, adapted to the subject of RADAR sampling, leads to the condition

$$f_{d_{max}} = \frac{PRF_{max}}{2} \Leftrightarrow PRF_{max} = 2 \cdot f_{d_{max}} = \frac{4 \cdot v_{r_{max}}}{\lambda} \tag{2.6}$$

where PRF_{max} is the maximum pulse repetition frequency, $f_{d_{max}}$ is the maximum unambiguous Doppler shift and $v_{r_{max}}$ is the maximum unambiguous speed measurable by the RADAR.

Optimization

With both non-ambiguity conditions previously determined ((2.3) and (2.6)), three regimes of operation can be identified to classify RADAR systems in terms of their corresponding operation pulse repetition frequency. Since the maximization of the unambiguous scanning range translates into the minimization of the *PRF*, systems designed to operate at low *PRF*s to achieve unambiguous measurements for all intended scanning ranges are designated *low PRF systems*. In a similar fashion, systems designed to be unambiguous in terms of Doppler shift measurements, which maximize *PRF* to achieve higher measurable Doppler frequency shifts (Nyquist's sampling theorem) are designated *high PRF systems*.

A third regime is characterized by an intermediate system designed to operate in a specific range of unambiguous scanning range and Doppler frequency shifts, satisfying both non-ambiguity conditions to meet its specifications. These systems are designated *medium PRF systems*.

An additional constraint is placed upon each of these regimes to ensure an optimal amount of information or measured data is passed through a communication channel, leading to correct and reliable results. This condition is placed upon the signal's effective bandwidth, a measure of a traffic stream used in dimensioning modern communication networks which stands for the mean expected bandwidth at which the signal is transmitted. This condition is given by

$$\beta \cdot \tau \geq \pi \tag{2.7}$$

where β is the effective bandwidth, τ the effective time duration of the signal and π the mathematical constant. This condition is derived and further explored in communication theory [14].

Resolution

The resolution of a RADAR is a concept related to a RADAR's ability to distinguish multiple targets that are closely spaced either in range, angle or Doppler frequency [13]. Examining the example of figure 2.5, in which two separate point scatterers are placed in the RADAR detection range, three possible scenarios yielding different results can be analysed.

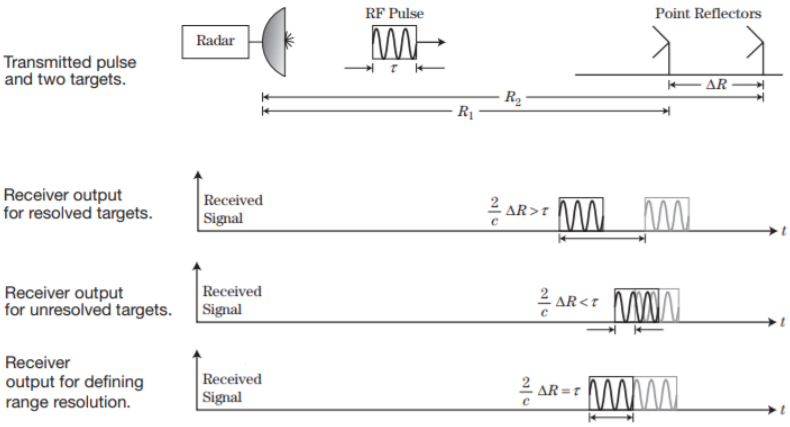


Figure 2.5: Multiple Target detection scenarios[4]

A single pulse being transmitted by the RADAR at a given repetition frequency (RF) will be echoed from two different point reflectors placed within its scanning range. Each of these reflectors is placed at a given distance relative to the pulse source, R_1 and R_2, and spaced by ΔR. Depending on the spacing between the reflectors, the echoed pulses may overlap, merging contructively, destructively or in some intermediate fashion. In a situation where these pulses overlap, the RADAR receiver would be unable to distinguish the echoed pulses, since these are received as a single pulse. It is said that these targets cannot be reliably resolved due to ambiguity. As such, the required distance between two distinct targets to be resolved correctly and reliably is given by

$$\Delta R = \frac{c \cdot \tau}{2} \Leftrightarrow \Delta R = \frac{c}{2 \cdot PRF} \tag{2.8}$$

where c is the speed of light (approximation) and τ is the pulse period (inverse of pulse repetition frequency, PRF). Consequently, RADAR parameters can be tuned, if possible, to increase resolution and enable the detection of lesser spaced objects, which in turn will affect the unambiguous scanning range as previously stated.

2.1.2 LIDAR

LIDAR (Light Imaging, Detection And Ranging), sometimes found under different designations such as LADAR (LAser Detection And Ranging), is another active remote sensing method which, similarly to RADAR (described in section 2.1.1), relies in electromagnetic waves for detection and ranging. However,

[4] adapted from [13]

as its name indicates, LIDAR uses light (electromagnetic waves in the visible range of the spectrum) in the form of a pulsated laser to measure distances between two points in space, the sensor (source) and a designated point (target). This designation is often used to refer to the surveying device that uses this method as well. LIDAR scanners can obtain three types of information [15]: Range to a target (Topographic LIDAR), chemical properties of the target (Differential Absorption LIDAR) and the velocity of said target (Doppler LIDAR). This thesis focuses on the use of topographical LIDAR systems for range detection, as represented in figure 2.6.

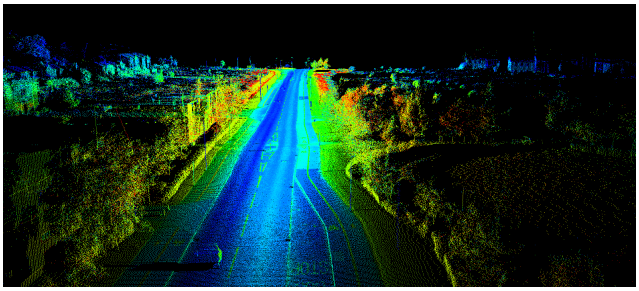


Figure 2.6: Topographic LIDAR Scan of a highway[5]

These systems are usually integrated in moving platforms and used to track both moving and fixed targets. Combined with the data provided from the system in which the sensor is mounted it can generate precise information about the shape of any detected targets as well as its surface characteristics, to an extent. As a physical sensing method, it is bound to be affected by noise, resulting in non-exact readings. However, these systems perform measurements with a high degree of precision, up to a millimetric scale.

Working Principle

LIDAR scanning is a surveying technique that makes use of electromagnetic waves, specifically light in the range of ultraviolet, visible or near infrared light beams. Due to the nature of electromagnetic waves, when colliding with an obstacle, they are reflected back into the sensor and picked up by a receiver, which then converts this measurement data into an estimation of the distance to target, as represented in figure 2.7.

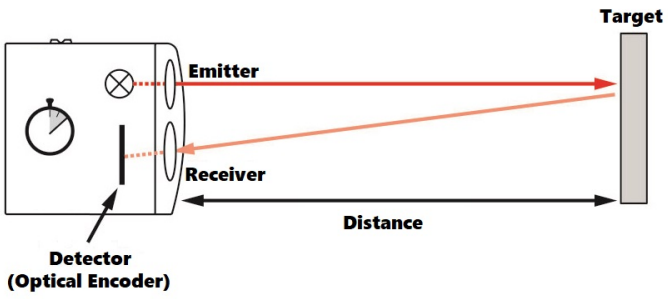


Figure 2.7: LIDAR working principle[6]

[5] https://www.geospatialworld.net/news/lizardtech-awarded-us-patent-lidar-point-cloud-compression-2/
[6] a

Utilizing precise estimations of the speed of light, detailed in table 2.1, and the principle of light reflection, the distance to the sensing target is given by

$$d_{target} = \frac{t_{travel}}{2} * v_{light} \tag{2.9}$$

In short, the distance between the target and the sensor can be derived by measuring half of the total time it takes for the light beam to travel from the emitter to the target and reflected back to the receiver.

This method enables the accurate estimation of distances between objects in space. However, a single light beam provides distance estimations to a single point only. Proximity sensing applications usually require a wider scanning area, spanning across a relatively wide angle centered on the point the sensor is mounted on depending on the specific application.

LIDAR sensors are able to extend the scanning range to a wide angle from the emitter origin point, as represented in figure 2.8.

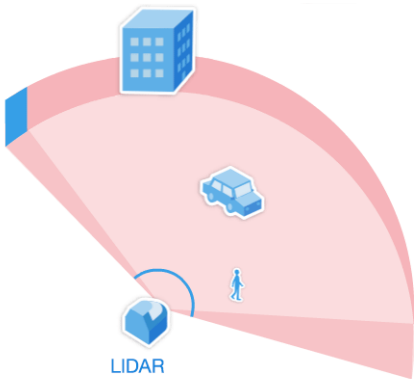


Figure 2.8: LIDAR scanning angle[7]

Due to the nature of the light beams emitted by the sensor, characterized by a high velocity in either sensing means (air or vacuum), the travel time between the emitter and the target is extremely short. This allows for the application of a single light beam to scan a wide angle when coupled with a rotational mechanism. The internal configuration of a general LIDAR scanner is represented in figure 2.9.

With the use of a servo motor and a mirror, the laser beam can be redirectioned, being able to scan several points in a plane at sensor elevation in succession in a short amount of time. This allows for a full 2D discrete scan of its surroundings, characterized by several measurements with an even angular spacing between them, $\theta_{resolution}$ (angular resolution). A simplified scheme of the 2D scanning is represented in figure 2.10.

[7] adapted from https://www.konicaminolta.com/cn-zh-cn/future/3dlr/index.html
[8] https://www.renishaw.com/en/optical-encoders-and-lidar-scanning–39244

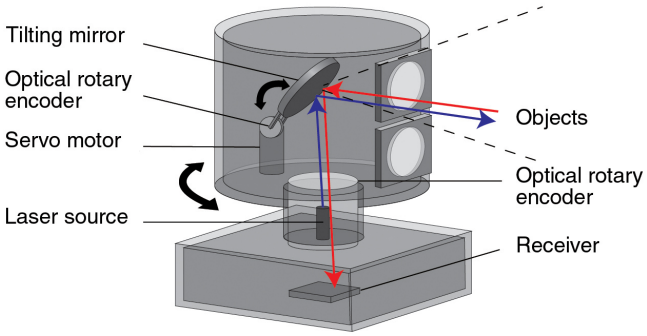


Figure 2.9: Component breakdown of LIDAR sensor[8]

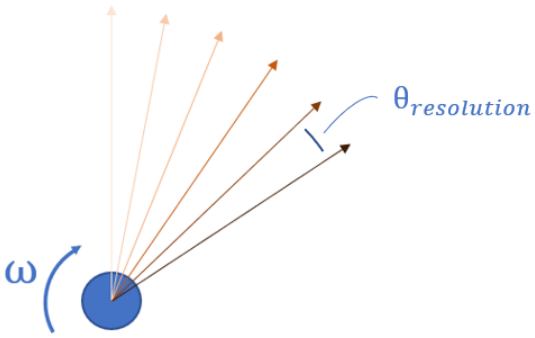


Figure 2.10: LIDAR scan angular sampling (top view)

This scanning method can be complemented with a servo tilting mechanism to enable the measurement of target elevation (in the case of 3D LIDAR systems).

Laser emitter

The light beam employed in LIDAR systems is a pulsated laser source emitting light in the wavelength range of near-infrared (IR, 780 - 3000 nm wavelength), visible (390 - 780 nm) or near-ultraviolet (UV, 200 - 390 nm) [16]. The specifications of each LIDAR system can vary according to specific applications, with factors such as reflectance of electromagnetic waves and ambient heat radiation contributions being determining factors for the precision of the measurements taken.

However, LIDAR systems are designed to be ”eye-safe” above all, since most of their applications result in possible exposure of the human eye to these light beams. As such, every LIDAR scanner must abide by the international standard EN 60825-1. This translates to limitations in pulse frequency and laser radiance. These specifications are fine tuned to fulfill the required safety conditions by lowering the sample time or, more commonly, shaping the laser pulse to reduce exposure to meet the maximum permissible exposure (MPE) to laser radiation of the human eye. Beam shaping can be achieved by either of two configurations, utilizing an afocal optical system comprised of two lenses or mirrors to expand and redirect the beam, as shown in figure 2.11.

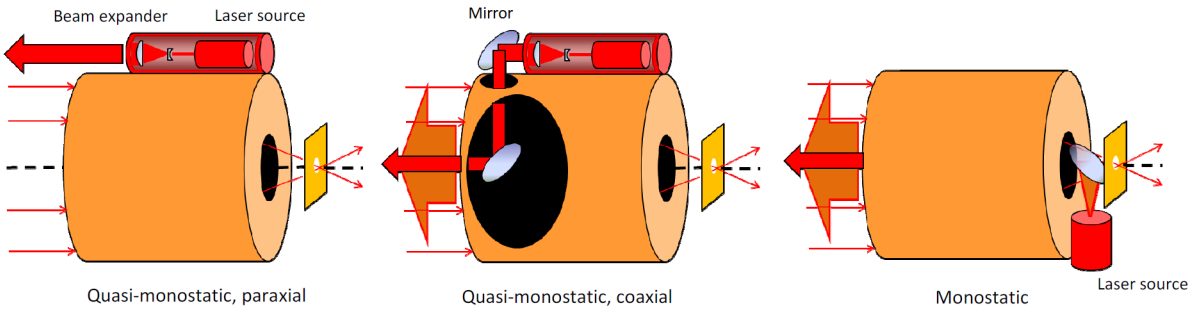


Figure 2.11: LIDAR beam shaping configurations [9]

Configurations where the laser source and receiver have a common optical axis are denominated monostatic, whereas configurations in which the two axes are not concentric are denominated bistatic. However, an intermediate case can be defined where the axes are approximately concentric, as is the case of most LIDAR systems. In this case, the configuration is given the designation of quasi-monostatic. Further differentiation can be made considering the redirectioning of the beam. Cases in which the beam source's optical axis is not concentric with the receiver's but is then redirected to the very same axis are then classified as quasi-monostatic coaxial configurations, with the opposite being given the designation of quasi-monostatic paraxial configuration.

Receiver

The reflected light beam provided by the emitter will then be captured by the receiving module. This component is generally coupled with the emitter module, being synchronized with its rotation speed. However, more complex configurations where a static receiver is present are possible, making use of a rotating mirror to redirect the received light beam into the receiver unit. By using a mirrored polygon, the scanning frequency can be increased for lower motor frequencies, with a scan being created for each polygon face. This effectively limits the field of view of the LIDAR scanner, as seen in figure 2.12.

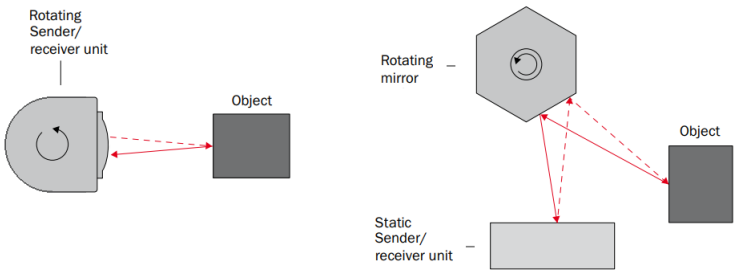


Figure 2.12: LIDAR receiver configurations[10]

In the case of a 3D LIDAR scanner, additional sensors, receivers or a combination of both at different tilting angles can be incorporated, granting the ability to scan multiple planes simultaneously and, with that, obtain information about the third dimension, elevation (figure 2.13).

[9] adapted from [16]
[10] adapted from [17]

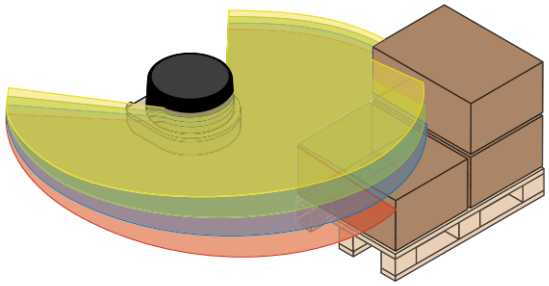


Figure 2.13: 3D LIDAR layer scan [17]

Velocity Estimation - Doppler LIDAR

Velocity estimation in LIDAR sensors is similar to RADAR systems, being based on the Doppler effect (described in section 2.1.1). Not every LIDAR sensor can perform velocity measurement. Frequency analysis, being a component of Doppler shift extraction algorithms, require a sufficiently long continuous wave as input, as short waves cannot capture most of the information needed for this method. Two alternatives to achieve these requirements are continuous wave systems, which cast a continuous laser beam at its target, and chirped pulse systems, which emit modulated long pulses to capture frequency shift data. Within continuous wave Doppler LIDAR systems, two distinct procedures can be used for estimation. Amplitude-Modulated Continuous Wave LIDAR systems (AMCW) use a continuous laser beam with modulated amplitude to illuminate its target, obtaining distance data in the process as any standard LIDAR system. However, by modulating the amplitude of short sections of this continuous beam, the reflected beams can be compared with their emitted counterparts. Then, by extracting the frequency shift of these signals (*Doppler shift*), the relative velocity can be estimated using equation (2.5). This process is very demanding in computational resources and, consequently, power consuming.

Frequency-Modulated Continuous Wave LIDAR sensors (FMCW) on the other hand make use of a stable known frequency laser beam modulated by triangular modulation (the only type of modulation that allows for simultaneous range and speed measurements [18]). This will produce a signal with gradually varying frequency, which will be used for scanning and as reference (Local oscillator, LO). Then, after receiving the light beams reflected off of the targets (into the receiving optical encoder, RX), these will be optically mixed with the undisturbed local oscillator to check for frequency shifts, with the result being a signal with the difference in optical frequencies (beat note) [19]. This process is represented in figure 2.14.

This beat note contains the *Doppler shift* information used for velocity estimation. In this example, a negative frequency shift is observed, meaning the LIDAR picked up an object moving in the opposite direction of the sensor.

This method, more recent than AMCW (first introduced in 2015 by Blackmore Sensors and Analytics, Inc. and released to the public for use with automated fleets in early 2019), is less complex, being lighter in computational resources used and making it faster and more efficient than its direct competitor.

[12] adapted from https://www.autosensorsconf.com/sites/autosensorsconf/files/assets/6C%20LiDAR%20Face-Off%20Blackmore_Curry.pdf

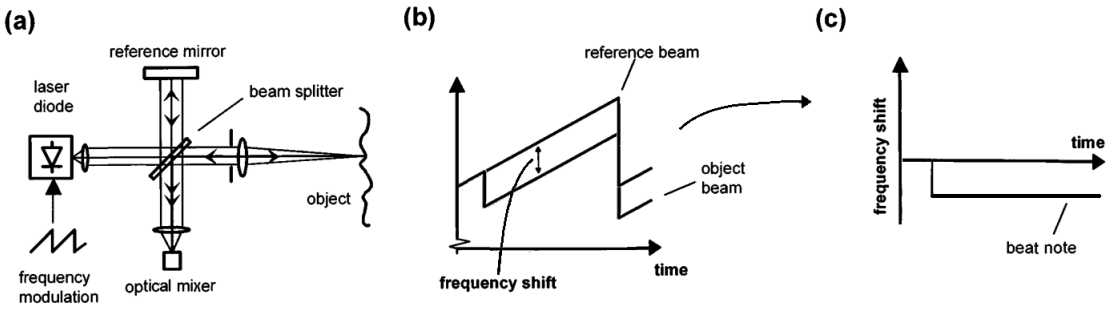


Figure 2.14: (a) Physical components of FMCW LIDAR, (b) reference and object beam comparison, (c) extracted beat note[12]

2.2 LIDAR vs RADAR - A theoretical overview

LIDAR and RADAR sensors both belong to the category of range-finder or proximity sensors, performing essentially the same general task of estimating relative distance to detected targets in their scanning areas and, in the case of RADAR systems, their velocity. With the many differences in algorithms, physics background and implementation methods, differences are bound to exist between the two.

Both of these sensors perform differently when working under different environments and subjected to noise from various sources.

From poor reflectivity of targets to wave scattering, these systems are characterized by slight variations of readings when compared to real distances, usually in the form of a percentage of the reading at maximum scanning range. In this section, the two devices are compared against each other in a theoretical approach, discussing the expected performance, advantages and disadvantages of each sensor in specific working conditions.

Scanning Range

Due to health and safety regulations imposed to LIDAR systems (as referenced in section 2.1.2), the legally allowed radiating power is limited. A laser sensor's maximum range is highly dependent in its laser power and wavelength, which in turn also determine the radiating power. Hence, the scanning range will also be affected by these limitations. Since RADAR systems are not harmful for a human, such limitations will not be imposed to RADAR sensors, which are able to reach longer ranges with a lower signal power.

Other impactful factors in the maximum range of these sensors include the absorption of waves in the LIDAR range by irradiated objects, which is not evidenced in the RADAR range, and the attenuation of electromagnetic waves by the atmosphere and its composition [20]. Figure 2.15 depics the spectrum of attenuation of electromagnetic waves within the earth's atmosphere.

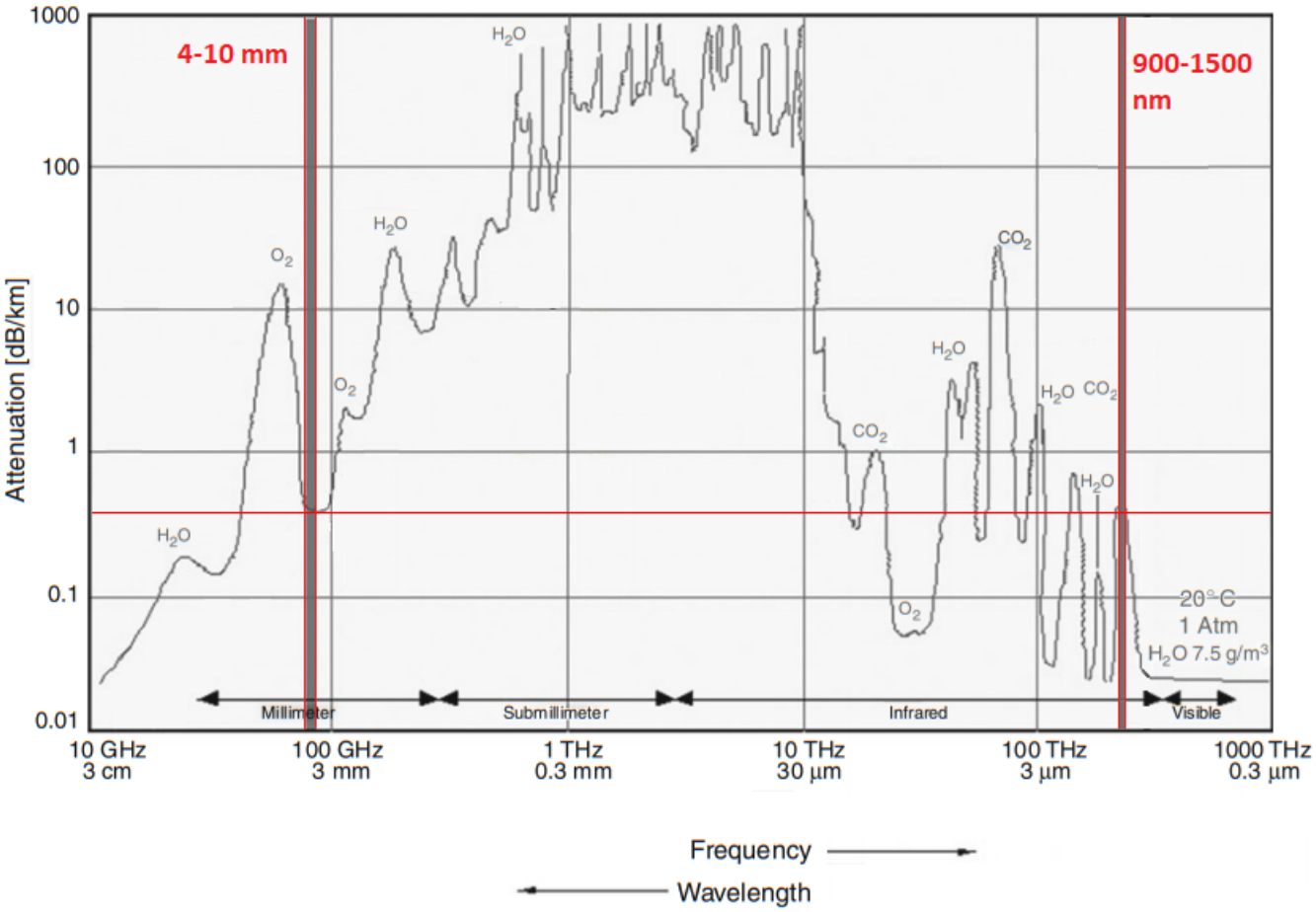


Figure 2.15: Spectrum of atmosferic attenuation of electromagnetic waves [20]

LIDAR beam waves (tipically within the 900-1550nm) are attenuated to a slightly higher degree than the RADAR's millimeter wavelength beams (tipically within the 4-10mm range).

These factors lead to an increased degradation of LIDAR waves as opposed to RADAR waves. Consequently, within the scope of scanning range, RADAR systems are expected to outperform LIDAR systems.

Lighting Conditions

Within the subject of lighting conditions, both sensors are expected to perform well during night-time since LIDAR sensors possess a light source of their own to perform the readings and radio waves are unaffected by radiation during the night. However, such is not the case during daytime. While RADAR sensors perform as expected under any lighting conditions due to their longer wavelength, solar radiation does influence LIDAR performance within this time frame. LIDAR beams are affected by solar radiation reaching the earth's surface since some of this radiation falls under the same wavelength range as this sensor's beam. In fact, solar radiation dominates the backscatter noise (noise associated to waves being reflected back to the sensor) in range sensing LIDAR during daytime [21]. As such, while both sensors

are expected to perform well during nighttime, the RADAR sensor is a superior option over LIDAR in environments exposed to significant solar radiation.

Wave Penetration

The transmitted wave's wavelength is the determining factor when analyzing the ability to penetrate or pass around objects and reach obscured targets. Due to the short wavelength of LIDAR systems, these are not able to penetrate objects and detect targets which may be obscured by these. RADAR systems, however, are characterized by a longer wavelength which, unlike laser based sensors, are able to penetrate or pass around certain objects which effectively grants the ability to detect obscured targets, albeit at a loss of beam power. An example of such phenomenon is shown in figure 2.16.

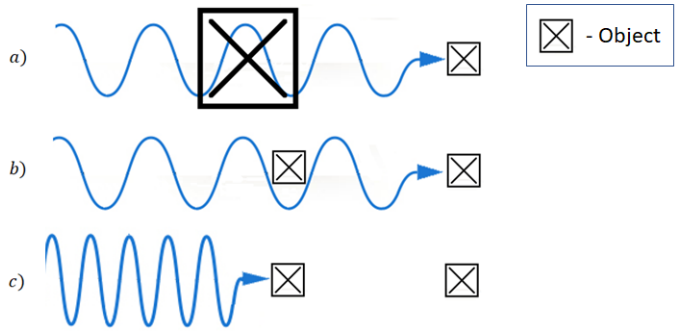


Figure 2.16: RADAR waves reaching obscured target (a,b) and LIDAR waves unable to detect obscured target (c)

Within the scope of autonomous systems, this information may prove essential to provide a complete scene of non-static environments.

Resolution

Angular resolution between the two sensors being studied is linked to model specifications such as LIDAR scanning speed and frequency as well as a RADAR's aperture angle, pulse width and frequency, which generally favors LIDAR systems due to their increased scanning frequency but does not offer a conclusive analysis in this scope. However, unambiguous range resolution is heavily dependant on pulse width and frequency. As previously stated, the difference in wavelength of the electromagnetic waves used in these two devices affects their ability to provide accurate readings of their surroundings. However, while the RADAR offers a few advantages over LIDAR in the domain of obscured targets and lighting conditions, the comparison between their expected performance in the domain of resolution does not follow the trend.

Due to the increased wavelength of RADAR waves, since detection of objects is dependant on reflection of the waves, it is possible for RADAR waves to miss objects of smaller dimensions which would otherwise be detected by LIDAR waves due to their much shorter wavelengths.

Weather Conditions

Proximity sensors as autonomous vehicle solutions are subject to all kinds of environmental conditions. In the case of LIDAR sensors, the most impactful adverse conditions are weather related. As a light-based system, being dependent on the transmission and reception of electromagnetic waves in the visible range of the spectrum, low visibility constitutes a considerable hurdle when designing such systems.

Despite both LIDAR and RADAR sensors utilizing electromagnetic waves for transmission, these are affected differently under these scenarios due to their wavelength differences. Figure 2.17 represents the electromagnetic wave attenuation spectrum in various weather conditions.

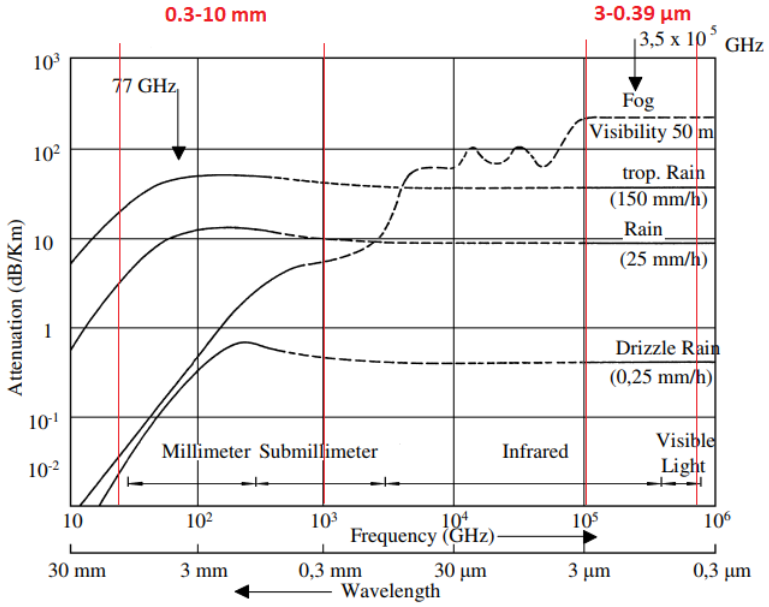


Figure 2.17: Attenuation of electromagnetic waves in various weather conditions[22]

As evidenced, attenuation due to weather interference is not homogeneous in the radio-infrared range (30mm-0.3μm). Consequently, LIDAR sensors, which operate in the near-ultraviolet to near-infrared range (0.39-3μm) are expected to be substantially more attenuated than RADAR, which operates in the microwave range of the spectrum (0.3-10mm) [23]. In general, LIDAR sensors experience a 25% decrease in performance when faced with low visibility weather such as fog or rain [22], which is much superior to the performance decrease of RADAR systems.

2.2.1 Reflectivity

Reflectivity is a key property of electromagnetic waves which greatly influences the performance of electromagnetic wave-based sensors. Due to the increased wavelength and wider beam form of radio waves used in RADAR systems, these are characterized by a low absorptivity when in contact with obstacles. For this reason, RADAR waves have a high reflectivity, often leading to multipath reflections where an emitted wave may be reflected multiple times and then received in a different direction of arrival

(different wave direction from when it was emitted by the source), generating ”ghost targets” (detections from non-existent obstacles) [24], as seen in figure 2.18. However, due to the nature of LASER emitters, LIDAR sensors do not suffer from ”ghost detections”, as the expected direction of arrival of a received beam is the same as the emitted beam, allowing for no disparity in angle of arrival.

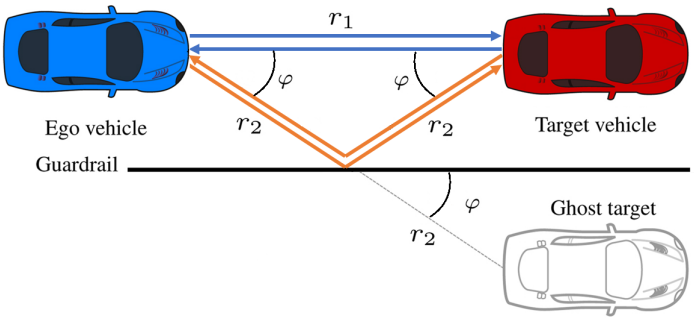


Figure 2.18: Multipath reflection example

2.3 Collision Avoidance

With the obstacle detection capabilities of the two systems described in section 2.1, this technology brings a significant contribution to the area of automotive automation, allowing for the application of algorithms to perform obstacle avoidance based on detections provided by these sensors. The purpose of obstacle avoidance or anti-collision systems is to prevent a fixed or, on the scope of this work, a small-scale autonomous robot to collide with any obstacles, whether if the robot, the obstacles or both are moving. It is essential for offline path planning as well as online reaction to unexpected changes in the environment along a predefined trajectory.

Due to the limited computational power of a dedicated processing unit integrated in a small-scale autonomous robot and since these algorithms are abundant, a selection of methods are explored, being computationally light and more suited for integration with the sensors studied in this work.

The concept of vehicle safety is introduced in the following section, being an integral part in the three selected collision avoidance methods that are explored in subsequent sections, stating their theoretical background and guidelines.

Vehicle Safety

An automated robot, as any electromechanical system, has an inherent response time to received inputs which, despite being designed to have a fast response, is not instantaneous. As such, and as an added precaution to ensure the ability to respond to any sudden obstacles, the concept of vehicle safety is defined.

Vehicle safety effectively translates into the definition of multiple zones within the robot’s observable environment (through any integrated sensors or interfaces that enable the collection of data from its

surroundings). Within each of these zones, obstacle detections will have different contributions to the collision avoidance algorithms, depending on the likelihood of a collision with the robot.

The four zones in which the robot's observable range can be decomposed into are the observation zone, warning zone, danger zone and collision zone. Figure 2.19 depicts a representation of these zones.

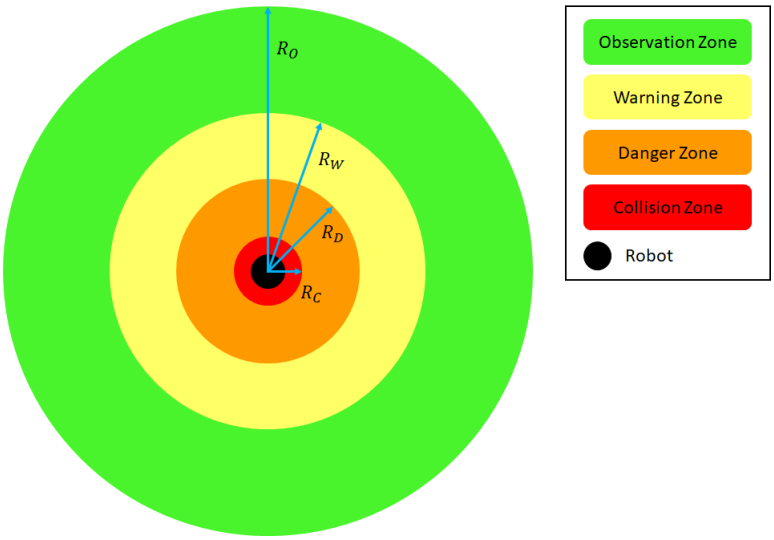


Figure 2.19: Obstacle detection zones

In decreasing range order, R_O is defined as the maximum range of the observation zone, R_W is the maximum range of the warning zone, R_D the maximum range of the danger zone and R_C the maximum range of the collision zone.

2.3.1 Potential fields method

Several obstacle avoidance methods based on artificial force potential fields have been proposed by Khatib [25]. These methods are based on the influence of artificial attractive and repulsive forces generated by obstacle detections and acting upon the moving robot. This enables for the correction of the robot trajectory while still aiming for a final target destination.

In this approach, obstacle detections are registered as repulsive forces proportional to the distance to the moving robot, while the waypoints along the robot's trajectory to a final destination are treated as attractive forces. The robot's trajectory is then computed in real-time, where the algorithm will compute the global potential field acting on the robot (acting as a charged particle) and defining the lowest potential value as the path to follow in that instant. Figure 2.20 depicts an example of a potential field acting on a moving robot entering the scene from the bottom left corner.

This method can be further improved by optimization algorithms such as Market-Based or Particle Swarm optimization (MBO and PSO, respectively) explored by Palm and Bouguerra [26, 27]. However, these may prove inadequate in specific scenarios due to local minima. Such issues can be minimized by reinforcement learning but at the cost of a higher computational effort.

[13] Adapted from https://www.cs.mcgill.ca/ hsafad/robotics/

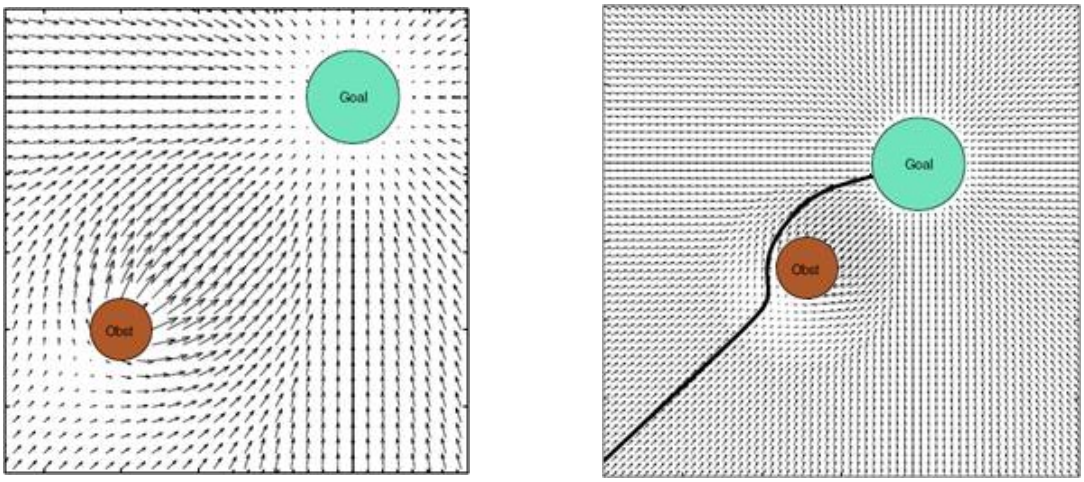


Figure 2.20: Generated potential field (left) and computed robot trajectory (right)[13]

2.3.2 Potential flow method

Similarly to the potential fields method described in section 2.3.1, another method of collision avoidance was developed based on a particular field of mechanics. Khosla and Volpe [28] proposed a different approach to the potential field method, the potential flow method.

The potential flow collision avoidance method is based on its fluid mechanics counterpart. Figure 2.21 represents the steady laminar flow of a fluid around a cylindrical object, represented by streamlines.

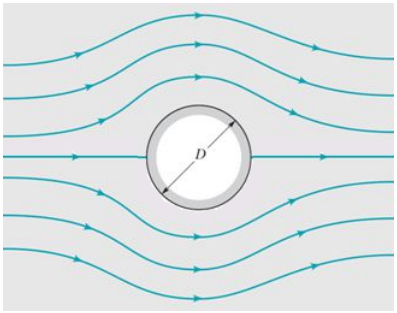


Figure 2.21: Steady, laminar fluid flow around a cylindrical object[14]

Due to the presence of an obstacle, the fluid is forced to flow around the obstacle, as represented by the deviated streamlines. Due to the laminar character of the flow, the streamlines return to the original configuration.

Similarly to fluid potential flow, this obstacle avoidance method uses the velocity potential to construct stream lines in the robot environment to achieve a natural and smooth trajectory correction. The algorithm defines which streamline to follow based on the conditions in a given instant.

This method can be applied to off-line scenarios with prior well known environments as well as on-line obstacle detection in unknown terrain.

[14] Adapted from https://slideplayer.com/slide/4568009/

2.3.3 Velocity obstacles method

The velocity obstacles method is a first-order obstacle avoidance method which uses the concept of velocity obstacles to achieve safe robot trajectories. Since it is a first-order method, it does not integrate velocities which result in positions as functions over time. Instead, the method is restricted to the robot feasible velocity space.

Velocity obstacles are a representation of robot velocities which, in the current scenario, would result in the collision of said robot with a fixed or moving obstacle in its path in the near future [29]. In the case of a static obstacle, this set of velocities is designated as a **collision cone**. Examples of these situations are represented in Figure 2.22.

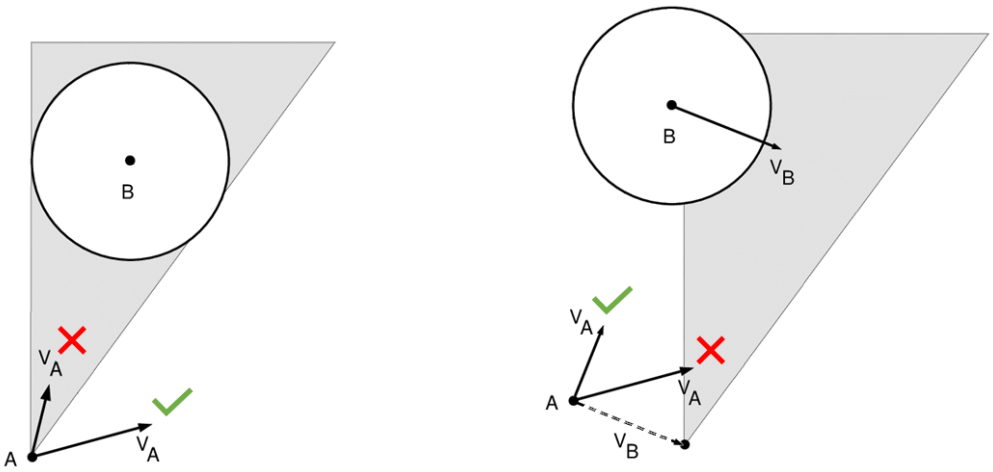


Figure 2.22: Collision cone (left) and velocity obstacle (right) examples[15]

The moving robot is represented by A, whilst fixed (Figure 2.22, left) and moving (Figure 2.22, right) obstacles are represented by B and gray areas represent their collision cone and velocity obstacles, respectively. The velocity obstacle is built by the displacement of the collision cone of an otherwise fixed obstacle by its velocity vector, V_B. Robot velocity vectors, V_A, outside the velocity obstacle and collision cones constitute successful avoidance vectors, while vectors that do not satisfy this condition will result in an impending collision with the obstacle.

Despite achieving acceptable results in obstacle avoidance with the absence of velocity readings from the obstacles, this method's performance and smoothness of the resulting trajectories are substantially improved if this information is available.

This method may be coupled with optimization algorithms such as Recursive Probability-based [30] or Artificial Bee Colony [31] decision making methods to optimize the choice of an avoidance trajectory.

[15] Adapted from [29]

Chapter 3

Collision avoidance implementation using RADAR and LIDAR

Following the theoretical analysis of the components studied in this thesis, testing trials must be performed to evaluate the validity of its expected results. As with any device in direct interaction with the environment, such expected outcomes may prove inaccurate or false due to noise or any external factors that may influence its behaviour, which are easily unaccounted for or neglected in ideal scenarios.

Two testing baselines are proposed and studied in this chapter. Firstly, a prototype of a real small scale automated vehicle with obstacle avoidance algorithms is developed to test the performance of the LIDAR and Doppler RADAR sensors in several aspects under the obstacle detection spectrum. Then, a simulated model based on the physical implementation is developed, simulating the behaviour of the sensors in a virtual environment with real conditions conditions to validate the theoretical results and to emulate the motion system of the physical implementation with its hardware constraints.

3.1 Physical Implementation

To evaluate the performance of the sensors subject to analysis in this work, a physical model was developed to integrate these devices in real, small scale obstacle detection and collision avoidance scenarios. Figure 3.1 depicts a diagram of the final physical implementation and the general flow of information in the system. PWM_i^c are the pulse width modulation generated command signals transmitted to the MD25 boards ($i = 1, 2, 3$), ω_i^c the commanded angular velocity of the wheels, x, y and θ the cartesian and angular positions of the robot, v_x, v_y and ω the cartesian and angular components of the robot's velocity, $d_{measured}$ and $v_{measured}$ the proximity and velocity of detected obstacles, x_{ref} and y_{ref} the cartesian components of the robot's reference and v_x^c, v_y^c and ω^c the cartesian and angular velocity components transmitted as commands to the robot by the collision avoidance system. In the following sections, the physical implementation is explored, describing each of the constituent elements of the model as well as assembly specifications and protocols used in the interactions between each of these interfaces.

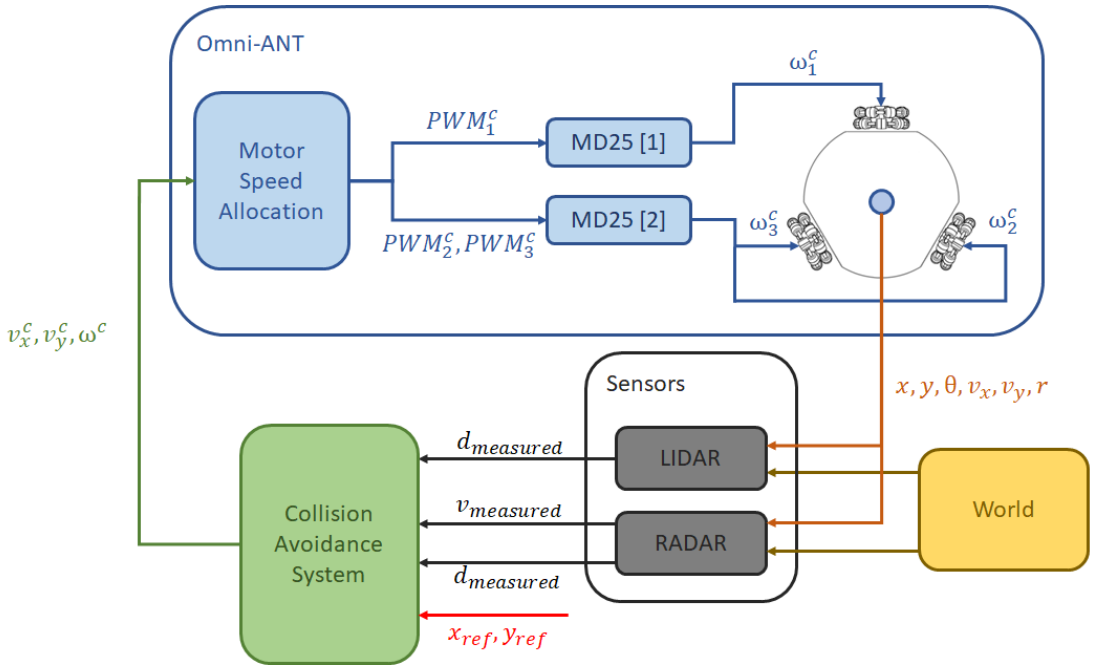


Figure 3.1: Physical implementation diagram

3.1.1 Omni-ANT Platform

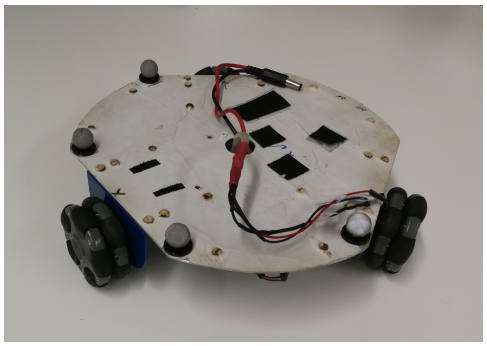


Figure 3.2: Omni-ANT platform

The Omni-ANT platform is an omnidirectional mounting platform equipped with a set of 3 58mm omnidirectional wheels, each powered by an EMG30 motor controlled by two MD25 motor controller boards. Both the devices were developed by Robot Electronics (a trading name of the company Devantech Limited). The main specifications for these devices are shown in tables 3.1 and 3.2.

This prototype was developed as a baseline for autonomous robotics projects, allowing for the integration of various devices such as sensors and actuators as well as small processing units to control these and the platform's motors, if necessary. It is also equipped with passive infrared markers, allowing for the location of the robot by external infrared cameras. A detailed representation of the Omni-ANT platform is shown in figure 3.3.

[1]When coupled with the manufacturer's EMG30 gear motor model, unavailable otherwise

Table 3.1: EMG30 motor specifications

EMG30 Gear Motor	
Rated Voltage	12 V
Rated Torque	1.5 kg/cm
Rated speed	170 rpm
Minimum Speed	1.5 rpm
Maximum Speed	200 rpm

Table 3.2: MD25 board specifications

MD25 Motor Controller Board	
Input Voltage	12 V
Available motor slots (JST ports)	2
Communication Protocols	Serial, I2C
Noise Suppression	Available[1]
Acceleration Regulation	Available

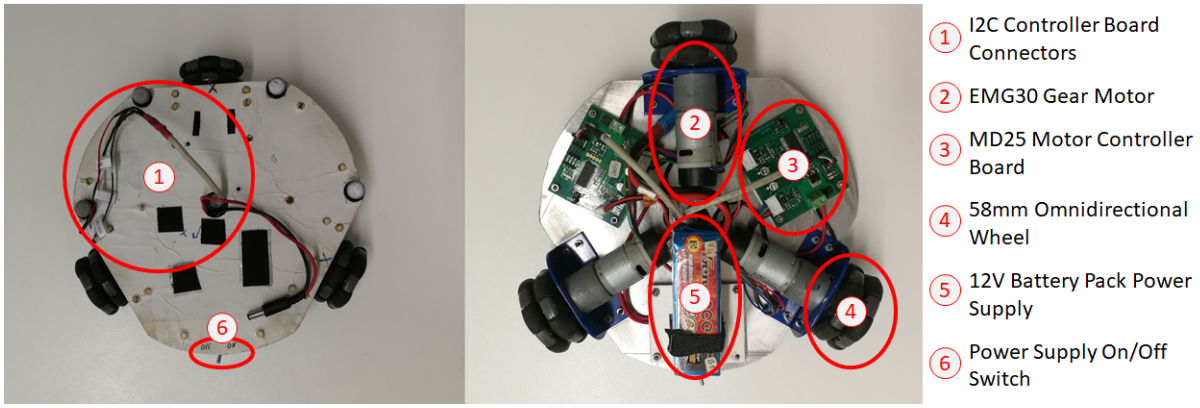


Figure 3.3: Omni-ANT assembly (top view (left) and bottom view (center))

In the following sections, a detailed description of the electronic devices integrated in and controlling this platform to simulate an autonomous robot is presented.

3.1.2 Raspberry Pi

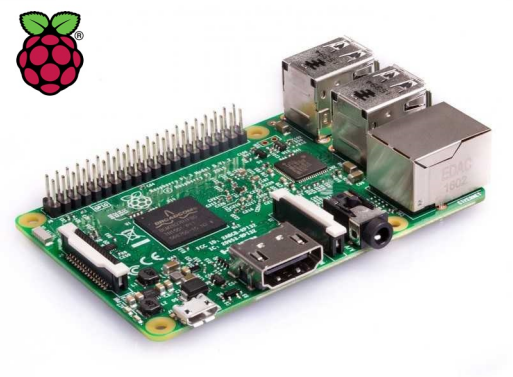


Figure 3.4: Raspberry Pi 3 Model B V1.2[2]

The Raspberry Pi 3 Model B V1.2 (figure 3.4) by the Raspberry Pi Foundation is a small and low-cost educational platform designed to be used in a wide range of applications, most commonly robotics and automation. This single-board computer has a high processing power for its small dimensions, being capable of executing programs autonomously and acting as a master device managing slave interfaces

[2]adapted from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

such as sensors and actuators as well as other processing units. The device's main specifications are presented in table 3.3.

Table 3.3: Raspberry Pi 3 Model B V1.2 main specifications

Raspberry Pi Model B V1.2 Main Specfications	
Processor	Quad Core 1.2GHz Broadcom BCM2827 64bit CPU
Internal Memory (RAM)	1 GB
GPIO[3] Interface	40-pin extended GPIO
USB ports	4 USB 2 ports
Video display output	HDMI
Storage	Micro SD Port
Power Supply	Switched micro USB 3-5V / 2.5A supply port

3.1.3 URG-04LX-UG01 LIDAR sensor

Figure 3.5: URG-04LX-UG01 LIDAR scanner[4]

The URG-04LX-UG01 (figure 3.5) is one among many LIDAR sensor models developed by Hokuyo Automatic Co.. Although being a more affordable option when compared to more advanced models by the same manufacturer, it poses a significant limitation to its applications. While other models have the capability of performing planar as well as elevation scanning (3 dimensions), the URG-UG01-04LX is only capable of performing readings in the planar space (2 dimensions), being unable to measure elevation of detected targets.

Due to the specifications of this work, such a limitation is not an issue since the robot prototype's vertical dimensions are negligible and planar movement constraints require two-dimensional scans only in the plane in which movement occurs.

This sensor is an FMPW (Frequency Modulated Pulsed Wave) LIDAR system, capable of measuring distance to irradiated targets without velocity sensing capabilities. The principle of measurement used in this device is based on the calculation of phase difference between the emitted and received pulse rather than time-of-flight (TOF), which minimizes the influence of an object's color and

[3]General Purpose Input/Output
[4]adapted from https://www.roscomponents.com/576-thickbox_default/urg-04lx-ug01.jpg

reflectance and effectively stabilizes the readings obtained.

The main specifications of this device are presented in table 3.4.

Table 3.4: URG-04LX-UG01 main specifications

URG-04LX-UG01 main specifications	
Scan area	240°
Wavelength	785 nm
Maximum range	4095, 5600 mm (2 modes)
Pitch angle	0.36°
Angular Resolution	0.36° (683 steps)
Radial Resolution	1 mm
Measurement accuracy	$\pm 3\%$ of measurement
Safety specifications	Laser Class 1[5]
Power Supply	5V DC
Connection interfaces	USB

3.1.4 TI mmWave AWR1642BOOST Radar sensor

Texas Instruments (TI) is a world renowned company that designs and manufactures several semiconductors and integrated circuits used by electrical appliances manufacturing companies all over the world. Among these devices, several models of sensors are available for a wide range of applications such as temperature, current and range sensing. Within the scope of range sensing, TI stands out as one of the main manufacturers of millimeter-wave (mmWave) sensors which enable range and angle as well as velocity sensing (through Doppler shift mechanics explained in section 2.1.1). The RADAR model used in this implementation is the AWR1642BOOST evaluation module (Fig. 3.6), a sensor belonging to the mmWave family. **This sensor is an FMCW (Frequency Modulated Continuous Wave) RADAR system, capable of providing location as well as velocity data from either static or moving obstacles**.

Figure 3.6: TI mmWave AWR1642BOOST evaluation board[6]

This model's main specifications are presented in table 3.5.

[5] [32]

[6] http://www.ti.com/tool/AWR1642BOOST?keyMatch=AWR1642BOOST&tisearch=Search-EN-everything&usecase=part-number#1

Table 3.5: AWR1642BOOST evaluation Module main specifications

AWR1642BOOST Evaluation module main specifications	
Microprocessor	Cortex-R4F
On-chip memory	1.5 MB
Power Supply	5.1V, 3A
Power Supply Jack	2.1mm barrel jack
Data Output Interfaces	USB UART, LVDS, CAN, GPIO, I2C[7]
Antenna Frequency	76-81 GHz
Transmitted wave frequency	4 GHz
Transmitter and receiver channels	2TX, 4RX

This sensor offers the possibility of utilizing a feature of the sensor's processing unit itself to run a clustering algorithm to group similar readings and publishing them as estimated coordinates and radial velocity values for the whole cluster as opposed to publishing the bulk of the raw data obtained by the sensor. Additionally, the sensor can be set to discard static object readings, restricting the publishing information to non-static objects.

3.1.5 Robot Operating System

Robot Operating System (ROS)[8] by the Open Source Robotics Foundation is a powerful framework designed to aid in the integration and communication between multiple electronics and robotics systems in a common setup or network.

Robotics projects integrating a wide variety of peripherals such as sensors and motors can become extremely complex and require robust algorithms for data transfer and general communications. Since most of the devices used in such projects are usually incompatible out-of-the-box or require specific additional modules (usually provided separately by the manufacturer) to be easily integrated into a prototype, the task of assembling a single platform to act as a master entity to each of these peripherals can become increasingly complicated.

ROS is a collection of libraries and tools designed to simplify data transfer between several separate devices such as sensors and actuators. A ROS network is constituted by a variety of entities:

- **Nodes:** The most basic structure of a ROS network. Nodes can be programmed to execute basic tasks such as reading data from a sensor and providing the raw data to the network or more complex tasks such as controlling motor inputs or performing localization or path planning. Each node can have publishing and/or subscribing capabilities, allowing it to publish relevant data to the network and/or subscribe to other nodes to receive data published by them. These publisher and subscriber nodes will remain active for the duration of their execution, publishing data or subscribing to other topics independently of whether any other nodes are subscribed to them or

[7] USB UART data output channel available out-of-the-box, other data channels require physical modification of the module
[8] https://www.ros.org

publishing the data they are subscribed to. Multiple publisher nodes can publish data to the same topic as well as multiple subscriber nodes can be subscribed to the same topic;

- **Services:** Integrated in nodes, services are defined by a pair of messages, a requesting message and its reply. Since simple publisher nodes are designed for many-to-many type communications, these will actively publish data or subscribe to other topics continuously. Instead, services are designed to act in a request/reply configuration, requiring a requesting message to be executed and sending a reply with relevant information to the requester.

- **Topics:** Similarly to channels, topics are identifiers under which messages will be published. Each topic is characterized by a name, usually defined according to the specifications of the message being published within. These identifiers grant a degree of anonymity to the active nodes on a ROS network, decoupling the production of information from its consumption. While nodes may not be aware of which other nodes they may be communicating with, they are still able to acquire relevant data from a known topic. This results in a high versatility and modularity of ROS nodes;

- **Messages:** Publisher nodes publish data under a specific topic in the form of messages. These messages are characterized by a specific structure defined by the publisher. The ROS libraries contain several message structures available for specific data types such as point cloud data or temperature readings. Any subscribing node requires the knowledge of the message structure to be able to process it. While nodes are not directly able to determine the published message structure and adapt accordingly, active topics can be queried by developers for message details to design subscriber nodes;

- **Master:** A ROS network requires a master device running a service known as *roscore*. This master device will handle essential tasks such as naming and registration services to the rest of the nodes in the network, tracking publishers, subscribers, topics and their associated nodes. This allows for nodes to be able to locate one another and establish peer-to-peer communication through a topic.

- **Bags:** Bags act as storage bins, where a tool known as *rosbag* records, plays back or even publishes the stored data within a bag file.

The ROS framework is highly user-friendly, providing a number of useful tools to perform some of the most common operations when developing software. A list of the most commonly used commands is listed below:

- ***roscore:*** This command starts the core process, defining the executing machine as a master within its network;

- ***roslaunch:*** Tool for running ROS packages containing multiple nodes with a single command. A *".launch"* file is used as an input to this command, within which the required parameters are stored and the desired nodes and packages are specified to be run within a single instance.

- ***rosnode:*** Used for obtaining information about active nodes within a network such as topics subscribed/publishing to;

- ***rostopic:*** Used for obtaining information about active topics within a network, such as their message type and interacting nodes or echoing published messages;
- ***rosrun:*** Tool used for running standalone nodes or editing packages and files contained within a package.

An example of a simple ROS network is shown in figure 3.7:

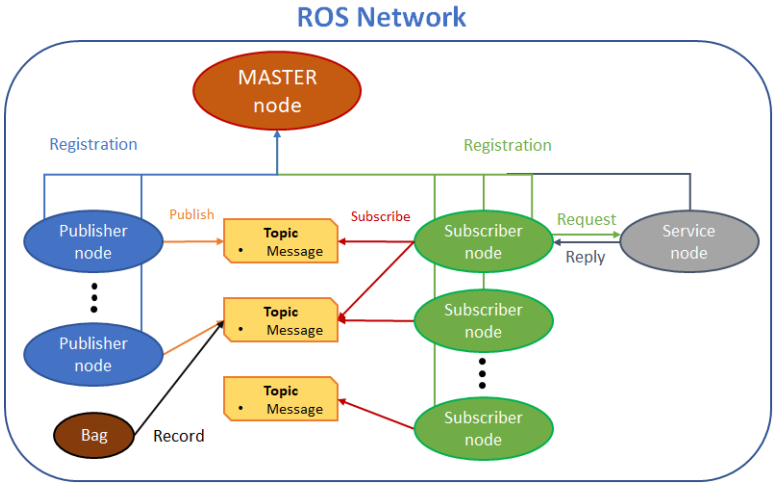


Figure 3.7: ROS network example

ROS also contains a set of built-in visualization tools, *RViz*. *RViz* can be used to represent geometry data, being capable of subscribing to topics and plotting 2D or 3D point clouds received within messages under that topic in 3D space. This tool is useful for the visual representation of sensor readings and algorithms such as path planning, localization or SLAM (Simultaneous Localization And Mapping).

ROS latest distributions are available in official stable releases for OS X, Linux-based operating systems such as Ubuntu as well as Debian or Debian-based operating systems such as Raspbian. Limited tools are also available for Windows.

3.2 Omni-ANT modelling

In this section, the numerical model of the Omni-ANT is derived, detailing the kinematics of the specific assembly of the platform as well as the dynamics of the motion system.

3.2.1 Kinematics

With the established assembly of the Omni-Ant platform in [10], the determination of the kinematic model of the prototype is necessary.

The definition of the physical model of the three-wheeled omnidirectional vehicle depends on its wheel configuration. The wheel configuration diagram for the Omni-Ant platform is shown in figure 3.8.

The vehicle's wheels are paired with the corresponding numbered motor (wheel n being powered by motor n). Due to the omnidirectional nature of the wheels and their configuration, the prototype is

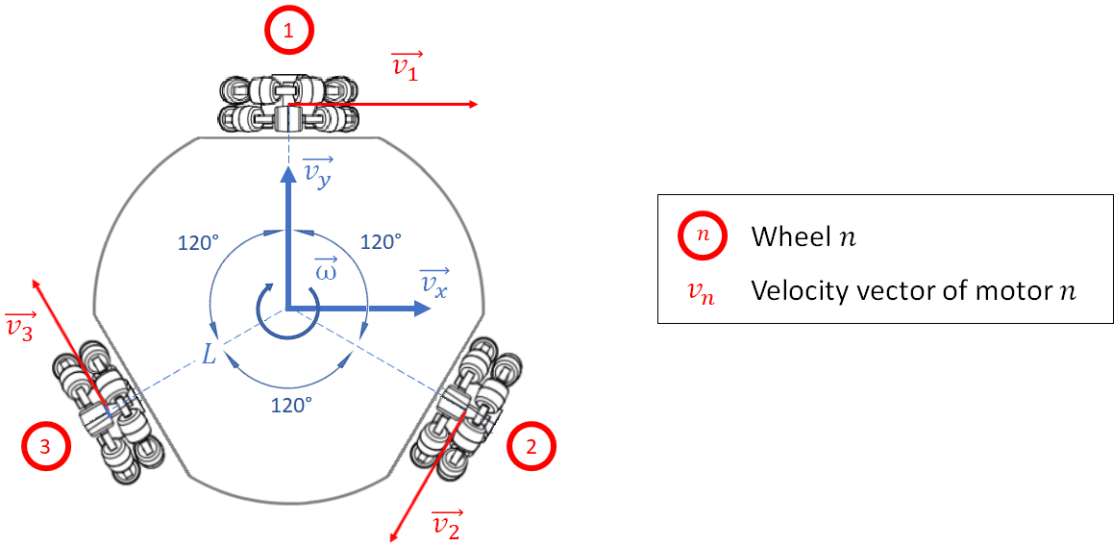


Figure 3.8: Omni-ANT platform wheel configuration (top view)

able to move in any direction without having to change its orientation. This translates into a holonomic behaviour, where the only constraints it is subjected to are positional constraints of the form (the robot is a rigid body, thus nondeformable and, as such, the relative position of its components are constrained to their initial state). With this configuration, the kinematics equations can be determined. By applying trigonometric relations, the translational and rotational movement relations in the local reference frame of the prototype are given by:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} 1 & -\cos 60° & -\cos 60° \\ 0 & -\tan 30° & \tan 30° \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{3.1}$$

where v_x is the x direction component of the velocity vector of the platform, v_y the y direction component of the same vector and v_n the linear velocity of wheel n, with $n = 1, 2, 3$. Assuming the no-slip condition, the linear velocity of a single wheel is given by:

$$v_n = \omega_n r_n \tag{3.2}$$

where r_n is the radius of wheel n and ω_n the angular velocity of the motor powering wheel n. As such, by combining equations (3.1) and (3.2), the relation between the robot's linear velocity components and the angular velocity of each of its wheels is given by:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} r_1 & -\frac{r_2}{2} & -\frac{r_3}{2} \\ 0 & -\frac{r_2\sqrt{3}}{2} & \frac{r_2\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.3}$$

Due to the holonomic nature of this wheel configuration, in addition to translational movement the plat-

form is capable of egocentric rotational movement. The angular velocity of the platform is given by:

$$\omega = \frac{v_1 + v_2 + v_3}{3L} \Leftrightarrow \omega = \frac{\omega_1 r_1 + \omega_2 r_2 + \omega_3 r_3}{3L} \tag{3.4}$$

where ω is the rotational speed of the platform and L is the distance between the rotational center and each of its wheels. Assuming the three wheels are identical ($r_1 = r_2 = r_3$), the result can be further simplified. Thus, the local kinematic model of the robot is given by:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = r \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.5}$$

To obtain the linear and angular velocity relations in the fixed reference frame, a coordinate transformation is required. As such, the rotational transformation from the robot (local) reference frame to the fixed reference frame given by:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = T' \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad , \quad T = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

is applied, where θ is the rotation of the local reference frame in relation to the fixed reference frame

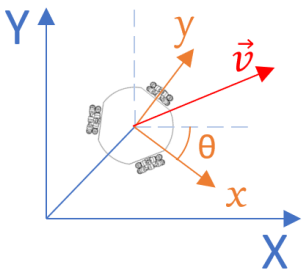


Figure 3.9: Local and fixed reference frames

(vehicle attitude, figure 3.9). As such, the relation between linear and angular velocities of the robot in the fixed reference frame and the angular velocity of the wheels is given by :

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = T' r \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = r \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.7}$$

where $\dot{\theta}$ is the rate of change of vehicle attitude in the fixed frame (equal to the angular velocity of the robot in its local reference frame).

3.2.2 EMG30 motor dynamics

In this section, the dynamics of the EMG30 motors used in the prototype is modelled. The EMG30 motor is an electromechanical component comprised of three main components that compose the acronym that designates this model, an encoder, a motor and a gearbox (with a 30:1 reduction factor). The modelling process is aimed at modelling the conversion of input voltage to the input current of the motor and subsequent conversion of this input current to the rotation speed of the motor.

The converter output can be defined by:

$$U = e + Ri + L\frac{di}{dt} \tag{3.8}$$

where U is the converter input voltage, e the back electromotive force (emf) voltage, R the resistor resistance, L the inductance and i the motor current.

The torque provided by the motor is not fully transmitted to the wheels due to being subjected to dissipative forces on the shaft. As such, the torque applied to the load (fraction of the total torque which will be provided to the wheels) can be defined by:

$$T_L = T_d - T_c - B\omega \tag{3.9}$$

where T_L is the torque provided by the motor to the load, T_d the developed torque, T_c the static friction torque and $B\omega$ the viscous friction (proportional to the angular speed of the motor, ω, by a factor B [33]).

Furthermore, the developed torque, the emf of the motor and the load torque can be correlated with the input current and developed angular speed of the motor by [33]:

$$T_d = K_s i \tag{3.10}$$

$$e = K_s \omega \tag{3.11}$$

$$T_L = J\dot{\omega} \tag{3.12}$$

where i is the supplied current, K_s the current to developed torque conversion factor and J the motor moment of inertia.

Nonlinear factors such as static friction, T_c, can be decoupled from the system and modelled as nonlinearities [34]. As such, combining equations (3.8)-(3.12) the linear dynamics model of the motor can be expressed by:

$$\dot{i} = -\frac{Ri}{L} - \frac{K_s\omega}{L} + \frac{U}{L} \tag{3.13}$$

$$\dot{\omega} = \frac{K_s i}{J} - \frac{B\omega}{J} \tag{3.14}$$

The state space model can then be derived. Considering as relevant output the motor angular velocity

ω, the motor system is defined as:

$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{K_s}{L} \\ \frac{K_s}{J} & -\frac{B}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} U \tag{3.15}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} \tag{3.16}$$

System identification is required to estimate the model parameters in order to achieve a valid model of the motor. Since this process is not within the scope of this work, the system identification performed by Gonçalves et al. [35] is used as reference. Table 3.6 contains the estimated motor parameters.

Table 3.6: Estimated motor parameters [35]

EMG30 estimated motor parameters	
Current-developed torque conversion factor - K_s	0.509
Inductance - L	$3.4 * 10^{-3} H$
Resistor - R	$7.101\ \Omega$
Viscous friction factor B	$9.31 * 10^{-4}$
Static friction factor- T_c	0.04
Motor moment of inertia - J	$5.67 * 10^{-3} kgm^2$

Due to input specifications of this electromechanical system, reference motor speeds need to be converted to voltage inputs for the motor to achieve the intended rotation speed. By the process of identification by Gonçalves et al. [35], the voltage converter estimated model is given by:

$$\omega(t) = \frac{a}{b}(1 - e^{-bt}) \tag{3.17}$$

where:

$$a = \frac{K_s U(t) - RT_c}{RJ} \tag{3.18}$$

$$b = \frac{K_s^2 + RB}{RJ} \tag{3.19}$$

Furthermore, the model is characterized by nonlinearities in the transmission of signals as well as safety thresholds to avoid malfunction of any component. Voltage signals are limited to the [-12;12] V range and any value outside this range is limited to the closest value within this range. Additionally, a dead zone nonlinearity can be identified below the motor's minimum required power to ensure motion of the shaft due to friction. Finally, as the scale in which these signals are evaluated is not continuous but rather discrete, a continuous voltage signal is quantized when received, resulting in a discrete scale of received voltage values. The real motor model can be described by the block diagram depicted in figure 3.10.

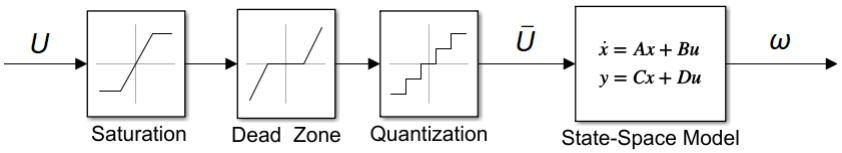


Figure 3.10: Motor block diagram

3.3 Data processing algorithms

In this section, the obstacle detection and collision avoidance system algorithms used in the computation of data within the various ROS nodes in execution are presented, stating the principles behind these methods and providing a mathematical breakdown of these tasks. A description of the odometry method is also presented. Figure 3.11 depicts the block diagram of the collision avoidance system block mentioned in figure 3.1. $\vec{v}$ and $\vec{u}$ are detection velocity and position vectors, respectively, $\vec{c}$ and r are the detected clusters' centroid position vector and radius and v_x, v_y and ω are the robot velocity commands in x, y and rotation components, respectively.

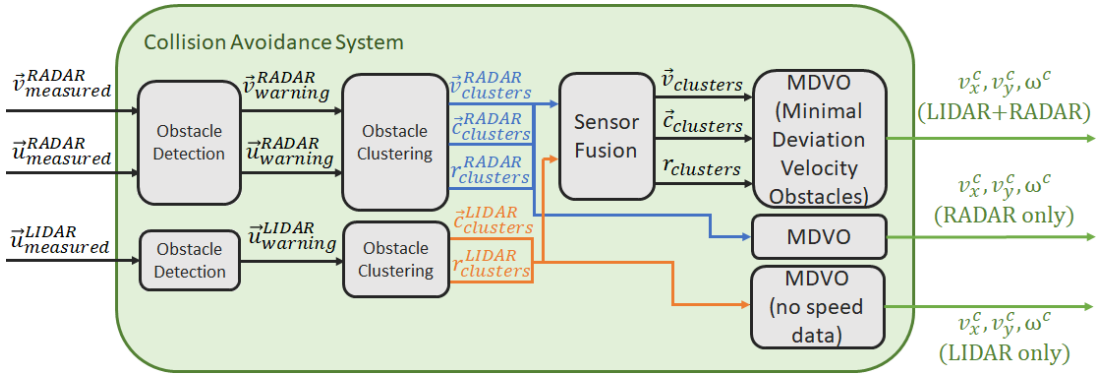


Figure 3.11: Collision avoidance block diagram

3.3.1 Obstacle definition

Preprocessing the raw sensor data is required in order to maximize the efficiency of the obstacle avoidance algorithm, as well as providing a more complete description of the surrounding environment through sensor fusion. Obstacle detection is limited to the robot's warning zone (see section 2.3, Vehicle safety). This zone, defined as **active avoidance zone**, is represented in figure 3.12.

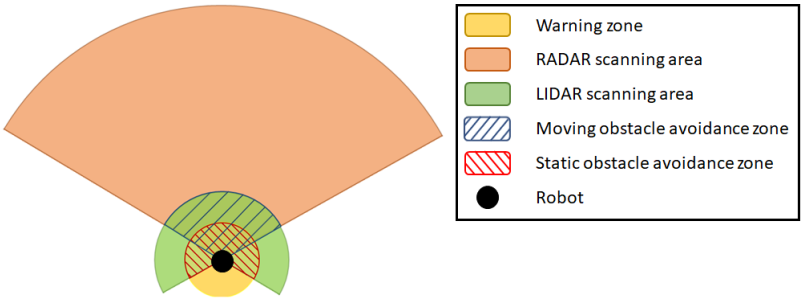


Figure 3.12: Active avoidance zones

Obstacle detection

Preceding the obstacle detection phase, the warning range definition is required to define at which distance an obstacle should be considered for analysis of an impending collision, which is coded into the algorithm as a static variable. **This range was set to $1m$ for static objects and no range restriction is applied to moving obstacles** (meaning every non-static obstacle, regardless of proximity, is checked for the possibility of a collision with the robot).

The obstacle detection process is initialized when sensor data is received. RADAR detections are received as relative position vectors, while LIDAR detections are received as a proximity value. As such, static RADAR detections that satisfy the following range condition are flagged as warning detections:

$$\|\vec{u}_m^{RADAR}\| \leq 1\ [m] \tag{3.20}$$

where $\vec{u}_m^{RADAR}$ are the RADAR detection position vectors, relative to the sensor, with m the index of the detection vector received. All non-static detections are flagged as warning detections, regardless of proximity.

Since the LIDAR sensor outputs detections as the proximity values of each detection, these can be directly scanned for warning detections. As such, LIDAR detections that satisfy the following condition are flagged as warning detections:

$$d_n^{LIDAR} \leq 1\ [m] \tag{3.21}$$

where d_n^{LIDAR} are the proximity values of LIDAR detections, relative to the sensor, with n the index of the detection value received. Then, since LIDAR scanning is discrete, by knowing the angular resolution of the LIDAR sensor (section 3.1.3) and the reading index, detection position vectors can be extracted:

$$u_y = \sin\left(-k\theta_{resolution} + 210^\circ\right) \tag{3.22}$$

$$u_x = \cos\left(-k\theta_{resolution} + 210^\circ\right) \tag{3.23}$$

$$\vec{u}_k^{LIDAR} = d_k \cdot (u_x, u_y) \tag{3.24}$$

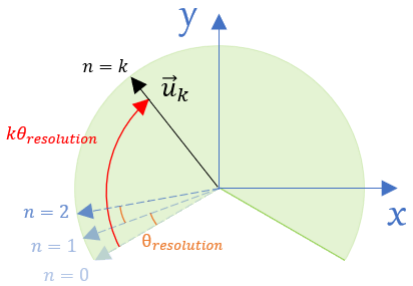


Figure 3.13: LIDAR diagram

where k is the detection index ($k = 0, 1, 2, ...$), $\theta_{resolution}$ the LIDAR angular resolution, d_k the proximity value of detection k and $\vec{u}_k^{LIDAR}$ the resulting position vector of detection k, relative to the sensor (figure 3.13). Any detections which verify the previous conditions are extracted and the remaining data

is discarded. Figure 3.14 depicts an example of an obstacle detection scene. The warning detection data is then fed to the clustering algorithm.

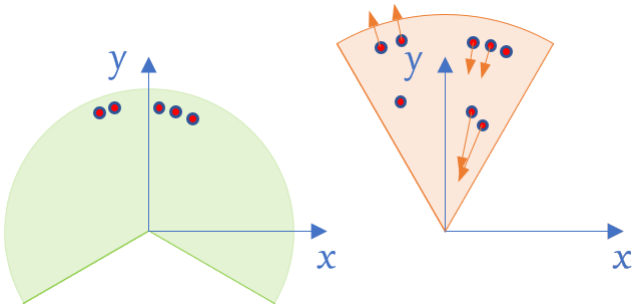


Figure 3.14: Obstacle detection scene example. LIDAR (left) and RADAR (right) sensor observation zones.

Clustering

The clustering algorithm is responsible for grouping the relevant raw range and velocity readings into groups to minimize processing time. A clustering algorithm was developed to allow for the integration of the standalone LIDAR and RADAR sensors as well as a sensor fusion approach, discarding the available clustering algorithm provided by the manufacturer in the case of the RADAR sensor.

Since the prototype is intended to detect obstacles on an unknown environment, the clustering algorithm is based on the proximity between detections. A threshold of $10cm$ is defined, meaning any pair of detections within this distance of each other will be grouped into the same cluster. The conditions that define this process are given by:

$$\left\|\vec{u}_i - \vec{u}_j\right\| \leq 10\ [cm] \tag{3.25}$$

$$\vec{c}_{cluster} = \frac{\sum_{n=1}^{o} \vec{u}_n}{o} \tag{3.26}$$

$$r_{cluster} = max(\|\vec{c}_{cluster} - \vec{u}_o\|) \tag{3.27}$$

$$\vec{v}_{cluster} = \frac{\sum_{n=1}^{o} \vec{v}_n}{o} \tag{3.28}$$

with i, j the warning detection position indexes ($i \neq j$), o the indexes of obstacle detections in a distinct cluster, $\vec{c}_{cluster}$ the cluster centroid, $\vec{v}_{cluster}$ the cluster velocity vector and $r_{cluster}$ the cluster radius. As such, warning detections within the minimum threshold distance from each other (3.25) are grouped into distinct clusters, computing the centroid (3.26), velocity (3.28) and radius ((3.27), the maximum distance from the cluster's centroid to any of the points assigned to said cluster) of each cluster afterwards. It is worth mentioning that the computation of the cluster data is initialized only when every detection has been assigned to a cluster. **Equations** (3.25)**-**(3.28) **are applied to RADAR warning detections, while only** (3.25)**-**(3.27) **are applied to LIDAR data** (due to the abscence of velocity data). Figures 3.15 and 3.16 depict the application and result of the clustering algorithm applied to the example in figure 3.14.

RADAR detected clusters are represented in purple and LIDAR detected clusters are represented in cyan.

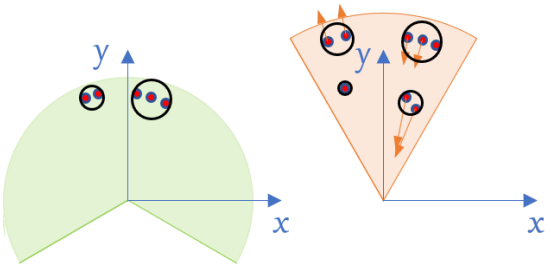


Figure 3.15: Detection grouping example.

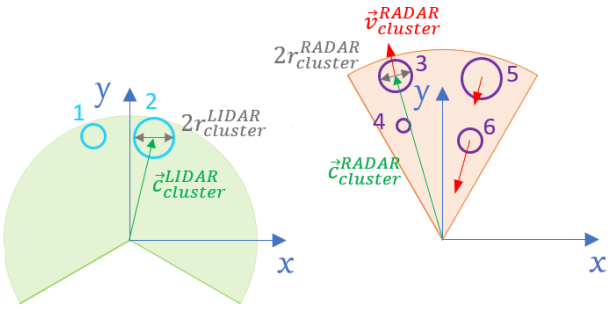


Figure 3.16: Clustering algorithm result.

In cases where only one of the sensors is active, cluster data is transmitted to the obstacle avoidance algorithm. However, if both LIDAR and RADAR sensors are active, a sensor fusion algorithm is applied prior to obstacle avoidance.

Sensor fusion

A sensor fusion algorithm was developed as a way to further complement the information of each of these sensors, allowing for a more accurate and complete description of the environment. Following the clustering of the RADAR and LIDAR data, moving objects characterized by the RADAR velocity readings are paired with their LIDAR cluster counterparts based on a threshold of said cluster's radius:

$$\|\vec{c}^{RADAR}_{cluster\ p} - \vec{c}^{LIDAR}_{cluster\ q}\| \leq r^{LIDAR}_{cluster\ q} \tag{3.29}$$

where $\vec{c}^{RADAR}_{cluster\ p}$ and $\vec{c}^{LIDAR}_{cluster\ q}$ are the centroids of RADAR detected cluster p and LIDAR detected cluster q, respectively, with p and q the RADAR and LIDAR cluster indexes, respectively, and $r^{LIDAR}_{cluster\ q}$ the radius of LIDAR detected cluster q. With this condition, obstacles with large dimensions characterized by LIDAR detected clusters with a large radius may be associated with multiple RADAR clusters due to the RADAR's wider angular resolution. As such, each detected cluster's velocity vector is based on an average of all the RADAR detected velocity vectors associated with said cluster:

$$\vec{v}_{cluster} = \frac{\sum_{n=1}^{r} \vec{v}^{RADAR}_{cluster\ n}}{r} \tag{3.30}$$

where r are the indexes of the RADAR detected velocity vectors associated with said cluster. With the pairing of the RADAR and LIDAR clusters, the cluster data fed to the obstacle avoidance algorithm is each cluster's centroid, $\vec{c}^{LIDAR}_{cluster}$, its respective radius, $r^{LIDAR}_{cluster}$, and velocity vector, $\vec{v}_{cluster}$ (from (3.30)). In the case of dynamic clusters (detected by the RADAR sensor) outside the LIDAR's observation zone (which may not be associated with any LIDAR clusters), the transmitted data pertaining to said clusters is instead their centroid, $\vec{c}^{RADAR}_{cluster}$, their radius, $r^{RADAR}_{cluster}$, and the velocity vector, $\vec{v}^{RADAR}_{cluster}$. Figure 3.17

depicts the result of sensor fusion applied to the example in figure 3.16.

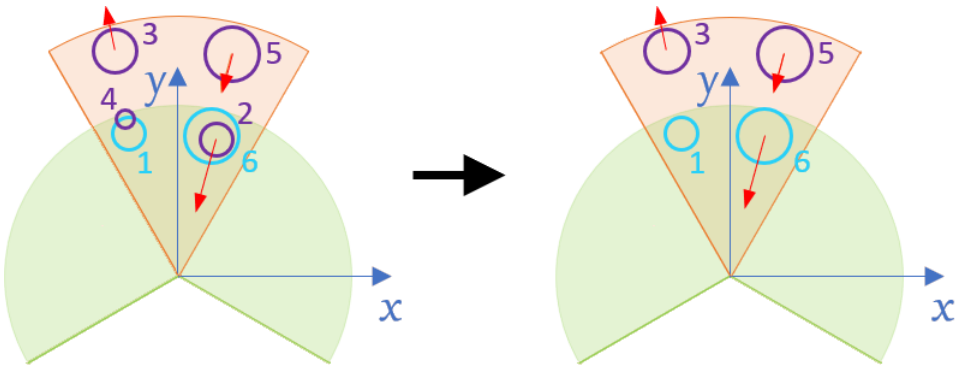


Figure 3.17: Sensor fusion result

3.3.2 Collision avoidance algorithm

The developed prototype is intended to navigate an unknown environment populated by obstacles towards a waypoint defined by the user. When encountering obstacles in its vicinity, the robot is intended to execute a maneuver which will result in the avoidance of any obstacles in its vicinity, subsequently resuming its path towards the end point. The **velocity obstacles** approach was selected as the most suitable approach to this problem (between the algorithms explored in section 2.3) due to the availability of obstacle velocity readings, provided by the RADAR sensor. Due to the ability to ”predict” if a collision may occur in the future with a moving obstacle in its current trajectory, this constitutes a significant advantage compared to other algorithms which may otherwise assume these obstacles are static and will remain within the robot's trajectory, always requiring an avoidance control to avoid said obstacle. The decision making process of this algorithm (the choice of which path to choose based on the current scenario) is aimed at minimizing the processing load on the robot's processor. This choice is based on the robot's otherwise unobstructed path, where the avoidance path chosen among the possible paths aims to minimize the deviation from its course. Thus, this algorithm is denominated **minimal deviation velocity obstacles**.

Minimal deviation velocity obstacles method

Based on the velocity obstacles approach, this algorithm builds the instantaneous collision cones based on the obstacles present in the current scene. Only obstacles within the robot's warning zone (see section 2.3, Vehicle safety) and at least one of the sensor's field of view (see figure 3.12) are considered for this step.

As an initialization step, a **set of feasible controls** is defined, containing a discrete set of velocity vectors in every direction, centered and relative to the robot, and a fixed norm (defined by a set cruise speed) to be used as robot commands. In the processing phase, the algorithm starts by defining the collision cone of each obstacle, containing all robot velocity vectors that would lead to a collision in the future, **assuming the obstacles are static**. This process is done by analyzing the cluster information

provided by the obstacle definition algorithm (section 3.3.1). The angular thresholds of the collision cone associated with a cluster are given by:

$$\begin{aligned} \beta &= \arccos\left(\frac{\vec{c}_{cluster}\cdot(0;1)}{\|\vec{c}_{cluster}\|\cdot\|(0;1)\|}\right) \\ \alpha_{left} &= \beta + \arctan\left(\frac{r_{cluster}}{\|\vec{c}_{cluster}\|}\right) \\ \alpha_{right} &= \beta - \arctan\left(\frac{r_{cluster}}{\|\vec{c}_{cluster}\|}\right) \end{aligned} \tag{3.31}$$

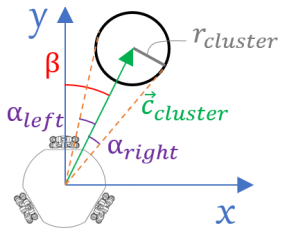


Figure 3.18: Collision cone threshold diagram

Figure 3.19 is an example of a scene where the Omni-ANT is navigating an environment containing both static and moving obstacles within its active avoidance zone. The collision cones relative to each of the obstacles are represented, as well as the velocity vectors of the robot and obstacles (orange vectors). Two examples of feasible robot trajectories outside of its collision cones are presented in figure

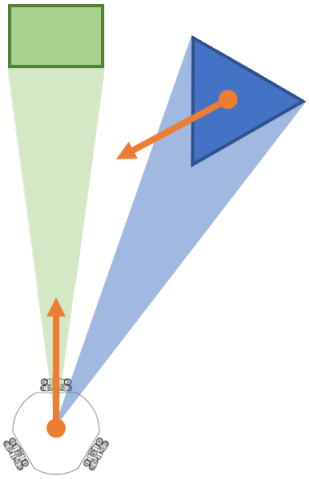


Figure 3.19: Velocity obstacles scene(static obstacle, rectangle, and dynamic obstacle, triangle)

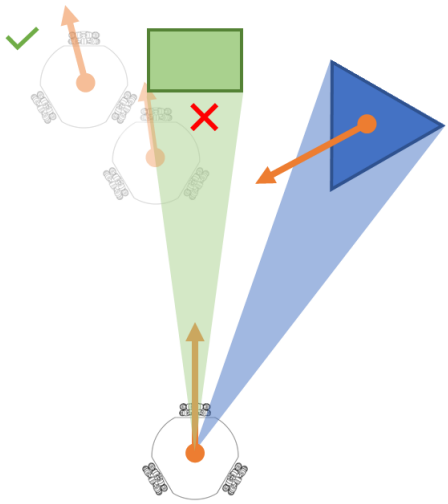


Figure 3.20: Avoidance controls example

3.20. As evidenced, avoidance controls (robot commands that ensure avoidance of obstacles) in the vicinity of the collision cone thresholds may result in collisions if these are built based on a point-particle model of the robot (rigid body defined by its mass concentrated in a single dimensionless point). As such, **the robot's dimensions must to be taken into account when determining the real collision cones**. Thus, a transformation is applied beforehand to correctly build the collision velocity space. The

application of a safety factor is also recommended due to the limited precision and noise of the sensing system. Let α_{left} and α_{right} be the left and right collision cone angular thresholds of a given obstacle, respectively. The corrected collision cones can be obtained through:

$$\begin{aligned} \alpha_{left\ corrected} &= \alpha_{left} - S_F R_{robot} \\ \alpha_{right\ corrected} &= \alpha_{right} + S_F R_{robot} \end{aligned} \tag{3.32}$$

where S_F is the safety factor ($S_F \geq 1$), $\alpha_{left\ corrected}$ and $\alpha_{right\ corrected}$ the corrected left and right collision cone angular thresholds, respectively. Figure 3.21 is a visual representation of this correction. Following this correction and, **contrary to the standard velocity obstacles approach in which the**

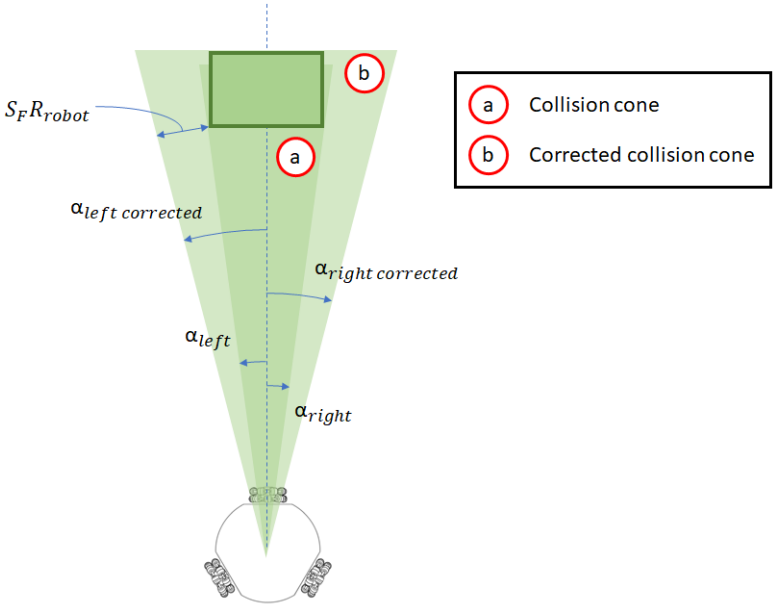


Figure 3.21: Collision cone correction

collision cones of non-static obstacles are displaced by the associated velocity vector, this algorithm will analyze each of these collision cones in separate. This results in the need to preserve solely the collision cone thresholds and the obstacle velocity vectors as opposed to the standard approach, which requires the computation of the feasible avoidance space. This decision was made to minimize the computational load and processing time. **However, the velocity vector of each of the obstacles must be subtracted to the feasible control prior to being tested:**

$$\vec{v}_{mod} = \vec{v}_{feasible} - \vec{v}_{cluster} \tag{3.33}$$

The orientation of the resulting vector is then compared with the angular thresholds of the detected obstacles. If a feasible control is found to result in an impending collision with any obstacle, it will be flagged as a non-avoidance control and removed from further comparisons. Thus, for a feasible control

to be considered an avoidance control, one of the following conditions must be satisfied:

$$\begin{aligned} \alpha_{left\ corrected} &\geq sgn(\vec{v}_{mod\ x}) \arctan\left(\frac{\vec{v}_{mod\ x}}{\vec{v}_{mod\ y}}\right) \\ \alpha_{right\ corrected} &\leq sgn(\vec{v}_{mod\ x}) \arctan\left(\frac{\vec{v}_{mod\ x}}{\vec{v}_{mod\ y}}\right) \end{aligned} \tag{3.34}$$

with $\vec{v}_{mod\ x}$and $\vec{v}_{mod\ y}$ the x and y components of the modified feasible vector $\vec{v}_{mod}$ being tested, respectively (figure 3.22).

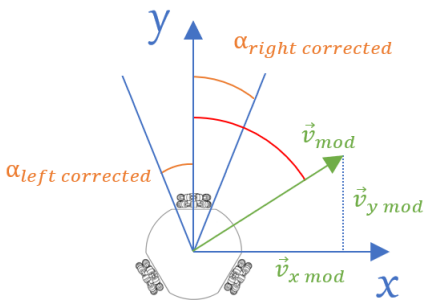


Figure 3.22: Avoidance control test diagram

Applying the collision cone correction to the example in figure 3.19, figure 3.23 depicts a comparison between the standard velocity obstacles method and the method developed in this work (minimal deviation method). Obstacle velocity vectors are represented in orange, feasible control avoidance vectors are represented in green and control vectors which would result in collisions are represented in red. The avoidance space (possible robot trajectories outside of any collision cones) of the traditional velocity obstacles method is represented in gray.

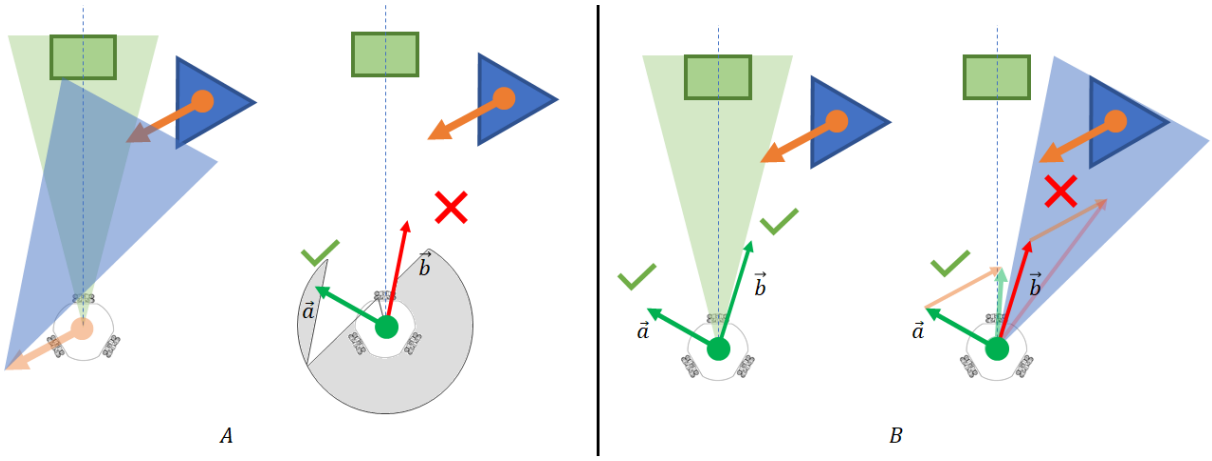


Figure 3.23: Feasible avoidance control test. Standard velocity obstacles (A) and minimal deviation method (B)

As evidenced, both methods are similar and effective at determining which feasible controls enable the robot to avoid collisions with any obstacles in the near future (command $\vec{a}$ is found to be a feasible

avoidance command and $\overrightarrow{b}$ is not suitable for avoidance). Thus, the minimal deviation method is used due to the lower computational load. Finally, the set of feasible avoidance controls (robot commands that ensure avoidance of every obstacle in the scene) is scanned for the control that minimizes the deviation from the robot's current path to ensure a smoother trajectory, selecting the avoidance vector that satisfies the condition:

$$min(\angle(\vec{v}_{robot}, \vec{v}_{avoidance})) \tag{3.35}$$

where $\vec{v}_{robot}$ is the robot's current velocity vector and $\vec{v}_{avoidance}$ the avoidance vector being tested. If, in any case, a feasible avoidance control is not found, the robot will perform a reverse maneuver, moving in the opposite direction of its current trajectory until a feasible avoidance control is found.

The inability to map the environment and the unavailability of a reference map poses a few limitations to navigation using this algorithm, where despite avoiding collisions with obstacles, it might not be able to reach its intended waypoint and instead become trapped in a trajectory loop. Limitations concerning the sensing system are present as well, where the RADAR sensor will only be able to sense the radial component of a moving obstacle's velocity (the contribution of a moving obstacle's real velocity vector in the direction of the robot). This results in a modified feasible control space which, despite different from the real feasible space, will ensure the avoidance of obstacles.

3.3.3 Odometry

The ability of a robot to accurately estimate their location over time is very important. Such a task can prove very challenging with the abscence of knowledge of the surrounding environment and its features. One of the most effective ways of a robot to locate itself in an unknown environment without GPS signal is odometry.

Odometry is the ability of a robot to estimate its current location based on the robot's previous position with the use of relevant sensor data. This can be done with the use of proximity sensors such as the LIDAR and RADAR sensors used in this work or motion sensors such as encoders. The former are most effective when a map of the environment is available, allowing for the matching of sensed features with map features and inversely estimate the current location based on the robot's perspective.

Other methods such as *SLAM* (Simultaenous Location And Mapping) can be used when a map of the current scene is not available (example in figure 3.24). In this case, a map of the environment is gradually generated with sensor data, resulting in a partial or complete mapping of the surroundings, allowing for the previous method to be applied. However, such algorithms can be very computationally demanding due to optimization processes.

In this work, due to the limited computational resources of the processing unit, a *SLAM* algorithm approach was discarded. Likewise, a map of the current scene is not provided due to the specifications of a modular prototype intended to be easily integrated in a variety of different environments in which a map of the scene may not be available. Due to these constraints, odometry through motion sensor data is regarded as the most suitable option for robot location, making use of the rotary encoders integrated in

[9] adapted from https://lejosnews.files.wordpress.com/2017/07/control.jpg

Figure 3.24: Robot trajectory (red) and SLAM generated map example[9]

the prototype's motors. The EMG30's encoder is a rotary absolute encoder, providing information about the absolute angular position of the shaft and, therefore, the wheel it is connected to. These readings are provided in the form of pulse counts, which can be translated into fractions of a complete wheel turn if the full turn count is known.

With the knowledge of the absolute position of the wheels at each point in time, the robot location can be estimated. However, odometry algorithms are subject to limitations that contribute to errors in estimations, both when reading the encoder data or in the subsequent processing steps. **Slipping** can greatly affect the accuracy of location estimations, due to the possibility of wheel movement not corresponding to the actual robot's movement (e.g. sudden changes in wheel speed such as acceleration and sudden braking). This can be mitigated by control, allowing for a more controlled motor response and fidelity to the inputs. **Reading errors** can affect the odometry estimation as well, although to a lesser degree. Equal dimensions of all the wheels in a system is ideal to avoid errors, such as the wheels used in this prototype. However, the **sample time of the processing algorithm** can be the most influential factor of its precision. Figure 3.25 depicts two distinct robot trajectories and resulting encoder readings. In both scenes, at the end of the robot trajectory, the readings of both pairs of encoders are identical,

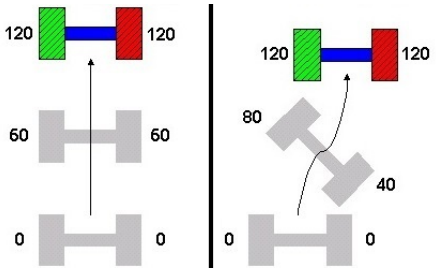


Figure 3.25: Encoder example: scene 1(left) and scene 2 (right)[10]

which translates into an identical absolute rotation in both pairs of wheels (in red and green). However, while scene 1 depicts a linear trajectory with the same inputs provided to both motors, scene 2 depicts an S-shaped trajectory, characterized by slight changes in motor speed. This results in two distinct end points characterized by the same encoder values. As such, the position of the robot cannot be correctly estimated (in most cases) using only the absolute encoder readings at the end of the robot's trajectory.

[10]https://groups.csail.mit.edu/drl/courses/cs54-2001s/images/Odometry.jpg

Thus, a **fast incremental odometry process is desireable**, processing data with an acceptable periodicity (sample time) and accounting for slight changes in motor speeds, which contributes to a higher degree of precision in estimations. This odometry method is designated as **dead reckoning**.

Dead reckoning odometry method

With the known encoder values, the prototype's kinematics model ((3.5), in section 3.2.1) can be used to estimate the robot's current location. By way of integration and applying transformation (3.6), robot position to motor rotation relation is obtained:

$$\begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} = r \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{bmatrix} \tag{3.36}$$

where X, Y, θ is the robot's pose in the fixed reference frame and ψ_i is the angular position of wheel i, with $i = 1, 2, 3$. Furthermore, due to the incremental nature of the algorithm, (3.36) can be modified:

$$\begin{bmatrix} X_{t=k} \\ Y_{t=k} \\ \theta_{t=k} \end{bmatrix} = r \begin{bmatrix} \cos\theta_{t=k-1} & \sin\theta_{t=k-1} & 0 \\ -\sin\theta_{t=k-1} & \cos\theta_{t=k-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{3L} & \frac{1}{3L} & \frac{1}{3L} \end{bmatrix} \begin{bmatrix} \Delta\psi_1 \\ \Delta\psi_2 \\ \Delta\psi_3 \end{bmatrix} + \begin{bmatrix} X_{t=k-1} \\ Y_{t=k-1} \\ \theta_{t=k-1} \end{bmatrix} \tag{3.37}$$

where $x_{t=k}$, $y_{t=k}$ and $\theta_{t=k}$ is the current robot pose, $\theta_{t=k-1}$ the robot attitude in $t = k-1$, $\Delta\psi_i$ the rotation of wheel i between instants $t = k-1$ and $t = k$ and $x_{t=k-1}$, $y_{t=k-1}$ and $\theta_{t=k-1}$ the previous robot pose.

Chapter 4

Simulator development

With the limitations of a physical implementation and the testing scenarios that can be built in a controlled environment, the creation of a simulation tool leads to a higher degree of freedom when simulating more complex scenes.

With the objective of creating a platform for testing the performance and future improvements to the algorithm, a simulator was designed to emulate the behaviour of the robot in a controlled scenario. This simulator is then validated with its physical counterpart in a real scenario.

The simulator was designed in *Matlab* 2018b's *Simulink* environment, making use of its 3D virtual world visualization and control systems design capabilities. In section 4.1 the dynamics of the modelled vehicle are described, followed by the specifications of the simulated sensing system in section 4.2. The functionalities of the virtual world are then explained in section 4.2. This chapter concludes with an overview of the *Simulink* simulator model. **The data processing algorithms used in the physical implementation (section 3.3) were reproduced in the developed simulator.**

4.1 Vehicle simulation

The Omni-Ant platform can be emulated in a virtual environment by a composition of its dynamics and kinematics models, allowing for the simulation of the robot's behaviour as a system upon receiving commands. Simulating this system in a virtual environment can be a powerful tool for developing controllers as well as evaluate the behaviour and develop algorithms based on outputs of any peripherals attached to it, such as sensors in an artificial environment that can easily be changed.

To simulate each of the motors' response to command inputs, the motor's dynamics model is required. Due to its electromechanical nature, this motor can be modelled as a simple first-order system to achieve acceptable results or, by way of identification, a more complex model closer to its real behaviour. Based on the identification of this specific motor by Gonçalves et al. [35], the parameters in table 3.6 and the model obtained in (3.15), (3.16) are used to derive a state space model of each motor,

given by:

$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = A \cdot \begin{bmatrix} i \\ \omega \end{bmatrix} + B \cdot U \qquad , \qquad y = C \cdot \begin{bmatrix} i \\ \omega \end{bmatrix} + D \cdot U$$

with (4.1)

$$A = \begin{bmatrix} -2.0885 \cdot 10^3 & -149.7059 \\ 89.7707 & -0.1642 \end{bmatrix}, \; B = \begin{bmatrix} 294.1176 \\ 0 \end{bmatrix}, \; C = \begin{bmatrix} 0 & 1 \end{bmatrix}, \; D = 0$$

where i is the current applied to the motor, U is the voltage input and ω is the motor's angular speed. It is worth noting that the 30:1 reduction gearbox present in the motor is included in the model identification.

The non-linearities mentioned in section 3.2.2 are modelled as a saturation in the [-12,12]V range and a quantization of the voltage signal received by the motor. This quantization is dependant on the resolution of the board's analog-to-digital converter (ADC) resolution. Since signals in this board are read with an 8-bit resolution, the quantization resolution can be calculated using:

$$Q = \frac{E_{FSR}}{2^M} \quad , \quad E_{FSR} = V_{RefHi} - V_{RefLow} \tag{4.2}$$

with Q as the quantization resolution, E_{FSR} the full scale voltage resolution, M the ADC resolution and V_{RefHi} and V_{RefLow} as the maximum and minimum values of the voltage scale, respectively. This leads to a quantization resolution of $0.09375V/bit$.

Additionally, a dead zone non-linearity is present, characterized by the effect of friction on the shaft which requires a minimum supply level to ensure motion. Using the steady-state relation between angular velocity and expected motor input voltage by Gonçalves et al. [35], the motor's angular velocity can be expressed as a function of the supply voltage:

$$\omega_{desired} = \frac{K_s U_{command} - RT_c}{K_s^2 + RB} \tag{4.3}$$

where $U_{command}$ is the voltage input command, $\omega_{desired}$ the desired angular velocity of the motor and K_s, R, B and T_c the systems identification factors provided in table 3.6. As evidenced, this is a linear relation horizontally shifted from the origin by a factor of $\frac{RT_c}{K_s}$, the voltage dead zone threshold. (Figure 4.1). Since the supplied voltage is actively regulated on the motor board, ensuring the correct supply to achieve the velocity reference provided, this factor can be discarded from the conversion block and modelled as a dead zone nonlinearity. As such, assuming open loop control, the static gain of the system is given by:

$$U_{command} = \frac{K_s^2 + RB}{K_s} \cdot \omega_{desired} = \frac{1}{1.9158} \cdot \omega_{desired} \tag{4.4}$$

Thus, the estimated dynamics model of each of the EMG30 motors developed in *Simulink* is shown in figure 4.2. Step and ramp responses of the modelled system are presented in figure 4.3, including models with no non-linearities for reference. As evidenced, the presence of a quantization may result in a small stationary error on a step input as well as small oscillations on ramp inputs. The system is

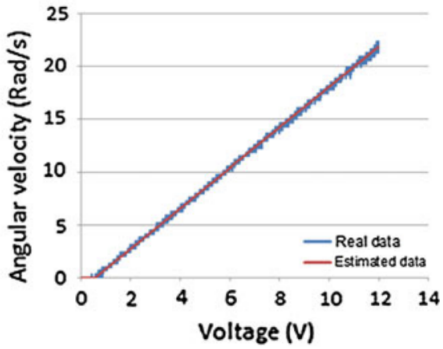


Figure 4.1: Voltage to angular velocity relation (adapted from [35])

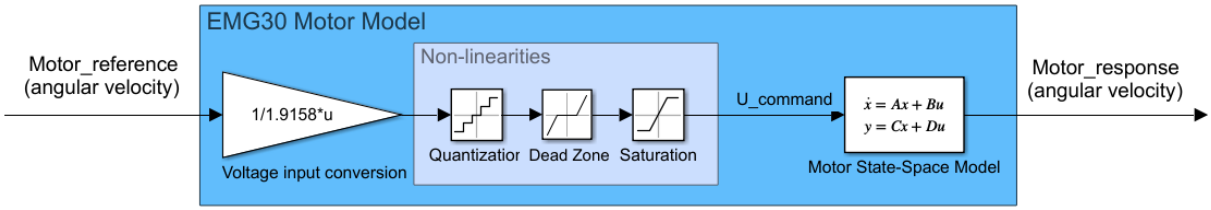


Figure 4.2: *Simulink* motor model

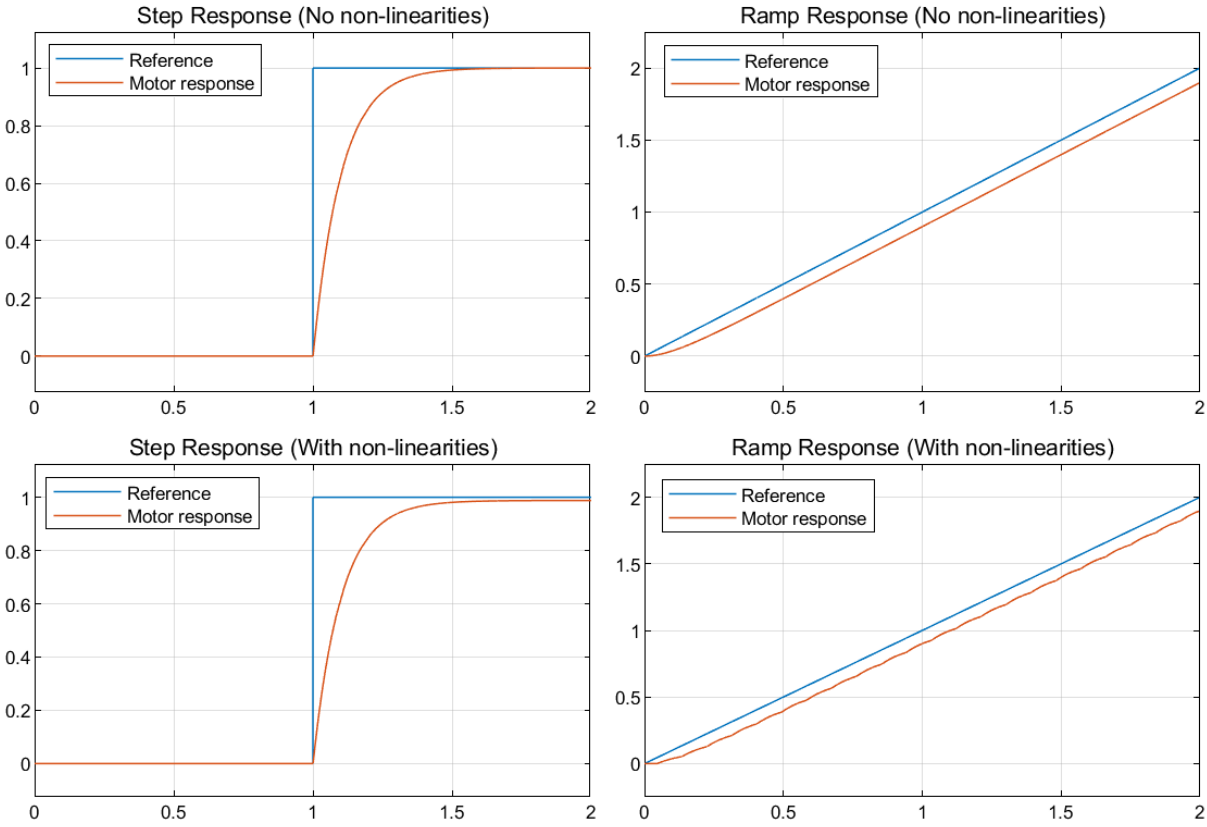


Figure 4.3: Motor response to step and ramp inputs

also characterized by a small stationary error when following ramp inputs, regardless of the presence of non-linearities.

The kinematic model of the robot (3.36) is used for the conversion of motor angular velocity into robot velocity values (direct model) and vice-versa (inverse model). Figure 4.4 depicts the robot dynamics model. With the purpose of providing the robot with waypoints to follow, with its path being autonomously

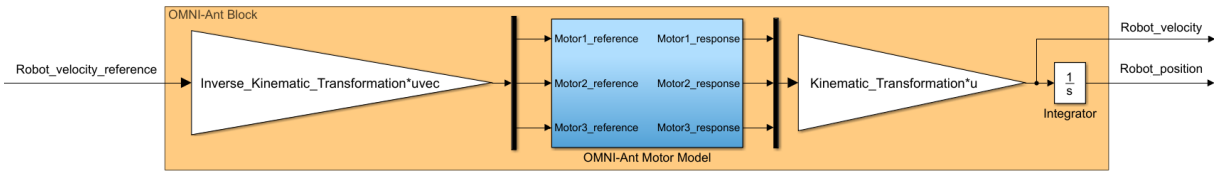


Figure 4.4: *Simulink* robot model

readjusted if needed, robot inputs are provided as the position of said waypoints. As such, similarly to the physical implementation in this work, commands are provided in the form of cartesian coordinates of these waypoints, relative to the starting point of the robot's trajectory due to the unknown environment. The robot is intended to move at a constant cruise speed (in steady-state conditions) to ensure correct sensor readings. This speed value is set to the selected cruise speed of the physical model, $0.3\ m/s$, although a different value can be used. Additionally, since position references are provided, the loop is closed to achieve position control. Velocity references are transmitted to the robot model based on the position error and the cruise speed. The position error determines the direction of the velocity reference vector and its magnitude is given by the set cruise speed value. Figure 4.5 depicts the full $Simulink$ Omni-ANT model with position control. The sensing system simulation is described in the following section.

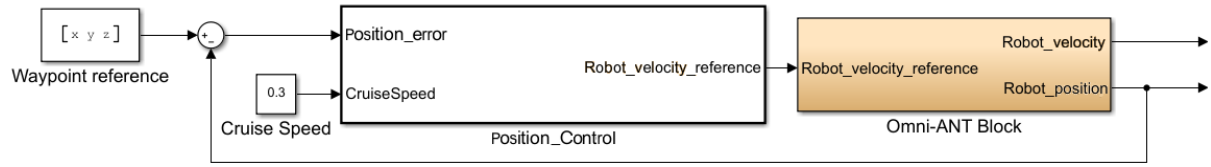


Figure 4.5: Final *Simulink* robot model

4.2 Sensor simulation

This section describes the specifications of the real sensor models applied to a virtual, simulated model of each of the sensors. Based on the real model of the URG-UG01-04LX used in the real implementation and its specifications (a field of view of $240°$ with a $0.36°$ angular resolution, as described in section 5.1.2), the 2D point cloud detection is simulated based on discrete angular detections of obstacles, to which a nominal error of $30mm$ ($< 1000mm$ range) or 3% of the real measurement ($> 1000mm$ range) is applied to simulate the noise in real measurements.

Likewise, the RADAR sensor simulation is based on the operation conditions of the real sensor, simulating a $60°$ field of view and mounted on the robot with a $15°$ resolution. As in the TI-mmWave AWR1642 sensor, the absolute velocity of detected obstacles within its scanning range is provided to the collision avoidance algorithm. Figure 4.6 depicts an example of a scene in which the robot and an obstacle are moving towards each other, with both sensors being used for data collection. Detections within the sensors' maximum scanning range are plotted in a top view representation of the scene (*Simulation environment (top view)*) while relevant detections within the defined warning (yellow) and danger (orange) range zones are represented in the robot's local reference frame plot (*Warning and danger range detections*). The clustering algorithm is applied to group the sensor readings into clusters, identifying each detected obstacle (outlined by a black rectangle). Each moving obstacle's absolute radial velocity is then matched with each detected cluster, plotting the velocity vector (RADAR readings) in the cluster's centroid using arrows matching the velocity direction (towards or away from the robot) and scaling with its value. The obstacles' velocity values are then represented in a separate plot (*Velocity detections*) with its respective value to avoid cluttering of the previous plot.

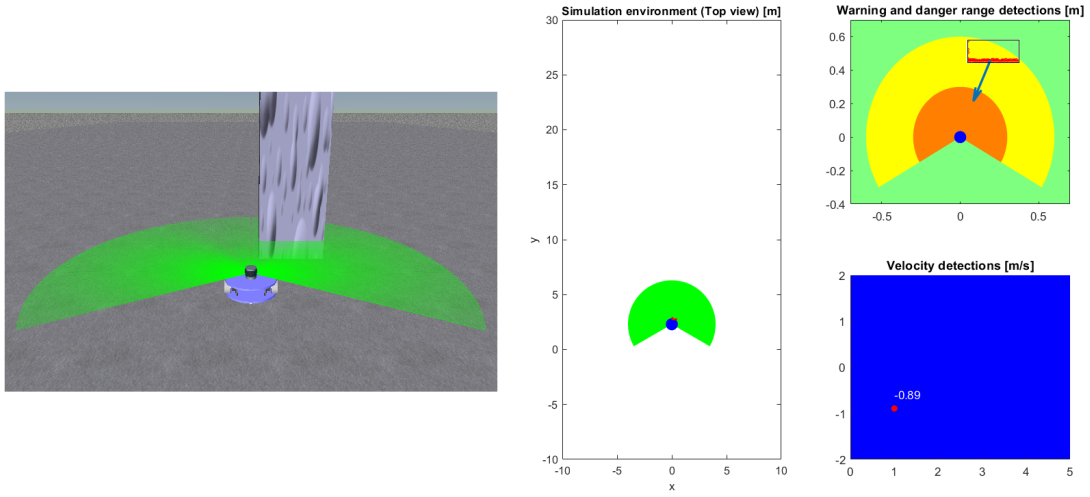


Figure 4.6: Sensors simulation example. Current scene (left) and simulated sensor output (right)

4.3 Virtual world

The virtual world was designed using *Matlab*'s 3D world editor. The simulator is equipped with a set of obstacles with several geometries which can be placed anywhere in the virtual world. These objects can be defined as fixed or moving objects, defined by a velocity vector (null for fixed objects). Figure 4.7 shows an example of a scene in the virtual world. The virtual world and the simulator itself can be edited by experienced users to include various other geometries and obstacles as well as define trajectories for any moving obstacle to follow.

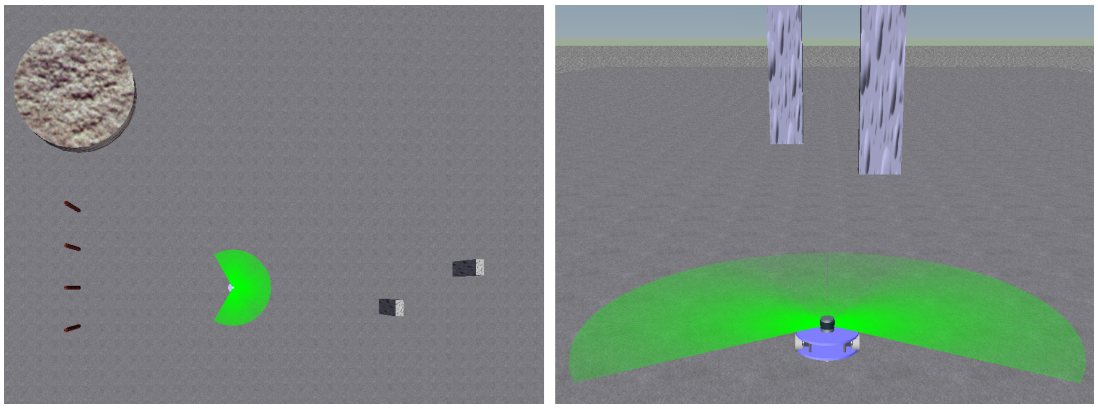


Figure 4.7: Virtual world top view (left) and third person robot view (right)

4.4 Simulator model

The final simulator model is presented in figure 4.8. The *Omni-ANT block* block represents the model described in section 4.1, the *Obstacle avoidance* and *Obstacle detection* blocks contain the algorithm in section 3.3, the *Virtual World Parameters* block contains the specifications of the virtual world described in 4.3 and simulation parameters and the *Sensor Model* block contains the simulated sensor model described in 4.2. Several parameters of the model can be defined by the user. Sensor activation is

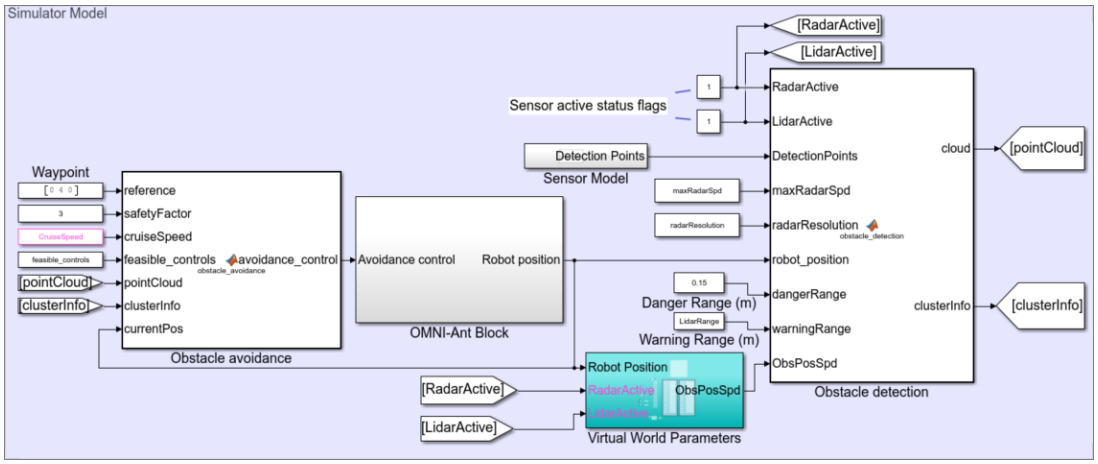


Figure 4.8: Simulator model (*Simulink*)

defined by the user as well, offering the possibility of activation of either or both sensors. As previously mentioned, the virtual world can be customized with the possibility of inclusion of other geometries of obstacles. Waypoints are defined by the user and the position and pose of each obstacle can be defined outside of the virtual world model as inputs. Data being transmitted between blocks such as sensing data or the robot's current trajectory can be visualized in its respective branch (by adding visualization blocks or exporting the data into *Matlab*'s workspace).

The simulator offers an interface for a graphical visualization of data such as detected points, clustering results, velocity readings, robot position and current scanning data (as seen in figure 4.6) as well as a tridimensional visualization platform of the current scene (as seen in figure 4.7).

Chapter 5

Experimental development

In this chapter, the software setup and prototype assembly are described, stating the various communication protocols, software and hardware connections used between the devices described in section 3.1 as well as visualization tools used. Figure 5.1 depicts the final configuration of the Omni-ANT prototype components. Figure 5.2 depicts the final assembly of the prototype.

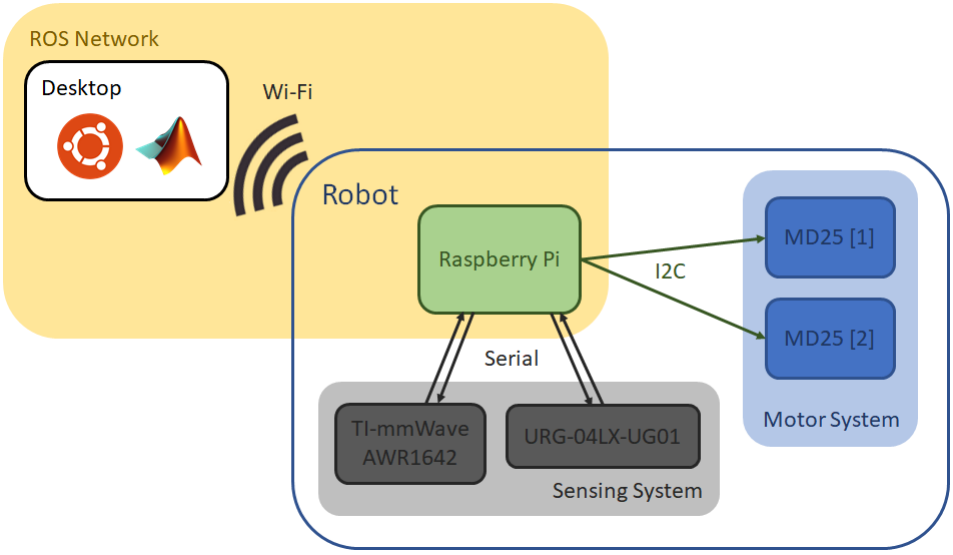


Figure 5.1: Connection protocol diagram

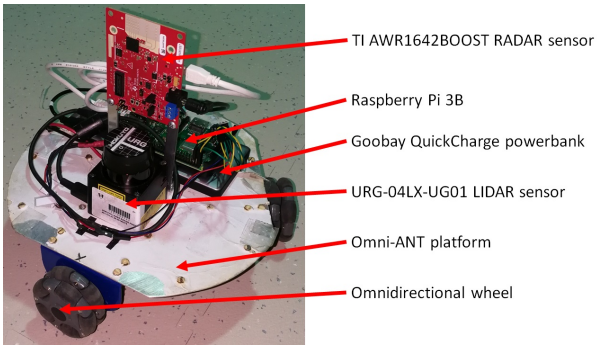


Figure 5.2: Final assembly

5.1 Component setup and configurations

5.1.1 Processing unit

In an initial approach, an Arduino Mega ADK was selected as the processing unit for controlling the motors and run obstacle detection and collision avoidance algorithms, based on [10] and [11]. However, this approach was discarded as the Arduino board allows for a single USB connection to be made, while at least two ports were needed for enabling simultaneous RADAR and LIDAR sensor connections, as the latter requires a USB port and the former requires physical modification of the board to allow for different data acquisition channels, which is undesireable due to the relatively high cost of the board, should a replacement be needed.

As such, the processing unit chosen for algorithm execution and network setup was the Raspberry Pi 3 Model B V1.2, described in section 3.1.2, due to its wide range of connectivity and communication options and protocols. The embedded processor, the Broadcom BCM2827, running at 1.2 GHz with 1GB RAM is also a significant performance improvement over the Arduino's ATmega2560 running at 16MHz with 8KB RAM and 256KB flash memory for storing code, of which 8KB are reserved for bootloader. This performance increase is considerable and possibly essential due to the relative computational bulk of the final implementation.

The Raspberry Pi was flashed through the NOOBS (New Out Of the Box Software) installer[1](via microSD card) with the desktop version Raspbian Buster operating system, a Unix-like system based on Debian Linux. The integrated development environment (IDE) used to develop the scripts running in this device is *Geany*, a lightweight *Python* 2.7 text editor bundled in the NOOBS installation. A Raspberry Pi-compatible version of ROS based on ROS Kinetic Kame, ROSberryPi, was installed on the processing unit. The various nodes involved were developed using Python 2.7, which is the latest version compatible with the standard version of ROS Kinetic. A more recent version of ROS, compatible with Python's 3.7 release is available. However, the older ROS Kinetic version was chosen to be used across all the platforms to avoid compatibility issues.

5.1.2 LIDAR sensor

The LIDAR sensor can be set to operate in two different modes. Its scanning area spans across a 240° egocentric area at either 4095 mm or 5600 mm maximum radial range. Although the sensor can be set up to operate at a higher detection range it does not guarantee the same accuracy specifications as the shorter range configuration. This phenomenon can be attributed to partial scattering of light waves which can introduce inaccuracies or non-detections of obstacles with small dimensions. For this reason, the sensor was set to operate at a 4095mm detection range as the additional range is not mandatory for short-range obstacle detection.

The data provided by the sensor is transmitted in 12 bits and encoded to 2 characters when operating at a 4095 mm detection range as opposed to 18 bits and 3-character encoding when operating at a 5600

[1] https://www.raspberrypi.org/downloads/raspbian

mm detection range. As such, the reply messages are set to a 2-character encoding which offers a nigh negligible improvement in processing time.

5.1.3 RADAR sensor

The AWR1642BOOST sensor requires a configuration step prior to data acquisition. A variety of configurations can be specified by the user, with a set of mandatory as well as optional configurations (in [36]). With the intention of complementing LIDAR range readings with their velocity counterpart, a higher emphasis is placed on the resolution of the obstacle velocity readings as opposed to their range readings. Consequently, the sensor was set to run at the highest velocity resolution configuration, without having a significant impact on range resolution (inversely proportional as mentioned in section 2.1.1). For the sake of simplicity, only the most relevant configurations used are mentioned.

The sensor was set to run at a 77 GHz frequency with a range resolution of 0.139 m and a velocity resolution of 0.04 m/s. A maximum unambiguous range of 14.23 m and a maximum radial velocity of 2.56 m/s are measurable with this configuration. The option of utilizing the sensor's internal processing unit for ignoring static objects and clustering of data is discarded.

5.2 Connections and communication protocols

5.2.1 ROS network setup

As described in section 3.1.5, ROS is used as the framework for setting up the communication network for data transmission between the modules involved in this implementation. A diagram depicting the developed ROS network is shown in figure 5.3. The entities involved in this network are explained in

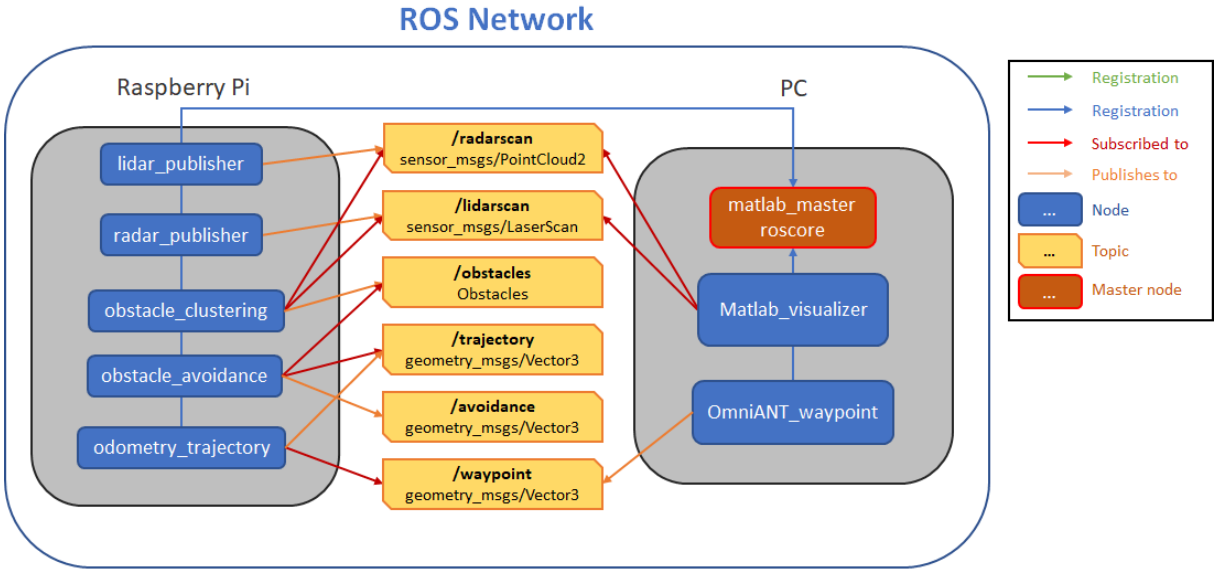


Figure 5.3: Implemented ROS network

detail in the following sections.

Nodes

As a crucial part of any ROS network, the **master node** is required to ensure correct communication between nodes. Generally, this node is not very computationally demanding when compared to other nodes running complex algorithms or gathering data from sensors. While this node can be run in the Raspberry Pi with negligible impact to its processing power, it was set to run on the Windows machine running ROS under Matlab to minimize the computational load on the Raspberry Pi, which will be running the obstacle detection and sensor publisher nodes, required to run on the prototype itself.

It is worth noting that as an additional condition for two-way communication between modules to function, each machine is required to export its IP address as well as the address of the machine running the master node, with the addition of the specific port used by the master node in this case (set to port 11311 as default in any ROS instalation). This allows the master node to obtain the location of every node in the network and nodes to be registered in a specific network if multiple networks are being run in the same environment.

The **sensor data publisher nodes** *lidar_publisher* and *radar_publisher* are the nodes publishing LIDAR and RADAR sensor information. These nodes are constrained to a set publishing frequency to ensure periodicity of the messages being published. The entirety of the LIDAR scanning range is published, characterizing non-detections as the maximum radial scanning range value and obstructions as the proximity at which these are detected. Similarly, due to the starting configuration of the RADAR sensor (section 3.1.4), RADAR detections are published in its entirety without discarding static detections, with location as well as velocity data of each detection.

Data being published by the sensors is processed in the **obstacle identification and collision avoidance system**, the *obstacle_clustering* and *obstacle_avoidance* nodes. The *obstacle_clustering* node is responsible for computing the detected obstacles by running a clustering algorithm on the LIDAR detections, identifying groups of points assumed to characterize a single obstacle (section 3.3.1). This node is set to process data from either proximity data alone (LIDAR detections) or from both proximity and velocity data from one (RADAR detections) or both sensors (by way of a sensor fusion algorithm explained in section 3.3.1).

These obstacle detections are then used by the *obstacle_avoidance* node to compute a feasible avoidance control based on its current trajectory to allow the prototype to avoid any present obstacles while minimizing the deviation from its otherwise unobstructed trajectory (explained in section 3.3.2).

The *OmniANT_waypoint* node prompts the user for the desired waypoint for the robot to reach. A signal is then received when the robot reaches its intended waypoint and the user is prompted for a new waypoint. This node was created as a user-friendly method to communicate the desired waypoint(s) to the robot, although it is not essential. The waypoint data can be directly published to the respective topic by other means with the correct message structure, ensuring the modularity of the prototype.

[1] adapted from http://tsgdoc.socsci.ru.nl/index.php?title=File:Matlab_Logo.png and https://www.raspberrypi.org/trademark-rules/

The **position and trajectory system** is implemented in the node *odometry_trajectory*. Based on the received waypoint data, a trajectory that minimizes distance travelled is computed for the robot to follow and reach the intended waypoint. This trajectory data is published and analyzed in the collision avoidance system to check for any future collisions (described in section 3.3.2). This node then receives the avoidance control data and communicates the command to the motors. Due to the unknown environment and the inability of external location, an odometry estimation algorithm is implemented to estimate the robot's current pose (explained further in section 3.3.3). **Odometry data is preserved between waypoint prompts until termination of the algorithm**.

The *matlab_visualizer* subscriber node is subscribed to both sensor detection topics. As any standard ROS subscriber node, these are set to be continuously active until termination while waiting for new messages to be published in the topics they're subscribed to. This node is used to visually represent the sensor data in a comprehensive graphical interface, as shown in figure 5.4. It is worth mentioning that the vertical axis of the rightmost figure represents velocity readings, while the horizontal axis represents the measured distance to the velocity reading.

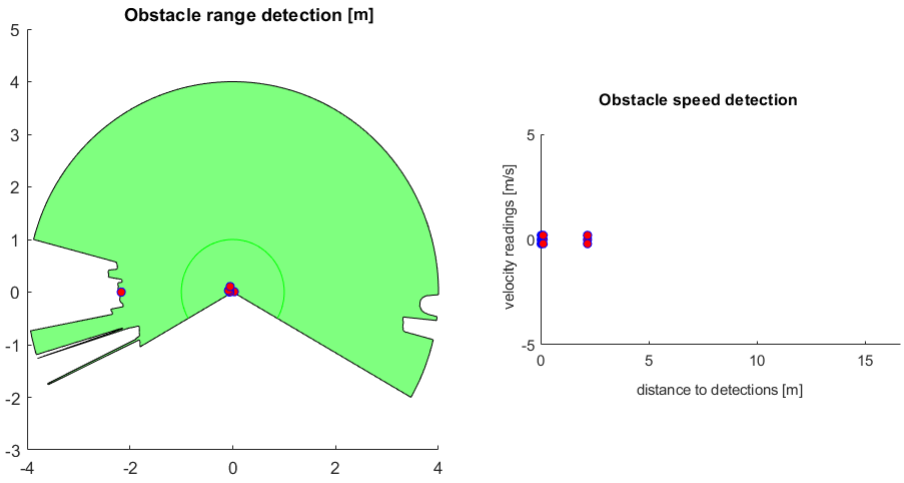


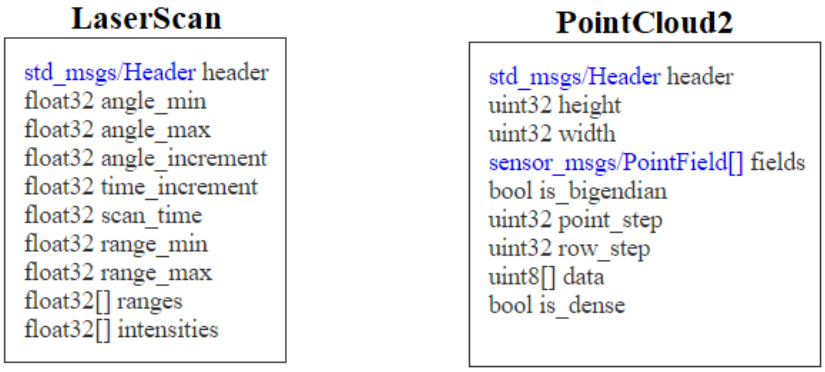
Figure 5.4: Matlab visualizer example

Topics and Messages

The active **detection topics** in this ROS network are */lidarscan* and */radarscan*. */lidarscan* is the topic under which the LIDAR sensor's raw data is published. This data is arranged into a preset message type provided by the baseline ROS installation, the *sensor_msgs/LaserScan* message type. Likewise, the */radarscan* topic message type is *sensor_msgs/PointCloud2*, another preset message structure present in the base ROS installation. Figure 5.5 depicts the structure of both message types.

Both of these message structures may contain a header with information about the time of publishing and frame ID. The *LaserScan* message type is published as a 2D continuous point cloud, carrying

[2]adapted from adapted from http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/LaserScan.html and http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/PointCloud2.html



(a) *LaserScan*'s structure (b) *PointCloud2*'s structure

Figure 5.5: Message structures[2]

information about obstructions of the LIDAR beams (*ranges[]*) within a range of angles (*angle_min-angle_max*) with a given resolution (*angle_increment*) and sample time (*time_increment*). *PointCloud2* on the other hand is published as a 2D point cloud containing solely the detected objects (*data*). The data is published as a byte array of length *row_step*, containing information about the relative coordinates of points in the relevant axes (*fields*, using the *sensor_msgs/PointField* structure).

Obstacles identified by the obstacle identification system are published under the */obstacles* topic. A new message type with a specific structure was developed to suit the content of the message, the *Obstacles* message (figure 5.6). The message contains information related to each detected cluster's

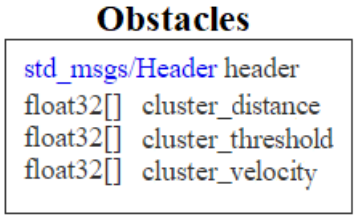


Figure 5.6: Obstacles message structure

proximity (*cluster_distance*) as well as its velocity (*cluster_velocity*, null if no RADAR data is available). Additionally, the message contains the aperture angle with respect to the robot's perspective in which the detected obstacle is contained for obstacle avoidance purposes(*cluster_threshold*).

The */waypoint*, */trajectory* and */avoidance* topics share the baseline ROS' geometry_msgs/Vector3 message type (figure 5.7).

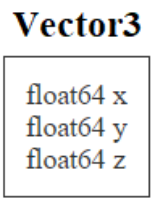


Figure 5.7: Vector3 message structure

The */waypoint* topic is used to publish waypoint information for the robot to follow, containing its

cartesian components (*x*,*y*,*z*). Due to the constraints of planar movement, the *z* cartesian component is discarded when analyzing the published message.

The */trajectory* and */avoidance* topics contain trajectory data characterized by a vector's cartesian components (*x*,*y*,*z*). However, while the */trajectory* topic contains the computed unobstructed trajectory to the defined waypoint, */avoidance* contains information about the avoidance control to avoid collisions, should obstacles be present.

5.2.2 Raspberry Pi

The Raspberry Pi acts as the master device to the various devices connected to it, sending commands to each of the peripherals, receiving data from the connected sensors and coordinating the execution of the obstacle detection and collision avoidance algorithms.

This device is capable of communicating in an array of protocols. Serial communication is used to transfer data between the sensing system and the processing unit. Power is supplied by a *goobay Quick Charge Powerbank 5.0*[3] through USB.

5.2.3 MD25

The MD25 motor controller boards are connected to the Raspberry Pi via an I2C cable. An I2C connection (Inter-Integrated Circuit) is a communication protocol designed to be used in master-slave configurations, whether when a master device is meant to control multiple slaves or multiple masters are meant to control a single or multiple slaves. Three lines are needed to establish an I2C connection, the ground line (GND), the clock line (SCL, Serial CLock) and a data line (SDA, Serial DAta).

Data is sent through the SDA channel in a bit-by-bit fashion, with the SCL line carrying the clock signal. The clock is controlled solely by the master, whereas any device within an I2C connection can transfer data over the data line.

The GND line is required due to the communications in the other channels being made through high/low voltage levels. A common ground shared by all the masters and slaves ensures the correct emission and reception of these signals. Slaves are assigned a unique address (which may be changed) that identifies them, allowing for a master to specify which slave to communicate with when sending a message.

Due to the difference in operating voltages between the Raspberry Pi (3.3 V) and MD25 boards ($5V$), a voltage level converter is needed to bridge the connection, preventing damage to the Raspberry Pi board.

Figure 5.8 depicts the physical connection between the Raspberry Pi board and one of the MD25 boards.

Two MD25 boards are needed to connect the 3 motors of the prototype since each board is capable of controlling up to two motors. Each motor is assigned a unique I2C address, independent of which board it is connected to. Communication between the MD25 boards and the Raspberry Pi board is

[3] https://www.wentronic.de/en/tab-and-phone/powerbanks-and-battery-packs/31015/quick-charge-powerbank-5.0-5.000-mah

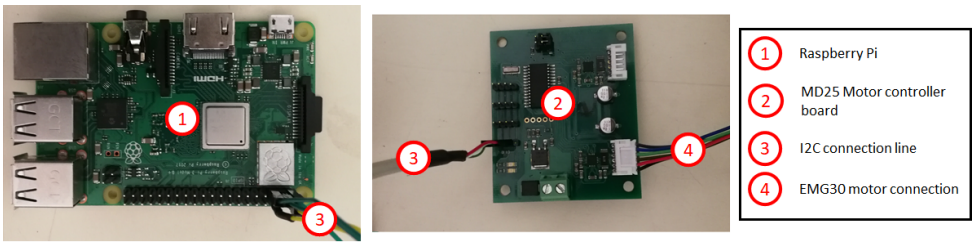


Figure 5.8: I2C connections, MD25 board (right) and Raspberry Pi (left)

established at a 100 Hz rate. The available functions and commands used to control the board in I2C mode can be accessed in [37]. These boards are externally powered by a 11.1V Li-Po battery (3S1P 25C).

5.2.4 LIDAR and RADAR sensors

The URG-04LX-UG01 is connected to the Raspberry Pi via a USB A - USB Mini B cable for data transmission as well as a power supply line. As such, this cable requires data transmission as well as power supply capabilities. The connection is shown in figure 5.9.

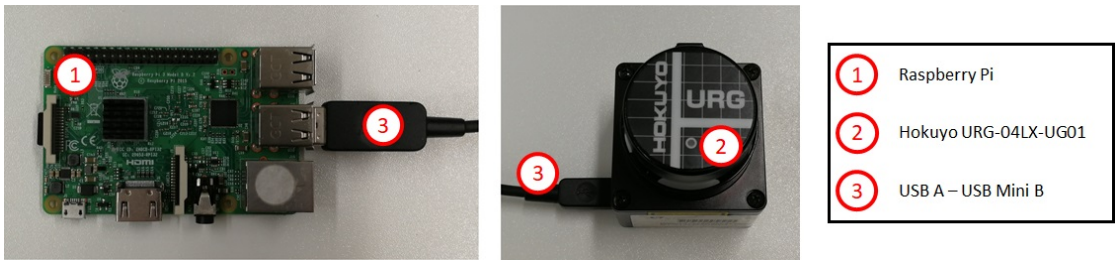


Figure 5.9: LIDAR sensor connection

The LIDAR sensor uses the serial protocol for communication, allowing for communication through a terminal such as PuTTY on any Windows or some Unix-like machines or any device capable of sending serial commands over USB. Communication between the device is made using the SCIP2.0 protocol on a request-reply basis, in which a host machine is required to send data request commands to which the sensor will reply with the ranging data at the time of the request. Communication is established with a baudrate of 115200 bps (bytes per second).

The AWR1642BOOST radar sensor is connected to the Raspberry Pi master through a micro USB-USB cable capable of data transfer. However, this connection is solely used for data transmission, with power being supplied through a separate line, a 2.1mm barrel jack cable connected to the *goobay Quick Charge Powerbank 5.0* through USB. Communication is established over serial connection with a baudrate of 921600 bps. The connections are shown in figure 5.10.

5.2.5 ROS nodes

The *Matlab* and the RViz visualizer can be simultaneously connected to the ROS network through the master node running on the *Matlab* machine. The *Matlab* visualizer is connected by a local connection

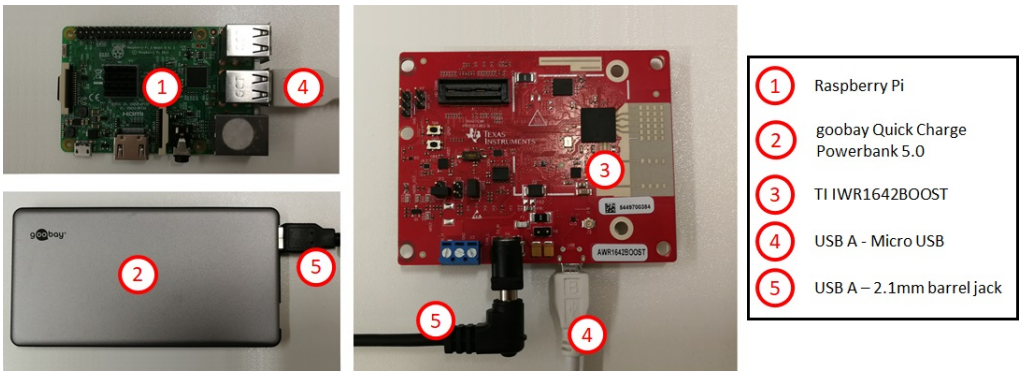


Figure 5.10: RADAR sensor connections

through *Matlab*, while the RViz visualizer running in an Ubuntu virtual machine is connected via its respective ROS node through Wi-Fi.

The Raspberry Pi, running the multiple ROS nodes described in section 5.2.1, is connected to the ROS network over Wi-Fi.

Chapter 6

Results

In this chapter, an analysis of the simulated and physical implementations is made, comparing the performance of obstacle detection and avoidance in specific scenarios. Static and dynamic trials were performed to evaluate the behaviour of the prototype when faced with static or a combination of static and moving obstacles obstructing its path. For simplicity, static obstacles were constrained to basic geometries, while dynamic obstacles are simulated by separate omnidirectional three-wheeled robots similar to the Omni-ANT (with no obstacle detection and avoidance capabilities) moving in a simple trajectory (straight line). To avoid any damage to the hardware, these robots were constrained in their movement speed, being set to move at a constant velocity lower than the Omni-ANT's cruise velocity. A *YouTube* playlist was created with demonstrations of the robot navigating test scenarios[1].

6.1 Case study scenarios

In order to analyse the performance of the developed simulated and physical implementations, a common ground truth is needed to ensure a correct and direct comparison. As such, two arenas were built in both real and virtual worlds with the purpose of simulating obstacle courses to be traversed by the Omni-ANT. Figure 6.1 depicts the two obstacle courses built in the real world. White tape is used to mark the position of obstacles and the starting point of the robot to ensure repeatability during testing.

The courses consist on a set of rigid objects placed between the robot and its intended goal point. Course 1 was built as a less dense obstacle course to allow for more path choices, while course 2 was assembled as a maze-like course with fewer navigation options. They were assembled within an arena equipped with the *Qualisys* system. The *Qualisys* system is a 3D infrared tracking system used to track marked objects in real time within a limited area contained in the infrared cameras' sensing space. The configuration of these cameras (depicted in figure 6.2) allows for the tracking of objects marked by no less than four infrared markers (which are visible in some of the obstacles in figure 6.1). Each tracked object requires a unique configuration of markers to ensure correct identification of said object by the system. Although very precise (5mm maximum absolute error in this camera configuration), this system

[1] https://www.youtube.com/playlist?list=PL-d8pRLUjxQ1GqVXmXKo9Kgd94PkKp8Do

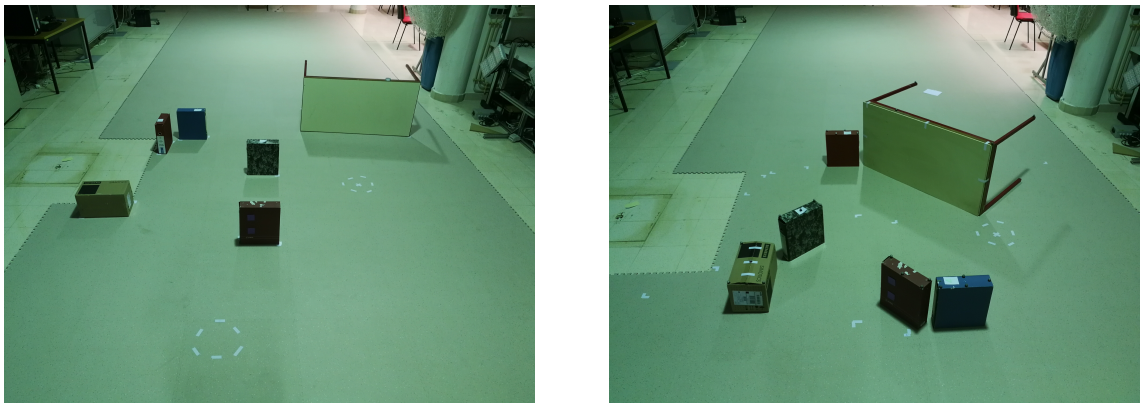


Figure 6.1: Arena configurations for course 1(left) and course 2(right) in the real world

is not without flaws, occasionally being unable to track objects for brief periods of time. Consequently, tracking data is expected to be recorded as segmented lines as opposed to continuous lines. In an initial

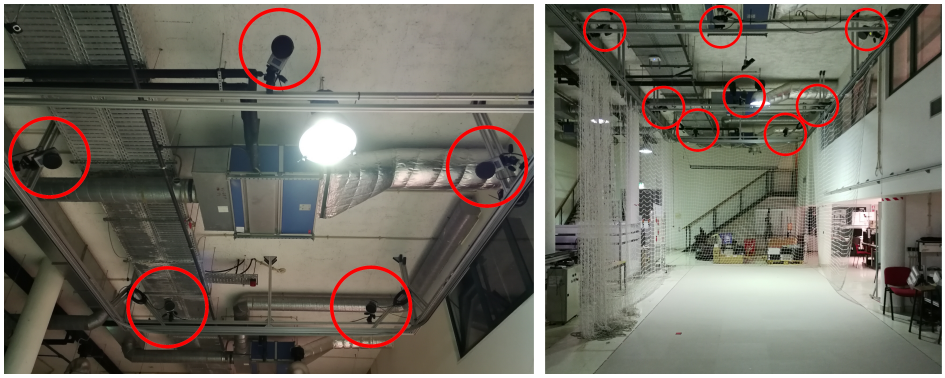


Figure 6.2: *Qualisys* camera configuration (in red)

approach, the *Qualisys* system was used to track both the location of the moving and static obstacles as well as the Omni-ANT. However, such an approach proved inaccurate due to the high number of markers simultaneously being tracked, leading to frequent loss of tracking or incorrect identification of certain objects. As such, the tracking of the static obstacles present in each course was performed beforehand, minimizing the number of objects being tracked simultaneously. This system was then used to track the movement of the Omni-ANT in the real world as well as other non-static obstacles within the obstacle course, preserving the obstacle configuration for each trial. Figure 6.3 depicts the static obstacle configuration obtained by infrared tracking.

With the tracking data of each obstacle and their known dimensions, both courses were reproduced in the simulator's virtual world, ensuring nearly identical conditions for the simulated and real experiments. Figure 6.4 depicts the reproduction of the real courses in *Simulink*'s virtual world.

With the reproduction of the real obstacle courses in the virtual world, a common ground truth is achieved for both experimental and simulated courses, allowing for direct comparison of the behaviour of the prototype in each case.

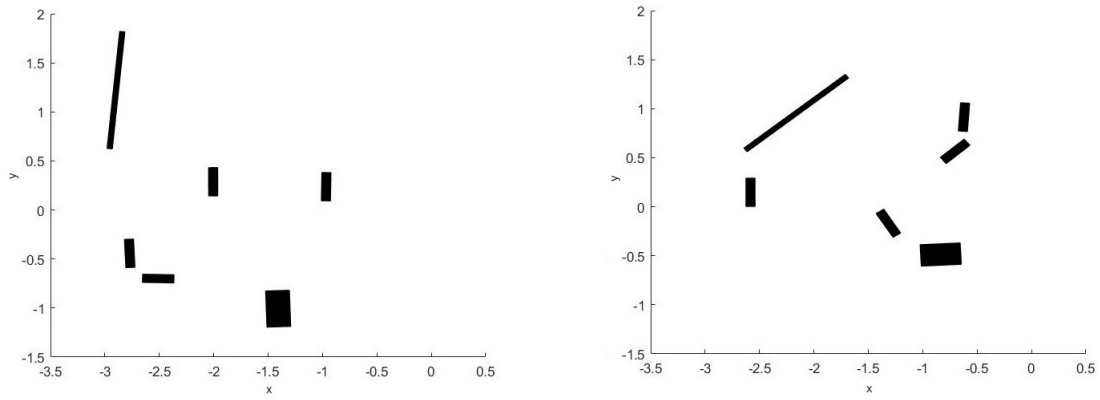


Figure 6.3: Static obstacle tracked configuration for course 1 (left) and course 2 (right), rotated $-90°$

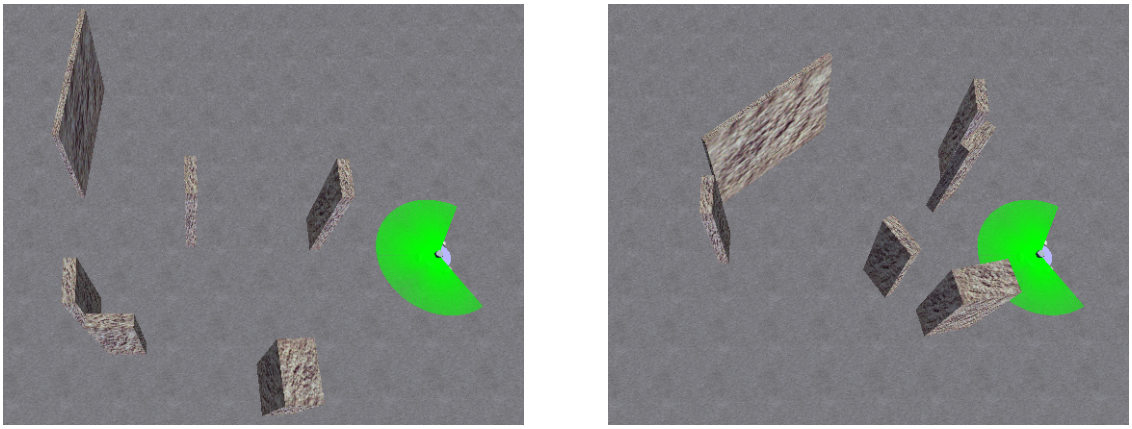


Figure 6.4: Course 1 (left) and course 2 (right) configurations in the virtual world

6.2 Simulation vs experimental results

The real and simulated Omni-ANT models were tried in a set of **four trials** with similar conditions:

- **Trial 1: Course 1 configuration** with the presence of **static** obstacles;
- **Trial 2: Course 1 configuration** with the presence of **static and dynamic** obstacles;
- **Trial 3: Course 2 configuration** with the presence of **static** obstacles;
- **Trial 4: Course 2 configuration** with the presence of **static and dynamic** obstacles.

Additionally, each trial was performed for all the different combinations of sensors being active for the entirety of a single trial:

- **LIDAR** sensor;
- **RADAR** sensor;
- **LIDAR and RADAR** sensors.

The goal of the robot is to navigate the environment, avoiding collisions and keeping a safe distance of no less than $0.2m$ from any obstacles. The robot is intended to reach a designated

waypoint located *4 m* in front of its starting point. The trial is considered to be complete once the robot reaches the vicinity of the designated waypoint or is otherwise unable to compute a feasible trajectory to avoid any obstacles, ceasing movement.

The Omni-ANT's and dynamic obstacles' real locations were tracked in each trial. In the following sections, the results of both the real and simulated tests as well as odometry data of the Omni-ANT are presented. Static obstacles (ground-truth, located by the tracking system) are represented by black rectangles ("Obstacles"), while dynamic obstacle paths are represented in pink ("Moving obstacle"). The robot's tracked (real) path is represented in blue ("Tracking"), the simulated virtual robot's path in green ("Simulated") and the odometry estimated path in red ("Odometry"). The minimum relative distance of the robot to any of the obstacles in the scene is also presented, with the minimum value registered along the trial represented in the plot's title. A representation of the error between odometry estimation and real, tracked position of the robot is also presented.

Course 1 results

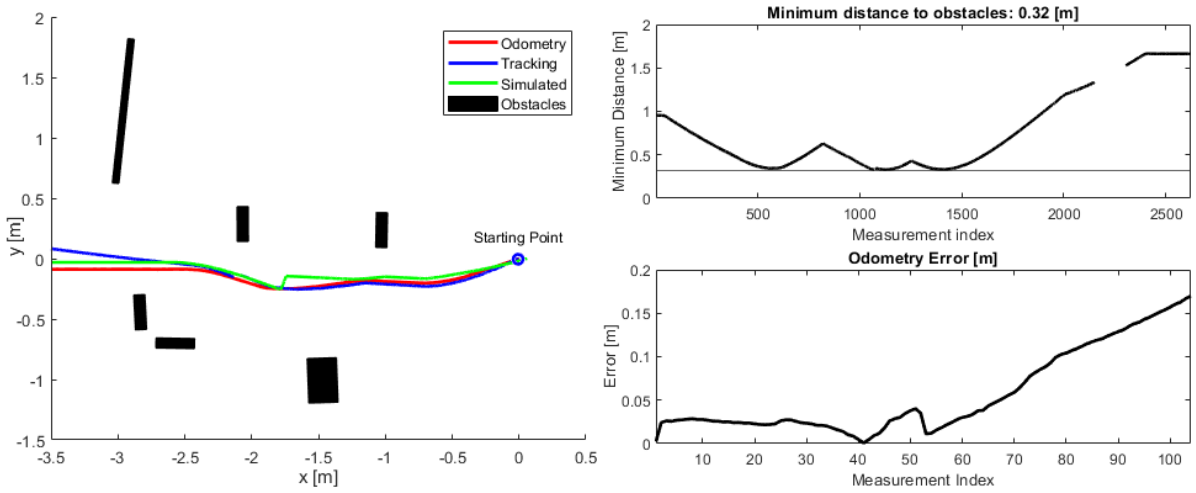


Figure 6.5: Trial 1, LIDAR and RADAR integration

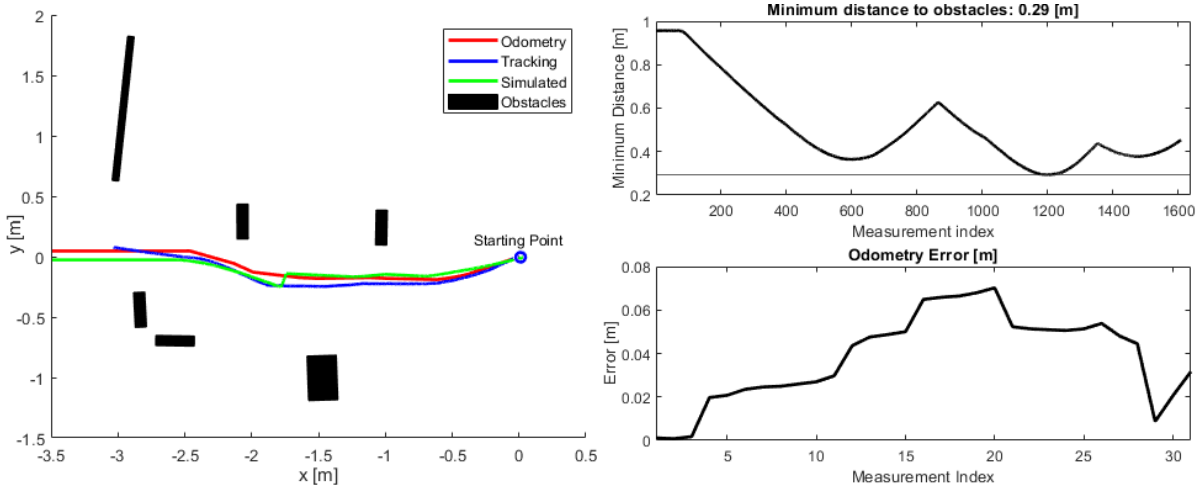


Figure 6.6: Trial 1, LIDAR sensor active

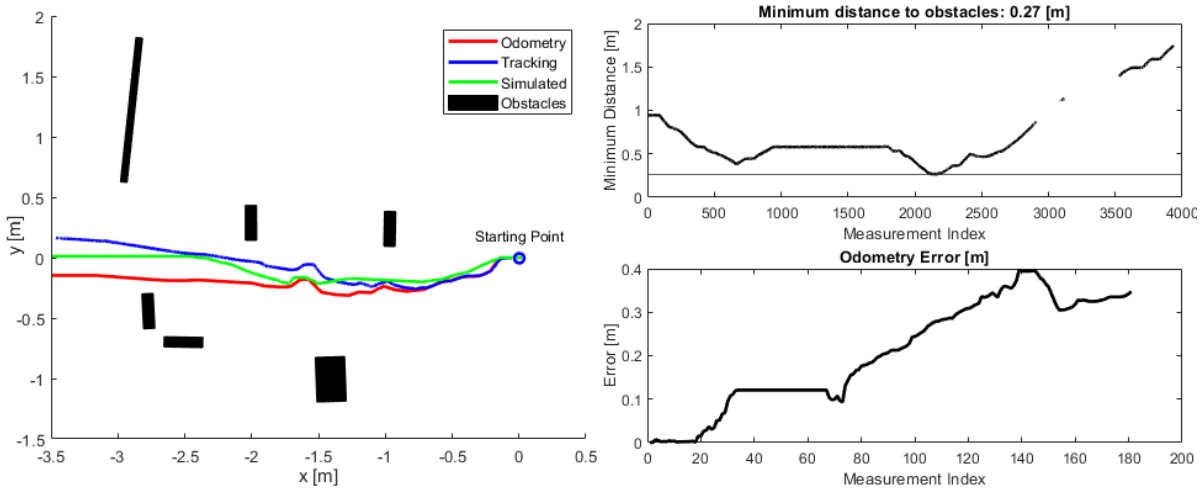


Figure 6.7: Trial 1, RADAR sensor active

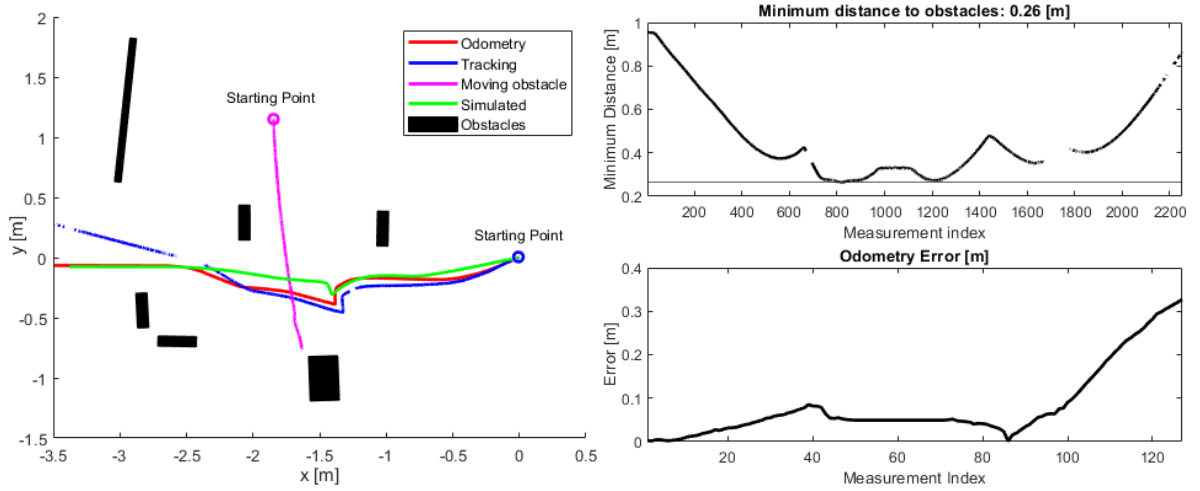


Figure 6.8: Trial 2, LIDAR and RADAR integration

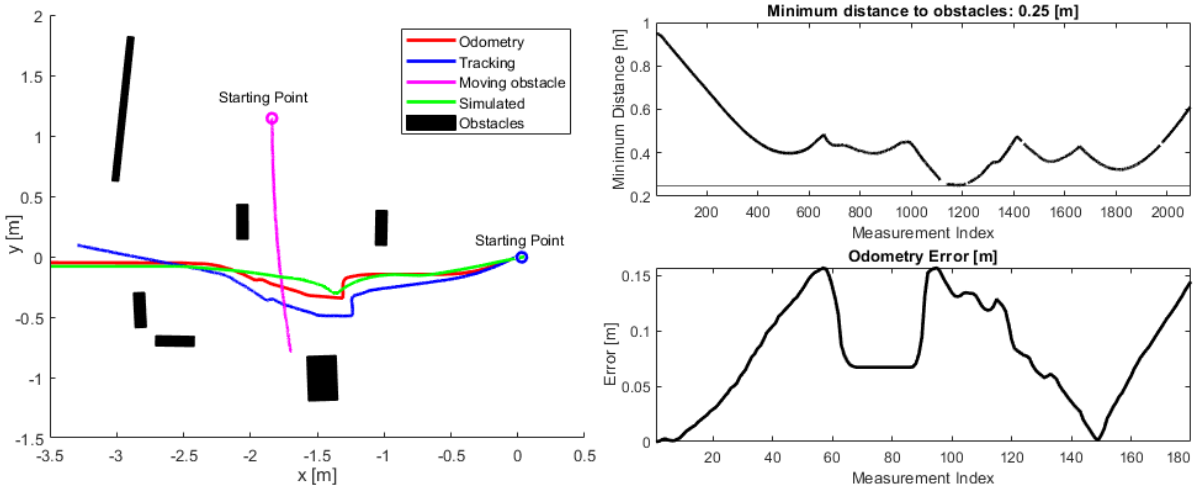


Figure 6.9: Trial 2, LIDAR sensor active

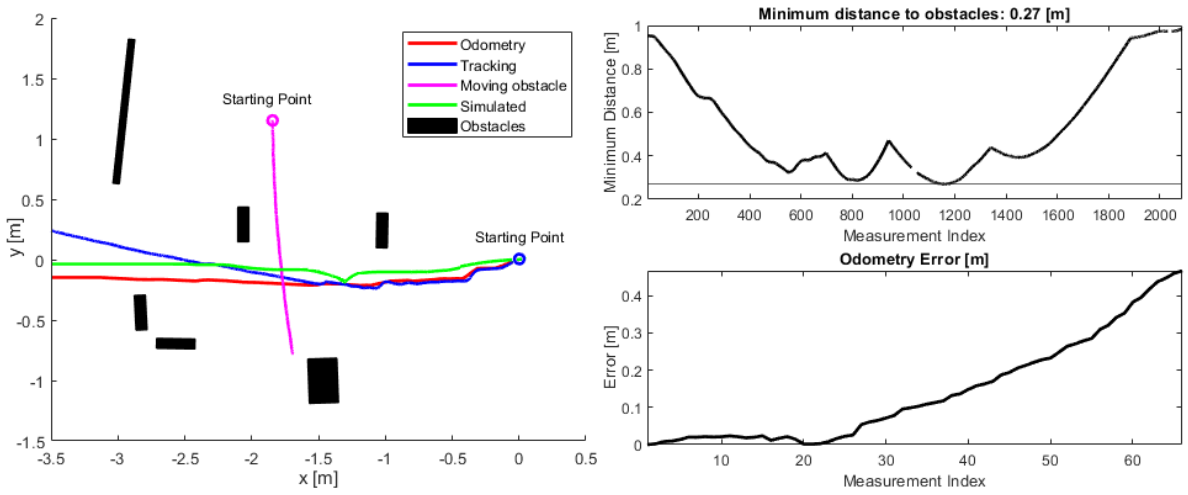


Figure 6.10: Trial 2, RADAR sensor active

Some observations can be made about the results of trials 1 and 2, run on course 1:

- All the trials were successful, with the robot navigating the course without colliding and keeping a distance above the designated safe distance from all obstacles;
- A number of trials were characterized by the loss of tracking of the robot (figures 6.6, 6.8, 6.9). However, the tracking system failed only in instances where the robot had already exited the course. Since tracking data is most relevant in the obstacle course section of the trial, such irregularities are negligible;
- The sensor fusion approach in trial 2 (figure 6.8) registered periods of loss of tracking during navigation through the obstacle-ridden section. However, these discontinuities in tracking were brief, with the tracking system being able to quickly localize and track the robot for the remainder of its navigation through the course. No significant data loss was registered, rendering the trial a success;
- The odometry algorithm achieved satisfactory results, estimating the current position accurately as evidenced by its comparison with the real, tracked position. As expected, this method suffered from innacuracies the further the robot travelled (most notably in figures 6.7-6.10). This can be attributed to a slight misalignment of the wheels' axes (intended to be mounted at a 120° angle between each pair) and the propagation of these errors in the odometry algorithm;
- The movement of the robot in trials where only the RADAR sensor is active appears to be slightly irregular, with frequent and sudden changes in trajectory (figures 6.7, 6.10). This can be attributed to sudden "ghost targets" (described in section 2.2) due to the reflectivity in the environment these trials were performed in, introducing unnecessary and sudden changes in robot trajectory before resuming its previous path. Nonetheless, the robot is able to consistently detect the existing obstacles and avoid them.

- This erratic behaviour is significantly attenuated in trials where sensor fusion is performed (figures 6.5, 6.8). This indicates that proximity-sensing via LIDAR sensor is more accurate and with less noise in this scenario, resulting in smoother avoidance trajectories which support the previous statement;

- While the simulated trajectories in sensor fusion (figures 6.5, 6.8)as well as LIDAR-based trials (figures 6.6, 6.9) are similar to their real, tracked counterparts, there is less similarity when compared to RADAR-based trial trajectories (figures 6.7, 6.10). This indicates a lesser fidelity of the simulator when simulating RADAR-based obstacle avoidance in real scenarios, where the reflectivity of the environment is a significant factor.

Course 2 results

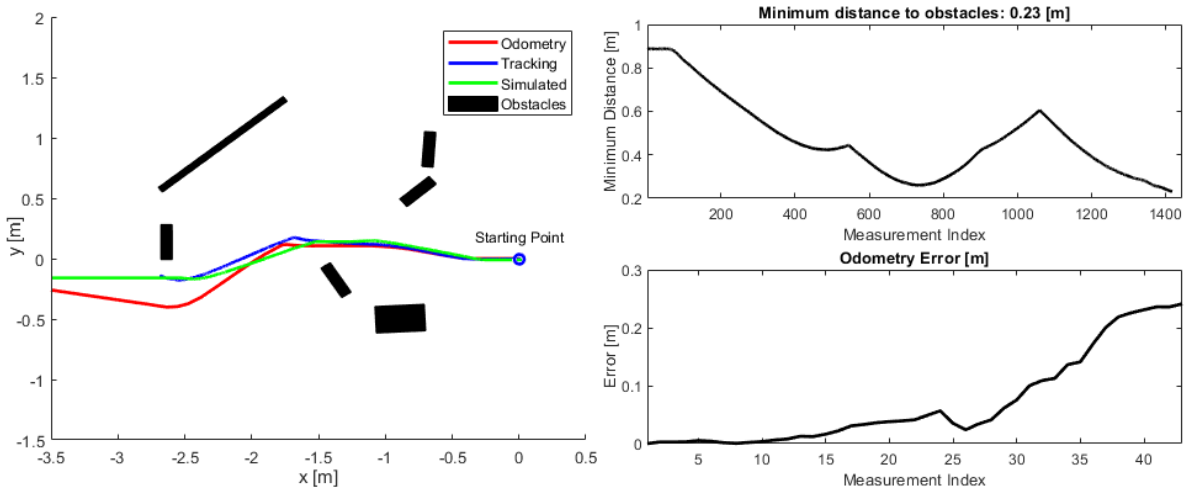


Figure 6.11: Trial 3, LIDAR and RADAR integration

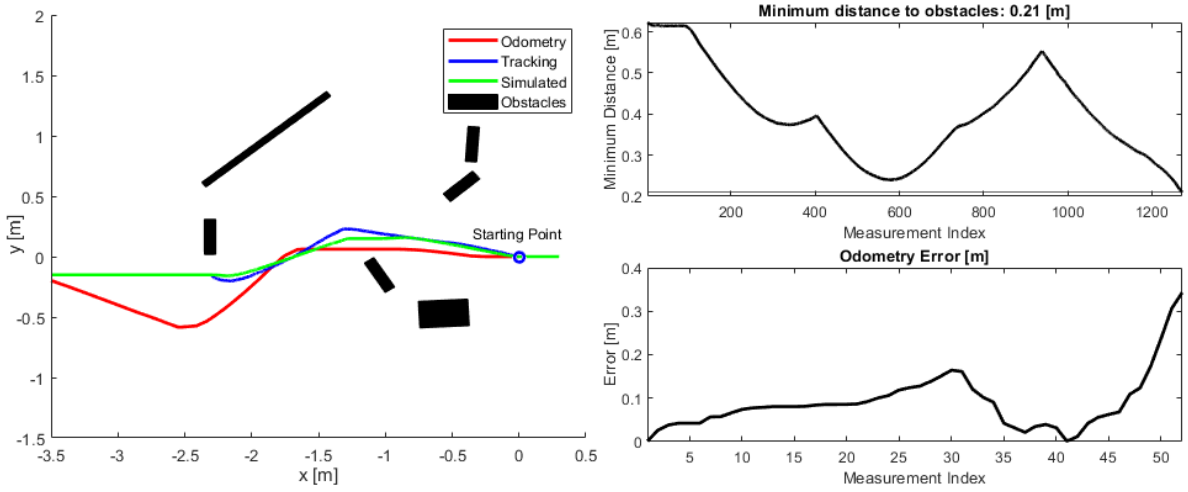


Figure 6.12: Trial 3, LIDAR sensor active

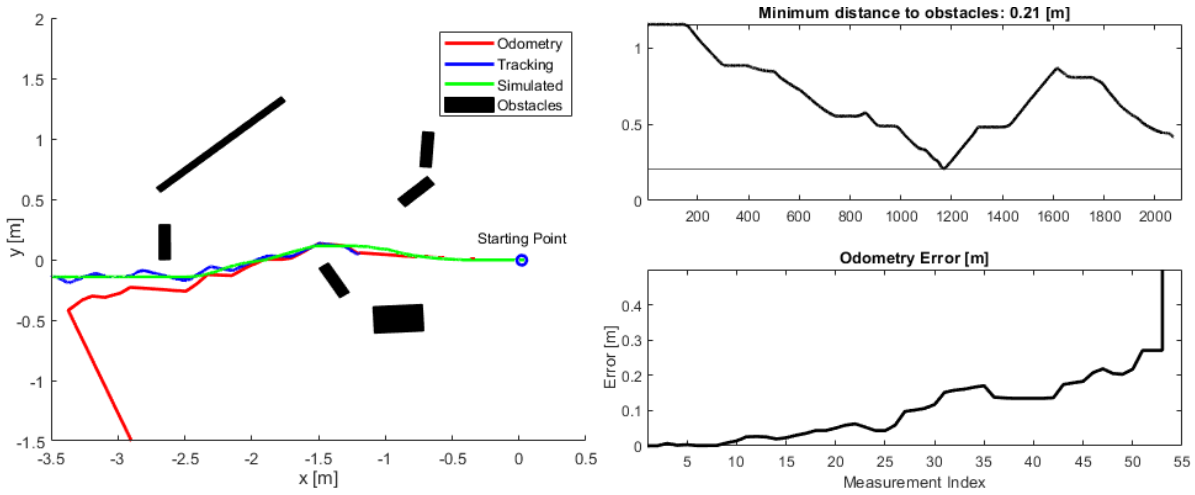


Figure 6.13: Trial 3, RADAR sensor active

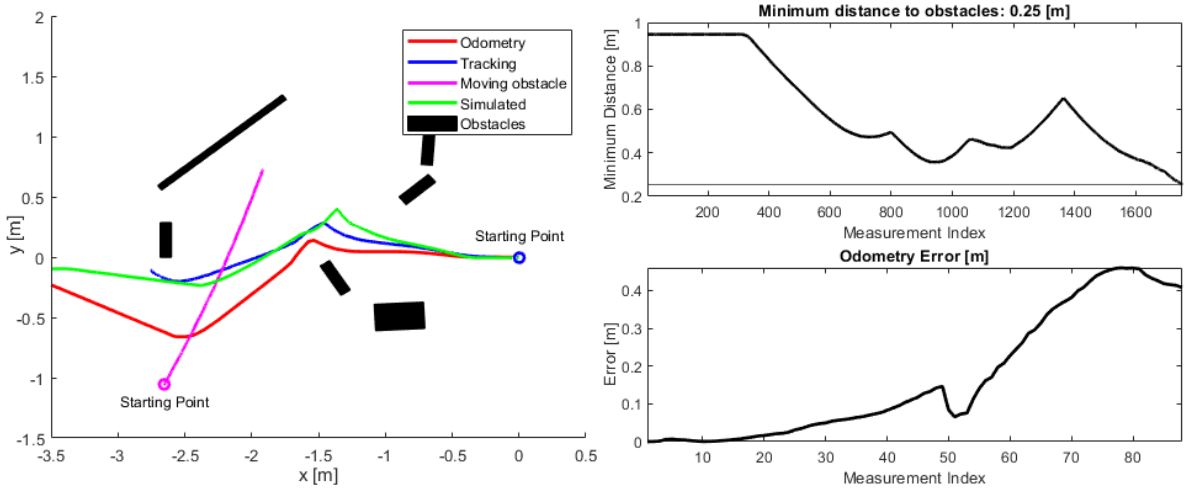


Figure 6.14: Trial 4, LIDAR and RADAR integration

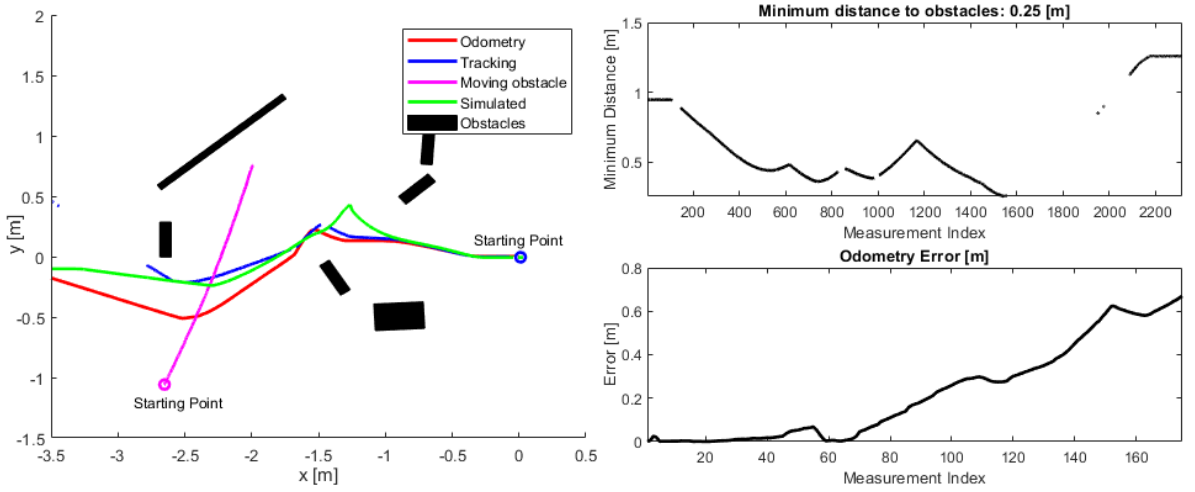


Figure 6.15: Trial 4, LIDAR sensor active

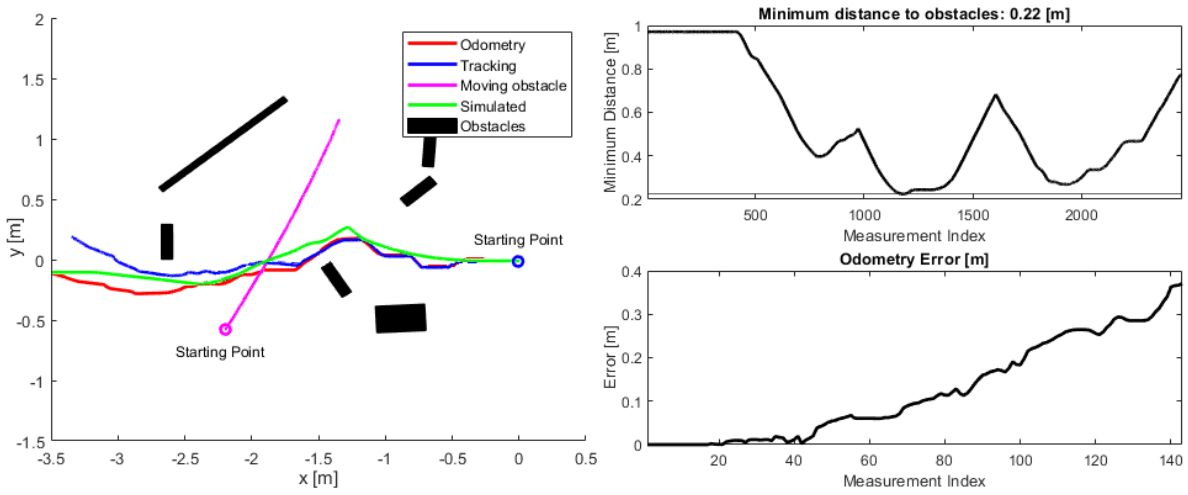


Figure 6.16: Trial 4, RADAR sensor active

Some observations can be made about the results of trials 3 and 4, run on course 2:

- Once again, all trials were completed successfully, with the robot navigating the obstacle course and arriving at its intended goal;
- Similarly to course 1, course 2's results show that the robot was able to navigate the obstacle course while keeping the minimum safe distance from any obstacles. However, the minimum safe distances registered along the trials on course 2 are lower than those registered on course 1. Such behaviour was expected due to the higher density of obstacles in course 2 when compared to course 1, with the robot having to traverse narrower paths to achieve its goal.
- Unlike the trials run on course 1, some of the trials performed on course 2 registered a loss of tracking during the initial phase of the trial, prior to the robot initiating its navigation during the obstacle course (figures 6.13, 6.16). However, this is noticeable prior to the robot initiating avoidance maneuvers, with the system being able to locate the robot during the initial section of the maze. Loss of tracking is practically nonexistant in the obstacle course portion of the trial. For this reason, the trials are considered a success;
- Once again, the odometry estimation algorithm is accurate at estimating the location of the robot in the initial part of the trials. However, small errors are propagated throughout the robot's path, contributing to an increasing disparity between the real, tracked position and the odometry estimation (most notably in figures 6.12, 6.14 and 6.15). This can be attributed to longer periods of strafing coupled with forward motion (due to the increased degree of avoidance required when compared to course 1) registered inaccurately due to the same inaccuracies in the assembly of the Omni-ANT platform previously mentioned;
- In the case of figure 6.13, the odometry algorithm fails to estimate the position of the robot towards the end of the trial, with a sudden and significant shift in trend. This phenomenon can be attributed

to a failure in the wheel encoders, leading to an incorrect incrementation in the odometry algorithm. However, this failure occurred after the maze portion of the trial, which maintains the trial's validity;

Chapter 7

Conclusions

In this chapter, an analysis of the developed work is made, discussing the conclusions of the implementation as well as the overall contributions this work poses to the scientific community and concluding with suggestions for future work and improvements in this field or based on this dissertation.

A few conclusions can be drawn pertaining the software and hardware implementations and results presented in the previous chapter:

- Velocity sensing (via RADAR) in the context of small-scale autonomous vehicles navigating static or otherwise non-significantly dynamic environments (dynamic obstacles moving at low velocities) does not offer an improvement in performance when compared to the avoidance system using solely proximity sensing, introducing erratic avoidance trajectories instead;
- Despite the presence of disparities between the tracking and simulated results, the behaviour of the models is similar, where both perform the same general avoidance maneuvers and follow a similar trajectory throughout the obstacle courses in each trial. However, there is a noticeable disparity in behaviour between the solely RADAR-based real and simulated models (due to high reflectivity of real environments). As such, the Omni-ANT simulator model is considered validated for LIDAR-based and sensor fusion trials;
- The virtual simulator can be used as a tool to run experiments in easily generated scenarios (within the validated conditions of sensor fusion or LIDAR-based trials) without risk of damage to any intervenient prior to the real world testing trials. However, it can not be used to predict the absolute outcome of a real trial in similar conditions without thorough testing. The sensors involved in this implementation as well as all the hardware are subject to mechanical failure as well as innacuracies. Analyzing the results from the real world trials, the robot trajectory is not as smooth and continuous as its simulated counterpart. These can be minimized by the implementation of specialized estimation algorithms such as Kalman filtering which, while effective, increase the computational bulk of such a system and is not feasible for implementation in the same conditions as the developed prototype;
- Due to the high degree of modularity that ROS grants to any machine running its software, the

prototype can easily be installed in a different ROS network, requiring no additional setup other than setting up the correct IP addresses of the machines involved and ensuring ROS version compatibility between nodes. This framework is an essential part of the system, which not only provided a simplified and efficient baseline for the interaction of the different peripherals of the prototype but also granted a high degree of versatility to the system, allowing for the communication and acquisition of data in several different terminals running ROS, which was a determining factor in the experimental portion of this work.

The main contributions of this work are the following:

- Major improvement of the Omni-ANT platform developed over [10], [11] and [12] in the form of the integration of a more powerful processing unit for execution of more computationally demanding algorithms and the addition of radar sensing capabilities;
- Development of a more versatile obstacle avoidance algorithm allowing for the usage of LIDAR or RADAR sensor readings exclusively or the simultaneous usage of both devices for sensor fusion;
- Development of a highly modular autonomous obstacle avoidance prototype with the integration of the ROS framework;
- Analysis of the performance of RADAR and LIDAR sensing technology applied to small-scale obstacle avoidance scenarios;
- Development of a virtual simulator of a small-scale vehicle with obstacle detection and collision avoidance capabilities based on proximity as well as velocity readings.

With the complexity associated with creating a fully autonomous obstacle detection and collision avoidance prototype, some guidelines are provided for future work based on this implementation:

- Implementation of Simultaneous Mapping And Localization (SLAM) on the ROS framework and trajectory planning algorithms (both real and simulated models);
- Integration of vision sensors (cameras) for obstacle identification based on intelligent systems analysis;
- Performance analysis of the implementation of either or both LIDAR and RADAR sensors in large-scale obstacle avoidance scenarios.

Bibliography

[1] K. Bimbraw. Autonomous Cars : Past , Present and Future. *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 01:191–198, 2020.

[2] Ü. Özgüner, C. Stiller, and K. Redmill. Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience. *Proceedings of the IEEE*, 95(2):397–412, 2007. ISSN 00189219. doi: 10.1109/JPROC.2006.888394.

[3] A. Yuste and M. Palma. Scanning our Past from Madrid: Leonardo Torres Quevedo. *Proceedings of the IEEE*, 93(7):1379–1382, 2005. ISSN 0018-9219. doi: 10.1109/JPROC.2005.851230.

[4] E. D. Dickmanns. Developing the Sense of Vision for Autonomous Road Vehicles at UniBwM. *Computer*, 50(12):24–31, 2017. ISSN 00189162. doi: 10.1109/MC.2017.4451214.

[5] S. T. Padgett and A. F. Browne. Vector-based robot obstacle avoidance using LIDAR and mecanum drive. In *SoutheastCon 2017*, pages 1–5, 2017.

[6] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu. The obstacle detection and obstacle avoidance algorithm based on 2-D lidar. In *2015 IEEE International Conference on Information and Automation*, pages 1648–1653, 2015.

[7] D. Hutabarat, M. Rivai, D. Purwanto, and H. Hutomo. Lidar-based Obstacle Avoidance for the Autonomous Mobile Robot. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, pages 197–202, 2019. doi: 10.1109/ICTS.2019.8850952.

[8] Y. Shim and G. Kim. Range Sensor-Based Efficient Obstacle Avoidance through Selective Decision-Making. *Sensors*, 18:1030, 2018. doi: 10.3390/s18041030.

[9] I. Ruiz, D. Aufderheide, and U. Witkowski. Radar Sensor Implementation into a Small Autonomous Vehicle. In U. Rückert, S. Joaquin, and W. Felix, editors, *Advances in Autonomous Mini Robots*, pages 123–132. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-27482-4.

[10] A. Casqueiro, D. Ruivo, A. Moutinho, and J. Martins. Improving Teleoperation with Vibration Force Feedback and Anti-Collision Methods. In L. Reis, A. Moreira, P. Lima, L. Montano, and V. Muñoz-Martinez, editors, *Robot 2015: Second Iberian Robotics Conference. Advances in Intelligent Systems and Computing*, volume 417, pages 269–281. Springer, Cham., 2016. doi: 10.1007/978-3-319-27146-0_21.

[11] D. Salvador. Formação de Veículos Holonómicos Terrestres com Sistema Anti-colisão. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2017.

[12] R. Severiano. Development of a Lightweight Visual-based Pose Estimation Sensor: Implementation and Validation. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2018.

[13] M. Richards, W. Holm, and J. Scheer. *Principles of Modern Radar: Basic Principles, Volume 1*. Electromagnetics and Radar. Institution of Engineering and Technology, 2010. ISBN 9781891121524.

[14] J. Gowar. *Optical Communication Systems (Optoelectronics)*. NewYork: Prentice-Hall, 1984.

[15] A. Jelalian. *Laser Radar Systems*. Artech House, 1992. ISBN 9780890065549.

[16] N. Baghdadi and M. Zribi. *Optical Remote Sensing of Land Surfaces*. Elsevier, 2016. ISBN 9781785481024.

[17] H. Weber. LIDAR Sensor Functionality and Variants. unpublished, 2018. URL `https://cdn.sick.com/media/docs/3/63/963/Whitepaper_LiDAR_en_IM0079963.PDF`.

[18] X. Mao, D. Inoue, S. Kato, and M. Kagami. Amplitude-Modulated Laser Radar for Range and Speed Measurement in Car Applications. *IEEE Transactions on Intelligent Transportation Systems*, 13(1): 408–413, 2012. ISSN 15249050. doi: 10.1109/TITS.2011.2162627.

[19] R. Frehlich and L. Cornman. Estimating spatial velocity statistics with coherent Doppler lidar. *Journal of Atmospheric and Oceanic Technology*, 19(3):355–366, 2002. ISSN 07390572. doi: 10.1175/1520-0426-19.3.355.

[20] J. M. Rüeger. *Propagation of Electromagnetic Waves Through the Atmosphere*, pages 48–49. Springer Berlin Heidelberg, 1996. ISBN 978-3-642-80233-1. doi: 10.1007/978-3-642-80233-1_5.

[21] W. Sun, Y. Hu, D. G. MacDonnell, C. Weimer, and R. R. Baize. Technique to separate lidar signal and sunlight. *Optics express*, 24(12):12949–12954, 2016.

[22] M. Kutila, P. Pyykönen, W. Ritter, O. Sawade, and B. Schäufele. Automotive LIDAR sensor development scenarios for harsh weather conditions. In *IEEE Conference on Intelligent Transportation Systems (ITSC) Proceedings*, pages 265–270, 2016. ISBN 9781509018895. doi: 10.1109/ITSC.2016.7795565.

[23] A. Hassen. *Indicators for the Signal Degradation and Optimization of Automotive Radar Sensors under Adverse Weather Conditions*. PhD thesis, TU Darmstadt, 2007.

[24] A. Kamann, P. Held, F. Perras, P. Zaumseil, T. Brandmeier, and U. Schwarz. Automotive Radar Multipath Propagation in Uncertain Environments. pages 859–864, 11 2018. doi: 10.1109/ITSC.2018.8570016.

[25] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Autonomous Robot Vehicles*, pages 396–404. Springer New York, 1990. ISBN 978-1-4613-8997-2. doi: 10.1007/978-1-4613-8997-2_2.

[26] R. Palm and A. Bouguerra. Navigation of Mobile Robots by Potential Field Methods and Market-based Optimization. In *5th European Conference on Mobile Robots, ECMR 2011, September 7-9, 2011, Örebro, Sweden*, pages 207–212, 2011.

[27] R. Palm and A. Bouguerra. Particle Swarm Optimization of Potential Fields for Obstacle Avoidance. In *Proceedings of RARM 2013*, pages 117–123, 2013.

[28] P. Khosla and R. Volpe. Superquadric artificial potentials for obstacle avoidance and approach. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1778–1784. IEEE, 1988.

[29] P. Fiorini and Z. Shiller. Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998. doi: 10.1177/027836499801700706.

[30] B. Kluge and E. Prassler. *Recursive Probabilistic Velocity Obstacles for Reflective Navigation*, pages 71–79. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-32854-4. doi: 10.1007/10991459_8.

[31] Z. Allawi. An ABC-Optimized Reciprocal Velocity Obstacles Algorithm for Navigation of Multiple Mobile Robots. The Second Engineering Conference of Control, Computers and Mechatronics Engineering (ECCCM2), 2015.

[32] P. Smalley. Laser safety: Regulations, standards, and guidelines for practice. *Lasers for Medical Applications*, pages 725–759, 2013. doi: 10.1533/9780857097545.4.725.

[33] R. H. Bishop. *The Mechatronics Handbook-2 Volume Set*. CRC press, 2002.

[34] I. Virgala, P. Frankovský, and M. Kenderová. Friction Effect Analysis of a DC Motor. *American Journal of Mechanical Engineering*, 1(1):1–5, 2013. doi: 10.12691/ajme-1-1-1.

[35] J. Gonçalves, J. Lima, P. Costa, and A. Moreira. Modeling and Simulation of the EMG30 Geared Motor with Encoder Resorting to SimTwo: The Official Robot@Factory Simulator. *Advances in Sustainable and Competitive Manufacturing Systems*, pages 307–314, 2013. doi: 10.1007/978-3-319-00557-7_25.

[36] MMWAVE SDK User Guide. URL `http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02_01_00_04/exports/mmwave_sdk_user_guide.pdf`. Accessed in 24/08/2019.

[37] MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive - I2C mode documentation. URL `https://www.robot-electronics.co.uk/htm/md25i2c.htm`. Accessed in 24/08/2019.