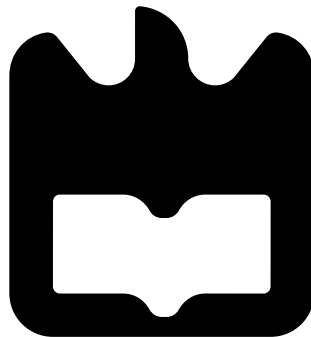




Francisco  
Marques dos  
Santos

Estudo do uso do radar na navegação em  
espaços interiores

Studying the use of radar for indoor navigation







**Francisco  
Marques dos  
Santos**

**Estudo do uso do radar na navegação em  
espaços interiores**

**Studying the use of radar for indoor navigation**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Artur Pereira, Professor Auxiliar do Departamento de Electrónica e Informática da Universidade de Aveiro e co-orientação científica do Doutor Eurico Pedrosa, Investigador da Universidade de Aveiro.



**o júri / the jury**

presidente / president

**Professor Doutor Pedro Nicolau Faria de Fonseca**

Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

**Professor Doutor António Paulo Gomes Mendes Moreira**

Professor Associado da Faculdade de Engenharia da Universidade do Porto

**Professor Doutor Artur José Carneiro Pereira**

Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientador)



## **agradecimentos**

Em primeiro lugar queria agradecer à minha mãe por todo o apoio ao longo do meu percurso académico. Ao professor Artur Pereira pela paciência e constante entusiasmo que me deu a resiliência para continuar e acabar o projeto. Ao meu co-orientador, Eurico Pedrosa, pela disponibilidade, espírito crítico e apoio constante ao longo do desenvolvimento deste trabalho. Finalmente, agradeço ao grupo de investigação RETIOT pela ajuda crucial nos testes práticos realizados ao longo desta dissertação. Muito obrigado a todos.

Este trabalho foi financiado e desenvolvido no âmbito do projeto RETIOT, POCI-01-0145-FEDER-016432.



## Resumo

Nos últimos anos, o uso de robôs móveis autônomos que operam em ambientes interiores aumentou significativamente. Eles são usados não apenas em contexto de indústria, mas estão a surgir nas nossas casas e em ambientes de escritório. Eles são tipicamente usados para transporte, telepresença e limpeza. A percepção para propósito de navegação autônoma do robô geralmente depende de medidor de distâncias a laser, sensores de profundidade de infravermelho ou a típica câmara. No entanto, os recentes avanços na tecnologia de radar de onda contínua modulada em frequência permitem uma nova solução para o problema de percepção. O foco principal deste trabalho é avaliar o Radio Detection And Ranging (radar) Frequency-Modulated Continuous-Wave (FMCW) como sensor para navegação autônoma interior. Para isso iremos comparar com o Light Detection And Ranging (LiDAR) 2D que é muito comum quando se toca a robôs autônomos.

Primeiro, será apresentado um estudo entre cada tecnologia, incluindo o princípio operacional de cada uma, o trabalho em que estão ser usados e quais as suas limitações que cada uma tem. Depois descobriremos como podemos usar o Robot Operating System (ROS) e algoritmos de última geração para combater o problema de navegação autônoma. Depois, descreveremos os componentes de hardware e software e como eles estão interconectados produzir uma plataforma robótica adequada que será usada para executar tarefas de navegação. Várias experiências foram implementadas para o uso de radar em navegação. Os resultados obtidos comprovam a validade do radar FMCW como sensor de obstáculos. Eles também mostram que existem casos onde o LiDAR não deteta ou deteta mal vários objetos onde são melhor detetados pelo radar. Também mostram que menos densidade nos dados não afeta significativamente a atualização do mapa de custos.

Finalmente, será realizado um trabalho exploratório que tenta usar o leituras de canal doppler do radar para ajudar em trajetórias de percurso mais seguras para ambientes interiores sociais.



## Abstract

Over the past few years the use of autonomous mobile robots that operate in indoor environments has grown significantly. They are used not only in industry settings but are creeping in on our homes and office environments. They are commonly used for application such as transportation, telepresence and cleaning. The robot's perception for navigation purposes usually depends on laser range finders, infra red depth sensors or the typical cameras. However recent advancements on Frequency Modulated Continuous Wave radar technology permits a new solution for this navigation task. The main focus of this dissertation is the evaluation of the FMCW radar as a sensor for indoor navigation. To do that we will compare it with the 2D-LiDAR, that is the very common when it comes to robotic sensors.

First, a study between each technology will be presented including the operating principle of each one, applications and what limitations they have. After that we will uncover how we can use the Robot Operating System (ROS) framework and state of the art algorithms to address the autonomous navigation problem. Afterwords we will describe the robot's hardware and software components and how they are interconnected to produce a suitable robotic platform that will be used to do navigation tasks. Several experiments were devised and implemented to evaluate the usage of the radar in navigation. The results obtained proved the validity of using the FMCW radar as an obstacle detector sensor device. They showed some type of objects not or poorly detected by a LiDAR can be better detected by the radar. They also showed that the less dense data produced by the radar does not have a significant impact in keeping the navigation costmaps updated. Finally an exploratory work will be conducted that tries to use the doppler channel readings of the radar to aid in more safe pathing trajectories for social indoor environments.



# Contents

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>Acronyms</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context . . . . .	1
1.2 Objectives . . . . .	1
1.3 Overview . . . . .	2
1.4 Document Structure . . . . .	2
<b>2 Range Sensing</b>	<b>3</b>
2.1 LiDAR . . . . .	3
2.1.1 Applications . . . . .	3
2.1.1.1 Autonomous Vehicles . . . . .	3
2.1.1.2 Robotics . . . . .	4
2.1.2 Operating Principle . . . . .	5
2.1.2.1 Time of Flight . . . . .	5
2.1.2.2 Phase Based . . . . .	5
2.1.3 Limitations . . . . .	6
2.2 MIMO Linear FMCW Radar . . . . .	7
2.2.1 Applications . . . . .	7
2.2.1.1 Pedestrian detection . . . . .	7
2.2.1.2 Localization of Land Vehicles . . . . .	8
2.2.1.3 Monitoring of human vital signs . . . . .	8
2.2.2 Operating principle . . . . .	8
2.2.2.1 Range Detection . . . . .	9
2.2.2.2 Velocity Detection . . . . .	10
2.2.2.3 Angle Detection . . . . .	10
2.2.3 Limitations . . . . .	12
2.3 Ultrasonic Sensors . . . . .	12

2.3.1	Applications . . . . .	12
2.3.1.1	Robotics . . . . .	12
2.3.1.2	Ground Vehicles . . . . .	12
2.3.1.3	Medicine . . . . .	12
2.3.2	Operating Principle . . . . .	12
2.4	Cameras . . . . .	13
2.5	Summary . . . . .	13
<b>3</b>	<b>Robot Navigation using ROS</b>	<b>15</b>
3.1	Robot Operating System . . . . .	15
3.1.1	ROS architecture . . . . .	15
3.1.2	ROS tools . . . . .	17
3.2	Navigation Concepts . . . . .	17
3.3	Path Planning and Motion Control . . . . .	18
3.3.1	Global Methods for Path Planning . . . . .	19
3.3.1.1	Dijkstra algorithm . . . . .	19
3.3.1.2	A* algorithm . . . . .	19
3.3.2	Local Methods for Motion Control . . . . .	20
3.3.2.1	DWA planner . . . . .	21
3.3.2.2	Trajectory Rollout . . . . .	22
3.4	ROS Navigation stack . . . . .	22
3.4.1	Requirements . . . . .	23
3.4.1.1	Transform Configuration . . . . .	23
3.4.1.2	Sensor sources . . . . .	23
3.4.1.3	Odometry . . . . .	24
3.4.1.4	Base Controller . . . . .	24
3.4.1.5	Map . . . . .	24
3.4.2	Navigation Node . . . . .	24
3.4.2.1	Global Planner . . . . .	26
3.4.2.2	Local Planner . . . . .	26
3.4.2.3	Local and Global Costmaps . . . . .	27
3.5	Summary . . . . .	28
<b>4</b>	<b>Evaluation Platform</b>	<b>29</b>
4.1	Hardware . . . . .	29
4.1.1	Turtlebot2 . . . . .	30
4.1.2	FMCW radar . . . . .	30
4.1.3	LiDAR . . . . .	32
4.2	Software . . . . .	32
4.2.1	ROS Interface . . . . .	33
4.2.2	Visualization of the radar point cloud . . . . .	34
4.2.3	Intensity Filter . . . . .	35
4.2.4	Navigation Module . . . . .	35

4.3	Summary . . . . .	36
<b>5</b>	<b>Results</b>	<b>37</b>
5.1	Static Obstacles in controlled environment . . . . .	37
5.1.1	Experimental setup . . . . .	38
5.1.2	Results . . . . .	39
5.1.2.1	Office Chair . . . . .	39
5.1.2.2	Four Legged Chair . . . . .	42
5.1.2.3	Garbage Bin . . . . .	44
5.1.2.4	Low height box . . . . .	45
5.1.2.5	Acrylic tube . . . . .	47
5.1.2.6	Robot (turtlebot2) . . . . .	49
5.1.3	Discussion . . . . .	49
5.2	Dynamic Obstacles in controlled environment . . . . .	49
5.2.1	Experimental Setup . . . . .	49
5.2.2	Results . . . . .	50
5.2.2.1	Person . . . . .	50
5.2.2.2	Mobile Robot . . . . .	51
5.2.3	Discussion . . . . .	51
5.3	Dynamic Obstacles in uncontrolled environment . . . . .	52
5.3.1	Experimental Setup . . . . .	52
5.3.2	Results . . . . .	53
5.3.3	Discussion . . . . .	54
5.4	Summary . . . . .	55
<b>6</b>	<b>Towards The Use Of Doppler To Enrich Obstacle Avoidance</b>	<b>57</b>
6.1	Layered Costmaps . . . . .	57
6.2	Inflation Layer . . . . .	57
6.3	Doppler Layer . . . . .	58
6.4	Discussion . . . . .	60
<b>7</b>	<b>Conclusion and Future Work</b>	<b>61</b>
7.1	Conclusion . . . . .	61
7.2	Future Work . . . . .	61
	<b>Bibliography</b>	<b>63</b>
<b>A</b>	<b>Simulation of radar range and velocity estimation</b>	<b>67</b>
A.1	Simulating range detection . . . . .	67
A.2	Simulating velocity detection . . . . .	70



# List of Figures

2.1	Example of LiDAR technology being used for road perception . . . . .	4
2.2	Example of two robots equipped with LiDAR . . . . .	5
2.3	Phase based method for measuring range . . . . .	6
2.4	2D LiDAR only detecting objects in one plane . . . . .	7
2.5	Representation of a chirp . . . . .	9
2.6	IF signal creation to retrieve range information of the target . . . . .	9
2.7	2D FFT for retrieving velocity information . . . . .	11
2.8	Angle of arrival estimation using more than one antenna . . . . .	11
2.9	Ultrasonic wave operating principle . . . . .	13
3.1	ROS communication architecture overview . . . . .	16
3.2	Problems regarding autonomous navigation . . . . .	18
3.3	Dijkstra algorithm . . . . .	20
3.4	Rviz view of the ROS Navigation Stack . . . . .	22
3.5	Example of the transform relationships . . . . .	23
3.6	Obstacle detections by both radar and lidar . . . . .	24
3.7	Example of a map created in the IRIS laboratory . . . . .	25
3.8	Navigation stack block diagram . . . . .	25
3.9	Cost cloud published in rviz . . . . .	27
3.10	Determination of the master costmap taking into account different costmap layers	28
4.1	Modified Turtlebot2 used in this work . . . . .	29
4.2	Turtlebot 2 platform . . . . .	30
4.3	Turtlebot 2 dimension specifications . . . . .	30
4.4	Texas Instruments AWR1642BOOST evaluation board . . . . .	31
4.5	Radiation pattern of the antenna . . . . .	31
4.6	Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder overview . . . . .	32
4.7	Block diagram of the software architecture designed for this work . . . . .	33
4.8	Part of the mmwave demo packet . . . . .	34
4.9	Visualization markers displaying radar data information . . . . .	34
4.10	Example of filtering the pointcloud . . . . .	35
5.1	Different obstacles used for the experiment . . . . .	38

5.2	Scenario Constructed for the experiment . . . . .	38
5.3	Demonstration of the course of test . . . . .	39
5.4	Robot colliding multiple times with office chair . . . . .	39
5.5	Navigation data with the office chair as an obstacle with LiDAR . . . . .	40
5.6	Robot avoiding with office chair . . . . .	40
5.7	Navigation data with the office chair as an obstacle with FMCW radar . . . . .	41
5.8	Trajectory of the robot for the office chair's case . . . . .	41
5.9	Robot getting stuck in four legged chair . . . . .	42
5.10	Navigation data with the four legged chair as an obstacle with LiDAR . . . . .	42
5.11	Robot avoiding the four legged chair . . . . .	43
5.12	Navigation data with the four legged chair as an obstacle with FMCW radar . . . . .	43
5.13	Trajectory of the robot for the four-legged chair's case . . . . .	44
5.14	Two instances of the robot's navigation with the garbage bin as an obstacle . . . . .	44
5.15	Robot avoiding garbage bin . . . . .	45
5.16	Navigation data with the garbage bin as an obstacle with FMCW radar . . . . .	45
5.17	Robot collision with box . . . . .	46
5.18	Robot avoiding box . . . . .	46
5.19	Navigation data with box as an obstacle and with the FMCW radar . . . . .	47
5.20	Robot getting stuck in acrylic tube . . . . .	47
5.21	Navigation data with acrylic tube as an obstacle and with LiDAR . . . . .	48
5.22	Robot avoiding acrylic tube box . . . . .	48
5.23	Navigation data with the acrylic tube as an obstacle with the FMCW radar . . . . .	49
5.24	Demonstration of the course of test . . . . .	50
5.25	Robot avoiding the person . . . . .	51
5.26	Navigation data with the person as a dynamic obstacle . . . . .	51
5.27	Robot avoiding dynamic robot . . . . .	52
5.28	Navigation data with the robot as a dynamic obstacle . . . . .	52
5.29	Experimental setup . . . . .	53
5.30	Four instances of the robot in the experiment . . . . .	54
6.1	A stack of costmap layers . . . . .	58
6.2	Inflation layer specifications . . . . .	59
6.3	Doppler layer manipulation of the master costmap . . . . .	60
A.1	Demonstration of the TX signal in blue and RX signal in other colours . . . . .	68
A.2	Range detection using FMCW radar . . . . .	68
A.3	Velocity detection using FMCW radar . . . . .	70

# List of Tables

A.1	Parameters used for simulation for range estimation. . . . .	67
A.2	Parameters used for simulation of velocity estimation . . . . .	70



# Acronyms

<b>ROS</b>	Robot Operating System
<b>SLAM</b>	Simultaneous localization and mapping
<b>AMCL</b>	Adaptive Monte Carlo Localization
<b>LiDAR</b>	Light Detection And Ranging
<b>IF</b>	Intermediate Frequency
<b>ADC</b>	Analog Digital Converter
<b>FFT</b>	Fast Fourier transform
<b>DSP</b>	Digital Signal Processor
<b>FMCW</b>	Frequency-Modulated Continuous-Wave
<b>radar</b>	Radio Detection And Ranging
<b>mmWave</b>	Millimeter Wave
<b>MIMO</b>	Multiple Input Multiple Output
<b>RISS</b>	Reduced Inertial Sensor System
<b>ECG</b>	electrocardiogram
<b>SNR</b>	Signal Noise Ratio
<b>PRM</b>	Probabilistic Roadmaps
<b>IRIS</b>	Institute of Robotics and Intelligent Systems
<b>PCL</b>	Point Cloud Library
<b>DWA</b>	Dynamic Window Approach
<b>VFH</b>	Vector Field Histogram
<b>TI</b>	Texas Instruments

<b>PIR</b>	Passive infrared
<b>GNSS</b>	Global Navigation Satellite System
<b>TLV</b>	Type Length Value

# Chapter 1

## Introduction

### 1.1 Context

The use of autonomous mobile robots has been rapidly expanding over the last few years. Whether it is in industry, our homes or in an office like environment, they provide endless applications. A few examples are the use in healthcare, military, agriculture, cleaning, construction and space. By definition a robot is a device able to perform human activities [1]. It can fulfill a wide variety of tasks that require the link between perception and action. A mobile autonomous robot is a system that has total mobility in its environment, that has limited human interaction and has the ability of perceiving what is around it. In terms of navigation purposes robots usually rely on LiDAR technology which has some limitations in accurately perceiving what is around them. This is usually the case when the mobile robot is exposed directly to sunlight, smoke, or when trying to detect objects of low height. Radar technology has constantly been upgraded and is not affected by the previous conditions. Due to recent advancements in terms of hardware and software, new high bandwidth low power radars are now available for the use in perception capabilities for the robot. Specifically the FMCW radar technology has been constantly improving in terms of being an obstacle detector and is a candidate to be used to address the past mentioned problems by being a supportive sensor in the robotic navigation module.

### 1.2 Objectives

The main idea behind this dissertation is to assess the use of radar technology in navigation, either as complement technology or by being the sole sensor component used for perception. Since LiDAR technology is one of the most common approach to sense the environment for navigation purposes, it was decided to use it for comparison. Thus, a primary objective of this work is to construct a robotic mobile platform equipped with a LiDAR and an FMCW radar to support the evaluation. Also a proper software architecture must be defined and implemented. Subsequently, a set of experiments must be planned and implemented, the results of which can provide the desired evaluation. The radar sensor provides Doppler data representing the radial velocity of objects in relation to the sensor. This data can be possible

used to enhance the navigation capabilities of the robot. As a final objective, the use of this Doppler data must be studied.

### 1.3 Overview

The work started with a study of the FMCW radar, focused on the operating principles and on typical applications. Since from the beginning it was decided LiDAR will be used for comparison purposes, the same kind of study was also conducted in this technology. Almost at the same time, a deep study of the Robot Operating System (ROS) was also performed. ROS is nowadays a quite popular environment to develop robotics applications and so it was decided to use it. A TurtleBot 2 was chosen as the basis for constructing the evaluation robotics platform. In terms of hardware, it was equipped with an FMCW radar, a Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder, and a computer unit. In terms of software, on top of ROS, several modules were developed. Several experiments were devised and implemented to evaluate the usage of the radar in navigation. Finally, an exploratory work was done to evaluate the use of the Doppler data provided by the radar. Also an adaptation of the ROS navigation stack is proposed in order to take advantages of the Doppler data.

### 1.4 Document Structure

The dissertation is arranged as follow. In Chapter 2 a comparative study is done between the LiDAR technology and the FMCW radar. It will give a brief overview of what work is being done using this sensors and what operating principle they rely on as well as describe their limitations. We will also explore the use of ultrasonic sensors and cameras. Chapter 3 will provide a brief overview of some basic concepts regarding this dissertation's work regarding ROS framework, navigation concepts and finally the description of the ROS navigation stack. In Chapter 4 we will discuss what hardware and software will be used in this work and how they are interconnected to create a suitable platform for indoor navigation using the previous highlighted sensors. In Chapter 5 we show the results of various experiments that try to compare the use of LiDAR and the FMCW radar as obstacle detectors. In Chapter 6 we will discuss how we can use the relative radial velocity of targets given by the FMCW radar to manipulate the costmaps of the navigation module in order to avoid incoming obstacles. Finally in Chapter 7 we will make a summary of the work done in this dissertation and what conclusions we found along the way. We will also discuss what future work can be done using this type of sensors for indoor autonomous navigation.

## Chapter 2

# Range Sensing

In order to do complex tasks and compute different complex algorithms robots require some type of information that relates to its state and environment. This information comes from its sensor sources. The most typical type of sensors used for perception are proximity sensors such as laser range finders (LiDAR), ultrasonic sensors or the use of a camera. However robots may have to deal with a wide variety of changing environments like variation in luminosity, weather conditions, presence of dust and smoke, object proprieties, etc. The previous highlighted sensors may show difficulty or be completely ineffective under these conditions.

The new emerging Millimeter Wave (mmWave) FMCW radar technology with high frequency and bandwidth of 77-81 GHz is now beginning to be suitable for use in robotic applications, and it may provide a solution for the previous conditions. In this chapter we will give a brief overview of current work being done using LiDAR and FMCW radar, how they work, and what limitations each sensor has. We will also evaluate the use of cameras and ultrasonic sensors.

## 2.1 LiDAR

LiDAR is a remote sensing technology (acquires information of an object without making physical contact) , that measures the distance to objects. It has been a staple in sensing technology in recent history. It has almost an unlimited number of applications [2] due to its generation of dense data. Some of the most standing out ones are in autonomous vehicles and robotics.

### 2.1.1 Applications

#### 2.1.1.1 Autonomous Vehicles

The use of LiDARs in autonomous vehicles has been a common trend for a while now. Fig. 2.1 illustrates how several of these sensors are used to provide full road environment perception. The LiDARs accurate depth information combined with a high field of view enables the development of advanced navigation systems that are able to perform self-driving.

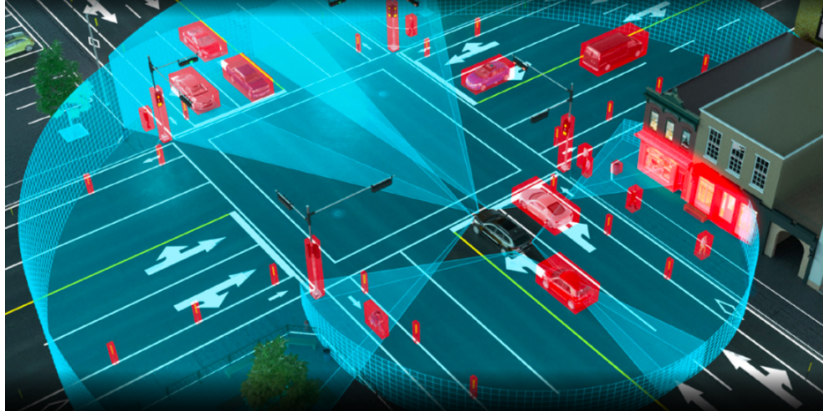


Figure 2.1: Example of LiDAR technology being used for road perception [3]

One application example of this is the work done by Dominguez et al [4] where a perception system for ground vehicles was implemented using segmentation, clustering and tracking techniques. However, it was noted by the authors that it was hard to distinguish between closely separated obstacles adding that a movement detection algorithm should be implemented to solve this problem.

Catapanga and Ramos [5] proposed a low cost solution for object detection, classification and ranging by the use of an inexpensive 2D pulsed light LiDAR. The system was able to provide reliable detection of 1 meter wide obstacles at distances less than 10 meters.

However there are some concerns about this type of technology. The CEO of Tesla, Elon Musk, has been very critical of it. Recently, at Tesla's first Autonomy Day event, he said *"Anyone relying on LiDAR is doomed. Doomed!"* [6]. He believes that cameras, radar and ultrasonic sensors are the future for car perception systems.

### 2.1.1.2 Robotics

LiDAR is also one of the major sensory components for robotics. Various robots rely on it for autonomous navigation. Fig. 2.2 shows an example of two robots (ROSBot and Knightscope) that use this type of sensors for perception.

In one of the most popular works with robotic navigation called *"The Office marathon"* [9], the PR2 robot was able to navigate autonomously for 26.2 miles in an office like environment. During the task, the robot was able to avoid shelves, tables, chairs and people as well as go through narrow spaces. The sensor device used in this case was the Hokuyo UTM-30LX Scanning Laser Rangefinder and the navigation system developed is now known as the ROS navigation stack. However it was noted that the robot had difficulty detecting low height and long width obstacles.

In a more recent case study [10], two different systems are proposed for the implementation of an autonomous mobile robot, both using a 2D LiDAR sensor under ROS. Various trials in an indoor environment were conducted with both of them. It was concluded that both performed reasonably well but had some difficulty detecting objects with lower or higher height from



(a) ROSbot - Autonomous Robot with Laser scanner RPLiDAR A2 [7]      (b) Knightscope autonomous security robot with a Velodyne LiDAR Puck [8]

Figure 2.2: Example of two robots equipped with LiDAR

the sensor, objects that are transparent, or even dark obstacles with high absorption of light waves.

## 2.1.2 Operating Principle

The way LiDAR technology works is straightforward: it emits a laser beam and it waits for it to bounce back. Based on the propriety of the reflected signal it can determine the distance of the obstacle it hits. By constantly spinning the mirrors at different angles (scanning) it gets angle and depth information about the environment by a set of points or in other words a point cloud. There are two main different operating principles to do this, **Time of Flight** and **Phase Based**, which are described below.

### 2.1.2.1 Time of Flight

In this approach a pulse of light is transmitted and, when it is done, an internal clock is started. The reflected pulse is captured by a photodetector which triggers the clock to stop. Being  $\tau$  the time taken by the reflected signal to comeback and assuming it traveled at approximately the speed of light ( $c$ ) then the distance to the object  $d$  is given by:

$$d = \frac{\tau c}{2} \quad (2.1)$$

This method produces very accurate results for a long range but requires high precision clocks. However, greater range capability leads to slower update rates since it has to wait more time for the pulse to comeback. It is typically not used for robotics since this type of systems has very high cost ranging from a few thousand dollars to upwards hundreds of thousands of dollars [11].

### 2.1.2.2 Phase Based

A more affordable approach is based on modulating the intensity of the laser at a specific frequency. Figure 2.3 showcases the resulting sinusoidal wave that is sent and the respective

returned signal. Being  $\Phi$  the phase difference between the transmitted signal and the reflective

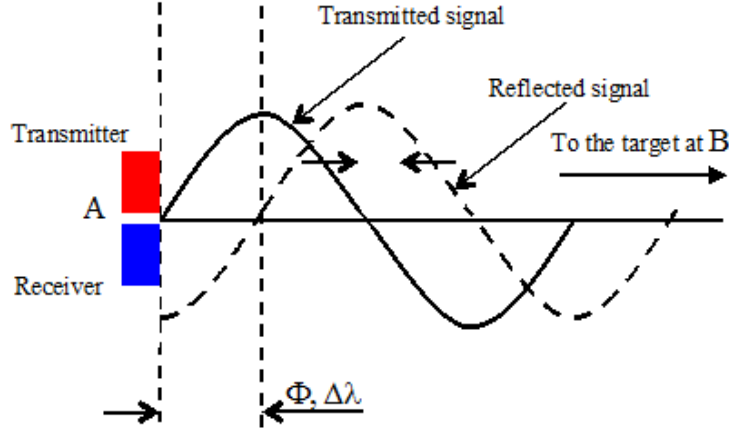


Figure 2.3: Phase based method for measuring range [12]

signal, and  $\lambda_m$  the wavelength of the sinusoid then the distance of the object is given by

$$d = \frac{\Phi \lambda_m}{4\pi} \quad (2.2)$$

This means that the maximum unambiguous distance that can be measured is  $\frac{\lambda_m}{2}$ , and its range resolution depends upon the resolution of the phase difference measurement as well as the wavelength used. Almost all robots use this type of LiDAR since it is usually cheaper.

### 2.1.3 Limitations

LiDAR technology has some problems associated with it:

- Even the most cheap devices have relative high operation costs that limit their use for small applications.
- Limited maximum detection range compared to radar technology (radar can achieve more than 100 meters).
- The sensor's moving parts are fragile when compared to a radar board
- Direct exposure to sunlight may negatively impact the sensor maximum range and accuracy or even prevent it from functioning properly.
- If it is a 2D-LiDAR it only senses in a horizontal plan

The last item on the list proves to be a significant problem because the sensor cannot detect objects that are above or below the height of the sensor (Fig. 2.4). The inability of detecting this type of objects may be crucial for the success rate of the robot being able of doing a certain navigation task safely.

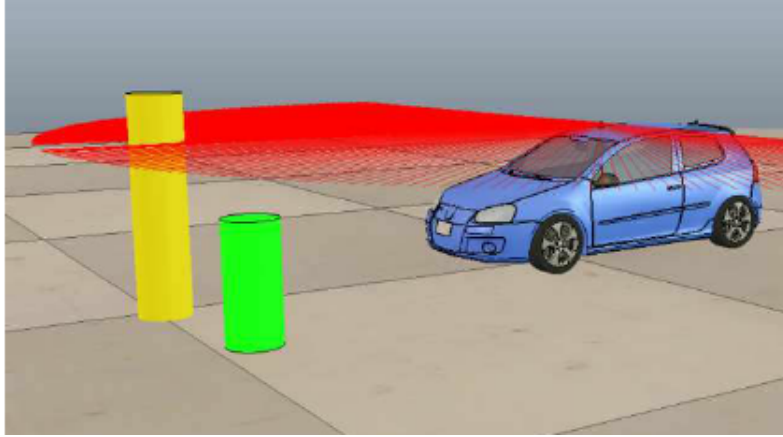


Figure 2.4: 2D LiDAR only detects objects in one plane (this plane is usually parallel to the ground (taken from [13])

---

## 2.2 MIMO Linear FMCW Radar

The use of radar technology has been around since the 19th century when it was first discovered that radio waves reflected off metallic surfaces, but serious development of this type systems only truly began in the 1930's for military applications [14]. This include the range and angle detection of ship and aircraft engines by measuring incoming signal fluctuations off an oscilloscope.

Nowadays there are various configuration types of radar systems [15], each one providing different types of applications. This usually include air traffic control, remote sensing, ground traffic control, space, etc. However we will focus on a specific configuration called Multiple Input Multiple Output (MIMO) linear (sawtooth) FMCW radars which are able to measure distance, velocity and also angular information.

### 2.2.1 Applications

#### 2.2.1.1 Pedestrian detection

In [16] three systems are implemented, the first one using a single radar measurement, the second using multiple, and the final one with additional tracking algorithms. Each system had the objective of distinguishing between people walking, vehicles and other objects. In the end it was concluded that that FMCW radar can be used for classification of inroad objects with relatively good accuracy in all systems.

Another case study is presented in [17] where by pre-processing the raw radar data and then using data processing pipeline, the system proposed was able to detect and track people in indoor environments using a 77 GHz MIMO FMCW radar.

### 2.2.1.2 Localization of Land Vehicles

Vehicle localization is often done using Global Navigation Satellite System (GNSS) accompanied by Reduced Inertial Sensor System (RISS). But is often the case in which GNSS signal can not be received correctly. This is often due the land vehicle being in an indoor environment or high rise building blocking the signal. In [18] it is proposed a radar based RISS that not only seeks odometer information but also radar readings. This proved to make the localization system of the vehicle more robust.

### 2.2.1.3 Monitoring of human vital signs

Monitoring vital signs of a patient such as the breathing rate or hearth rate may be critical in some situations. In [19] a system for detecting this type of signals using FMCW radar is proposed. By doing a phase analysis of the radar signal it was concluded that the proposed system measurement was in tune with a reference ground truth signal. It was also concluded that by increasing the Signal Noise Ratio (SNR) of radar it could estimate the heartbeat (electrocardiogram (ECG)) waveform of multiple targets.

## 2.2.2 Operating principle

The MIMO FMCW radar can retrieve range, velocity and angle of multiple targets. The following sections will explain how to calculate each one of these components.

An FMCW radar transmits a signal called a “chirp”. A chirp is an electromagnetic sinusoid whose frequency increases linearly with time. A chirp is characterized by a start frequency ( $f_c$ ), bandwidth ( $B$ ) and duration ( $T_c$ ). The slope ( $S$ ) of the chirp is the rate at which the frequency of the chirp increases and is given by:

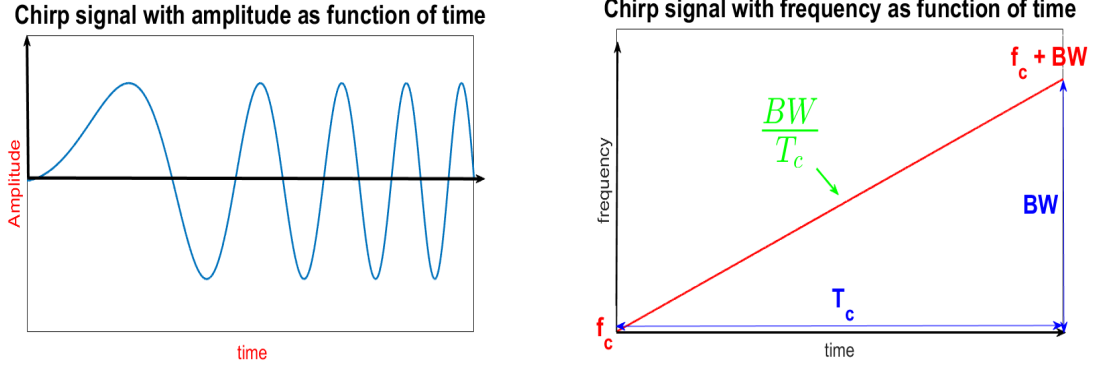
$$S = \frac{B}{T_c} \quad (2.3)$$

Figure 2.5a shows the plot of the amplitude in function of time for the described chirp. It shows that the frequency of the sinusoid is increasing with time. Another way of illustrating the chirp is by its frequency plot (Fig. 2.5b). The latter version is often preferred because it is simpler and more intuitive.

Object detection follows these steps:

1. The chirp is transmitted by the TX antenna
2. The chirp is reflected off an object and the reflected chirp is received at the RX antenna
3. The RX signal and TX signal are ‘mixed’ and the resulting signal is called an ‘Intermediate Frequency (IF) signal’

Figure 2.6 shows how the IF signal is generated through both the RX and TX signals for one target detection.



(a) Representation of the chirp, with amplitude as function of time (b) Representation of the chirp, with frequency as function of time

Figure 2.5: Representation of a chirp

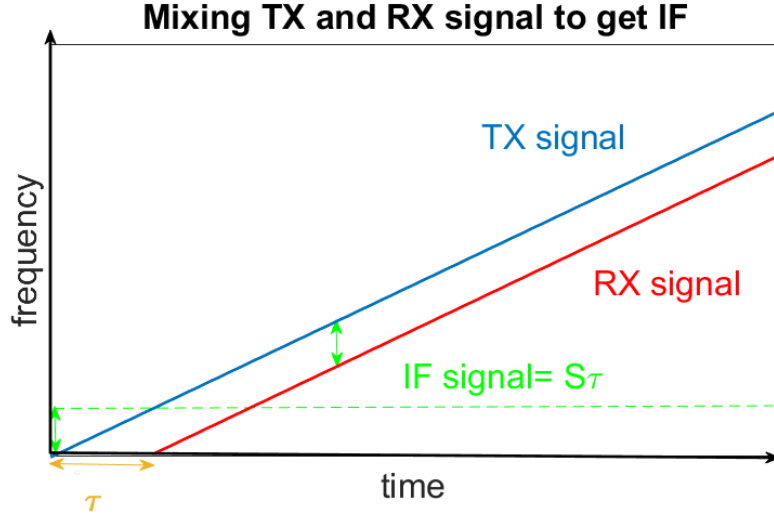


Figure 2.6: IF signal creation to retrieve range information of the target

### 2.2.2.1 Range Detection

The RX signal is a sum of delayed versions of the TX signal. This means that the resulting IF signal will be a combination of sinusoids, each one corresponding to the reflection on an object. For object  $i$  the corresponding frequency  $f_i$  is given by:

$$f_i = S\tau_i \quad (2.4)$$

where  $\tau_i$  is the round trip delay of the wave and is given by:

$$\tau_i = \frac{2d_i}{c} \quad (2.5)$$

where  $d_i$  corresponds to the distance of the object to the radar and  $c$  is the speed of light. Putting this together the distance to the object  $i$  is given by:

$$d_i = \frac{f_i c}{2S} \quad (2.6)$$

To determine the range of all objects detected, we need to find the according frequency components in the IF signal. To do this the analog signal is first sent to an Analog Digital Converter (ADC) and with the resulting digital output the Fast Fourier transform (FFT) is computed by a Digital Signal Processor (DSP). Each peak in the FFT that meets a certain SNR threshold corresponds to a different range detection.

However since the ADC has a limited sampling rate, the valid range of frequencies in the FFT is also limited, this meaning there is a maximum amount of ranges the radar can detect. The range resolution is also finite since the FFT has a limited amount of samples. In short, the higher the bandwidth of the chirp the better resolution we get.

#### 2.2.2.2 Velocity Detection

This type of radar also provides an estimate on the radial velocity for each detected range. To do this more than one chirp is needed. A set of  $N$  chirps is called a frame. The radar constantly sends these frames, each one corresponding to an observation. Since the time between frames is low the corresponding range (or first) FFTs will have identically located peaks (the same obstacle is in the same range for all chirps in a frame). However this set of peaks have different phases between each other. By calculating a second FFT (Doppler-FFT) of this vector of phases we can extrapolate a set of relative radial velocities for each range detection. Radial velocity information is given by:

$$v_i = \frac{\lambda \omega_i}{4\pi T_c} \quad (2.7)$$

where  $\lambda$  is the wavelength of the chirp,  $\omega_i$  is the phase difference between chirps for the respective objects in the Doppler FFT and  $T_c$  is the time required to send a single chirp. In the end we get a matrix that relates radial velocity and range. Figure 2.7 illustrates this process.

#### 2.2.2.3 Angle Detection

Finally, to determine the angle for each target more than one receiver antenna is needed. This is where the MIMO configuration comes in. Figure 2.8 shows that for the same target the distances to the two antennas is different. This difference makes it so for the same peak in the range FFT and doppler FFT the phase of the received IF signal is different in the two antennas. If we know this phase difference we can retrieve angle of arrival of the target. The phase difference  $\Delta\phi$  is given by:

$$\Delta\phi = \frac{2\pi\Delta d}{\lambda} \quad (2.8)$$

By basic geometry we know that  $\Delta d = L\sin(\theta)$ , where  $L$  is the distance between the antennas and  $\theta$  is the angle of arrival.

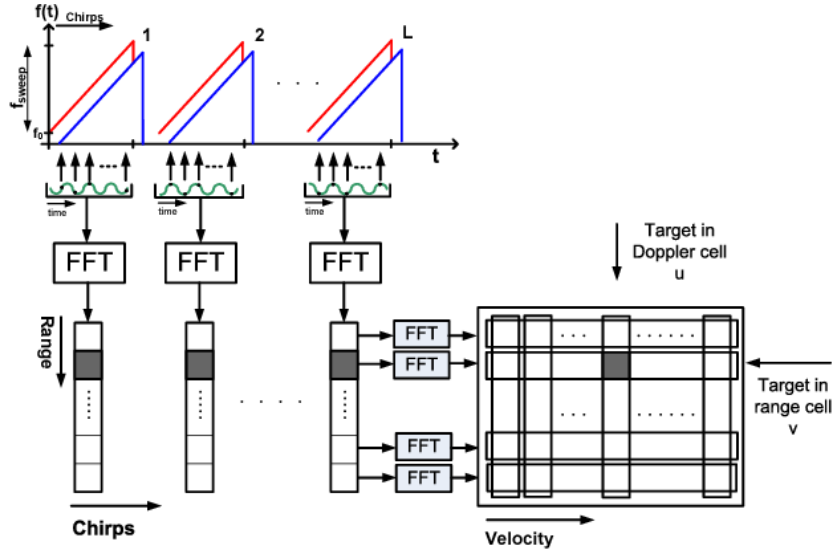


Figure 2.7: At the same range peaks the sinusoid has different phases, by calculating the 2D FFT (or doppler FFT) the radial velocity information can be retrieved (taken form [20])

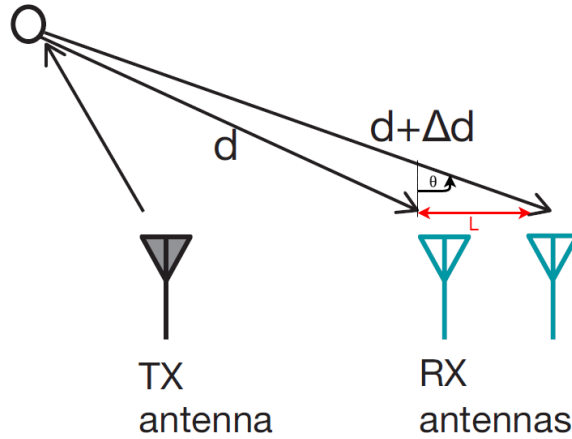


Figure 2.8: More than one antenna is required to estimate the angle of arrival (adapted form [21])

$$\Delta\phi = \frac{2\pi L \sin(\theta)}{\lambda} \quad (2.9)$$

Then the angle of arrival is given by

$$\theta = \sin^{-1}\left(\frac{\lambda \Delta\phi}{2\pi L}\right) \quad (2.10)$$

The above equation makes it possible to estimate the angle of arrival. But since  $\phi$  has a non linear dependency of  $\sin(\theta)$  then the angle of arrival accuracy depends on the angle of arrival. The closer it is to  $0^\circ$  the more accurate is the result.

### 2.2.3 Limitations

FMCW radar technology has some problems associated with it, some of them being:

- Limited range and angular resolution when compared to LiDAR which make difficulty to distinguish between two close targets
- Low pointcloud density
- Radar cannot provide the exact geometry of an object because of the longer wavelength
- Can have intermodulation products that result in ghost objects

## 2.3 Ultrasonic Sensors

One of the various ways of sensing the range of an obstacle, is by using ultrasonic sensors. Using sound waves, an ultrasonic sensor is able to measure range. These type of sensors have high reliability and outstanding versatility and can be used for wide variety of applications.

### 2.3.1 Applications

#### 2.3.1.1 Robotics

One examples of robotic applications is the work done by Byoung-hoon Kim [22] where by using a multi modulation processing of the ultrasonic sensors the proposed system was able to estimate the 3D coordinates of a moving object while being robust for noisy environments.

#### 2.3.1.2 Ground Vehicles

Ultrasonic sensors is also widely used for ground vehicles. On the work proposed by Enas Odat [23], a traffic sensing system is constructed using Passive infrared (PIR) sensors and ultrasonic range finders. It was able to calculate vehicle detection, speed estimation, and vehicle type classification. However the variability of the sensors readings, thermal noise and environmental conditions may produce unwanted reading errors.

#### 2.3.1.3 Medicine

Ultrasonic sensors are also used in medicine. For example in the work done by Ibrahim AlMohimeed et al [24], a wearable and flexible ultrasonic sensor was developed for monitoring of skeletal muscle contraction. The tissue thickness variations were successfully measured in accordance with the muscle contraction performed.

### 2.3.2 Operating Principle

To retrieve range information about the environment the ultrasonic sensors work similarly as the Time of flight LiDAR. But instead of light waves the sensor uses sound waves. A transducer sends and receives ultrasonic pulses that relay back information about an object's proximity.

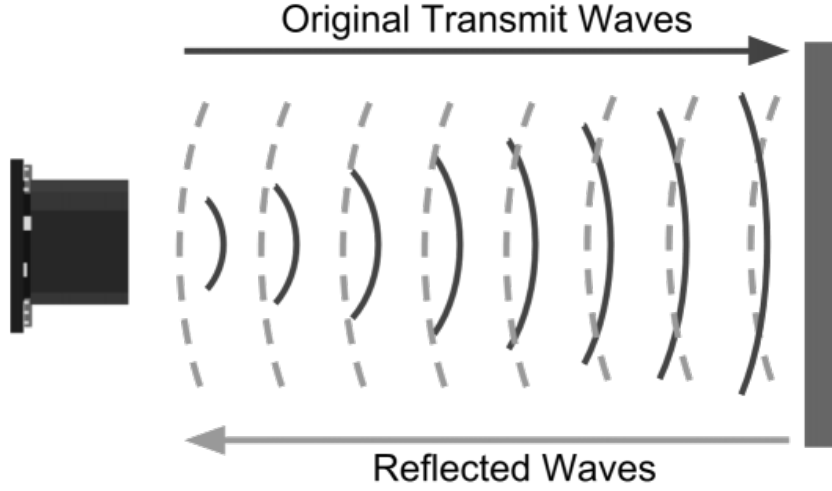


Figure 2.9: Ultrasonic wave operating principle (taken from [25])

## 2.4 Cameras

Besides the previous highlighted sensors, cameras can be used to determine the distance of obstacles. There are three image based object distance measurement techniques, (1) using two cameras i.e stereovision, (2) using a single camera and (3) time-of-flight camera [26]. The first method is highly accurate but is costly when compared to the monovision and it is not accurate for high distances. In the time of flight technique the distance is obtained using the time it took for the light to be reflected back. This method is a simple way to retrieve range from obstacles and it is also very fast that can reach up to 160 frames per second. It is also efficient having lower computation needs when compared to stereovision. However this technique comes with the problems of background lighting, interference from other time of flight cameras and multiple reflections.

## 2.5 Summary

In this chapter we presented various types of proximity sensor technologies. First, for the FMCW radar and LiDAR, we discussed on what contexts this sensors are used for and what current work is being done with them. Then we explain the operating principle of each one of them to retrieve information from the environment as well as simulating the range and velocity estimation provided by the radar. We also showed which limitations each sensor has. Finally we also provide an overview about ultrasonic sensors and cameras. In appendix A we provide a simulation code on how to estimate range and velocity using the operating principles of FMCW radar. This simulation serves as an additional contribution for better understanding of how these principles work.



## Chapter 3

# Robot Navigation using ROS

When it comes to navigation we need a wide variety of modules that interconnected produce an intelligent system that makes the robot go from one point to another while avoiding obstacles. The first problem that needs to be taken into account is where the robot is and where it is trying to go, for that we require a localization system and a map for a global reference. The second issue is finding the best route to a predefined goal. For that various algorithms and systems are already proposed and ready for use. ROS [27] is the most indicative place for the use or construction of software for our robot.

### 3.1 Robot Operating System

Creating software for robotic applications is not an easy task to do from scratch. It usually involves very complex code to achieve even the simplest applications due to the wide variety of hardware and data that robots rely on. ROS fixes this issue by being a general purpose framework for robotics. Despite its name, it is not an operating system, being more a kind of middleware, since it handles communication between programs in a distributed system. You can either construct a program that does all the computation needed in your application or you can have sub programs with each one having a specific functionality, the latter being often preferred.

ROS provides hardware abstraction, tools for introspection, message-passing and more. Also it is open source which means we can use them for virtually any means which we deem necessary for the development of our application, for example adapting pre established code for a specific case. This greatly facilitates the entrance of new developers to learn and do research in the field of robotics.

#### 3.1.1 ROS architecture

The ROS architecture is based on a peer to peer network between processes usually referred to as *Computation Graph* [28]. This architecture is comprised by different concepts that we define below:

- **Nodes** - A node can be defined as a process or piece of software that performs some

type of computation. In ROS, for a typical application there are various nodes running in parallel that pass information between each other. For example, controlling the robot movement given the input image of a camera, can be done by a node acting as a driver between the camera firmware and the ROS environment, a node that processes the image using ROS pre established software and finally a node that uses the processed information to calculate the velocity of the robot and transmit it to the locomotion system of the robot.

- **Messages** - ROS uses defined data structures called messages to represent information. This makes it so ROS tools can generate the appropriate source code in the selected language (C++ or python in this case). Each message can be broken down into more messages and so one and so forth until we arrive at the primitive data types of the given programming language.
- **Topics** - Nodes can send messages by conceptually publishing to a topic or receive them by subscribing to a topic. For example a node can be subscribed a velocity message and then after some data processing publish a smoothed version of it into another topic.
- **Master** - Provides registration and naming for each node (helps nodes find each other). It is also responsible for organizing communication between nodes. Finally it also provides the Parameter service (described bellow).
- **rosout** - This can be viewed as the ROS equivalent of stdout. This acts as a log reporting mechanism which is useful when debugging large amounts of code.
- **Parameter Server** - This server ROS tracks different parameter values defined by the running application. It is useful for setting or getting different parameters dynamically without having to restart the application. Parameter values can also be uploaded using roslaunch files.

Figure 3.1 gives a brief overview of how each of these concepts are put together and generate the ROS computation system.

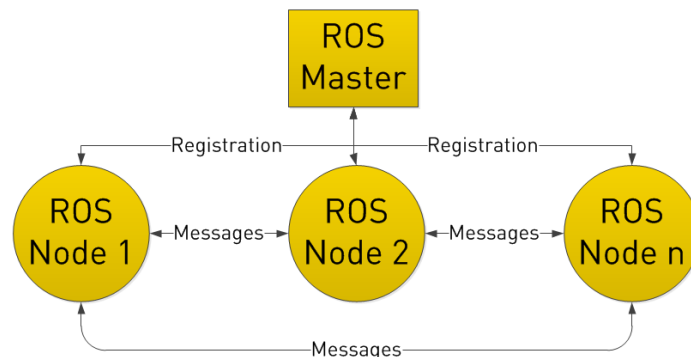


Figure 3.1: ROS communication architecture overview adapted from [29]

### 3.1.2 ROS tools

Now that we described the inner workings of ROS we can use its tools to retrieve and analyze information regarding it. ROS provides a wide range of tools for development and debugging. We will use the following ROS tools:

- **rviz** - A graphical interface that enables a 3D visualization environment that allows the view of the different components of ROS like the robots frame transforms, the map, pointclouds, visualization markers etc...
- **roscap** - Roscap will subscribe to one or more topics and the serialized message data will be stored in a file as it is received. These files are called "bags" and contain information about all the ROS topics chosen and can be played back at a predefined rate and can be started in any moment we want.
- **rqt\_graph** - To visualize how the various nodes and topics are interconnected the use of rqt\_graph is the best for this case. It can be used to produce a graph of all of them and how they communicate between each other
- **roscore** - This launches a process that the ROS System needs in order to properly setup communication between nodes. It starts the ROS Master, Parameter Server and roscout. Using **roslaunch** will automatically run a roscore if it was not initiated before.
- **roslaunch** - Used for running multiple ROS nodes as well as setting parameters in the server. This is done by launching ".launch" files that tell which nodes to load, which parameters to set and what machines it will be used on.
- **roslaunch** - Allows you to run a specific executable in a predetermined package.
- **rostopic** - Is used for displaying debug information and interact with a given topic. This may be the messages published by a topic, finding a topic by its type, showing information about a topic, listing all the topics in the ROS system. among other things.
- **rostopic** - Can be used to display debug information about a node, regarding publications, subscriptions and connections.
- **rostopic** - Used for manage parameters in the Parameter Server.

## 3.2 Navigation Concepts

There are four main problems associated with robotic autonomous navigation. They are Mapping, Localization, Path Planning and Motion Control [30], as shown in Fig. 3.2 .

Various algorithms have been developed over the years to address these problems.

In regards to creating a suitable map, The most popular approach available in ROS is an improved version of Rao-Blackwellized particle filters such as the ones described in [31]. This approach has proven to be an effective way to solve the Simultaneous localization and

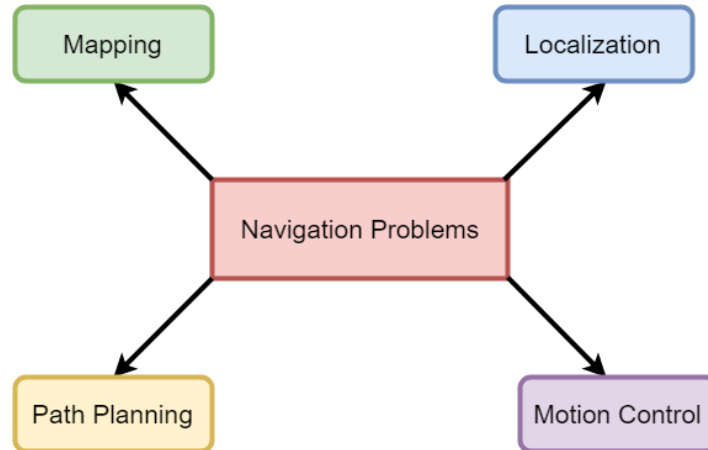


Figure 3.2: Problems regarding autonomous navigation

mapping (SLAM) problem and will be used later on in this work to produce a valid map of our indoor environment.

With the grid map built we now need to estimate the robot's position in said map. An easy solution to this problem is relying on the robot's odometry information inferred by the robot's encoders and inertial sensors such as accelerometers and gyroscopes. This is called dead reckoning and is an easy and low cost solution for the localization problem. However since the sensor data is integrated over time, this leads to the accumulation of errors which make this approach not feasible for long navigation tasks. To fix this issue various algorithms were developed being the most popular ones based on particle filters. The Adaptive Monte Carlo Localization (AMCL) [32] algorithm is the standard choice in this case. It takes into account a group of particles, each one corresponding to a certain robot state (position and orientation in this case). As the robot moves the least probable states are filtered out and the particles should over time converge to the actual position of the robot.

Assuming the robot can localize itself on the map with a reasonable error we can start sending navigation goals to the robot. To reach the goal the robot must be able to find a path that optimizes the travel cost while avoiding obstacles. The outputted plan can vary depending on the algorithm used. This type of planning is often referred to as a global path planning that will be discussed further in a section 3.3 .

After an optimal plan is computed the final step is to determine the best velocity command that will be sent to the locomotion system. The ROS navigation system follows a similar approach to the one used in [33]. A set of velocities are simulated during a given set of time and the corresponding predicted trajectories are computed. This subject is often referred to as local path planner methods that will also be further detailed in the following section.

### 3.3 Path Planning and Motion Control

Obstacle avoidance is a major subject in robotics, as failure in this systems may result in crashes that have catastrophic consequences such as hardware being badly damaged or being

completely nonfunctional. This subject is usually separated into two categories local path planning and global path planning [34].

Global methods assume the environment is completely known a priori and can therefore compute an optimal safe path around known obstacles using a variety of different strategies. These models prove to be very reliable for static environments. However these models are usually computational expensive which lead to very slow update times. For fast changing environments such as is in the case for highly populated indoor scenarios they prove to be insufficient. To improve the quality of navigation capabilities, local path planning techniques are added to increase the responsiveness of the robot. These techniques have a much faster update time since they use a smaller version of the world around them. However this type of approaches have trouble with local minimum cases where the robot can get stuck (for example the U-shape case).

### 3.3.1 Global Methods for Path Planning

Path planning is crucial for the robot to handle safe trajectory around obstacles. In this work the information regarding this obstacles is based on a grid map built by SLAM and new observations retrieved by the robot's sensors. Planning an optimal safe path can be done using a wide variety of methods like Visibility Graphs, Generalized Voronoi Diagram, and Probabilistic Roadmaps (PRM) [35]. However in this work we will use the PRM method. This method samples its environment and creates an occupancy graph. Then by inserting the initial position and the goal position the path can be computed using dijkstra, A\* or Rapidly exploring Random Trees search algorithms. However, the ROS navigation packages only use the first two so we will only focus on these ones.

#### 3.3.1.1 Dijkstra algorithm

Assuming we have an occupancy map we can now start to explain how does a robot compute the shortest path between the starting position and the end goal. Edsgar W. Dijkstra proposed an algorithm that solves this problem by computing the shortest cost path for a given graph. In our case we have cells on an occupancy grid map. The shortest path can be calculated executing the algorithm described in Figure 3.3. Each cell is numbered with the number of steps it needs to get to the starting position. This continues until we reach the goal cell and we get the minimum path to reach it. However if we have a map with cells that have variable cost in the algorithm above we do not iterate  $n$  with plus one, but with the cell cost. This makes it so for this case that shortest path geometrically may not be the shortest path when the costs are taken into account.

#### 3.3.1.2 A\* algorithm

In the previous case we search for cells in all directions, however there is a more efficient way of doing it by adding more information. The A\* algorithm besides the cost from the steps to the start position  $g(x, y)$ , it takes into account an extra *heuristic function*  $h(x, y)$  that gives

---

**Algorithm 10.1: Dijkstra's algorithm on a grid map**

---

```
integer n ← 0 // Distance from start
cell array grid ← all unmarked // Grid map
cell list path ← empty // Shortest path
cell current // Current cell in path
cell c // Index over cells
cell S ← ... // Source cell
cell G ← ... // Goal cell
```

---

```
1: mark S with n
2: while G is unmarked
3:   n ← n + 1
4:   for each unmarked cell c in grid
5:     next to a marked cell
6:     mark c with n
7:   current ← G
8:   append current to path
9:   while S not in path
10:    append lowest marked neighbor c
11:    of current to path
12:    current ← c
```

Figure 3.3: Dijkstra algorithm taken from [36]

---

a preferred direction for the search. The cost function in this case can be shown as:

$$f(x, y) = g(x, y) + h(x, y) \quad (3.1)$$

### 3.3.2 Local Methods for Motion Control

In contrast to global path planning where a large portion of the environment is generally assumed to be known, local methods take into account a partial world view for motion planning capabilities. The most popular of this type of methods are [37]:

**Dynamic Window Approach (DWA)** - [34] Takes into account a subset of admissible velocities and the robot's dynamic constraints and simulates them for a given time frame outputting various trajectories. Then using these trajectories derives the best trajectory given a certain cost function.

**Trajectory Rollout** [33] - Similar to the DWA planner except the sampling method for the control space is different.

**Elastic Band** - [38] In this case bubbles are formed that are defined as the maximum local subset of free space around a given robot configuration that can be travelled without collision. Given these bubbles a set of "elastic bands" are connected to form a collision free trajectory from the initial position to the goal without collision. When it comes to

avoid an obstacle that obstructs the global plan, this technique tends to deviate from it minimizing the bubble band tension.

**Timed elastic band** - [39] This planner is an extension of the "elastic band" that takes into account time intervals in between the robot's configurations (position and orientation).

By utilizing a multiple objective optimization function, that takes into account the robot's acceleration and velocity limits as penalties and taking into account execution time, shortest path and clearance of obstacles, it determines an optimal trajectory. With this a suitable local planner that is appropriate for avoiding dynamic obstacles is created.

**Clothoid Tentacles** - [40] This is an empirical approach to the local path planning problem. It generates various "tentacles" and take the form of clothoids. It then chooses the best one taking into account the best obstacle clearance, curvature and distance to the global path planner.

**Vector Field Histogram (VFH)+** [38] - For this approach a histogram of the ammount of obstacles in a certain direction is first created. Then using a cost function that takes into account the alignment of the robot towards the goal, the difference between the new direction and the current wheel orientation and the difference between the previously selected direction and the new direction is computed. The best trajectory is then chosen taking into account this information.

For our case study we will only study in detail the first two since they are already implemented in ROS.

### 3.3.2.1 DWA planner

The DWA planner is the most standard approach when it comes to local path planning in ROS. It outputs rotational and translational velocities by generating multiple trajectories for different types of velocity sample search space and choosing the best one. The algorithm goes as follows [37]:

- Start with a set of velocities pairs (translational and rotational)  $\{(R_{vR_x}, R_{\omega_z})\}$  that are obtainable by the robot.
- Genarate in the form of arcs the projected trajectories obtained using the previous velocity sample space.
- Dismiss velocities that result in the robot colliding with an object in a given time frame. With this we are left of a subset of admissible velocities  $V_a = (R_{vR_a}, R_{\omega_a})$  in which:

$$V_a = (R_{vR_a}, R_{\omega_a}) \Rightarrow \begin{cases} R_{vR_x} & \leq \sqrt{2 * dist(R_{vR_x}, R_{\omega_z}) * R_{\dot{v}_{xb}}} \\ R_{\omega_z} & \leq \sqrt{2 * dist(R_{vR_x}, R_{\omega_z}) * R_{\dot{\omega}_{zb}}} \end{cases} \quad (3.2)$$

Where  $R_{\dot{v}_{xb}}$  and  $R_{\dot{\omega}_{zb}}$  are the braking accelerations of the robot and  $dist$  the distance to the closest obstacle found in the trajectory.

- Dismiss velocities that don't respect the robot's acceleration limits in a given simulated time frame. With this we are left with a subset of velocities called dynamic window  $V_d = (R_{vR_d}, R_{\omega_d})$  in which:

$$V_d = (R_{vR_d}, R_{\omega_d}) \Rightarrow \begin{cases} R_{vR_x} & \in [R_{v_a} - R_{v\dot{R}_x} * t, R_{v_a} + R_{v\dot{R}_x} * t]. \\ R_{\omega_z} & \in [R_{\omega_a} - R_{\omega\dot{z}} * t, R_{\omega_a} + R_{\omega\dot{z}} * t]. \end{cases} \quad (3.3)$$

- Finally we choose the most optimal trajectory and in consequence velocity pair taking into account a given objective cost function.

### 3.3.2.2 Trajectory Rollout

The Trajectory Rollout planner follows the same logic as above but in this case the sampling method for the control space is different. In this a set permissible velocities in each simulation is calculated including acceleration limits for the entire simulation time [37]. Instead of searching the space of feasible trajectories, we search the space of feasible controls. In Trajectory Rollout set of permissible velocities in each simulation is calculated including acceleration limits for the entire simulation time, while in Dynamic Window Approach it is limited to one simulation step.

## 3.4 ROS Navigation stack

The ROS navigation stack is a set of software packages that properly combined can make a robot navigate autonomously. Figure 3.4 shows an example of **turtlebot2** using the navigation stack to drive autonomously.

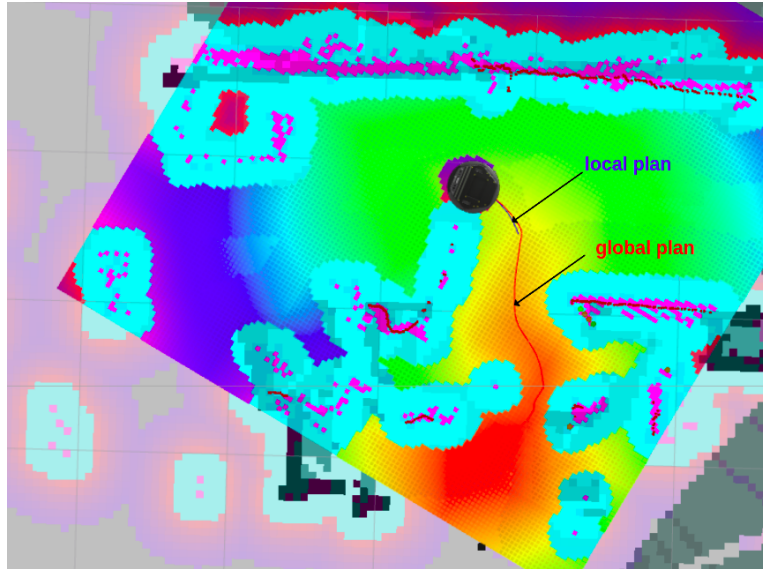


Figure 3.4: Rviz displaying the ros navigation stack components while **turtlebot2** is navigating.

### 3.4.1 Requirements

Before running the ROS navigation stack we first need to properly setup the robot to meet its requirements.

#### 3.4.1.1 Transform Configuration

The Navigation stack requires that the relationships between the different frames must be published in `tf` or `tf_static` topic. This makes it so the robot perceives what is around it correctly. Figure 3.5 shows all the frames and their transforms with each other for **turtlebot2**. However the main transforms that we need to worry about are between the sensors and the base link frame.

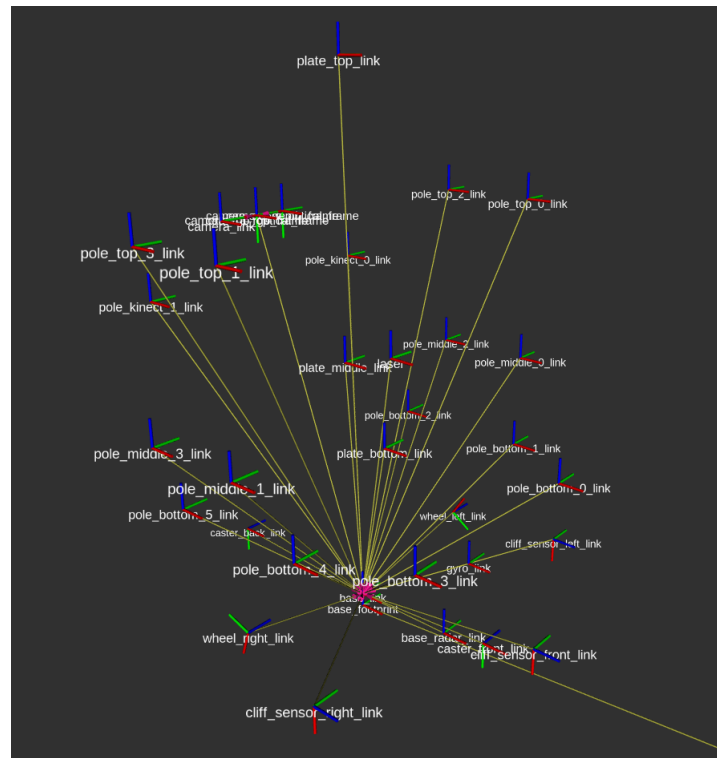


Figure 3.5: Example of the transform relationships

#### 3.4.1.2 Sensor sources

To avoid obstacles we need some type of sensors that can detect them. Before running the stack we need to make sure they are publishing information. This information needs to be in either a `PointCloud2` or `LaserScan` message format. Figure 3.6 shows an example of the published data from both in `rviz`.

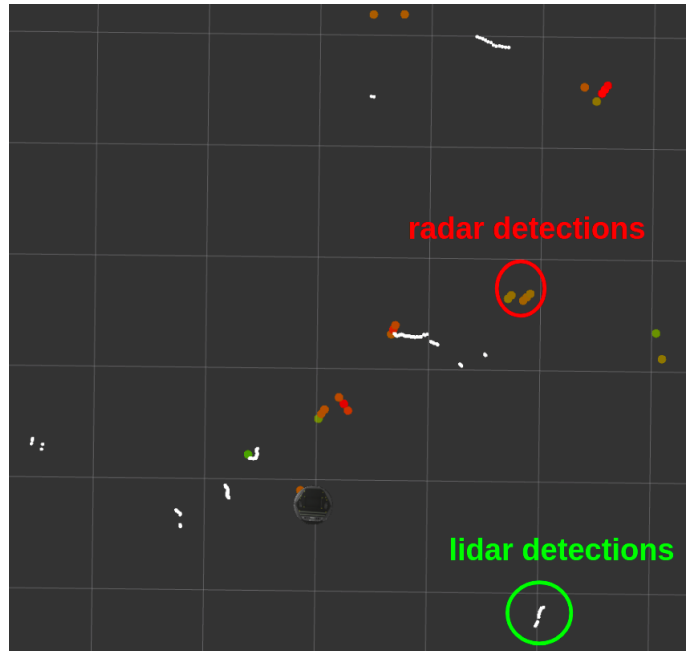


Figure 3.6: Obstacle detections by both radar and lidar

#### 3.4.1.3 Odometry

We need to have a rough estimation of the robot localization. Therefore we require a topic that publishes odometry information.

#### 3.4.1.4 Base Controller

This module will subscribe to the velocity message outputted by the navigation stack and convert them into the appropriate motor commands to send to the mobile base that will actually make the robot move.

#### 3.4.1.5 Map

This part is not mandatory but it helps to have some sort of map being published to use as a global reference for the robot. It is used by `amcl` node to correctly localize the robot and to mark previously detected static obstacles when the map was built. Fig. 3.7 shows an example of a map that might be used by the ROS navigation stack. If the setup is done correctly we can now run the navigation stack.

### 3.4.2 Navigation Node

Now that we have all the things we need for navigation we need an entity that actually processes all this information in an intelligent way to determine the best velocity command for the robot. This is done by the `move_base` node and its peripherals. Figure 3.8 shows an overview of the different components of the navigation stack [41].



Figure 3.7: Example of a map created in the IRIS laboratory

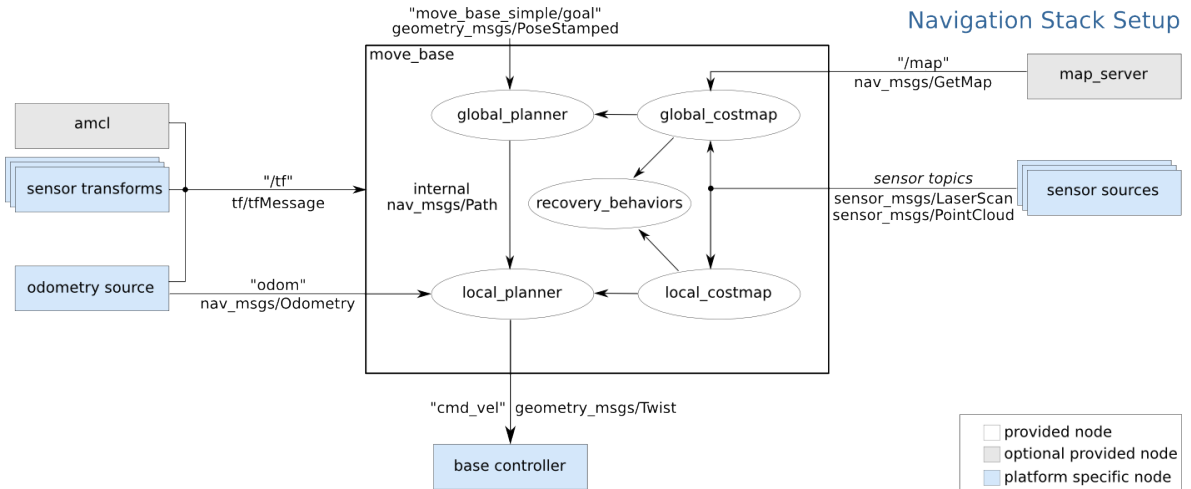


Figure 3.8: Navigation stack block diagram taken from [41]

The move base node links together the global and local planner as well as a local and global costmap to achieve the end goal provided by an action server. It also loads a set of determined recovery behaviors if the planners fail to produce a valid path.

The global and local planners are plugins specified by the user. This choice will affect the behavior of the robot depending on the planners architecture and parameters. It is integrated in ROS two global planners, the **navfn** and **global\_planner**, and three local planners the **TrajectoryPlannerROS**, **DWAPlanner** and **carrot\_planner**.

In this work it will be used **navfn** as our global planner and **TrajectoryPlannerROS** as our local planner.

### 3.4.2.1 Global Planner

The job of this component is to produce the best trajectory for a robot to take with the amount of information given by the global costmap. This plan can be updated when the robot gets stuck or by using a user specific frequency, the latter being usually preferred. The algorithms used to get this path are usually **Dijkstra** or **A\*** that are described in the previous sections 3.3.1.1 and 3.3.1.2.

### 3.4.2.2 Local Planner

The local planner takes into account the trajectory given by the global planner and tries to compute velocity commands that follows it. However, the given path may be too close to a detected obstacle and in order to avoid it the robot must deviate from the given plan to avoid collision. The function of the local planner is to avoid dynamic obstacles that appear while still trying to follow the global plan and goal. This type of local planner is usually **TrajectoryPlannerROS** or **DWAPlaner** that were explained in the previous sections 3.3.2.1 and 3.3.2.2 and uses the local costmap to do so. Below, we explain how **TrajectoryPlannerROS** generates the best trajectory using a determined cost function.

The algorithm to get the best trajectory goes as follows:

1. Discretely sample the velocity space ( $d_{vx}$  and  $d_{vtheta}$ )

$$\begin{aligned}d_{vx} &= (\text{max\_vel\_x} - \text{min\_vel\_x}) / \text{vx\_samples} \\ d_{vtheta} &= (\text{max\_vel\_theta} - \text{min\_vel\_theta}) / \text{vtheta\_samples}\end{aligned}$$

2. For each sampled velocity predict its trajectory in a given time frame (`sim_time`).
3. Evaluate the cost of each trajectory by using the value cost function
4. Pick the one with lowest cost and publish the associated velocity.
5. Repeat for a given rate (`controller_frequency`)

The cost function used to evaluate a trajectory is given by:

$$\begin{aligned}\text{cost} &= \text{pdist\_scale} * \text{path\_dist} + \text{gdist\_scale} * \text{goal\_dist} \\ &\quad + \text{occdist\_scale} * \text{max\_obs\_cost}\end{aligned}$$

where **path\_dist** is the distance from the endpoint of the trajectory to the global path in map cells, **goal\_dist** is the distance from the endpoint of the trajectory to the local goal (goal that is within the local costmap taking into account the global plan) in map cells, and **max\_obs\_cost** is equal to the maximum obstacle cost (given by the local costmap) of all the points along the trajectory. Fig. 3.9 displays an example of these values when the robot is navigating.

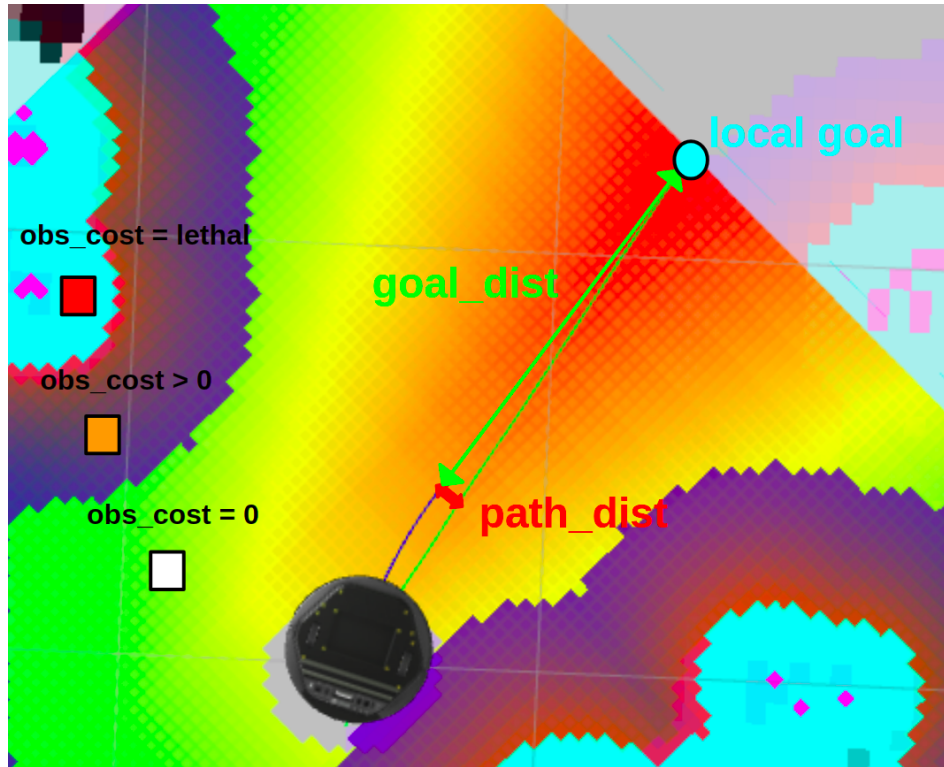


Figure 3.9: Cost cloud published in rviz. Red cells correspond to low cost and green cells correspond to a high cost

### 3.4.2.3 Local and Global Costmaps

The global and local costmaps share the same class, the `Costmap2DROS`. This class consists of a layered costmap that takes into account various layers defined by the user.

#### Available Layers

- **Static Layer** - Retrieves static information from the `/map` topic and marks them as lethal objects (Typically only used in global costmap).
- **Obstacle Layer** - Marks objects retrieved from our sensor sources with lethal value. It also raytraces observations to clear out space.
- **Inflation Layer** - Inflates the detected obstacles taking into account the robot radius and inflation radius. The closer the cells are from a lethal obstacle the more value they will have.

Figure 3.10 shows how the combined layers produce the master costmap that will be used by the planners [42].

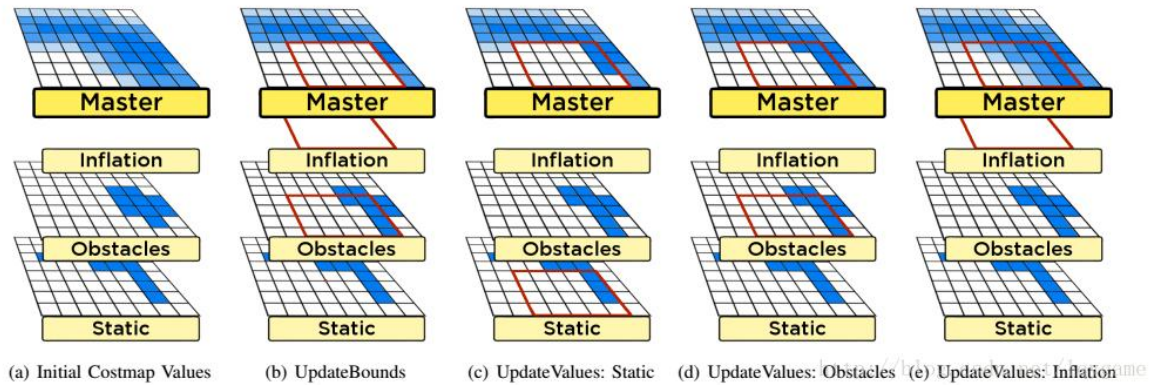


Figure 3.10: Update Algorithm - In (a), the layered costmap has three layers and the master costmap. The obstacles and static layers maintain their own copies of the grid, while the inflation layer does not. To update the costmap, the algorithm first calls the `updateBounds` method (b) on each layer, starting with the first layer in the ordered list, shown on the bottom. To determine the new bounds, the obstacles layer updates its own costmap with new sensor data. The result is a bounding box that contains all the areas that each layer needs to update. Next, each layer in turn updates the master costmap in the bounding box using the `updateValues` method, starting with the static layer (c), followed by the obstacles layer (d) and the inflation layer (e). from [42]

### 3.5 Summary

In this chapter we described the inner workings of the ROS framework and what tools can be used in order to develop a certain application. We also discussed what types of problems exist in regards to autonomous navigation. Finally we showed how the ROS navigation stack tackles these issues in order to construct a safe navigation module for a robot.

## Chapter 4

# Evaluation Platform

In this chapter we will describe the principal components in terms of hardware and software and how they are interconnected to create a suitable robotic platform that will later be used for evaluating the performance of LiDAR and FMCW radar as obstacle avoidance sensors.

### 4.1 Hardware

The basic hardware used in this work is a modified version of the turtlebot2 platform. The platform was modified in order to include a processing unit, a 2D scanning LiDAR and a FMCW radar. The modified version is displayed in Fig.4.1.

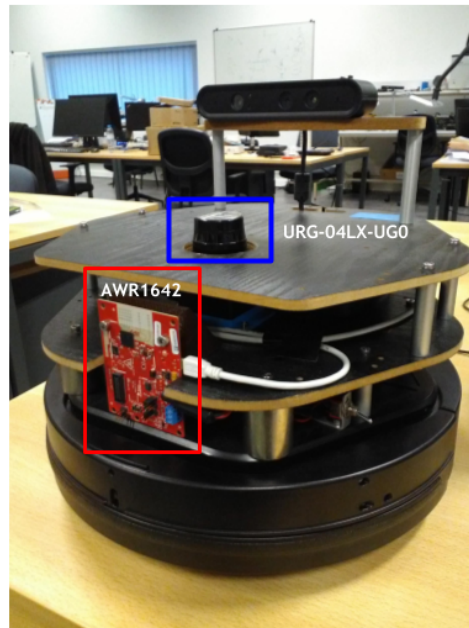


Figure 4.1: Modified Turtlebot2 used in this work

#### 4.1.1 Turtlebot2

TurtleBot 2 (Fig. 4.2) is one of the most popular low cost personal robots around. It is completely run by open source software which makes it exceptional for research and educational purposes. The robot has been developed by the Korean company Yujin Robotics in collaboration with Willow Garage. Its differential kinematics mobile platform can be used for multiple applications, due to the huge number of available ROS packages.



Figure 4.2: Turtlebot 2 platform

When it comes to technical specifications the robots dimension is 354 x 354 x 420 mm as shown in Fig. 4.3, its weight is 6.3 Kg with a max payload of 5 Kg which means it is able to attach lots of sensor devices with if needed. Its maximum translational speed is 0.7 m/s and the maximum rotational speed is  $180^\circ/\text{s}$ . It is equipped with a gyroscope with 1 axis( $110^\circ/\text{s}$ ), an odometer at 52 ticks/encoder and bumpers on left, right and center among other components.

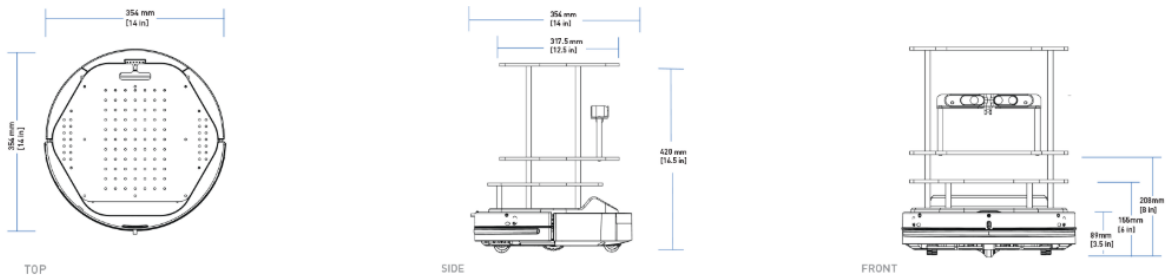


Figure 4.3: Turtlebot 2 dimension specifications

#### 4.1.2 FMCW radar

The radar board chosen for this work is the mmWave Texas Instruments (TI) AWR1642BOOST (Fig.4.4). This is a recently distributed FMCW radar appropriate for short range applications.

It is an easy-to-use evaluation board for the AWR1642 automotive radar sensor which is connected to the micro-controller unit (MCU) LaunchPad. To develop software it has on-chip C67x DSP core and low-power ARM Cortex-R4F controllers which include onboard emulation for programming and debugging. It requires a 5V @ 2.5A supply brick with a 2.1-mm barrel jack to run. The device supports a wide RF bandwidth of 77-81 GHz that permits good range, velocity and angle resolution. These last parameters depend on the configuration fed to the device.

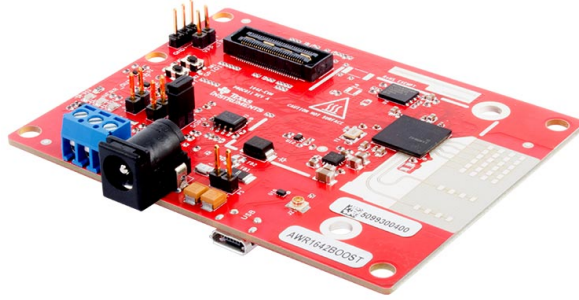


Figure 4.4: Texas Instruments AWR1642BOOST evaluation board

The radiation pattern of the antenna in the horizontal plane (H-plane  $\Phi = 0$  degrees) and elevation plane (E-plane  $\Phi = 90$  degrees) is shown by Figure 4.5.

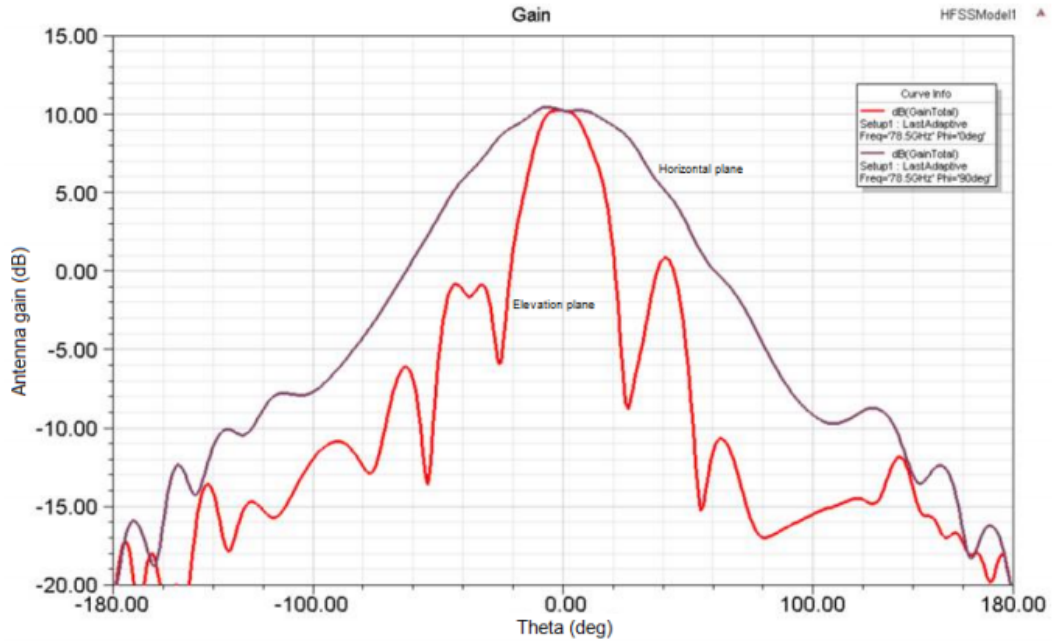


Figure 4.5: Radiation pattern of the antenna from [43]

### 4.1.3 LiDAR

Besides the FMCW radar the Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder (Fig. 4.6a) is also attached to the platform.

This sensor is an inexpensive 2D-LiDAR that is based on phase difference measurement. It retrieves information of the surrounding environment by scanning an area of  $240^\circ$  with  $0.36^\circ$  angular resolution. Its maximum range is about 4 meters and its range resolution is 1mm. Its scan update time is 100ms/scan and its weight is 360 g. As for the power supply it only needs a 5V DC provided by the USB connection as shown in Figure 4.6b.

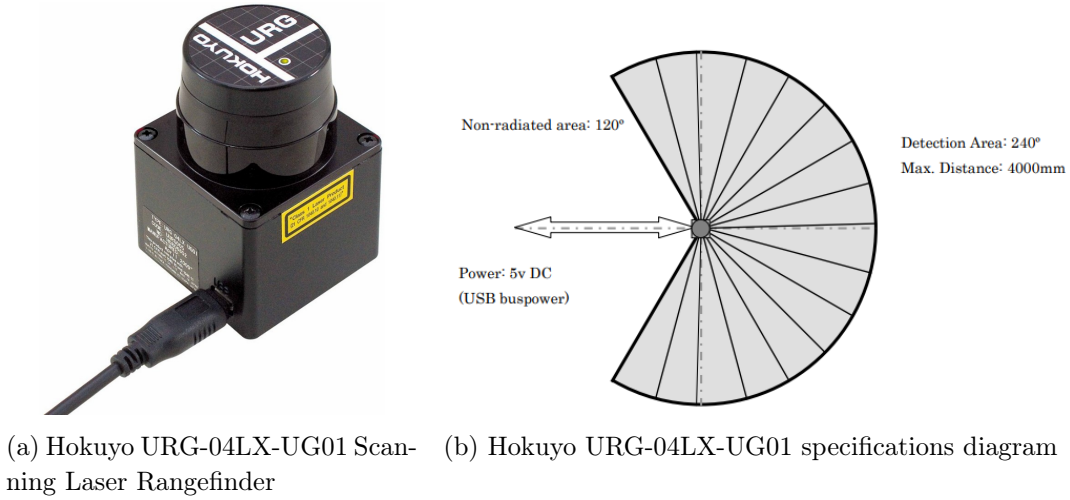


Figure 4.6: Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder overview

## 4.2 Software

Figure 4.7 shows the block diagram describing the different modules used to have a proper autonomous navigation platform.

First of, a map of the surrounding environment must first be created. There are multiple packages for doing this in ROS, but in our case we used the package **gmapping** with 2D-LiDAR as input. The robot also needs to be able to localize itself, for that we will use again the 2D-LiDAR for input in the package **amcl**. This will update the localization of the robot taking into account the odometry information and the observable environment. With localization and mapping problem taken care of we now must feed the Navigation Module with obstacle detectors. For that we use the 2D-LiDAR and the radar data as sensor sources. However the radar data is not compatible with the ROS navigation module. It sends Type Length Value (TLV) data that must be decoded and converted to a ROS message format. To do this conversion we have a block called ROS Interface. After this is done we encountered false positive obstacles detected by the radar so we added a new module that processes the radar message to combat this issue, we call it an Intensity Filter. Finally, with this processing done, we feed the Navigation Module with the processed radar data in ROS format or/and

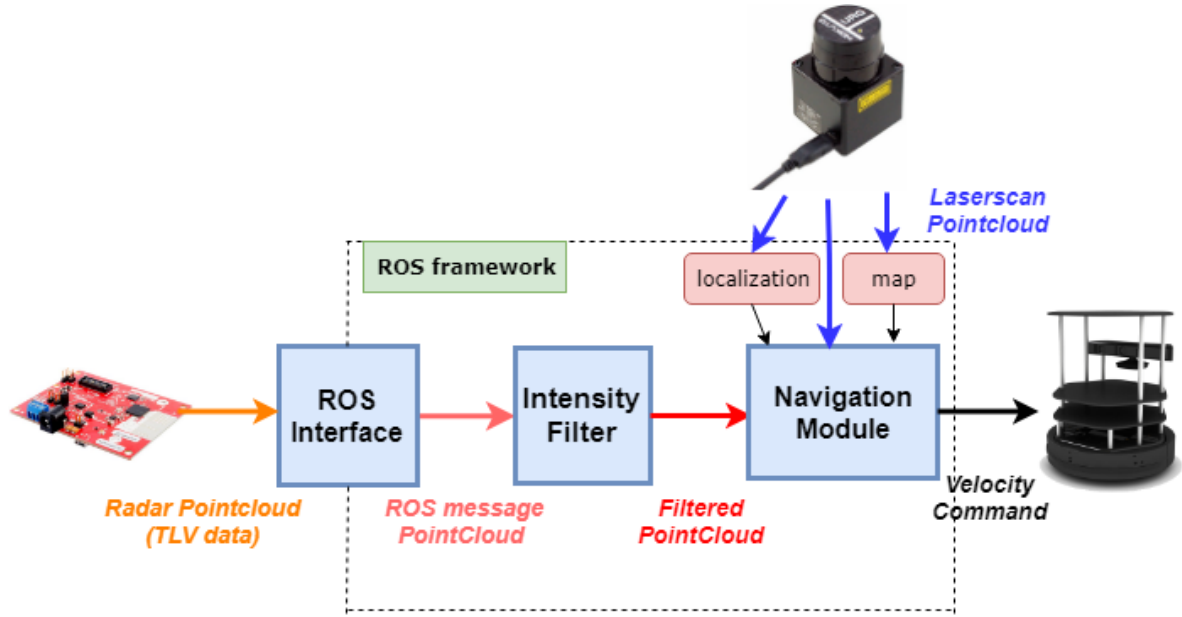


Figure 4.7: Block diagram of the software architecture designed for this work

the 2D-LiDAR as obstacle detectors.

#### 4.2.1 ROS Interface

In order to use the radar for obstacle detection we must first convert the TLV data of the robot to the ROS message format. To do this Texas Instruments provides a ROS package that interfaces radar data to the ROS framework [44]. The incoming radar TLV data from the radar is decoded in order to create a **PointCloud2** type ROS message. This point cloud follows the detected objects frame described in the mmWave demo data structure [45] represented in Fig. 4.8. Each point has 6 fields:

- **x (m)** - position x of the detected object in the frame of the radar.
- **y (m)** - position y.
- **z (m)** - position z (for 2D devices this is equal to zero).
- **range (m)** - range of the object relative to the radar frame.
- **doppler (m/s)** - radial velocity of the object relative to the radar frame.
- **intensity** - power of the received signal corresponding to that object.

The characteristics of the radar data, such as publishing rate, range resolution, maximum range, velocity, resolution and maximum velocity depends on the chirp profile configuration file loaded in the radar. The easiest way to create a chirp configuration file is using the mmWave Demo Visualizer. With it you can auto generate a configuration file given a set of specifications. Another way of doing this is manually. This however requires the understanding of the radar operating principle and the configuration commands.

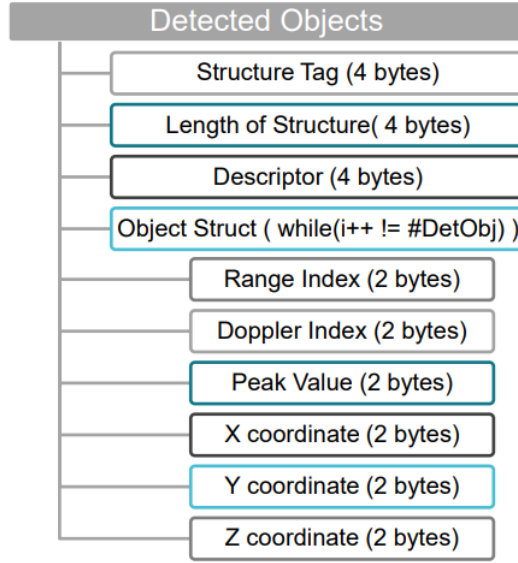
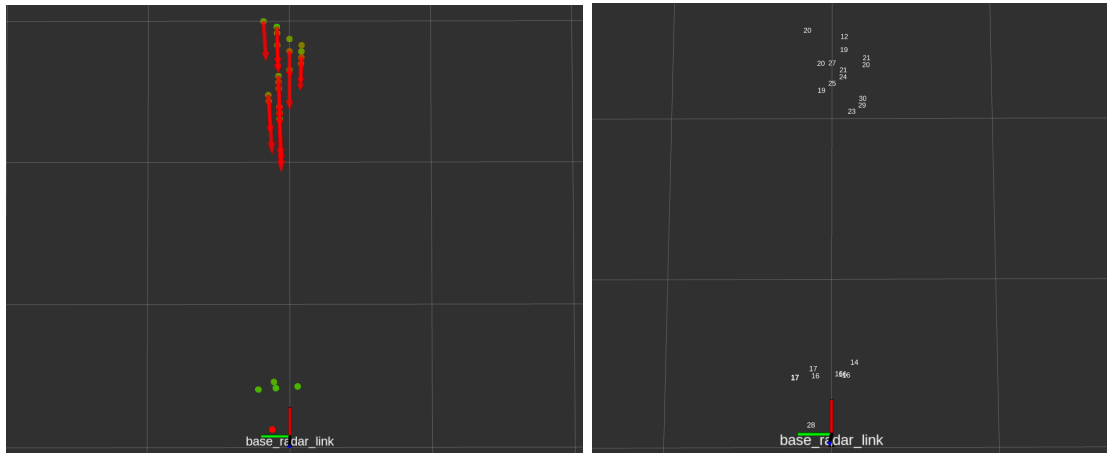


Figure 4.8: Part of the mmwave demo packet containing object detection information. This fields will be used to construct the ROS PointCloud2 [45].

#### 4.2.2 Visualization of the radar point cloud

Plotting the points in the XYZ space is not enough to fully visualize the radar data sent since each point also gives velocity and intensity information. We can better visualize it by using markers such as arrows or text in rviz. Figure 4.9a displays the radial velocity of each point with an arrow. The width of the arrow indicates how fast in the direction of the radar the object is going. Figure 4.9b shows the intensity values of each object in text. This type of visualization will be useful when we want to filter the cloud.



(a) Arrow markers displaying the points radial velocity (b) Text markers displaying the points intensity values

Figure 4.9: Visualization markers displaying radar data information

### 4.2.3 Intensity Filter

Since we are dealing with point clouds coming from the FMCW radar and the 2-DLiDAR than we need some type of ways to handle and manipulate them. For that the open source libraries called Point Cloud Library (PCL) is the more indicated place to process and manipulate this type of information. A pointcloud is a collection of multi-dimensional points and is commonly used to represent three-dimensional data [46]. These points are often just designed to locate points in x,y,z but more dimensions can be added as is for the FMCW radar which has 6 dimensions. PCL provides open source, state of the art library modules that enables filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation.

In the point cloud there may be some points that have undesirable characteristics, such as points with low intensity that lead to false detections or outside of the radar operating range. To remove these points we use **passthrough filters** that specify the range of values a given field can have in order for a point to be kept in the point cloud.

For example, if we are only interested in obstacles that are moving between 0.5 m/s and 1.0 m/s (radial velocity), this can be done by using a passthrough filter on the doppler channel. Figure 4.10 shows an example where we delete detections close to the radar by filtering the point cloud by intensity. After various tests it was decided that we will use an intensity filter of 16, which means all target points that have an intensity bellow that will be filtered out due to having a low SNR.

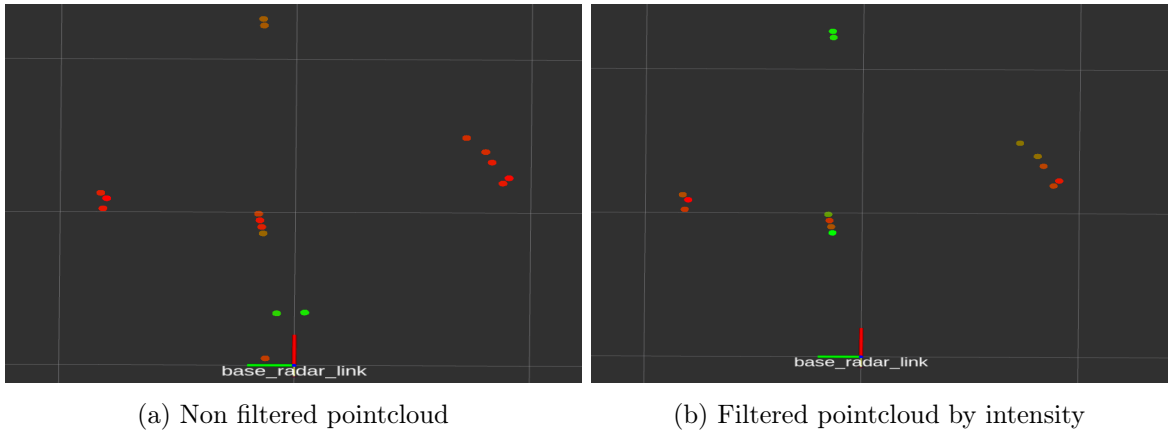


Figure 4.10: Example of filtering the pointcloud

### 4.2.4 Navigation Module

The navigation module first needs a map in order to have a global frame as reference. After that we need to localize the robot in said map, for that we use the AMCL node. Finally we need range sensors that detect obstacles. With the processed radar data we can now feed it to our navigation system. We can either feed the LiDAR or/and FMCW radar as obstacle detectors. The navigation module will then use all this information to compute the velocity command that will make the turtlebot2 move.

### 4.3 Summary

In this chapter we overviewed the technical specifications of each hardware component in the navigation platform that will be used on in this work. We also overview what software and how it is interconnected to properly setting up the turtlebot2 robot for indoor navigation.

## Chapter 5

# Results

With the configuration described in the previous chapter we can now start to experiment with the robotic platform to execute certain navigation tasks. In this chapter we present various experiments that were conducted to evaluate the performance of the FMCW radar. The experiments were divided into three groups. In the first, we try to evaluate the detection capabilities of objects that are poorly or not even detected by the LiDAR. In the second we try to evaluate the performance of the FMCW radar for dynamic obstacles in a controlled space. In the third experiment we try to evaluate how the robot handles dynamic obstacles in a an uncontrolled environment using the FMCW radar as an obstacle detector.

### 5.1 Static Obstacles in controlled environment

The 2D-LiDAR has trouble detecting objects that are above or below the plane where the sensor is. So low height obstacles are difficult to detect. Also obstacles that are highly reflective or highly absorbent are also not appropriate for detection using this sensor. However the FMCW radar may show better performance at detecting this type of obstacles. Taking this into consideration a test was devised in a controlled environment that compares the performance of each sensor for these types of obstacles. The objects evaluated were two types of chairs, a garbage bin, a low height box, a transparent acrylic tube and finally a robot (in this case another turtlebot2) that is certainly going to be detected by both. The list of objects used as obstacles are displayed in Fig. 5.1.

To ensure the experiment is done in a controlled way the scenario shown in Figure 5.2a was constructed. This is approximately a 4 by 4 meter area with 1 meter high walls with the addition of a half a meter wall in length in the middle. This environment optimizes the robots localization system (AMCL) as well as make sure we only concentrate with one specific object at a time. Using a SLAM package developed at IRIS [47] a map is first created (Figure 5.2b) that will later be used for localization purposes.

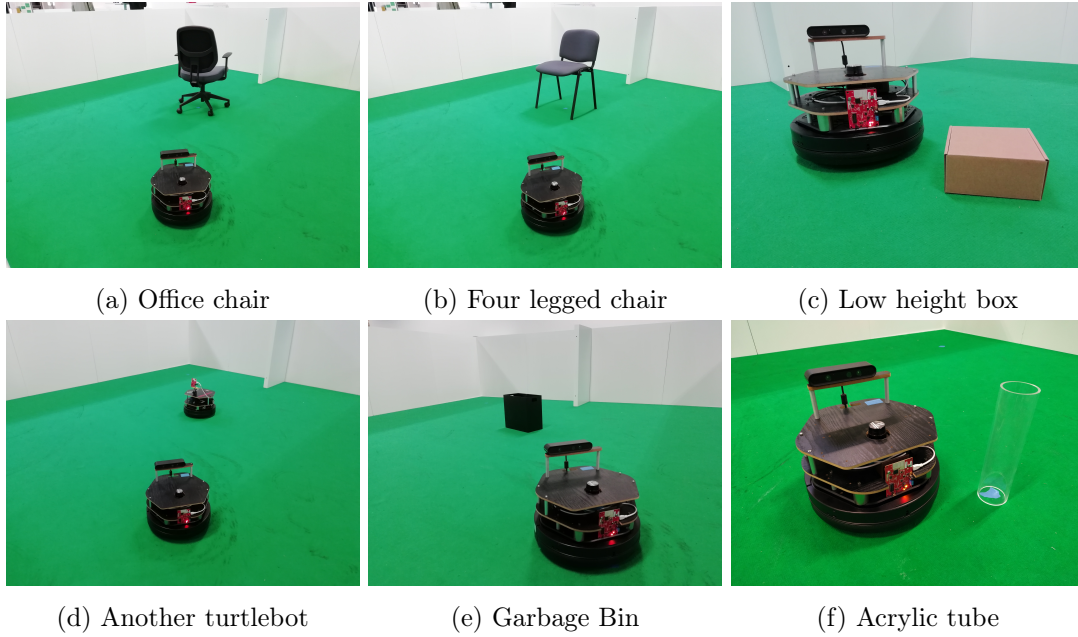


Figure 5.1: Different obstacles used for the experiment

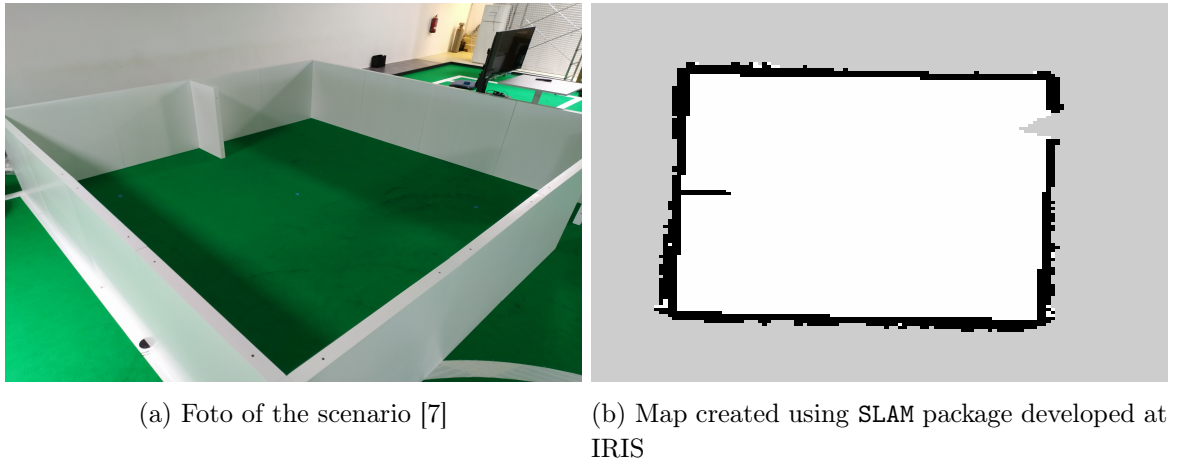


Figure 5.2: Scenario Constructed for the experiment

### 5.1.1 Experimental setup

With the described scenario we setup the robots path to make five loops between two goals, positions A and B, positioning in between an obstacle as illustrated in Figure 5.3. If the obstacle detection system fails then the robot should collide with said object; if it succeeds the robot should go around the object leaving in between a relatively safe distance. The navigation data was recorded in a rosbag file in order to be analyzed later.

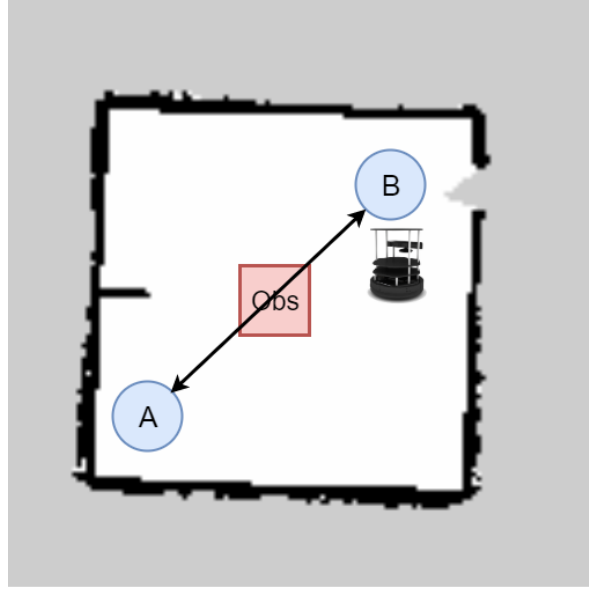


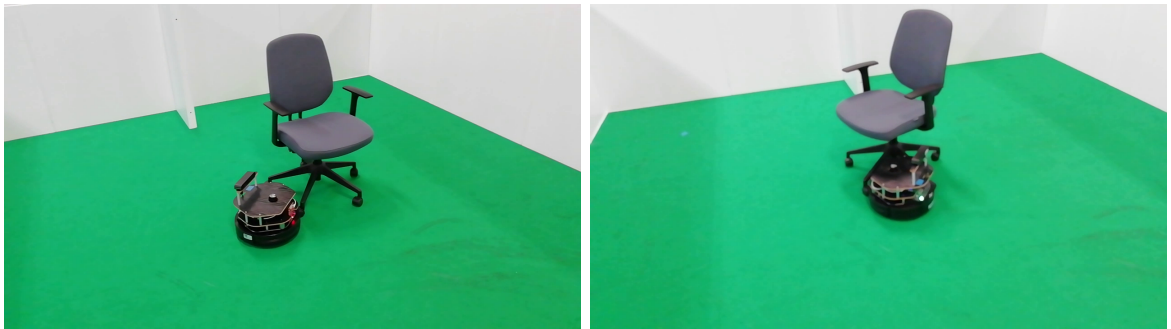
Figure 5.3: Demonstration of the course of test, the robot will go back and forth from position A to position B 5 times while trying to avoid the obstacle in the middle

## 5.1.2 Results

In this section we show the results of how the robot performed in avoiding the previous obstacles using the FMCW radar and the 2D-LiDAR.

### 5.1.2.1 Office Chair

In the LiDAR's case the robot disregarded the chair going in a straight line and pushing it until it was away from the experimental setup has shown in Figure 5.4. This was due to the

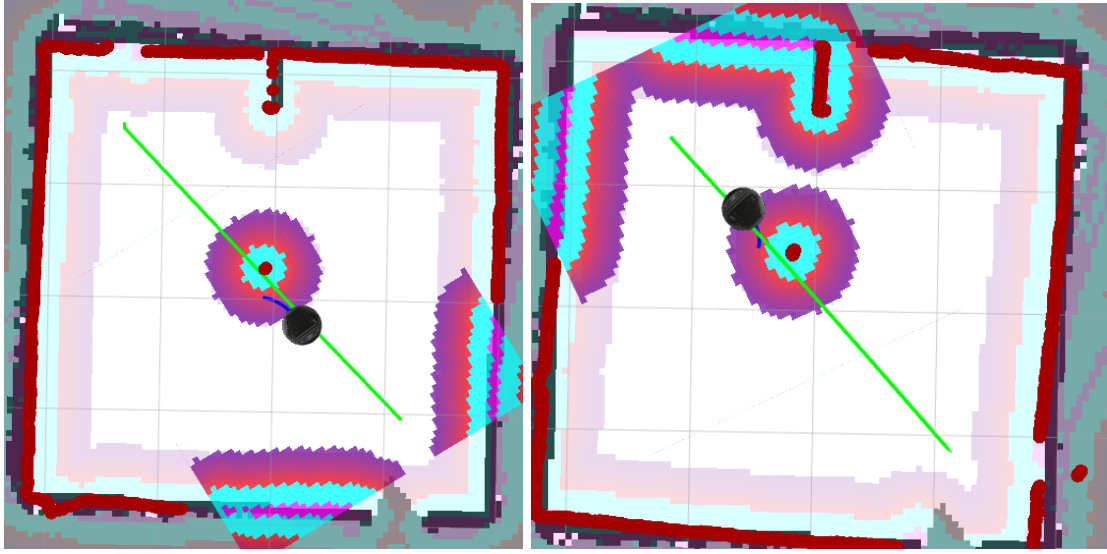


(a) Robot collision with office chair number 1

(b) Robot collision with office chair number 2

Figure 5.4: Robot colliding multiple times with office chair

2D LiDAR only detecting the leg of the chair and not the wheels. This means that the robot only perceived a single point as being occupied and not the full area of the chair. Figure 5.5 shows two instances of the experiment in rviz. As we can see the robot only perceives a single



(a) Robot only perceiving a single point as an obstacle (b) Robot only perceiving a single point as an obstacle

Figure 5.5: Rviz display of the navigation data with the office chair as an obstacle and the LiDAR as an obstacle detector

point as being occupied.

With the FMCW radar the chair is detected almost immediately, this makes the global planner and motion controller to be able to adjust in a very comfortable way and with it the robot is able to avoid collision. Figure 5.6 shows some instances of the experiment. Since the radar has small field of view (120 degrees) the robot might not detect the obstacle when it is passing by it that in fact happens once, leading the robot to scrape the chair. Figure 5.7

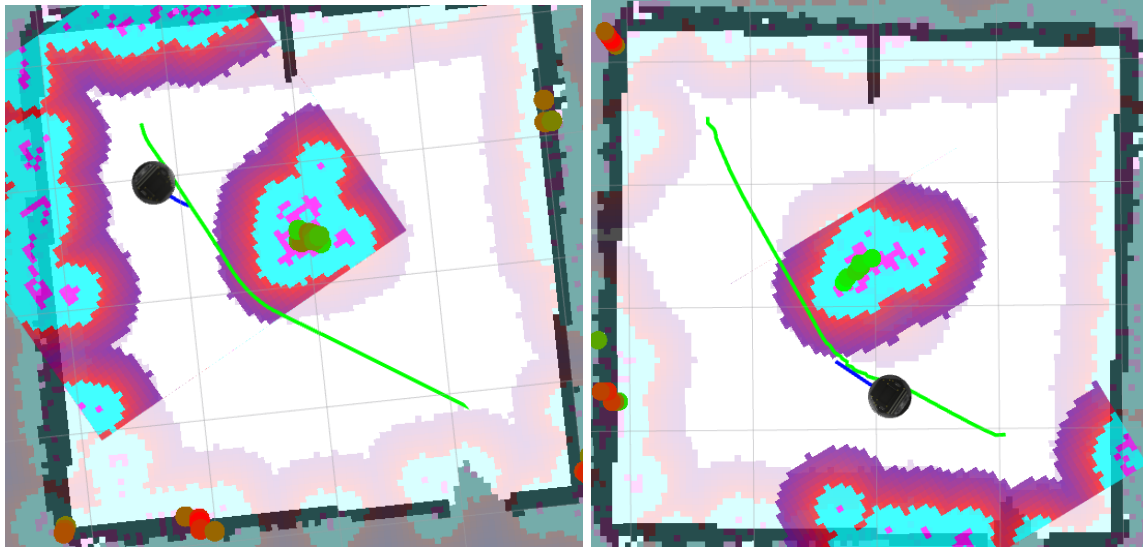


(a) Robot avoiding office chair number 1

(b) Robot avoiding office chair number 2

Figure 5.6: Robot avoiding with office chair

shows two instances of the experiment in rviz. Analysing the navigation data we can clearly see that the robot detects multiple points in the office chair's case. Figure 5.8 shows the robot trajectories in the LiDAR and the FMCW radar cases as well as an approximation of the invalid space the center of the robot can not go through in dimmed red. As we can see the



(a) Robot circumventing the office chair 1

(b) Robot circumventing the office chair 2

Figure 5.7: Rviz display of the navigation data with the office chair as an obstacle and the FMCW radar as the obstacle detector

robot flat out ignores the obstacle with LiDAR while in the FMCW radar case it avoids it at all times.

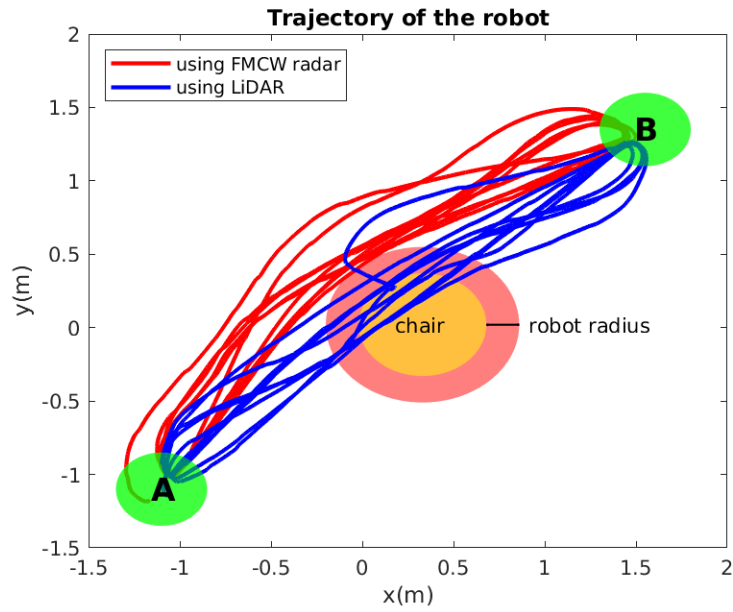
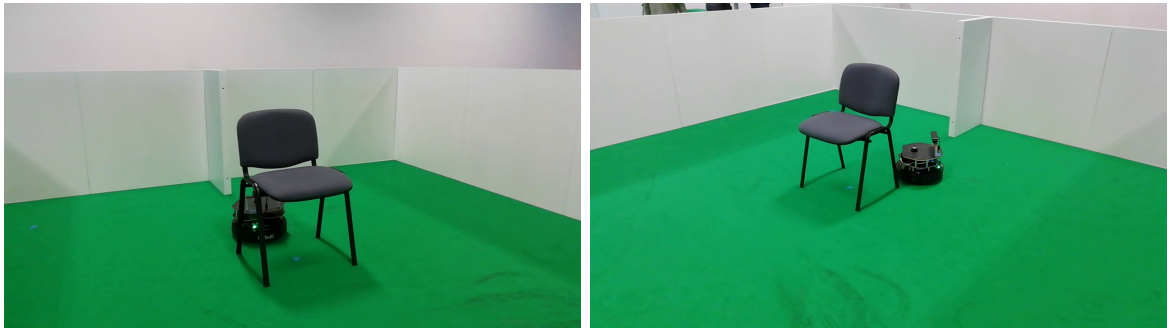


Figure 5.8: Trajectory of the robot for the office chair's case

### 5.1.2.2 Four Legged Chair

Using LiDAR the robot did not succeed in producing a safe trajectory. First off it went in a straight line just as the previous case almost hitting the legs of the chair. However when it got to close to it, it came to a halt and kept oscillating for a long time until it either collided with the leg or scraped it. This was due to the LiDAR only detecting this type of obstacle at small distances. This behavior repeated itself throughout the task, leading to the chair being pushed several times. Figure 5.9 shows two instances where the robot gets stuck near the chair. This type of behavior is indicative that the robot's motion controller is stuck in local minimum, in other words computing a safe trajectory around the obstacle is countered by its force to follow the goal and path. Figure 5.10 shows three instances of the experiment in rviz

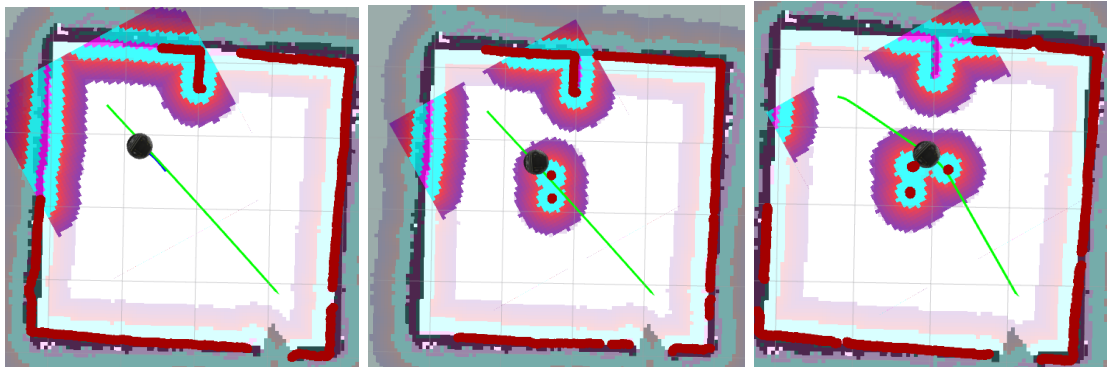


(a) Robot getting stuck in chair number 1

(b) Robot getting stuck in chair number 2

Figure 5.9: Robot getting stuck in four legged chair

where the robot uses the 2-DLiDAR has an obstacle detector. As we can see the robot only



(a) Robot not perceiving any type of obstacle in its way

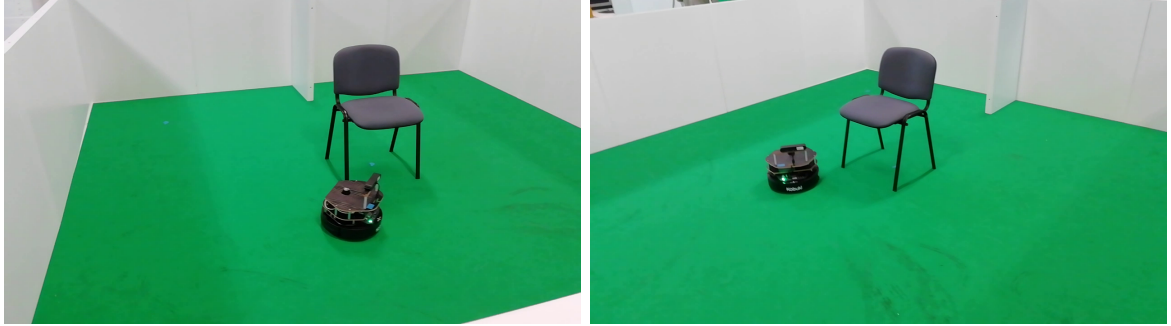
(b) Robot detecting 2 legs of the chair

(c) Robot getting stuck in chair due to being too close to it

Figure 5.10: Rviz display of the navigation data with the four legged chair as an obstacle and with the 2D-LiDAR as an obstacle detector

detects the chair's legs when it got close to it. This made the robot get stuck in between the chair's legs.

Using the FMCW radar proved to be much better with the robot safely circumventing around the chair at safe distance for all duration of the task. Figure 5.11 shows two instances of the turtlebot avoiding the chair at a safe distance. Figure 5.12 shows an instance of the



(a) Robot avoiding four legged chair number 1      (b) Robot avoiding four legged chair number 2

Figure 5.11: Robot avoiding the four legged chair

experiment in rviz where the robot uses the FMCW radar has an obstacle detector. In this

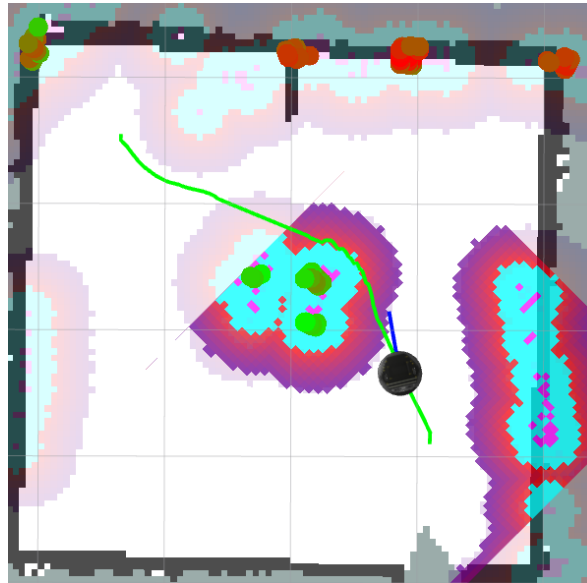


Figure 5.12: Rviz display of the navigation data with the four legged chair with the FMCW radar as an obstacle detector. The robot circumvents the four legged chair

case the legs of the chair are detected immediately which makes the robot circumvent safely without getting stuck. Analysing the data we see that when the robot is facing it it detects almost all legs immediately. This makes it so the robot is aware of it at all times and planning around it.

Figure 5.13 shows the trajectory of the robot for the LiDAR and FMCW radar and an approximate position of the invalid space the four legs of the chair create coloured in dimmed red. The center of the robot should not passthrough this area since it will lead to collision. As

we can see the trajectory produced by the FMCW radar case is much less susceptible to the object than in the LiDAR case.

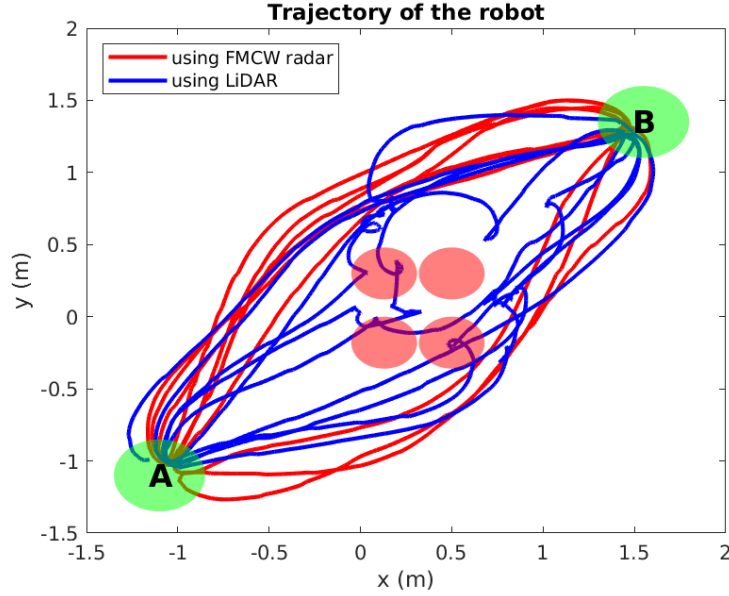
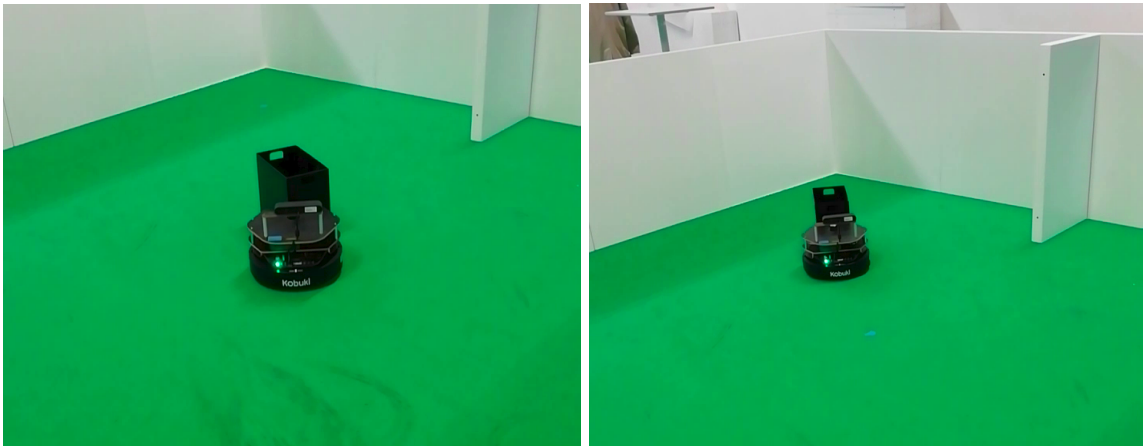


Figure 5.13: Trajectory of the robot for the four-legged chair's case

### 5.1.2.3 Garbage Bin

In the LiDAR's case the robot went in a straight line as in the wheeled chair's case pushing the garbage bin until it reached its first goal. This happened because the LiDAR was unable to properly detect the garbage bin at a certain angle. Figure 5.14 shows two instances of the turtlebot hitting the garbage bin.

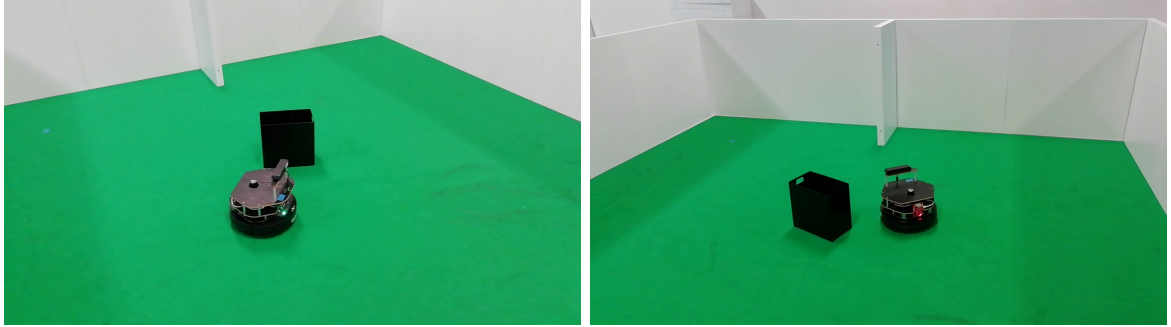


(a) Robot crashing into garbage bin

(b) Robot pushing garbage bin

Figure 5.14: Two instances of the robot's navigation with the garbage bin as an obstacle

Using the FMCW radar, the robot properly detected the garbage bin and went around it easily. Figure 5.15 shows two instances of the turtlebot avoiding the garbage bin at a safe distance. Figure 5.16 shows three instances of the experiment in rviz where the robot uses the

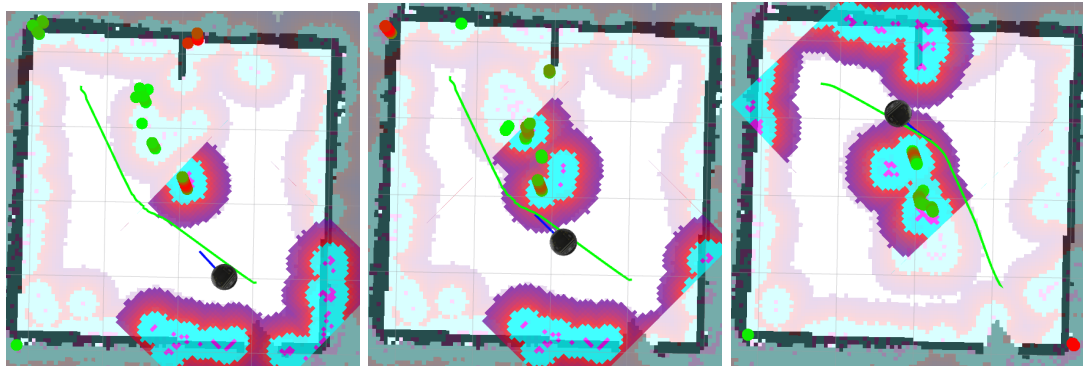


(a) Robot avoiding garbage bin number 1

(b) Robot avoiding garbage bin number 2

Figure 5.15: Robot avoiding garbage bin

FMCW radar has an obstacle detector. Analysing the navigation data we see that the robot



(a) Robot avoiding the garbage bin 1

(b) Robot avoiding the garbage bin 2

(c) Robot avoiding the garbage bin 3

Figure 5.16: Rviz display of the navigation data with the garbage bin as an obstacle and with the FMCW radar as an obstacle detector

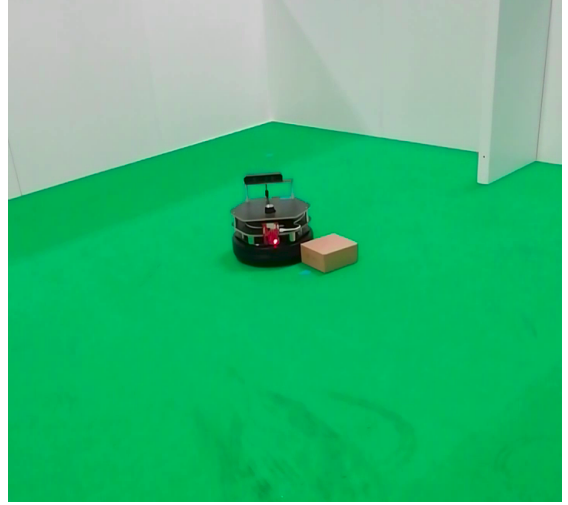
perceives the grabage bin correctly but it also detects obstacles behind it. However the robot still made its course without any trouble.

#### 5.1.2.4 Low height box

In the LiDAR's case the robot did not detect the low height box throughout all the course. This makes sense since this sensor is built to detect objects that are the same height as the sensor. In other words it detects only the horizontal plane of the 2D-LiDAR. This failure of detection made it so the robot could not replan its course and in consequence it collided with said box. Figure 5.17 shows two instances where the robot collided with the box using LiDAR as a sensor source.



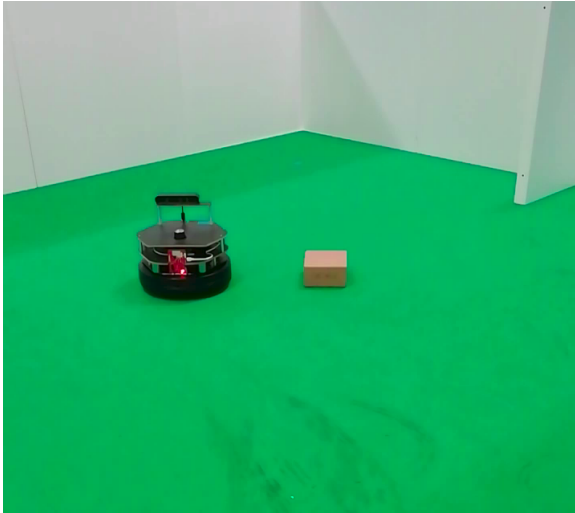
(a) Robot collision with box number 1



(b) Robot collision with box number 2

Figure 5.17: Robot collision with box

For the FMCW radar case, initially the robot also did not detect the box. However by decreasing the threshold of the passthrough filter of intensity to 14 we made it so it can detect it. With this modification the robot was able to complete the test without colliding with the box. Figure 5.18 shows two instances of the robot performing the course while avoiding the box with the previous modification. However this change may result in the FMCW radar having false detections due to now having a lower level of SNR. Figure 5.19 shows an instance



(a) Robot avoiding box number 1



(b) Robot avoiding box number 2

Figure 5.18: Robot avoiding box

of the experiment in rviz where the robot uses the FMCW radar as an obstacle detector. As we can see with the change the robot perceives the box with the threshold change. This made

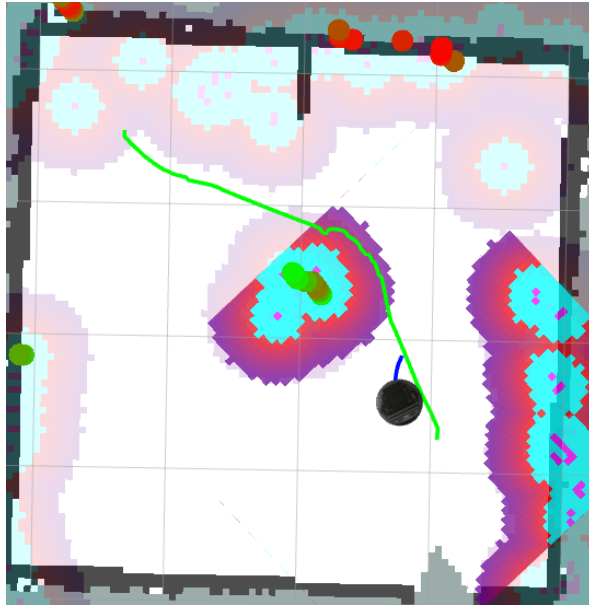
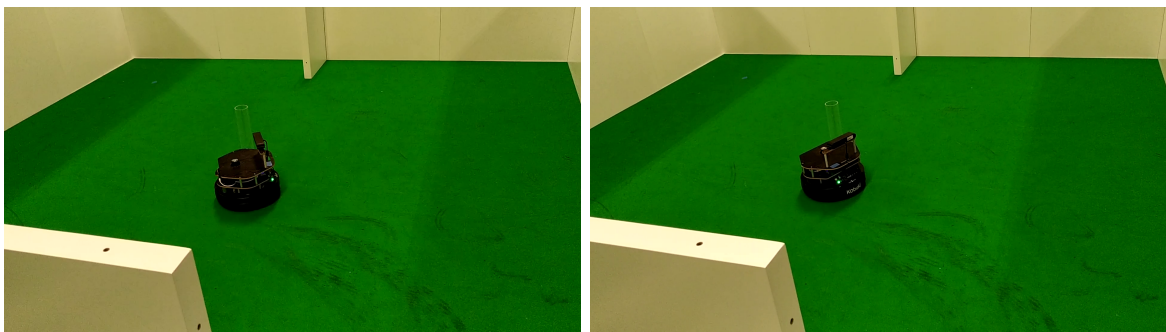


Figure 5.19: Rviz display of the navigation data with the low height box as an obstacle and the FMCW radar as an obstacle detector

it so the robot successfully finished the course without hitting it.

#### 5.1.2.5 Acrylic tube

When it comes to the Acrylic tube using the 2-D LiDAR it falsely detected targets where there where none. This is due to the physical proprieties of the object being unsuitable for LiDAR technology to handle. The robot in its course got stuck near the obstacle due to think it had crashed into it. However this was not the case and the robot did not succeed in the experiment. Figure 5.20 shows two instances of the robot getting stuck in the acrylic tube. Figure 5.21 shows an instance of the experiment in rviz where the robot uses the 2D-LiDAR



(a) Robot getting stuck acrylic tube number 1 (b) Robot getting stuck in acrylic tube number 2

Figure 5.20: Robot getting stuck in acrylic tube

as an obstacle detector. As we can see the robot perceives obstacles where there are none.

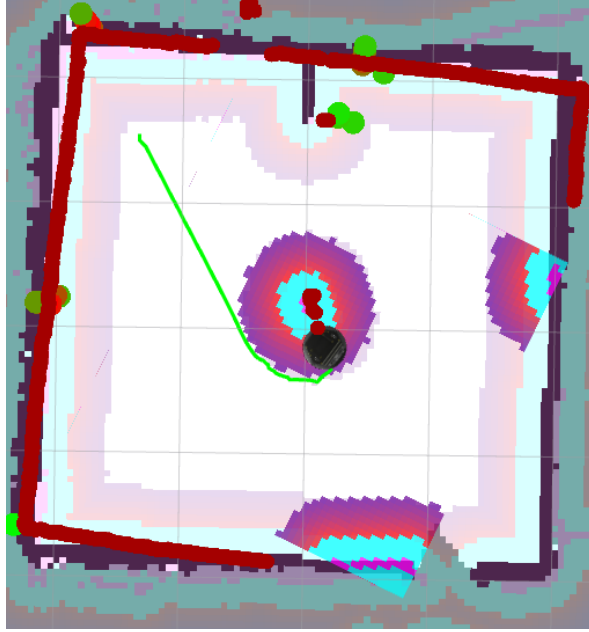
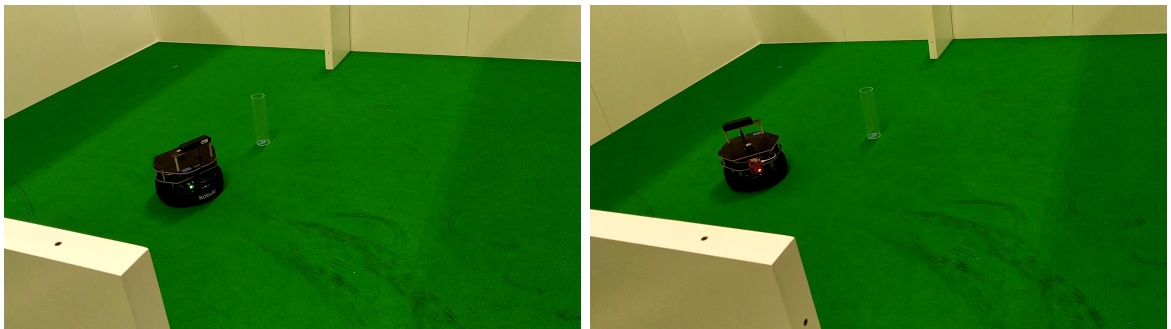


Figure 5.21: Rviz display of the navigation data with acrylic tube as an obstacle and the 2D-LiDAR as an obstacle detector

This made the robot behave in a strange way that was not the behavior we want to see for the experiment.

In the FMCW radar case the robot was able to properly detect the tube and thus avoiding it all the course. The behavior of the robot was better than the previous case since it went through all course without hitting the tube in any way. Figure 5.22 shows two instances of the robot avoiding the acrylic tube. Figure 5.19 shows three instances of the experiment in



(a) Robot avoiding acrylic tube number 1

(b) Robot avoiding acrylic tube number 2

Figure 5.22: Robot avoiding acrylic tube box

rviz where the robot uses the FMCW radar as an obstacle detector. As shown in the figure we see that the robot avoids the obstacle without any problem.

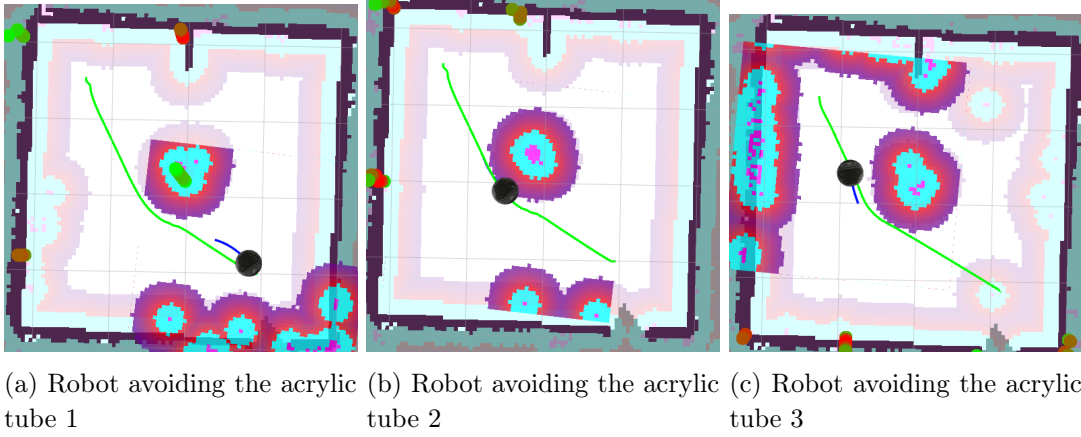


Figure 5.23: Rviz display of the navigation data with the acrylic tube as an obstacle and with the FMCW radar as an obstacle detector

#### 5.1.2.6 Robot (turtlebot2)

In this last case both the FMCW radar and the 2D-LIDAR performed reasonably well avoiding the obstacle with ease for all the trajectory.

### 5.1.3 Discussion

In this experiment we conclude that there are multiple types of obstacles that the 2-D LiDAR is unable to properly detect. This poor detection led to the robot crashing into the objects making it so the navigation task presented ended in failure. However using the FMCW radar proved to have a better perception of all the obstacles in the experiment and with it the indoor navigation tests were concluded with success with the robot circumventing them in a safe manner. We conclude that there are advantages in using the FMCW radar as an obstacle detector.

## 5.2 Dynamic Obstacles in controlled environment

In the previous experiment we only dealt with static objects. In this new test we want to see how the FMCW radar handles dynamic objects in a controlled space. For that we send multiple goals to the robot and use an additional turtlebot2 or a person to obstruct its path. In other words we have the robot trying to reach multiple goals multiple times and the dynamic obstacle will try to make more difficult by obstructing the planned paths of the robot. The main goal of this experiment is to see if the robot can handle unstructured environments that are often the case in indoor scenarios.

### 5.2.1 Experimental Setup

For this test we send four goals to the robotic platform, A,B,C,D in zigzag fashion and we will place a dynamic obstacle obstructing it as shown in Figure 5.24.

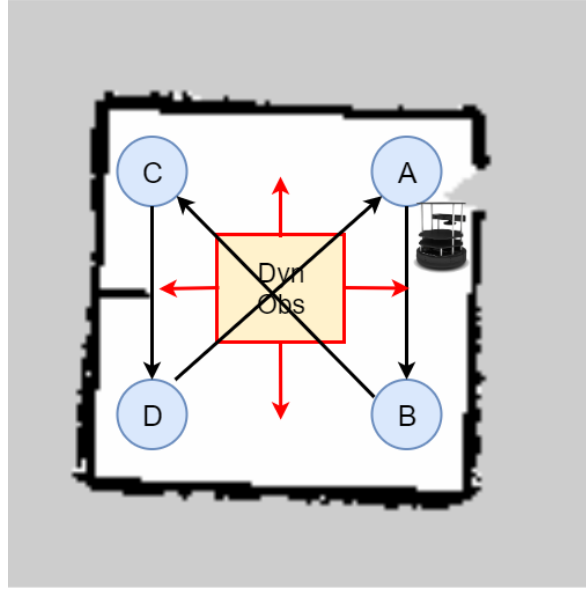


Figure 5.24: Demonstration of the course of test, the robot will run through A-B-C-D while trying to avoid the dynamic obstacle obstructing its path

For the dynamic obstacles we will use firstly a person and in a second trial a mobile robot. These obstacles will try to obstruct the path of the robot throughout the course. For this case we will only use the FMCW radar as an obstacle detector.

## 5.2.2 Results

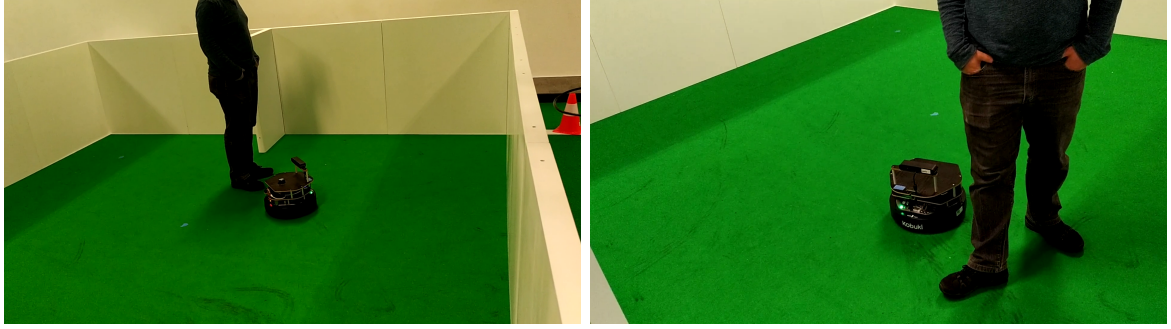
In the following sections we show the results for the person and the mobile robot as obstacles for the experiment.

### 5.2.2.1 Person

Throughout all the test the robot successfully avoided the person only using the FMCW radar as an obstacle detector. Figure 5.25 shows two instances of the robot avoiding the person in the experiment.

However the robot was only able to perceive the legs of said person and not its feet. This may lead to cases where the robot shocks in the person if it is too close.

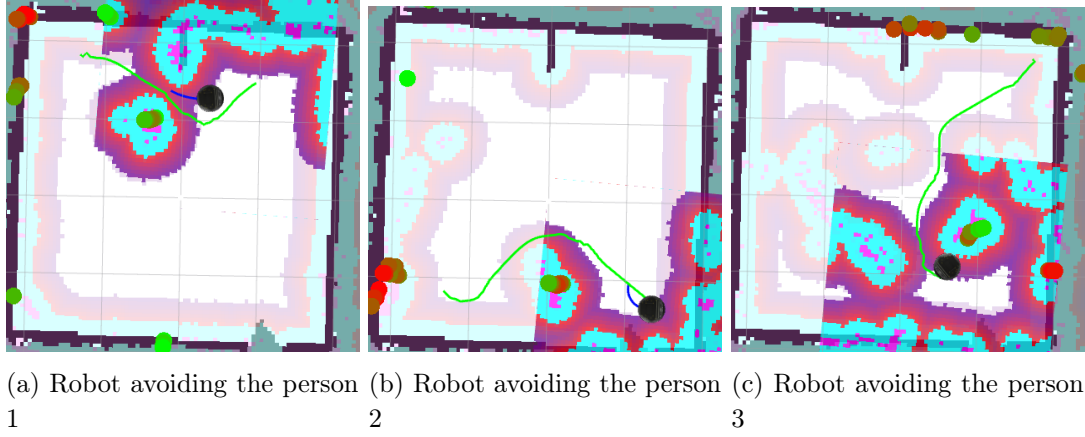
Figure 5.26 shows three instances of the experiment in rviz. Analysing the navigation data we see that the robot perceives the person and goes around it easily most of the time. However when the person got too close to the robot, it stopped completely and its recover behaviors were activated. We conclude that using the FMCW radar, the robot is able to avoid moving people which may indicate that this sensor can be used for social environments. We also conclude that it is better suited for detecting obstacles that are at least half a meter away from the robot and that are facing it directly due to small field of view ( $120^\circ$ ).



(a) Robot avoiding the person number 1

(b) Robot avoiding the person number 2

Figure 5.25: Robot avoiding the person



(a) Robot avoiding the person  
1

(b) Robot avoiding the person  
2

(c) Robot avoiding the person  
3

Figure 5.26: Rviz display of the navigation data with the person as a dynamic obstacle and with the FMCW radar as an obstacle detector

#### 5.2.2.2 Mobile Robot

For the mobile robot case the results were similar to the previous case. The robot was able to detect and avoid the other robot when it is facing it.

Figure 5.27 shows two instances of the robot avoiding the dynamic robot.

Figure 5.28 shows three instances of the experiment in rviz. As we can see from the figures our robot is able to avoid the mobile robot at all times in the experiment.

#### 5.2.3 Discussion

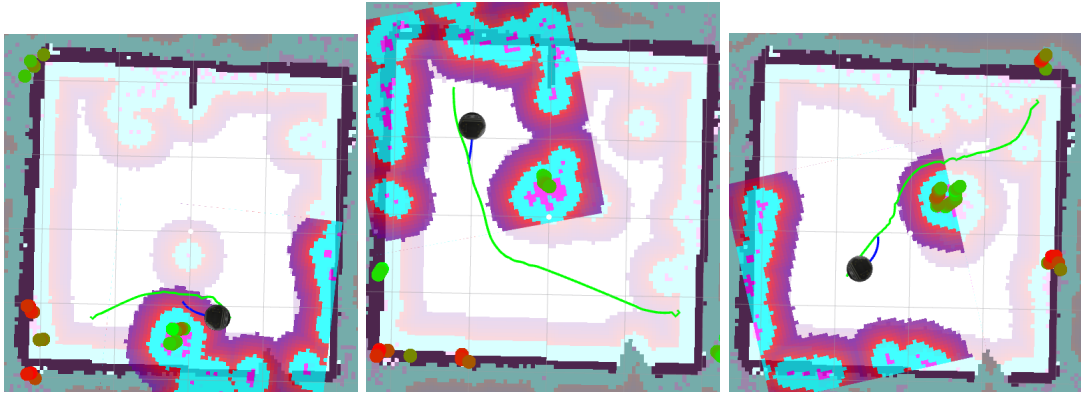
The robot was able to handle dynamic obstacles using only the FMCW radar, this meaning that it might be feasible to use this sensor for obstacle avoidance in ever changing indoor environments.



(a) Robot avoiding dynamic robot 1

(b) Robot avoiding dynamic robot 2

Figure 5.27: Robot avoiding dynamic robot



(a) Robot avoiding the robot 1

(b) Robot avoiding the robot 2

(c) Robot avoiding the robot 3

Figure 5.28: Rviz display of the navigation data with the robot as a dynamic obstacle and with the FMCW radar as an obstacle detector

## 5.3 Dynamic Obstacles in uncontrolled environment

In the previous section we used a small confined space for the robot to operate in. This made it so the robot localization was easy to determine. It also was in an almost closed space, which means he can raytrace the environment to clean previous detected obstacles. However in this next experiment we seek to analyse how the robot behaves when it is exposed to an open space. For that we make an experiment in the IRIS laboratory. In the following test we want to answer if the FMCW radar is able to clear previously obstructed spaces (by the person in this case) by raytracing its environment.

### 5.3.1 Experimental Setup

First a map was created of the IRIS laboratory using the package `gmapping` provided by ROS (Fig. 3.7). Then using the map and the package `AMCL` we ensure that the robot localization is fairly reasonable. After that we set up the robot to do a certain navigation task. The robot starting position and goal were set as shown in Figure 5.29 in the IRIS

laboratory.



Figure 5.29: Experimental setup showing the initial position and the goal sent to the robot. A person will be put to obstruct the path of the robot

In this case a single person was instructed to actively obstruct the robot’s forward movement until the robot reaches its first goal. After reaching it the person gets out of the environment and the robot is instantaneously given a second goal which in this case is its the starting position. The experiment was repeated five times with the FMCW radar and the 2D-LiDAR.

### 5.3.2 Results

As expected, using the 2D-LiDAR, the robot was able to detect and avoid the person in all 5 cases. However it should be noted that in one of these cases the robot tried to avoid it by going through an obstructed space (that was not the person) due to a missed detection. This lead to collision. The robot was also able to clear the previously obstructed spaces, getting to the starting position without avoiding past marked obstacles.

The FMCW radar was also able to detect the obstructing person and managed to plan around it in all cases as shown in Figures 5.30a, 5.30b and 5.30c. Since the radar cloud is less dense, clearing marked obstacles was slower than in the LiDAR case. However this did not impact the overall performance of the navigation task in a significant way since it still went to the starting position in an almost straight line (Figure 5.30d).

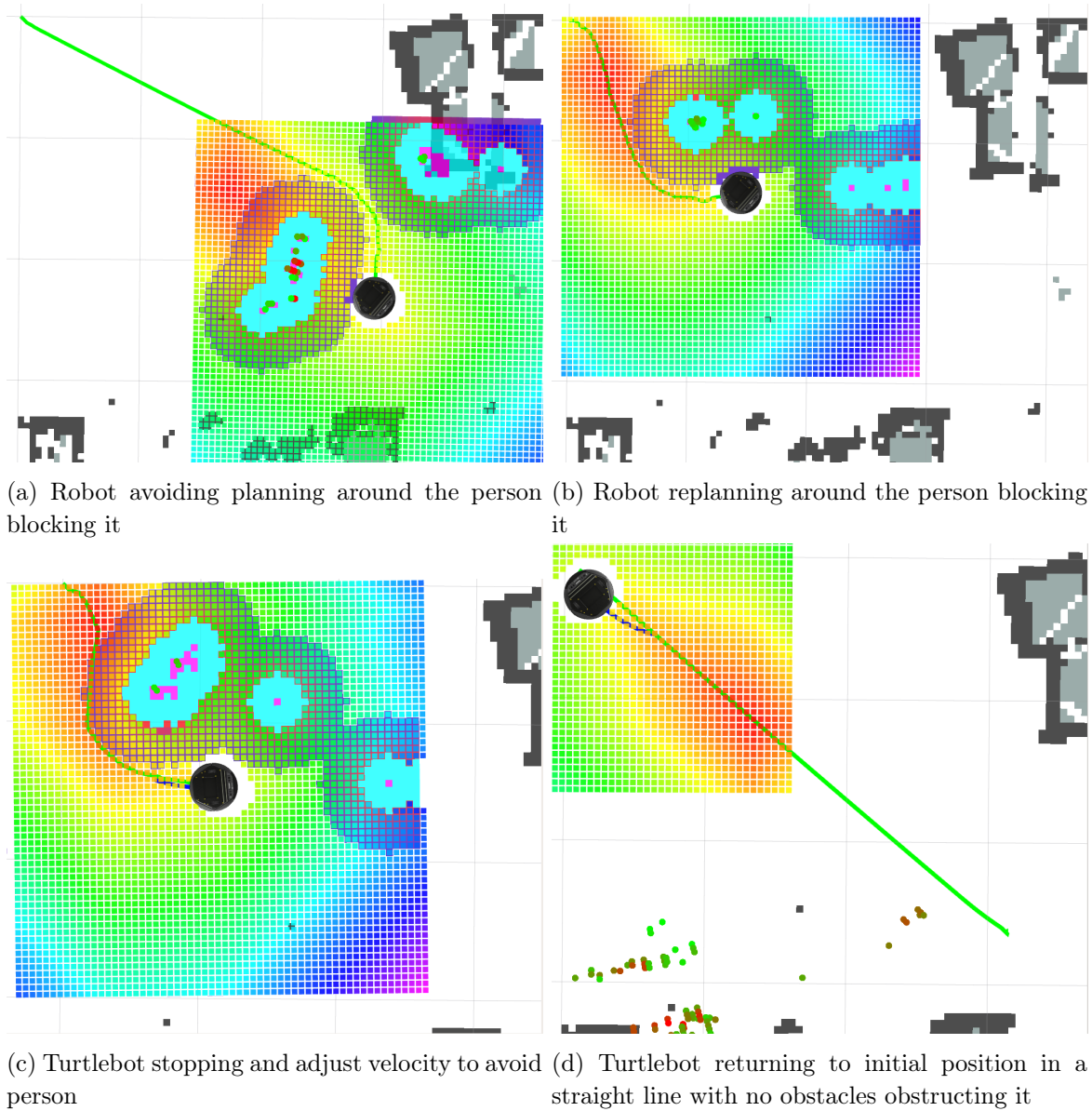


Figure 5.30: Four instances of the robot in the experiment

### 5.3.3 Discussion

The FMCW radar was able to perform obstacle avoidance of dynamic obstacles (in this case a person) that continuously obstructed its path at 5 different times in the same environment which may suggest the use of the FMCW radar as an alternate sensory unit over the LiDAR. We also found that the FMCW radar is able to clean the dynamic obstacle presented.

## 5.4 Summary

In this chapter we presented various experiments used to evaluate the performance of the FMCW radar as an obstacle detector and using in some of them the 2-DLiDAR as a comparative source. In the first experiment we concluded that there are various objects in which the 2-DLiDAR is unable to detect due to the height or proprieties of the object. However this obstacles were correctly detected by the FMCW radar. This means that this type of sensors are advantageous for scenarios in which this type of obstacles are commonly used (office environments). We also verified how the FMCW radar handles dynamic obstacles, as we successfully made the robot avoid them only using the FMCW radar as an obstacle detector. Finally we conducted an experiment in an unstructured environment with a dynamic obstacle (person). It terminated with success in all cases. With this we conclude that the FMCW radar is a viable sensor to support obstacle avoidance.



## Chapter 6

# Towards The Use Of Doppler To Enrich Obstacle Avoidance

If the velocities of objects relative to the (moving) robot are available, this information can be used to compute better paths. In this chapter we present an exploratory work on how this can be implemented, using the doppler data provided by the radar. The approach proposed is based on adapting the layered costmap, at the inflation layer by substituting it by another that takes into account velocity information.

### 6.1 Layered Costmaps

As said before the global and local costmaps used in ROS Navigation Stack are both layered costmaps. This means that they are composed by independent components with each one affecting the resulting master layered costmap for specific environmental contexts. For example the classic *static layer* takes information from a published map topic and based on the position of the robot marks some pre-determined obstacles while the inflation layer propagates the cost of obstacles radially. Figure 6.1 show some different layers for different contexts.

The set of layers used follows a specific hierarchy that determines what order and how they overwrite the master costmap. This is an important part to take into consideration because the priority of the different layer plugins will determine the final costmap that will be used by each planner (global or local).

### 6.2 Inflation Layer

When it comes to define values for the costmap occupancy grid the inflation layer is usually the case in the ROS navigation stack. Inflation is the process of propagating cost values out from occupied cells that decrease with distance [48]. Figure 6.2 shows an overview of the different values each cell can have (0 to 254) and what they mean when it comes to the robot perception.

All cells in the costmap further than the inscribed radius distance and closer than the

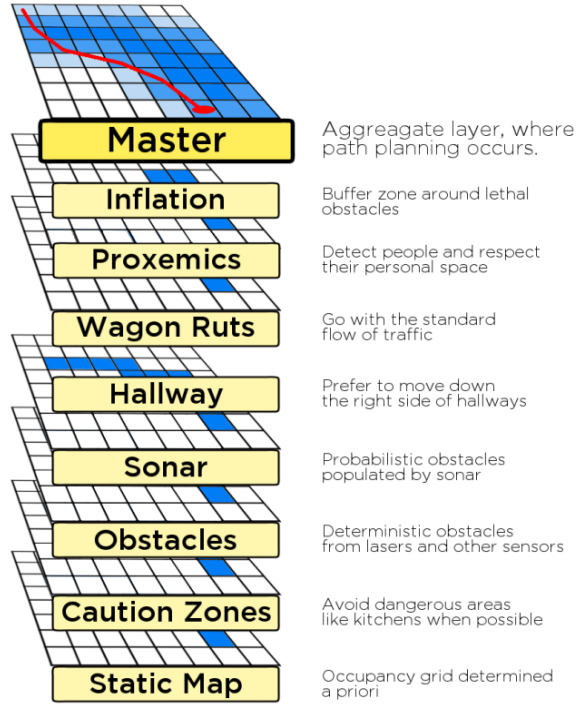


Figure 6.1: A stack of costmap layers, showcasing the different contextual behaviors achievable with the layered costmap approach from [42]

inflation radius distance away from an actual obstacle is given by:

$$cell_{cost} = 253 * e^{(-k*(dist-r))} \quad (6.1)$$

where  $k$  is the cost scaling factor that determines how fast the decaying of the function is,  $dist$  is the distance from the obstacle to the robot and finally  $r$  is the robot radius. This means that cells near obstacles will have a value that will tend to 253 while cells far away from the obstacle will have values trending to 0. Note that  $dist$  must be always greater than  $r$  because if it isn't than it means the robot collided with said obstacle. This inflation is only taking into account distance from the obstacle as input which means that the values of the cells will be propagated radially.

### 6.3 Doppler Layer

In the previous case, inflation only took into account the distance of obstacles for computing what value each cell in the costmap will have. In unstructured environments there may be obstacles that have negative relative radial velocity in regards to the robot, or in other words incoming obstacles. This obstacles may crash into the robot since the velocity information is not considered by the robot. To surpass this we want to design a layer that not only takes into account the distance but also the relative radial velocity of the object to determine the value of cells in the costmap. To do this we designed a layer that adjusts the cell values of the

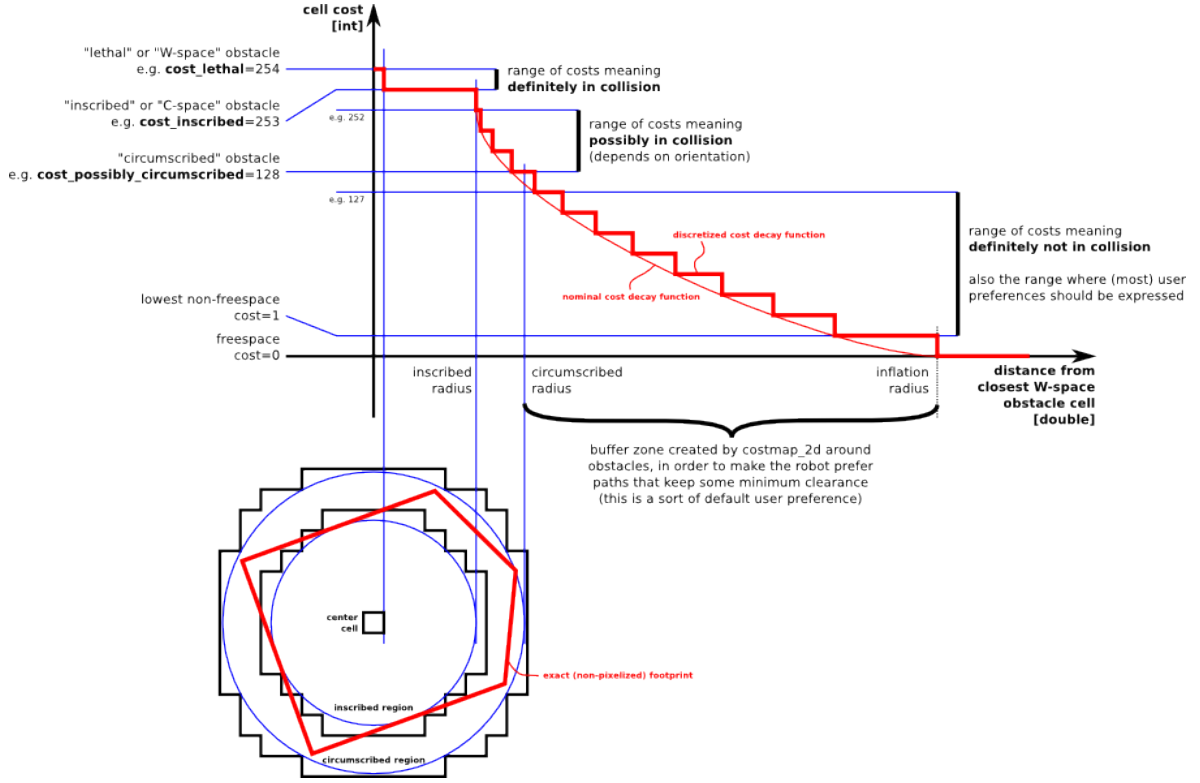


Figure 6.2: Inflation layer specifications from [48]

master costmap taking into account the radial velocity of said objects taken from the FMCW radar. Taking into account the relative radial velocity  $v_r$  the cell value of the surrounding cell's is given by:

$$\begin{aligned}
 cell_{cost} &= Ae^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)} \\
 \sigma_x^2 &= cov * factor * |v_r| \\
 \sigma_y^2 &= cov \\
 x &= dist * \cos(\theta) \\
 y &= dist * \sin(\theta)
 \end{aligned} \tag{6.2}$$

where  $cov$ ,  $factor$  and  $A$  are parameters adjusted by the user,  $\theta$  is the angle between the cell where we want to calculate its value and the direction of the velocity vector of the target object, and finally  $dist$  is the distance between the two previous cells. This function is a gaussian distribution that takes into account velocity and distance from the obstacle. With this cost function we can manipulate the costmap cell values to have higher values in the direction of incoming objects. Figure 6.3 shows an example of the doppler layer changing the 2D master costmap taking into account the velocity and distance of the target obstacle.

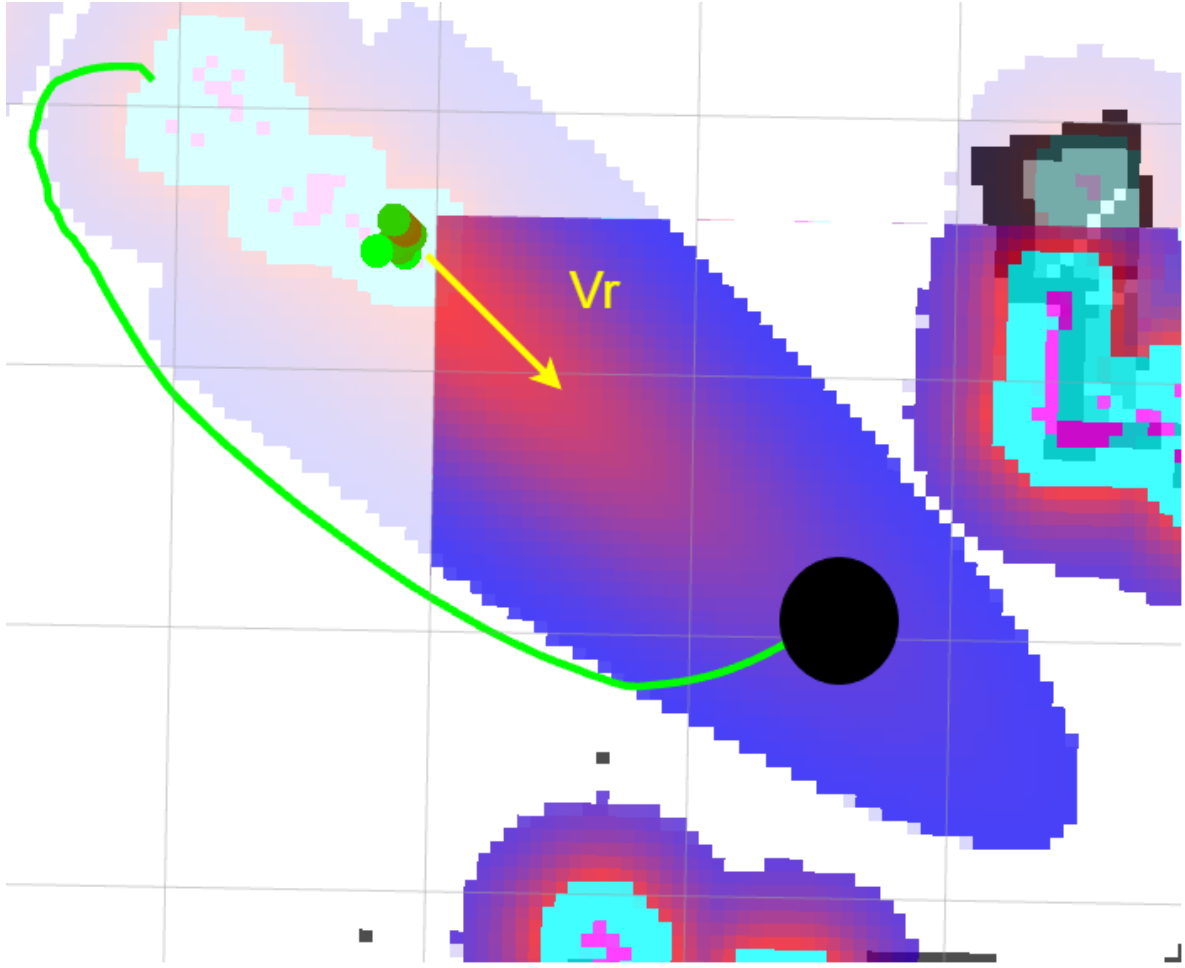


Figure 6.3: Example of doppler layer manipulation of the master costmap with the parameters  $A=253$ ,  $\text{factor}=20$  and  $\text{cov}=0.1$

## 6.4 Discussion

By using the doppler layer we have concluded that the trajectory of the robot can be altered taking into consideration the relative radial velocity of the target obstacles. This may be useful for high populated indoor environments where people constantly are shifting from one place to another. This makes the robot safe from incoming obstacles and therefore is a good support for our navigation system. There are some limitations for this layer however, the first one being that the velocity resolution of the radar is limited which may produce bad results when the radial velocity of the objects is low. Another limitation is that the global costmap update is usually around 1 Hz, and a big amount of information may be outdated when the control loop occurs which may lead on late replanning by the global planner. For the local planner another problem is that its limited dimensions do not allow the prediction of long range obstacles since it only works in the vicinity of the robot.

## Chapter 7

# Conclusion and Future Work

### 7.1 Conclusion

LiDAR technology is one of the most common approach to sense the environment for navigation purposes. However, new radar devices are appearing that, by their cost and dimensions, could be an appealing alternative or complementary technology for the same purpose. In this work, the FMCW radar was evaluated as a potential sensing device for navigation purposes.

To support the evaluation, a Turtlebot2 platform was adapted in order to equip it with a LiDAR and an FMCW radar. Afterwards, a software architecture was defined allowing the evaluation. A set of experiments were devised and implemented and the results obtained proved the validity of using the FMCW radar as an obstacle detector sensor device. In one hand, they showed some type of objects not or poorly detected by a LiDAR can be better detected by the radar. In the other hand, problems that could arise from the fact that the radar produce less dense data does not showed to be significant in keeping the costmap updated.

As a final work, an exploratory work on the use of the Doppler data provided by the radar was conducted. It was proposed a manipulation of the layered costmap approach used by the navigation stack in ROS to define a new way of inflating the obstacles taking into account its relative radial velocity in relation to the robot.

### 7.2 Future Work

Taking into account the work done in this dissertation there are multiple objectives that we can try to perform in the near future. Firstly, in the first experiment we only experiment with a small amount of different obstacles. The number of objects can be expanded to try and find more where FMCW radar detects and where the 2D LiDAR does not detect. Another objective to take in mind is the further improvement of the plugin costmap layer proposed in this work. The parameterization and optimization may be key factor for the successful use of this layer to address unstructured dynamic environments. Lastly in this work we only tackled the obstacle avoidance capabilities of the FMCW radar, in the future we can try to use it for mapping or even localization purposes.



# Bibliography

- [1] M. R. Pedro Lima, “Mobile robotics,” University Lecture, Instituto Superior Tecnico, University of Lisbon, 2002.
- [2] Admin, “100 applications or uses of lidar technology,” Dec 2017. [Online]. Available: <http://lidarradar.com/apps/100-applications-or-uses-of-lidar-technology>
- [3] “Leddartech developing lidar sensor solutions for autonomous vehicles,” Apr 2018. [Online]. Available: <http://insideunmannedsystems.com/leddartech-developing-lidar-sensor-solutions-for-autonomous-vehicles/>
- [4] R. Dominguez, E. Onieva, J. Alonso, J. Villagra, and C. Gonzalez, “Lidar based perception solution for autonomous vehicles,” in *2011 11th International Conference on Intelligent Systems Design and Applications*. IEEE, 2011, pp. 790–795.
- [5] A. N. Catapang and M. Ramos, “Obstacle detection using a 2d lidar system for an autonomous vehicle,” in *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. IEEE, 2016, pp. 441–445.
- [6] M. Burns, “‘anyone relying on lidar is doomed,’ elon musk says – techcrunch,” Apr 2019. [Online]. Available: <https://techcrunch.com/2019/04/22/anyone-relying-on-lidar-is-doomed-elon-musk-says/>
- [7] Instructables, “Rosbot - autonomous robot with lidar and husarion core2 controller,” Aug 2017. [Online]. Available: <https://www.instructables.com/id/ROSBot-Autonomous-Robot-With-LiDAR/>
- [8] “78 Ways Lidar Brings Mobile Robots to Life,” Mar 2019. [Online]. Available: <https://velodynelidar.com/newsroom/8-ways-lidar-brings-mobile-robots-to-life/>
- [9] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, “The office marathon: Robust navigation in an indoor office environment,” in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 300–307.
- [10] S. Gatesichapakorn, J. Takamatsu, and M. Ruchanurucks, “Ros based autonomous mobile robot navigation using 2d lidar and rgb-d camera,” in *2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*. IEEE, 2019, pp. 151–154.

- [11] T. B. Lee, "How 10 leading companies are trying to make powerful, low-cost lidar," Feb 2019. [Online]. Available: <https://arstechnica.com/cars/2019/02/the-ars-technica-guide-to-the-lidar-industry/>
- [12] H. P. K.K, "Measuring distance with light." [Online]. Available: <https://hub.hamamatsu.com/us/en/application-note/measuring-distance-with-light/index.html>
- [13] O. Yalcin, A. Sayar, O. Arar, S. Akpınar, and S. Kosunalp, "Approaches of road boundary and obstacle detection using lidar," *IFAC Proceedings Volumes*, vol. 46, no. 25, pp. 211–215, 2013.
- [14] J. A. Grant, M. J. Brooks, and B. E. Taylor, "New constraints on the evolution of carolina bays from ground-penetrating radar," *Geomorphology*, vol. 22, no. 3-4, pp. 325–345, 1998.
- [15] Admin, "Different types of radar systems," Jan 2018. [Online]. Available: <http://lidarradar.com/definition/different-types-of-radar-systems>
- [16] S. Heuel and H. Rohling, "Pedestrian recognition based on 24 ghz radar sensors," in *11-th INTERNATIONAL RADAR SYMPOSIUM*. IEEE, 2010, pp. 1–6.
- [17] N. Knudde, B. Vandersmissen, K. Parashar, I. Couckuyt, A. Jalalvand, A. Bourdoux, W. De Neve, and T. Dhaene, "Indoor tracking of multiple persons with a 77 ghz mimo fmcw radar," in *2017 European Radar Conference (EURAD)*. IEEE, 2017, pp. 61–64.
- [18] A. Abosekeen, A. Noureldin, and M. J. Korenberg, "Utilizing the acc-fmcw radar for land vehicles navigation," in *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE, 2018, pp. 124–132.
- [19] M. Alizadeh, G. Shaker, J. C. M. De Almeida, P. P. Morita, and S. Safavi-Naeini, "Remote monitoring of human vital signs using mm-wave fmcw radar," *IEEE Access*, vol. 7, pp. 54 958–54 968, 2019.
- [20] C. Schroeder and H. Rohling, "X-band fmcw radar system with variable chirp duration," in *2010 IEEE Radar Conference*. IEEE, 2010, pp. 1255–1259.
- [21] C. Iovescu and S. Rao, "The fundamentals of millimeter wave sensors," *Texas Instruments, SPYY005*, 2017.
- [22] B.-h. Kim, J.-s. Choi, S.-i. Ko, and M. Park, "Improved active beacon system using multi-modulation of ultrasonic sensors for indoor localization," in *2006 SICE-ICASE International Joint Conference*. IEEE, 2006, pp. 1366–1371.
- [23] E. Odat, J. S. Shamma, and C. Claudel, "Vehicle classification and speed estimation using combined passive infrared/ultrasonic sensors," *IEEE transactions on intelligent transportation systems*, vol. 19, no. 5, pp. 1593–1606, 2017.
- [24] I. AlMohimeed, "Development of wearable ultrasonic sensors for monitoring muscle contraction," Ph.D. dissertation, Carleton University, 2013.

- [25] R. Burnett, “Understanding how ultrasonic sensors work,” Sep 2019. [Online]. Available: <https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm>
- [26] P. Alizadeh, “Object distance measurement using a single camera for robotic applications,” Ph.D. dissertation, Laurentian University of Sudbury, 2015.
- [27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [28] “Understanding the ros computation graph level.” [Online]. Available: [https://subscription.packtpub.com/book/hardware\\_and\\_creative/9781783551798/1/ch01lvl1sec12/understanding-the-ros-computation-graph-level](https://subscription.packtpub.com/book/hardware_and_creative/9781783551798/1/ch01lvl1sec12/understanding-the-ros-computation-graph-level)
- [29] I. Baranov, “Ros 101: Intro to the robot operating system,” Jan 2014. [Online]. Available: <https://www.clearpathrobotics.com/blog/2014/01/how-to-guide-ros-101/>
- [30] N. Buniyamin, W. Ngah, and Z. Mohamad, “Robot global path planning overview and a variation of ant colony system algorithm,” vol. 5, 01 2011.
- [31] G. Grisetti, C. Stachniss, W. Burgard *et al.*, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE transactions on Robotics*, vol. 23, no. 1, p. 34, 2007.
- [32] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, “Robust monte carlo localization for mobile robots,” *Artificial Intelligence*, vol. 128, no. 1, pp. 99 – 141, 2001.
- [33] B. P. Gerkey and K. Konolige, “Planning and control in unstructured terrain,” in *ICRA Workshop on Path Planning on Costmaps*, 2008.
- [34] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
- [35] E. Tsardoulis, L. Petrou, A. Iliakopoulou, and A. Kargakos, “On global path planning for occupancy grid maps,” 09 2013.
- [36] M. Ben-Ari and F. Mondada, *Mapping-Based Navigation*. Cham: Springer International Publishing, 2018, pp. 165–178. [Online]. Available: [https://doi.org/10.1007/978-3-319-62533-1\\_10](https://doi.org/10.1007/978-3-319-62533-1_10)
- [37] M. Słomiany, P. Dąbek, and M. Trojnecki, *Motion Planning and Control of Social Mobile Robot – Part 2. Experimental Research: Progress in Automation, Robotics and Measurement Techniques*, 01 2020, pp. 524–537.
- [38] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

- [39] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, “Efficient trajectory optimization using a sparse model,” in *2013 European Conference on Mobile Robots*. IEEE, 2013, pp. 138–143.
- [40] C. Alia, T. Gilles, T. Reine, and C. Ali, “Local trajectory planning and tracking of autonomous vehicles, using clothoid tentacles method,” in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 674–679.
- [41] “Move base documentation.” [Online]. Available: [http://wiki.ros.org/move\\_base](http://wiki.ros.org/move_base)
- [42] D. V. Lu, D. Hershberger, and W. D. Smart, “Layered costmaps for context-sensitive navigation,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 709–715.
- [43] “2017 awr1642 evaluation module (awr1642boost) single-chip mmwave sensing solution user’s guide.” [Online]. Available: <http://www.ti.com/lit/ug/swru508b/swru508b.pdf>
- [44] “Ti mmwave ros driver setup guide - dev.ti.com.” [Online]. Available: [http://dev.ti.com/tirex/content/mmwave\\_training\\_1\\_6\\_1/labs/lab0006-ros-driver/lab0006\\_ros\\_driver\\_pjt/TI\\_mmWave\\_ROS\\_Driver\\_Setup\\_Guide.pdf](http://dev.ti.com/tirex/content/mmwave_training_1_6_1/labs/lab0006-ros-driver/lab0006_ros_driver_pjt/TI_mmWave_ROS_Driver_Setup_Guide.pdf)
- [45] “mmw demo data structure v0 - ti e2e community.” [Online]. Available: [https://e2e.ti.com/cfs-file/\\_\\_key/communityserver-discussions-components-files/1023/mmw-Demo-Data-Structure\\_5F00\\_8\\_5F00\\_16.pdf](https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/1023/mmw-Demo-Data-Structure_5F00_8_5F00_16.pdf)
- [46] “About - Point Cloud Library (PCL).” [Online]. Available: <http://pointclouds.org/about/>
- [47] E. Pedrosa, A. Pereira, and N. Lau, “A Non-Linear Least Squares Approach to SLAM using a Dynamic Likelihood Field,” *Journal of Intelligent & Robotic Systems*, vol. 93, no. 3-4, pp. 519–532, mar 2019. [Online]. Available: <http://link.springer.com/10.1007/s10846-017-0763-7>
- [48] “Costmap documentation.” [Online]. Available: [http://wiki.ros.org/costmap\\_2d](http://wiki.ros.org/costmap_2d)

## Appendix A

# Simulation of radar range and velocity estimation

### A.1 Simulating range detection

In this section we try to simulate in MATLAB the process of retrieving the range from different targets using the FMCW radar operating principle. . Table A.1 shows the paramaters used for the radar and the vector of distances of the supposed objects to be retrieved, where  $T_c$  is the time of chirp and  $F_s$  is the sampling frequency of the ADC. Figure A.1 shows the

Table A.1: Parameters used for simulation for range estimation.

Paramater	Value
$f_c$	77 GHz
Bandwidth	4 GHz
$T_c$	40us
$F_s$	8 MHz
Object distance	[0.2345 0.0469 1.1 3.0 5 5.3] m

TX signal in blue and RX signal as the other colors in a plot of time vs frequency. Mixing both signals and passing the resulting IF signal by a low pass filter we then compute the FFT of the IF signal. Fig A.2a shows the result. By using equation 2.6 we can retrieve the distance vs power of the signal plot (Fig. A.2b). As we can see the power of the signal is stronger in the distances that were chosen a priori. To extract them we must choose a certain threshold of SNR. Note that the sampling frequency limits the maximum distance of detection. The code used to generate the simulation is shown bellow.

```
1 % RETIOT Range Detection: Simulatio of object range detection
2
3 %% Initialization
4 clear ; close all; clc
5
6 fc=77*10^9;           % 77 GHz
```

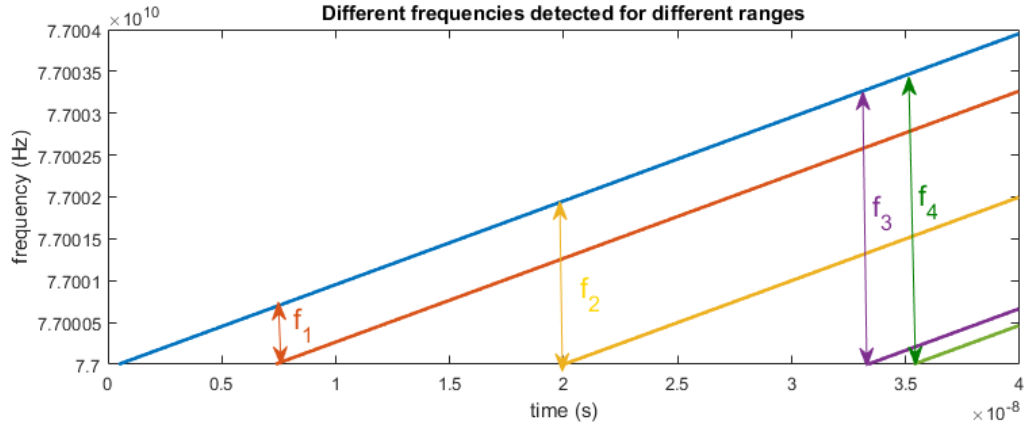
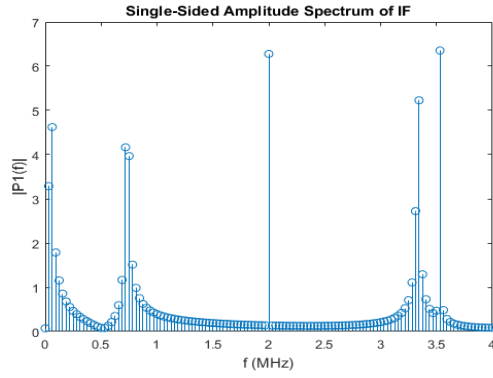
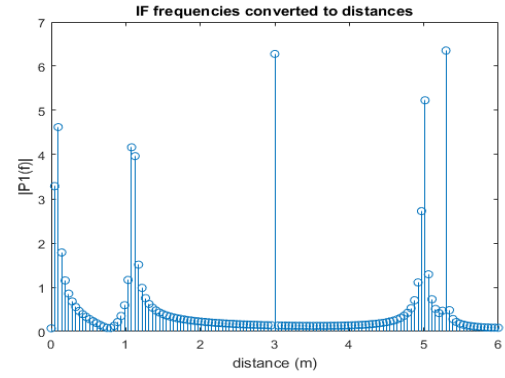


Figure A.1: TX signal in blue and RX signal in other colours, each strip corresponds to a different range detection



(a) FFT of IF signal



(b) Frequencies converted to distances plot

Figure A.2: Range detection using FMCW radar

```

7 BW=4*10^9;           % 4 GHz
8 Tc=40*10^-6;         % 40 us
9 S=BW/Tc;             %
10
11 dist_obj=[0.0469+0.0469/2 1.1 3.0 5 5.3]; %m
12 n_objects=length(dist_obj);
13 c=3*10^8;            %m/s
14 round_trip_delay=2*dist_obj/c; % s
15
16 N=10000000;          % # samples
17 dt=Tc/N;             % Ts
18 t=[0:dt:Tc];         % time vector
19
20 %%
21 f1=fc+S*t;
22 TX_signal=sin(2.*pi.*f1.*t);
23
24 n_zeros=round(round_trip_delay/dt); % # samples

```

```

25 %%
26 plot(t(1:10:10000),f1(1:10:10000));
27 RX_signal=zeros(1,length(TX_signal));
28 for i=1:n_objects
29 tmp=length(TX_signal)-n_zeros(i);
30 f2=[zeros(1,n_zeros(i)) f1(1:tmp)];
31 RX_signal_i=0.1*sin(2.*pi.*f2.*t);
32 RX_signal=RX_signal+RX_signal_i;
33 plot(t(1:10:10000),f2(1:10:10000));
34 hold on;
35 end
36
37 axis([0 4*10^-8 fc fc+4*10^6]);
38 %%
39
40 IF_signal=RX_signal.*TX_signal;
41
42 %%
43 B = fir1(70,0.1,'low');
44 IF_signal_F=filter(B,1,IF_signal);
45 Fs_ADC=8*10^6;
46 Ts_ADC=1/Fs_ADC;
47 n_skip=round(Ts_ADC/dt);
48 tmp=1:n_skip:length(t);
49 IF_signal_sampled=IF_signal_F(tmp);
50 %plot(IF_signal_sampled(1:1000));
51
52 L = length(IF_signal_sampled); % Length of signal
53 N=256;
54
55 F=fft(IF_signal_sampled,N);
56 P2 = abs(F);
57 P1 = P2(1:N/2+1);
58 %P1(2:end-1) = 2*P1(2:end-1);
59
60 f3 = Fs_ADC*(0:(N/2))/N;
61 figure;
62 stem(f3/10^6,P1);
63 title('Single-Sided Amplitude Spectrum of IF')
64 xlabel('f (MHz)')
65 ylabel('|P1(f)|')
66 f3(2)
67 dres=f3(2)*c/(2*S)
68 fmax=Fs_ADC/2
69 dmax=fmax*c/(2*S)
70
71 Threshold=3.5;
72 idx=find(P1>Threshold);
73 number_objects_found=length(idx);
74 for i=1:number_objects_found
75     dcalc_obj(i)=f3(idx(i))*c/(2*S);
76 end

```

```

77 %%
78 dres=f3(2)*c/(2*S);
79 d=[0:dres:dres*length(f3)-dres];
80 figure;
81 stem(d,P1);

```

## A.2 Simulating velocity detection

In this section we try to simulate in MATLAB the process of retrieving the velocity from different targets using the FMCW radar operating principle. Table A.2 shows the parameters used for the radar and the velocity and range of the supposed object to be retrieved, where  $N_c$  is the number of chirps used.

Table A.2: Parameters used for simulation of velocity estimation

Paramater	Value
$f_c$	77 GHz
Bandwidth	4 GHz
$T_c$	40us
$F_s$	8 MHz
$N_c$	64
Object distance	5 m
Radial velocity	7 m/s

By computing the FFT of the  $N_c$  phase differences we get the 2-D doppler FFT for the object at 5 m. Fig A.3a shows the resulting 2D-Doppler FFT that displays which phases are stronger. By using equation 2.7 we can retrieve the velocity vs power of the signal plot (Fig. A.3b). As we can see there is a peak near the 7m/s velocity that means this parameters can

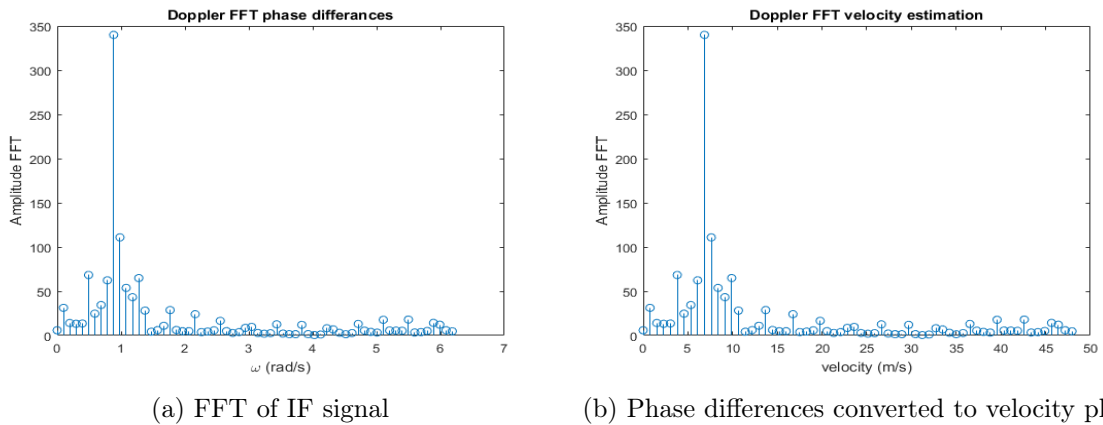


Figure A.3: Velocity detection using FMCW radar

be used to compute the radial velocity of the target with fairly good precision. The code used

to generate the simulation is shown bellow.

```

1  % RETIOT Velocity Detection: Simulation of velocity detection
2
3  %% Initialization
4  clear ; close all; clc
5  %%
6  %
7  %
8
9  fc=77*10^9;           % 77 GHz
10 BW=4*10^9;           % 4 GHz
11 Tc=40*10^-6;         % 40 us
12 S=BW/Tc;             %
13 c=3*10^8;            %m/s
14 N=20000000;          % # samples
15 dt=Tc/N;             % Ts
16 t=[0:dt:Tc];         % time vector
17
18 my_vector=[];
19
20 N_chirps=64;
21 velocityMax=c*pi/(fc*4*pi*Tc);
22 distance_start=5.0;
23 velocity_object=7.0; %m s
24
25 dist_obj(1)=distance_start;
26 for i=2:N_chirps
27     dist_obj(i)=dist_obj(i-1)+velocity_object*Tc;
28 end
29
30 %% N Chirps
31
32 f1=fc+S*t;            %TX signal frequency
33 TX_signal=sin(2.*pi.*f1.*t);
34
35 for i=1:N_chirps
36     round_trip_delay=2*dist_obj(i)/c; % s
37     n_zeros=round(round_trip_delay/dt); % RX signal
38     tmp=length(t)-n_zeros;
39     f2=[zeros(1,n_zeros) f1(1:tmp)];
40     offset=2*pi*fc*dt*n_zeros;
41     RX_signal=0.1*sin(2.*pi.*f2.*t-offset);
42     RX_signal(1:n_zeros)=0;
43     IF_signal=RX_signal.*TX_signal;
44
45     %% IF_signal sampling
46     B = fir1(70,0.1,'low');
47     IF_signal_F=filter(B,1,IF_signal);
48     Fs_ADC=4306*10^4;
49     Ts_ADC=1/Fs_ADC;
50
51     n_skip=round(Ts_ADC/dt); %skip some samples

```

```

52     tmp=1:n_skip:length(t);
53     IF_signal_sampled=IF_signal_F(tmp);
54
55     %% FFT IF signal
56     N=256;
57     F=fft(IF_signal_sampled,N);
58     P2 = abs(F);
59     P1 = P2(1:N/2+1);
60     P2_ang = angle(F);
61     P1_ang = P2_ang(1:N/2+1);
62
63     Threshold=3.5;
64     idx=find(P1>Threshold);
65
66     phasor(i)= F(idx);
67 end
68
69 %% FFT IF signal
70 doppler_FFT=fft(phasor);
71
72 dw=2*pi/length(phasor);
73 w=[0:dw:2*pi-dw];
74
75 stem(w,abs(doppler_FFT));
76 xlabel('\omega (rad/s)');
77 ylabel('Amplitude FFT');
78 title('Doppler FFT phase differances');
79
80 vres=c*dw/(fc*4*pi*Tc)
81 vmax=c*pi/(fc*4*pi*Tc)
82 idx=find(abs(doppler_FFT)==max(abs(doppler_FFT)));
83 v_calc=c*w(idx)/(fc*4*pi*Tc)
84 v=[0:vres:2*vmax-vres];
85
86
87 figure;
88 stem(v,abs(doppler_FFT));
89 xlabel('velocity (m/s)');
90 ylabel('Amplitude FFT');
91 title('Doppler FFT velocity estimation');

```