UNIVERSIDADE D
COIMBRA

Carlos André Seara da Silva

Robot Navigation in Highly-Dynamic Environments

Master's Dissertation in MEEC, supervised by Professor Doctor Lino Marques and Doctor Sedat Dogru and presented to the Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Coimbra

September 2022

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Robot Navigation in Highly-Dynamic Environments

Carlos André Seara da Silva

September of 2022

Robot Navigation in Highly-Dynamic Environments

Supervisors:

Lino José Forte Marques

Sedat Dogru

Jury:

Manuel Marques Crisóstomo

Ana Cristina Barata Pires Lopes

Lino José Forte Marques

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

September of 2022

Acknowledgements

In this short text, I would like to thank everyone who, directly or not, contributed to not only this work, but to the entirety of my academic journey.

First of all, I would like to thank my supervisors, Prof. Dr. Lino Marques and Dr. Sedat Dogru for all their guidance, help and encouragement during this last year, as well as for the opportunity to work on something that interests me so much, as is the area of mobile robotics. A very special thank you to Dr. Sedat Dogru in particular, without whose expertise and willingness to assist, teach and help me in anyway possible, I'm certain this work would not have been concluded.

To my parents and my brother, the biggest thank you possible for all the unconditional support throughout all my life and for giving me the possibility of being here, right now, writing this text. To my brother in particular, I would like to thank for being the best friend I could ask for, and for his willingness to participate in the testing of this work even though it must have been an insufferable process.

I would like to thank all my friends as well for always being there, for the good times, the bad times, and for the times to come. I cherish you all very much.

Finally, I would like to thank my great friend Stitch for his infinite love and appreciation. He will not read this, for he is a dog, but he also is an amazing friend and companion. Thank you, buddy.

Resumo

Os robôs móveis autónomos têm vindo a evoluir ao longo dos anos, tendo progredido de máquinas de propósito único usadas em ambientes isolados e destinadas puramente a robôs, para máquinas de elevada complexidade que coexistem e interagem com os seres humanos nos seus ambientes naturais. Estes robôs têm visto a sua presença aumentar significativamente em várias áreas de aplicação, com especial destaque para a crescente aceitação e importância dos robôs industriais em ambiente fabril e dos robôs de serviços em áreas públicas. Este tipo de ambientes, em que o robô pode vir a encontrar obstáculos em lugares inesperados sendo, portanto, obrigado a re-planear o seu trajeto de acordo com o movimento dos mesmos, são considerados ambientes dinâmicos. É, portanto, importante garantir que os robôs têm a capacidade de navegar de forma segura e sem colisões em ambientes dinâmicos, algo que é possível através dum planeamento de trajeto que leva em consideração o movimento dos obstáculos.

Esta dissertação apresenta uma nova abordagem para a prevenção de colisões com obstáculos dinâmicos, em particular seres humanos, levando em consideração aquelas que, de acordo com o seu movimento, serão as suas posições futuras. A nossa abordagem gera um mapa de grelhas de ocupação, representado como um mapa de custos, de acordo com estas posições futuras, melhorando-o através da aplicação de restrições de índole social e por fim usando o algoritmo A* para encontrar um caminho sem colisões e socialmente aceitável. O uso do algoritmo A* não é vinculativo, podendo ser utilizados outros algoritmos de planeamento de caminhos. Neste método levamos em consideração o movimento dos obstáculos para prever a área que o robô deve evitar num futuro próximo de modo a não colidir com um obstáculo, ao invés de assumir um ambiente estático. A nossa abordagem também incorpora a incerteza associada à estimativa do movimento dos obstáculos nos custos atribuídos às celulas do mapa de custos. Asseguramos ainda que os caminhos planeados não causam embaraço ao movimento dos obstáculos, levando o robot a descrever uma trajetória mais aceitável do ponto de vista social através da inflação do custo das células do mapa de custos

pertencentes à área situada ao longo da direção de movimento do obstáculo.

A solução proposta foi validada através de simulações e de experiências usando o hardware real, isto é, uma plataforma robótica. Em ambos os casos, foram testados cenários nos quais a ocorrência de colisões seria certa senão pela capacidade do robô de as evitar. Os resultados obtidos demonstram que a aplicação de estratégias de atribuição de custos que não levem em consideração o movimento dos obstáculos não são viáveis para a navegação em ambientes dinâmicos. Em forte contraste, os testes realizados a capacidade da nossa abordagem de, na maioria dos casos, evitar colisões e levar o robô a executar trajetórias socialmente mais aceitáveis, passando por trás dos obstáculos quando adequado.

Abstract

Autonomous mobile robots have evolved from single purpose machines used in isolated, custom built robot-specific environments, to complex multi-purpose machines interacting with humans in their natural environments. The presence of such robots has also increased in many application areas, with industrial robots in factories and service robots in public areas becoming more and more important and accepted. These environments, where the robot may find obstacles in unexpected locations and need to replan its path in accordance with their motion, are considered dynamic environments. Is is, therefore, important to assure that robots are capable of navigating in a safe and collision-free manner in dynamic environments through adequate path planning that takes into consideration the motion of the obstacles.

This dissertation presents a novel dynamic obstacle avoidance approach, aiming to avoid particularly humans, taking into account their future poses as predicted by their motion. The approach generates an occupancy map, represented as a costmap, in accordance with the future poses, improves it with social constraints, and uses A* to find a collision free and socially acceptable path. However, the approach is not limited to A*, it can use other path planning algorithms when required. We take into account the motion of the obstacles to predict the area that the robot should avoid going through in the future in order to prevent a collision, instead of assuming a static environment. Our approach also incorporates the uncertainty in the estimate of the obstacles' motion into the costs assigned to the cost-map. We ensure that the generated paths do not hinder the obstacle's motion and lead the robot through a socially more acceptable trajectory by inflating the costs of the costmap cells located along the obstacle's moving direction.

The proposed solution was validated through both simulation and real-world experiments where collisions would be bound to happen unless the robot had the ability to avoid them. Results show that cost assignment techniques that do not account for the motion of the obstacles are not reliable for navigation in dynamic environments. In contrast, our experiments

revealed our approach to be able to, on most cases, avoid collisions and lead the robot to perform more socially acceptable trajectories by passing the obstacles through their backs when appropriate.

Contents

List of Acronyms

ABC Artificial Bee Colony.

ACO Ant Colony Optimization.

AMCL Adaptive Monte Carlo Localization.

APF Artificial Potential Field.

BIT Batch Informed Trees.

DBSCAN Density Based Spatial Clustering of Applications with Noise.

DEEC Departamento de Engenharia Eletrotécnica e de Computadores.

DWA Dynamic Window Approach.

EA Evolutionary Algorithms.

GA Genetic Algorithms.

LiDAR Light Detection and Ranging.

MCL Monte Carlo Localization.

MSE Mean Squared Error.

OMPL The Open Motion Planning Library.

PRM Probabilistic Roadmap.

PSO Particle Swarm Optimization.

ROS Robot Operating System.

RRT Rapidly-exploring Random Tree.

SLAM Simultaneous Localization and Mapping.

VO Velocity Obstacle.

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

Autonomous mobile robots have evolved from single purpose machines used in isolated, custom built robot-specific environments, to complex multi-purpose machines interacting with humans in their natural environments [1]. The presence of such robots has also increased in many application areas, with industrial robots in factories and service robots in public areas becoming more and more important and accepted. It is, therefore, important to assure that robots are capable of navigating in a safe and collision-free manner through adequate path planning in dynamic environments [2].

Common robot navigation in dynamic environments usually does not take into account the motion of the obstacles over time, as the term *dynamic obstacles* is frequently used to refer to unforeseen obstacles and not necessarily moving ones. This can lead to erratic navigational behavior, because the robot must change its planned path frequently due to the assumption that the obstacles are not moving [1]. That is the case with, for example, the approach proposed by Ghorbani et al. [3], as well as with Zeng et al. [4], and with the solution for dynamic environment navigation introduced by Hossain and Ferdous [5]. There are also some approaches that, while explicitly considering the obstacle's motion, work mainly on a local scale. That is the case, for example, with the modified Dynamic Window Approach [6]. Although this allows the robot to navigate safely, these approaches' reactive rather than predictive nature can once again lead to inconsistent planning due to their limited scope [7]. Therefore, theese approaches are more adequate for navigation in unknown environments than for situations where the robot has information about the structure of the environment, as can often be assumed to be the case in indoor navigation.

1.2 Goals and Contributions

The main goal of this work is the development of a robot navigation strategy tailored for use in dynamic environments. We propose a novel dynamic obstacle avoidance approach, aiming to avoid particularly humans, taking into account their future poses as predicted by their motion. The approach generates an occupancy map, represented as a costmap, in accordance with the future poses, improves it with social constraints, and uses A* to find a collision free and socially acceptable path. However, the approach is not limited to A*, it can use other path planning algorithms when required. We take into account the motion of the obstacles to predict the area that the robot should avoid going through in the future in order to prevent a collision, instead of assuming a static environment. Our approach also incorporates the uncertainty in the estimate of the obstacles' motion into the costs assigned to the cost-map. We ensure that the generated paths do not hinder the obstacle's motion and lead the robot through a socially more acceptable trajectory by inflating the costs of the costmap cells located along the obstacle's moving direction.

This work was developed in the scope of the UltraBot project, giving origin to an article [8] to be published in the Proceedings of the Fifth Iberian Robotics Conference (ROBOT2022).

1.3 Document Overview

The structure of this dissertation and content of each chapter are the following:

- **Chapter 2** presents the reader with some essential concepts for understanding the robot navigation problem, introduces the Robot Operating System (ROS) and performs a review of the existing works on robot navigation in dynamic environments;
- **Chapter 3** presents and explains our approach to costmap construction, starting with the perception-related processes executed in order to extract meaningful information for the cost assignment process;
- **Chapter 4** contains experimental results, obtained both through simulations and real-world experiments. A discussion on these results and their meaning is also held in this chapter;
- **Chapter 5** reflects upon the overall results of this work and proposes some improvements to be made in future work.

2 Background

In this chapter we present the basic concepts associated with robot navigation and how interconnected they are in that process, introducing some seminal works focused on each of them. The reader is also introduced to the Robot Operating System (ROS) and given a brief explanation of its building blocks and utilities. Finally, a discussion regarding the existing approaches to robot navigation in dynamic environments is presented as well, substantiated with brief analyses of some important works on this topic.

2.1 Robot Navigation: Basic concepts

Autonomous navigation refers to the manner in which a robot finds its way in the environment it is integrated in, relying solely on its decisions and not on the intervention or control of an external agent, such as a human [9]. This decision process is controlled by the *a priori* knowledge the robot may have regarding the environment in the form a map, the sensory information it acquires through a perception process, depending as well on the goal position or series of positions it is desired to reach. [10].

The navigation problem can then be generally summarized into four different subtasks [11] (Fig. 2.1) [12]:

- **Localization** - Answers the "where am I?" question, i.e., determines the robot's pose relative to the environment;
- **Mapping** - Answers the "What does the world look like?" question [13], i.e., provides the robot with a useful representation of its surroundings;
- **Path planning** - Answers the "how should I get there?" question, i.e., what is the most adequate path that allows the robot to reach its goal;
- **Motion control** - Moves the robot along the calculated path, i.e., generates velocity commands that allow it to achieve the desired path.

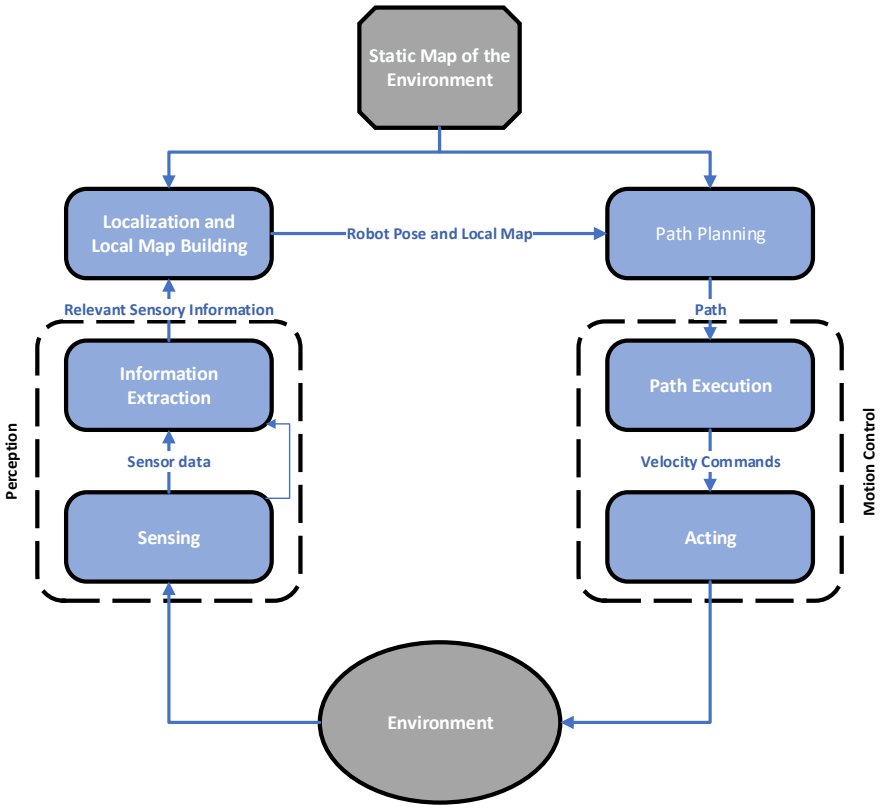


Figure 2.1: Flow diagram of the robot navigation loop with a static global map of the environment

It is clear that the three first elements require the robot to have the ability to acquire meaningful information regarding its surroundings in real-time. This process of acquisition is named *perception* and comprises one of the backbones of *autonomous navigation*.

In this section we present and explore these key components, starting with *perception*, and show their interconnection in the navigation process. A larger emphasis is given to *path planning* because it is the element most directly connected to the subject of this dissertation.

2.1.1 Perception

Perception can be defined as the ability of a system to acquire knowledge about its environment [10]. This is achieved through the use of sensors for data acquisition, which then requires further processing to be transformed into useful information.

Sensors can be, in a simplistic manner, classified as either *proprioceptive* or *exteroceptive*, depending on whether they acquire data relative to internal state of the robot or the surrounding environment [10].

Although proprioceptive sensors, such as wheel encoders, are useful for the *localization*

task, navigation depends mostly on information regarding the robot's surroundings, which enables the system to react in a timely and adequate fashion to occurrences in the environment, e.g. the appearance of unforeseen obstacles. It is, therefore, very common for mobile robots to be equipped with laser rangefinder devices [14] or with vision sensors, such as depth cameras [15, 16].

2.1.2 Mapping

Mapping can be defined as the process in which a robot uses the information collected in the perception process to enhance its knowledge of the surrounding world through the construction of a *map*, i.e., a physical model of its environment [17]. This process represents a challenge due to a number of factors, the most notable of which are [18]:

- **Size** - The larger the environment in comparison to the robot's perceptual range the more difficult it becomes to build an accurate and coherent map;
- **Noise in perception and actuation** - The noisier the sensors and the less accurate the actuators, the more difficult mapping becomes as the sources of information regarding the environment and the robot's movement are less trustworthy;
- **Perceptual ambiguity** - The more homogeneous an environment is, the harder it becomes to differentiate between different locations;
- **Closing the loop** - Loop closure refers to situations where the robot may have followed different paths leading to the same initial point, which can erroneously lead to two different maps corresponding to the same physical area.

One of the most popular environment representation methods are occupancy grid maps (Fig. 2.2). In an occupancy grid map the environment is tesselated into cells, creating a discrete approximation of space where each cell stores a probabilistic estimate of its state [19], i.e, the probability of being occupied.

In a way, the mapping process happens in every iteration of the navigation loop, using the data obtained in the perception process, from which results a dynamic map of the robot's immediate surroundings, i.e, a *local map*. However, it is also useful for the robot to keep a larger scope record of the environment, particularly of its static elements, e.g. its structure. Such a map can be termed a *global map* and only needs to be acquired once due to its static nature.

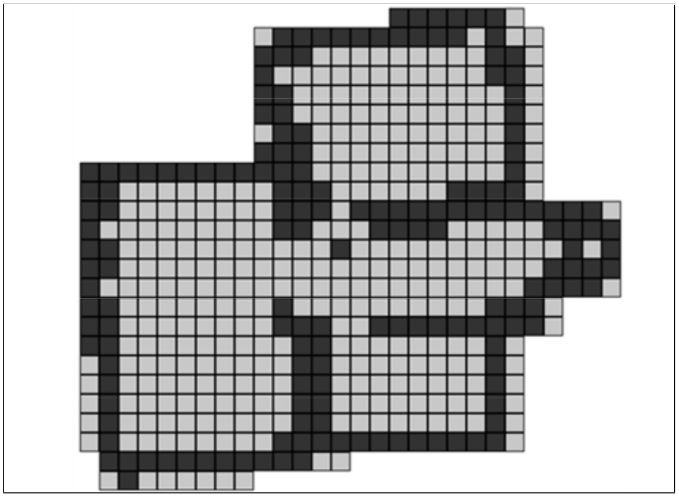


Figure 2.2: Example of an occupancy grid map. Black cells represent occupied space, grey cells represent free space and the white area represents unexplored space. Taken from [20] © 2010 IEEE

A very widely adopted approach to global map building is SLAM. SLAM stands for Simultaneous Localization and Mapping and, as the name suggests, allows the robot to construct a map of its environment while simultaneously localizing itself in relation to it, thus making SLAM a significantly more difficult challenge than simply localizing the robot in relation to a known map or mapping with known poses [18]. There exist SLAM approaches for the construction of both 2D [21] and 3D [22] maps using laser rangefinder sensing devices, as well as using vision sensors, in what is typically called Visual SLAM [23] [24].

2.1.3 Localization

The localization problem refers to the robot's ability to determine its pose in the environment at a given moment in time in relation to some form of environmental representation, i.e., a map [18]. It is one of the key elements of robot navigation, as it becomes much more difficult for the robot to traverse safely in its environment if it does not accurately know where it is, making it also harder for the system to reach its goal positions, given that it has no way of accurately comparing its current pose to the desired one.

The localization process is, in its different approaches, supported by both the knowledge of the environment the robot may have and the information gathered by its sensors. Sensors may be exteroceptive, acquiring information regarding the environment, or proprioceptive, which gather data regarding the internal state of the such robot, such as wheel encoders, that allow the robot to determine an estimate of how much it has moved, i.e, perform odometry. Odometry can also be performed using other type of exteroceptive sensory devices, such as vision sensors, in what is usually designated visual odometry [25] [26].

One of the most widely used methods for mobile robot localization [27] is Monte Carlo Localization [28]. MCL uses a particle filter [29] to estimate the robot's pose,where each particle corresponds to a possible pose. As the robot moves, the particles are updated to reflect the new robot's state prediction. Eventually, they should converge to the system's correct position and orientation [18].

2.1.4 Path planning

Path planning is concerned with finding a path that will lead the robot to its target. Typically, there will be a multitude of paths that allow the robot to achieve its goals. However, a path planner will look for a path that is optimal according to some criterion, such as distance, safety, or computation time [9].

Path planning approaches can be divided into two categories, based on the level of knowledge regarding the environment they require or take into account: global path planning and local path planning [30].

Global path planning techniques use a map of the environment, typically one that has static elements regarding the environment's structure and is periodically updated with newly discovered information such as dynamic obstacles. This map is used to compute a complete path from the robot's starting point to its goal position [2]. Thanks to its knowledge of the environment, the robot can assure, depending on the global planning method, the optimality of the generated path in accordance to established metrics, although the search for an optimal path undoubtedly increases the computational complexity of the process [31].

Global path planning algorithms can be split into three categories [2]:

- **Graph search-based algorithms** - Graph search-based path planning algorithms transform the path planning problem into a graph search one, in a two-step process: first, a connectivity graph is built based on the environment's map; then, a graph search is performed, where the computation of an optimal solution is executed [10]. They give guarantees in terms of finding a path if one exists, i.e. are *complete*, and of path optimality, assuming that the available map has an adequate resolution. This need for high enough map resolution comprises one of the downsides of graph search-based algorithms, given that their ability to generate a path and its quality are highly reliant on the map's resolution. Besides, their performance degrades in high dimensional environments [32]. Some of the most notorious graph search-based path planning approaches include Dijkstra's algorithm [33] and A* [34];

- **Sampling-based algorithms** - Sampling-based path planning algorithms attempt to capture the connectivity of the environment through sampling, i.e., only a subset of points and their connectivity is evaluated, in contrast to graph search-based approaches, where all possibilities are assessed. This kind of approach is advantageous in the sense that it allows for fast performance even in difficult path planning problems; However, they give no guarantee regarding the optimality of the generated path or of even actually finding a path if one exists. This means that they are not necessarily complete, at least if not given sufficent runtime, which may in some cases be infinite [32]. Some of the more widely used sample-based planning algorithms are [2, 32] are Probabilistic Roadmap (PRM) [35, 36, 37], Rapidly-exploring Random Tree (RRT) [38] and its variants [39] and Batch Informed Trees (BIT) [40] and its evolutions [41];

- **Evolutionary algorithms** - Evolutionary Algorithms (EA) are population-based algorithms that mimic or use mechanisms involved in natural evolution for approximation and optimization of a solution [42]. A *population* is composed of *individuals*, which are possible solutions for the problem at hand and which are evaluated on their *fitness* according to a quality function. As in natural evolution, fitter *individuals* are more likely to survive and are therefore preferred by EA [43]. These *individuals* become the seed for the next generation of the population, undergoing a number of modification operations that mimic genetic changes [43], such as recombination and mutation [44]. These new candidates are evaluated through the aforementioned quality function and compete with the old ones based on their fitness. This process is iterated until a solution with the desired quality is found or a computational limit is reached [44]. EAs typically employed in global path planning include [2] Genetic Algorithms (GA) [45], Ant Colony Optimization (ACO) [46], Artificial Bee Colony (ABC) [47] and Particle Swarm Optimization (PSO) [48].

Local path planning techniques, as the name suggests, work on a more local scale, both in spatial and temporal terms, when compared to global planners, meaning that they may only use a fraction of the map or even rely solely on the sensory information gathered during navigation for its planning process [9, 31]. They can, therefore, be considered more reactive approaches in comparison to the more deliberative character associated with global planning. Furthermore, the fact that the planning is done at a smaller scope decreases the computational complexity of the process, thus improving its computational performance.

These sort of approaches often bridge the gap between path planning and motion control [2], meaning that the planning may be done directly in the velocity space rather than through the use of a map as is the case with global planning, which once again highlights the short temporal horizon considered in the planning process.

There is a wide variety of local path planning methods, which makes them harder to categorize as was done with the global planning techniques. In light of this, the categorization of the most popular methods will be done as classic approaches and newer ones, such as soft computing based approaches.

Some of the classic approaches [49] to local path planning include Artificial Potential Field (APF) [50, 51] based approaches, methods that apply the Velocity Obstacle (VO) [52, 53, 54] and ones based on the Dynamic Window Approach (DWA) [55, 6, 56]:

- **Artificial Potential Field** - The APF approach is essentially a reactive strategy, in which the system's goal is assigned an attractive potential, while obstacles have a repulsive potential. The sum of all forces allows the robot to determine the moving direction and velocity to adopt.

- **Velocity Obstacle** - The VO concept is used in velocity space based planning. It consists in establishing a forbidden velocity cone related to a certain obstacle, which in a given instant contain all the velocities that, if adopted by the robot, would lead to a collision with said obstacle, taking into account its velocity profile. Choosing a velocity outside that cone allows the robot to avoid the collision.

- **Dynamic Window Approach** - DWA involves the sampling of possible angular and linear robot velocities. The trajectories resulting from the adoptance of these velocities over a short interval of time are determined and rated based on, for example, their safety and proximity to the goal. The velocities leading to the highest rating trajectory are then adopted by the robot.

There has also been a strong focus in recent years in research on the application of soft computing techniques to local path planning [57], particularly concerning the use of fuzzy logic [58] and neural networks [59] based approaches.

2.1.5 Motion Control

Robot motion control is related to the actual execution of a path or plan by a robot. That means, it consists on the translation of a path or instruction to velocity commands

that the robot can execute. This process depends on the model that governs the robot's motion, i.e. differential drive model [60], and the kinematic constraints it may be subject to, based on which a robot is generally classified as *holonomic* or *non-holonomic*, depending on whether or not it can move freely in every one of the physical dimensions it operates in.

As explained in 2.1.4, the matter of motion control is usually directly addressed in local path planning techniques.

2.1.6 Costmaps

Costmaps are a type of grid-based environment representation typically used in robot navigation which can be seen as an evolution from the previously presented occupation grid maps [61], where instead of probabilities of occupation, cells store traversal cost values. While costmaps can be used solely for identifying unknown, free and occupied spaces, they allow for far richer cost assignments representing navigational soft constraints [61] to be made, such as regarding the traversability of the terrain [62] or social constraints [63, 1].

Using a single, monolithic costmap can, however, lead to some information loss regarding the origin of the data, which can create problems in the cost update process. Consider, for example, a robot equipped with two 2D LiDAR sensors placed at different heights which also possesses a static occupation grid map of the environment. Situations may arise where an obstacle of limited height, e.g. a vase, is present at a location marked as free on the static map, and that is only detected by one of the sensors due to the difference of height in their placement. This could potentially lead an occupied area to be marked as free on the costmap and vice versa.

Lu et al. [61] proposed a layered approach to costmap construction (Fig. 2.3) to address the shortcomings of monolithic costmaps. Each layer handles the data relative to a specific functionality, e.g social constraints, or to specific information sources, e.g a static map. The data for each of the layers is accumulated into the master costmap, in an ordered update process, i.e., the order and way in which the layers are superimposed onto each other is arbitrarily defined.

2.2 ROS and its Navigation Stack

Robot Operating System (ROS) is a flexible and widely used open-source framework for the development of robot software. It offers a ready-to-use collection of tools, libraries and

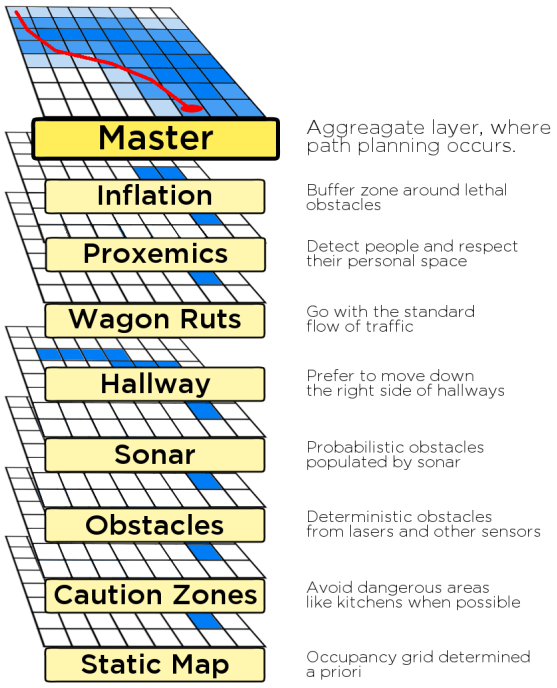


Figure 2.3: Example of a layered costmap. Each layer is responsible for a different functionality, allowing for a set of different contextual behaviors to be achieved. Taken from [61] © 2014 IEEE

conventions that aim to simplify the creation of varied, complex and robust robot behaviors and its replication across a wide variety of robotic platforms [64] allowing the developer to abstract away from the specificities of the hardware. The fundamental building blocks of ROS are *nodes*, *messages*, *topics* and *services*.

Nodes are processes that perform computation tasks [65]. ROS is designed with system modularity in mind, meaning that a single system is typically comprised of many nodes. These nodes are contained in *packages*, and while a *package* can be composed of multiple nodes, there is no obligation of executing all of a *package's* nodes simultaneously.

Nodes communicate with each other through *messages*. ROS provides a large number of message types that are adequate for the different types of information a node may be trying to transmit, also allowing for the creation of custom message types. Messages can also be composed of other messages and arrays of other messages [65].

Nodes don't communicate with each other directly. Rather, they publish their messages to *topics*, in what assumes the form of broadcast communication. The nodes interested in the information published in a certain topic will subscribe to it, without a limit to how many topics a node can simultaneously subscribe to. There may also be multiple concurrent publishers for a single topic, although, generally, publishers and subscribers are not aware of each others' existence [65].

Although the topic communication model is quite flexible, it is not appropriate for syn-

chronous transactions that certain nodes may require. To address this, ROS introduces *services*. *Services* can be considered very simple nodes whose execution can be called upon by other nodes. They follow a strict structure, being defined by a string name and a pair of messages: one for the request and one for the response. Unlike what happens with topics, where multiple nodes may publish information to the topics, only one node can advertise a service with a particular name, i.e., there cannot be two nodes providing two homonymous services.

One of the most enticing features of ROS is the collection of nodes it offers for robot navigation, aptly named the *Navigation Stack*[1] (Fig. 2.4). In very broad terms, the *Navigation Stack*'s role is to generate safe and feasible control inputs for a mobile robot given a goal pose by processing data from various sources, such as odometry, sensors and a map of the environment [66]. Therefore, this requires nodes that can handle each of elements of robot navigation presented in 2.1.

For *localization*, the most widely used package is **amcl**[2] (Adaptive Monte Carlo Localization), which is an implementation of a variation of the Monte Carlo Localization introduced in 2.1.3.

There are as well multiple packages for *mapping* that implement different SLAM algorithms, such as **gmapping**[3] and **slam_karto**[4].

ROS *Navigation Stack* employs both a global path planner and a local path planner. The global planner generates a path to the goal, while the local planner is responsible for the actual execution of said plan, therefore assuming more the role of a motion controller than a path planner. Among the available global planners for ROS are **carrot_planner**[5] and **global_planner**[6], which has implementations of both A* and Dijkstra's algorithm. Libraries such as **OMPL**[7] also offer a multitude of implementations of sampling-based global planning algorithms. As for local planners, or, perhaps, more correctly termed *motion controllers*, some of the available packages for ROS are **base_local_planner**[8] and **dwa_local_planner**[9], which is an implementation of the previously discussed Dynamic Window Approach.

[1] http://wiki.ros.org/navigation
[2] http://wiki.ros.org/amcl
[3] http://wiki.ros.org/gmapping
[4] http://wiki.ros.org/slam_karto
[5] http://wiki.ros.org/carrot_planner
[6] http://wiki.ros.org/global_planner
[7] https://ompl.kavrakilab.org
[8] http://wiki.ros.org/base_local_planner
[9] http://wiki.ros.org/dwa_local_planner

The *Navigation Stack* employs *layered costmaps* to guide the path and motion planning processes. Some of the available layers include the **static layer**, which handles the costs related with the static map of the environment, the **obstacle layer**, which processes sensory information to mark the locations of detected obstacles in the costmap, and the **inflation layer**, whose role is to create new costs around the obstacles which allow the costmaps to account for the robot's dimensions. These costmaps are handled by **costmap_2D**[10] package.

The functioning of all these parts is orchestrated by the **move_base**[11] package, an element that is essentially the brain of the *Navigation Stack*.

ROS also provides tools for data visualization and troubleshooting, such as **rviz**[12], **rqt_tf_tree**[13] and **rqt_graph**[14]. **rviz** is a 3D visualization tool that, in a way, allows the user to see the robot's perception of its environment. It can reproduce a lot of different information, such as the environment's map, a 3D model of the robot and it's pose, as well as the sensory information captured by the system, displaying data originated from, for example, cameras and LiDARs. **rqt_tf_tree** provides a graphic interface that allows the user to visualize the interconnections between the different frames of reference present in the robot, such as a sensor's reference frame and the robot's own frame of reference, being particularly useful for troubleshooting. **rqt_graph** presents a graph-like representation of the currently running nodes and the interconnections they may establish through the *topics* they publish or subscribe to, which can be convenient for troubleshooting.

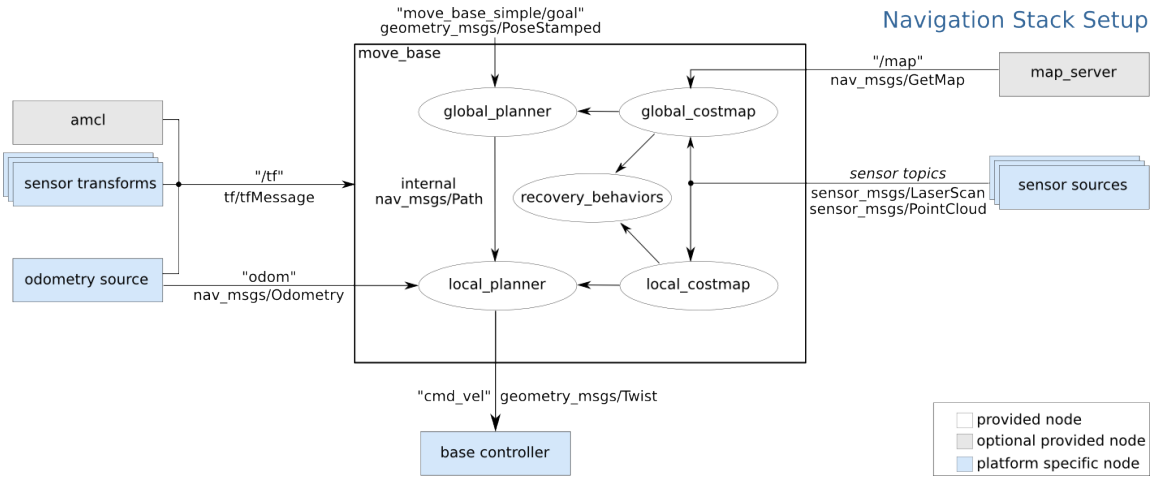


Figure 2.4: Structure of the ROS *Navigation Stack*. Taken from [67]

[10]http://wiki.ros.org/costmap_2d
[11]http://wiki.ros.org/move_base
[12]http://wiki.ros.org/rviz
[13]http://wiki.ros.org/rqt_tf_tree
[14]http://wiki.ros.org/rqt_graph

2.3 Robot Navigation in Dynamic Environments

Out of all the concepts related to robot navigation presented in 2.1, it is clear that path planning is the one that requires special care in order to accomodate the challenges that come with navigating in an environment where the obstacles may be moving.

There are many approaches to robot path planning in dynamic environments. Nevertheless, plenty of them do not account for the motion of obstacles over time, treating them as static entities. This leads to inconsistent and incoherent behavior from the robot when faced with moving obstacles, e.g. people. Therefore, it is important to add a predictive component to the path planning process.

The velocity obstacles concept presented by Fiorini et al [52] proposes a method of planning in the velocity space that accounts for the relative velocity of the robot and the obstacle to establish a forbidden velocity cone containing the set of robot velocities that would lead to a collision. Van Den Berg et al. [53] extended this work through a generalization of the VO concept which addresses the difficulty of applying the Velocity Obstacle approach to kinematically constrained robots. However, both of these approaches, and eventually other approaches based on planning in velocity space, find their collision avoidance capabilities constrained by the short-time planning horizon inherent to motion control when compared to path planning based techniques.

There has been strong interest in the past few years in research on the application of soft computing techniques to robot path planning [57], particularly using fuzzy logic and neural networks. Nasrinahar et al. [58] proposed a fuzzy logic based planning approach that effectively seeks to differentiate static and dynamic obstacles, establishing different behavior rules for each. However, there appears to be no actual integration of data regarding the obstacle's speed into the behavior decision process. Furthermore, rule-based planning methods can somewhat limit the robot's ability to react to dynamic obstacles due to the finite set of behavioral rules it adheres to, which coupled with the fact that the obstacle's motion is not taken into consideration, can cause some inconsistent behavior in the robot. Singh et al. [59] present a neural network based path planning solution that explicitly addresses the matter of moving obstacles. In each iteration the robot's workspace is divided into 5 equal segments each of 30°with a predefined length. They determine whether or not each segment is blocked by an obstacle or how long it would take for an obstacle to enter the area covered by the segment. This time (which is 0 if the segment is blocked by a static obstacle), calculated for each of the segments, is input to a neural network, which will select

the segment the robot should progress through, as well as the speed to be adopted. As with the approaches previously presented, there is the downside that collision avoidance is achieved by changing the speeds rather planning a full path, as well as the time and the data that is necessary to train the network.

In the field of social robotics there are some very interesting approaches to be considered as well. Kollmitz et al. [1] introduce a layered cost-map based approach to socially aware robotic navigation, which uses a social cost model to represent the social preferences of humans based on a Gaussian distribution. They have a static layer where costs related to the environment (e.g. the structure of the environment) are assigned and multiple dynamic layers where the assignment of costs is done accordingly to the previously mentioned probability distribution. Each dynamic layer represents one instance of the predicted human trajectory from time step i until step (i + 1). This approach provides interesting results, however, large memory requirements arise from large maps, fine space or time resolutions or a large prediction look-ahead. Yang et al. [68] propose in their work a formulation of a Gaussian function that models the individual space around a character in a virtual environment that can be applied to robot planning in the presence of moving obstacles. It enlarges the character's footprint according to its status, particularly when it is moving, as there is an inflation of the character's individual space along its moving direction.

3 Methods

3.1 Overview

The cost-map based navigation approach proposed in this dissertation focuses on dynamic obstacles and can be integrated with most of the existing path planning techniques that rely on cost-maps. We take into account the motion of the obstacles, assuming a linear motion model, as well as their footprints, which are approximated by a circle, to predict the area that the robot should avoid going through in order to prevent a collision. We also incorporate the uncertainty in the estimate of the obstacles' motion into the cost assignment process, while ensuring that the generated paths do not hinder the obstacles' motion, conditioning them into assuming a more socially acceptable character through the inflation of the cost-map cells located along the obstacles' moving direction.

This method requires detection, tracking and knowledge of the obstacle's position, velocity and footprint radius. To obtain this information, we also present a perception method tailored for use with a 2D LiDAR.

The solution developed in this work can be, therefore, divided into two blocks/stages:

- **Perception block** - Given the 2D LiDAR readings and a 2D Occupation Grid map of the environment, performs obstacle detection, tracking and velocity and radius estimation;

- **Cost-map update block** - Given the obstacle's position, velocity and radius, predicts the area that the robot should avoid going through in order to prevent a collision and marks it in the cost-map.

These two blocks are executed sequentially on each iteration of the method (Fig. 3.1).

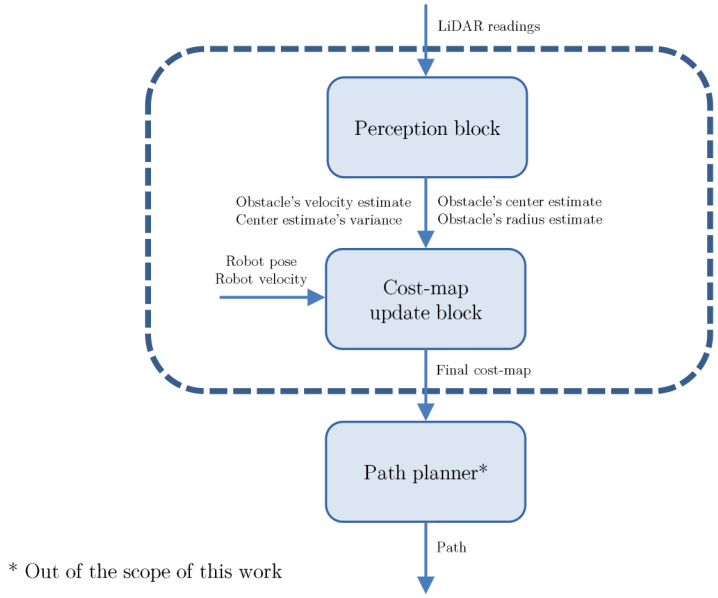


Figure 3.1: Block diagram describing one iteration of the proposed approach

3.2 Perception block

This block is responsible for detecting the obstacles and estimating their positions, velocity and radius (Fig. 3.2). This is achieved through the use of the Kalman filter [69], where the measurements correspond to obstacles' position and radius determined purely through the combined use of the LiDAR readings and the environment's occupation grid map that, as previously mentioned, is presumed to be available. As explained in 3.1, the obstacles are assumed to move according to a linear motion model, i.e., with constant velocity.

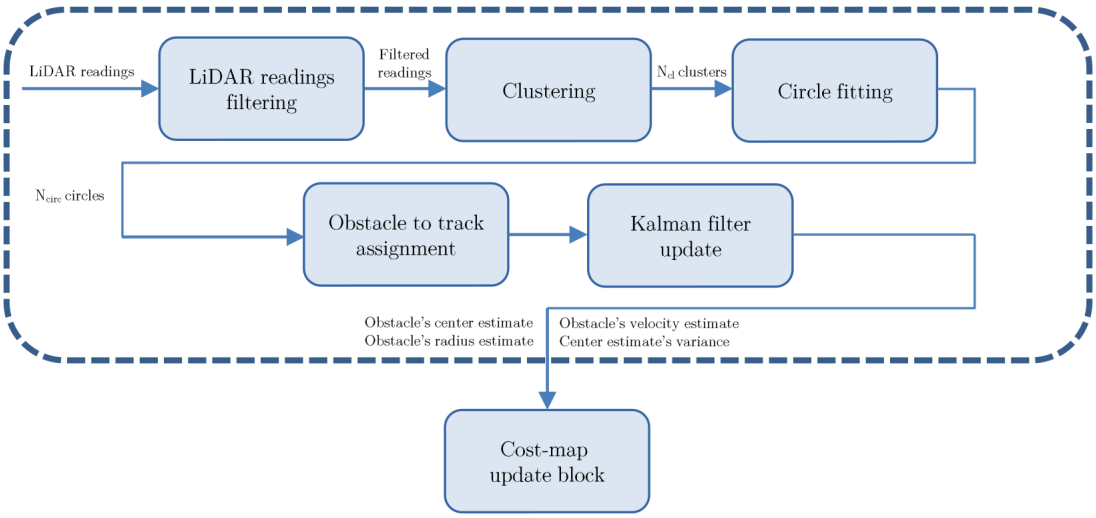


Figure 3.2: Block diagram describing one iteration of the perception block

3.2.1 LiDAR readings filtering

A 2D LiDAR sensor returns a set of N readings at a frequency f, where the value of N is dictated by the sensor's aperture angle and angular resolution and f is an intrinsic parameter of the sensor, which is also characterized by its minimum and maximum range. Each individual reading is represented in polar coordinates and so is composed of an angle ϕ and a distance ρ. ϕ represents the angle formed between the line segment that connects the sensor to the point in the real world that the reading corresponds to and the LiDAR's frame of reference X axis, while ρ is the length of said line segment (Fig. 3.3).

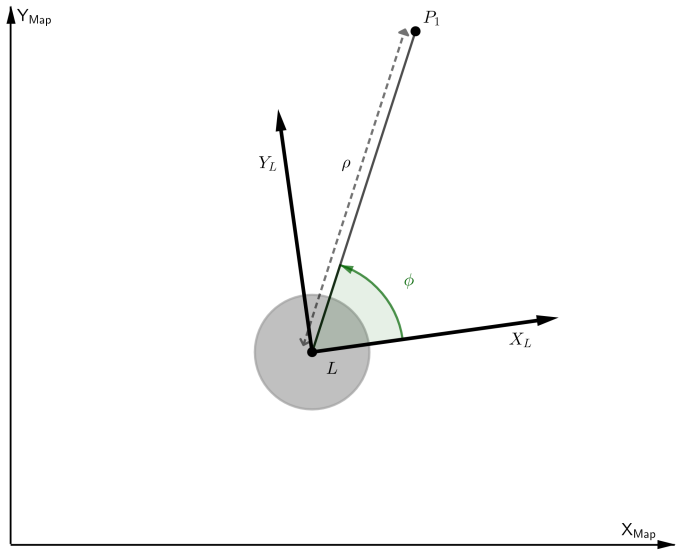


Figure 3.3: A diagram showing a LiDAR reading. L is the sensor's position (mounted on the robot) and the grey circle represents the robot's footprint. X_L and Y_L comprise the LiDAR's frame of reference, while P_1 is the point to which the reading corresponds to.

Not all the of the readings returned by the sensor are necessarily valid ones. For example, when a reading corresponds to a point situated further away than the LiDAR's maximum range, its ρ may assume an invalid value or a value larger than the maximum range. In these cases, the reading must be discarded. An analogous situation occurs when the reading corresponds to a point whose distance to the sensor is smaller than its minimum range. This is the first and simplest stage of filtering, after which what remains is a set of N_v readings.

Given that an occupation grid of the environment is available, we can analyze it in order to find out what readings concern parts of the environment, e.g. walls, and subsequently discard them, and what readings are useful and may correspond to dynamic obstacles. To achieve this it is crucial to have both the map and the LiDAR data in the same frame of reference, which will be the map's The first step in this process is to convert the LiDAR readings from polar to cartesian coordinates (in the LiDAR's reference frame). This is a

very simple procedure, which requires but the direct application of the following equations:

$$^{L}P_{i_x} = \rho_i \cos \phi_i \tag{3.1}$$

$$^{L}P_{i_y} = \rho_i \sin \phi_i \tag{3.2}$$

where $i = 1, 2, ..., N_v$. Transforming these points from the sensor's reference frame to the map's coordinate system implies the application of a transformation matrix ${}^{M}_{L}T$, which essentially represents the map's coordinate system in the sensor's frame of reference:

$$^{M}P_i = {}^{M}_{L}T \cdot {}^{L}P_i, \;\; i = 1, 2, ..., N_v \tag{3.3}$$

Each of the points $^{M}P_i$ is then individually evaluated using the occupation grid map. A square, centered on the map cell to which the currently analyzed point $^{M}P_i$ belongs to, is established. If any cell belonging to said square is marked as occupied, the point $^{M}P_i$ is discarded. This procedure allows accounting for possible small errors in the robot's localization, in the map and in the LiDAR readings, all of which are rather natural and bound to happen. The length of the square's side is a key parameter in this process. A very small value may be unable to deal with inaccuracies in the aforementioned factors, while a very large one will render the filtering process ineffective, because too many points will be discarded. An obvious case of this would be the detection of an obstacle close to a wall. The readings corresponding to it would be thrown away due to their proximity to the wall and the excessive square side length. At the end of this procedure we are left with a final set of N_f points, where $0 \leq N_f \leq N_v$.

3.2.2 Clustering

The previously filtered points must be clustered in order to be interpreted as parts of individual entities, i.e., obstacles. To achieve this we decided to apply the DBSCAN [70] algorithm, as we concluded that its characteristics, such as little sensitivity to noise/outliers in the data, affordable computational complexity and, most importantly, the fact that it doesn't require previous knowledge of the number of clusters in the data [71], made it an adequate choice for the task at hand.

DBSCAN algorithm is a density-based clustering algorithm, meaning that it attempts to group together segments of 2D data based on the proximity between a minimum number of points according to an arbitrarily defined distance measure, such as the Euclidean distance

[72] we adopt in this work. These two factors are actually the only two parameters of the algorithm:

- ***MinPts*** - The minimum number of points required to form a cluster;
- $\boldsymbol{\mathcal{E}}$ - The maximum allowed distance between two points for them to be considered neighbors and, therefore, possibly part of the same cluster.

These two parameters allow the classification of the analyzed points into three different categories:

- ***Core points*** - Points that have $MinPts$ or more points situated at a distance equal to or less than $\mathcal{E}$;
- ***Border points*** - Points that can be reached from a core point but have less than $MinPts$ situated at a distance equal to or less than $\mathcal{E}$;
- ***Noise*** - Points that cannot be reached from a core point.

In each iteration of the algorithm, the first step is randomly selecting a point A which hasn't previously been analyzed. If said point is a *core point*, then a cluster is formed containing this point's neighbors and, if they too are *core points*, their neighbors as well. In case the selected point is not a *core point*, then it is labeled as *noise*, although it can later become a *border point* if it is found to be reachable from a *core point*. At the end of this process we will have N_{cl} clusters, each one considered to be a dynamic obstacle. All the points classified as *noise* are eliminated.

The choice of the $MinPts$ and $\mathcal{E}$ parameters is highly dependent on the context of application. In the case of our work, given that we intend to identify moving obstacles which are quite likely to be human beings, the height at which the 2D LiDAR is placed is extremely important. If the sensor is relatively close to the floor, the readings obtained will correspond to a person's legs. This means that there will be a small number of points that belong to a possible cluster, requiring a lower value for $MinPts$. Furthermore, it would not be recommendable to set a very small value for $\mathcal{E}$, because that could lead to two different clusters relative to the same person, one corresponding to each of the person's legs. If the LiDAR is placed at a person's torso height, then these parameters aren't so critical to the success of the algorithm. There will be a larger number of readings that correspond to the person to be detected, leading to a wider range of adequate values for $MinPts$. These readings would also be closer together than in the previously discussed scenario, diminishing the strictness in the choice of $\mathcal{E}$.

3.2.3 Circle fitting

The clusters obtained in the previous step will have an arbitrary shape. For example, if the target is a person directly facing the robot, then the LiDAR readings that regard that person will, assuming that the sensor is placed at torso height, very likely form a line segment or at best an arc. This means that the resulting clusters' shapes may not necessarily reflect the obstacle's real morphology or be able to properly convey its dimensions.

To overcome this, we chose to approximate the obstacle's footprint by a circle. For this purpose, we use Coope's circle fitting method [73]. Coope transforms the nonlinear circle fitting problem into a linear least squares problem through variable substitution, thus reducing the mathematical complexity of determining a solution and increasing its robustness to outliers when compared to a nonlinear least squares approach.

Consider the following equation of a circle:

$$(x - c_x)^2 + (y - c_y)^2 = r^2 \tag{3.4}$$

where (x, y) are the coordinates of a point belonging to said circle, and (c_x, c_y) are the coordinates of its center. This can be rewritten as:

$$x^2 + y^2 = 2xc_x + 2yc_y + r^2 - c_x^2 - c_y^2 \tag{3.5}$$

Applying the following variable substitution to 3.5

$$\begin{aligned} A &= 2c_x \\ B &= 2c_y \\ C &= r^2 - c_x^2 - c_y^2 \end{aligned} \tag{3.6}$$

we get the following formulation

$$Ax + By + C = x^2 + y^2 \tag{3.7}$$

Applying this to all the points we wish to fit a circle to, which in our case are the points belonging to a cluster, (c_x, c_y) and r can be determined through linear least squares approximation. This procedure must be applied individually to each of the previously defined N_{cl} clusters.

After this procedure, and in order to account for a possible failure in the filtering process presented in 3.2.1, the resulting circles are first evaluated in terms of their dimensions. It is quite unlikely that the robot will encounter a dynamic obstacle with very large or very small dimensions, at least in an indoor environment. It is then plausible to assume that both those cases are result of a failure in the LiDAR reading filtering process. To solve this we apply both a minimum, r_{min}, and a maximum, r_{max}, radius threshold to the obtained circles. Any circle whose radius lies outside this range is discarded.

The circles that remain after the radius based filtering are then subject to another stage of evaluation focused on the mean squared error between the distance from the estimated circle center to the cluster points and the estimated circle radius. If this MSE is larger than a preset MSE_{thresh} threshold, the circle is discarded. This stage of filtering is effective in situations where a cluster erroneously corresponds to an environmental element, e.g. a wall, whose shape could hardly be approximated by a circle in an accurate manner. At the end of this stage we will have N_{circ} obstacles, $0 \leq N_{circ} \leq N_{cl}$.

3.2.4 Obstacle to track assignment

Having identified the dynamic obstacles present in the environment at a certain moment, it is crucial to be able to match them with the ones detected in previous iterations. In this process, each obstacle is analyzed individually, and therefore it must be executed for all of the obstacles detected in an iteration.Let i denote the perception block's current iteration and $i-1$ the previous one. In keeping with this notation, $N_{circ_{i-1}}$ denotes the number of obstacles detected in the previous iteration, $i-1$, while N_{circ_i} refers to the number of obstacles detected in the current iteration i.

The procedure starts with the calculation of the distance of the present iteration's currently analyzed obstacle O_j^i, $j = 1, 2, ..., N_{circ_i}$ to all of the obstacles detected in the previous iteration $i-1$ that haven't been matched yet. An initial correspondence is established between the $i-1$ iteration obstacle O_k^{i-1}, $k = 1, 2, ..., N_{circ_{i-1}}$ whose distance to O_j^i is minimal. Therefore, it is assumed that O_h^{i-1} and O_j^h are the same obstacle, which will be referred to by only O_j^i. Given that the perception cycle is executed at the same frequency f at which the LiDAR readings are returned, it is not plausible that an obstacle traveled a significant distance in the interval between sensor updates. Therefore, the initial match is only kept if the distance traveled by obstacle O_j^i over successive iterations $i-1$ and i, $d_{O^i_{j_{i-1,i}}}$, is inferior to a preset threshold d_{max}. Previous iteration obstacles that are matched with current

iteration ones are flagged as so.

A buffer is maintained for storing the obstacles that aren't matched for a maximum of 5 iterations. These are then reused in the tracking process as part of the previous iteration obstacle group until they are matched or deleted from the buffer.

3.2.5 Kalman filter update

Each of the $N_{circles}$ detected obstacles will keep a Kalman filter used for velocity estimation, as well as for refining both its radius and position estimation, which can fluctuate due to the lack of stability associated with the LiDAR readings over time (e.g, the number of readings corresponding to a certain obstacle may vary, which in turn may cause the circle fitting algorithm to estimate a different center and radius for the same obstacle in different iterations).

The Kalman filter is a recursive algorithm used for estimating the state of a process or system, usually modeled in state-space format, given the measurements of its states obtained over time [74]. It allows to reduce the uncertainty associated with the states' estimates through a weighted fusion of the information given by system's model and the obtained measurements.

The process model describes the evolution of the system's states from time step $k-1$ to time step k and is governed by the linear stochastic difference equation

$$x_k = A_{k-1}x_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.8}$$

where A_{k-1} is the state transition matrix at time step $k-1$, which relates the state at $k-1$ to the state at step k, B is the control-input matrix, which relates the control input u to the state x and w_{k-1} is the process noise vector, characterized by a zero-mean normal distribution, $w_{k-1} \sim \mathcal{N}(0, Q)$.

The measurement vector z_k contains measurements of the system's states at time step k and is characterized by

$$z_k = Hx_k + v_k \tag{3.9}$$

where H is the measurement matrix, relating the states to the measurement vector z_k, and v_k is the measurement noise vector, which follows a zero-mean normal distribution, $v_k \sim \mathcal{N}(0, R)$, independent from that of process noise's probability distribution.

It must be noted that although the covariance matrices Q and R should reflect the statis-

tical properties of the process and measurement noise respectively, in real world applications these matrices are sometimes only partially known or even not known at all. In some cases, they may be difficult to determine because the noises may not follow a Gaussian distribution. Therefore, they must be approximated and tuned by the user based on their knowledge and confidence in the process model and in the measurements fed to the filter.

The Kalman filter can then be divided into two stages: prediction and correction. In the prediction stage, the Kalman filter uses the previous time step's $k-1$ state and error covariance estimates to determine an *a priori* estimate of both elements for the current time step k, as shown in the following set of equations

$$\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^- + Bu_{k-1} \tag{3.10}$$

$$P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + Q \tag{3.11}$$

where $\hat{x}_k^-$ and P_k^- are, respectively, the *a priori* state and error covariance estimates.

In the correction stage, the Kalman filter uses feedback in the form of the measurement vector z_k to refine the *a priori* estimates. This process is guided by the confidence attributed to the process model and the measurements, which is reflected in the predefined Q and R matrices, respectively. The equations for this stage are

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3.12}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k) \tag{3.13}$$

$$P_k = (I - K_k H)P_k^- \tag{3.14}$$

where K_k is the Kalman gain and $\hat{x}_k$ and P_k are the *a posteriori* estimates of the state and error covariance respectively, all relative to time step k.

Given our goal of estimating the obstacle's velocity, as well as improving its position and radius estimate, the state vector x for an obstacle A is

$$x = \begin{bmatrix} P_A^T & v_A^T & r_A \end{bmatrix}^T \tag{3.15}$$

where $P_A = [P_{A_x}, P_{A_y}]^T$ is the obstacle's position and is equal to $[c_x, c_y]^T$, $v_A = [v_{A_x}, v_{A_y}]^T$ is its velocity vector and r_A the estimated radius.

Assuming, as mentioned in 3.1 a linear motion model for the obstacles, i.e. that they

move with a constant velocity vector, the state transition matrix A_{k-1} assumes the form

$$A_{k-1} = \begin{bmatrix} 1 & 0 & \Delta_t & 0 & 0 \\ 0 & 1 & 0 & \Delta_t & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

where Δ_t is the time interval that elapsed from time step $k-1$ to k and depends on the frequency f of the LiDAR, which typically assumes a value of 10 Hz or higher.

In our case, B is naturally 0. Since the only states we have measures of are the position and the radius, H assumes the following form

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

Matrix Q is a diagonal matrix, i.e., it is assumed that the noises affecting the states are statistically uncorrelated between themselves, and the same applies to matrix R.

$$Q = \begin{bmatrix} \sigma_{P_x}^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{P_y}^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{v_x}^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{v_y}^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_r^2 \end{bmatrix} \tag{3.18}$$

$$R = \begin{bmatrix} \sigma_{z_{P_x}}^2 & 0 & 0 \\ 0 & \sigma_{z_{P_y}}^2 & 0 \\ 0 & 0 & \sigma_{z_r}^2 \end{bmatrix} \tag{3.19}$$

3.3 Cost-map update block

This block, which is summarized in Fig. 3.4, starts by estimating the possible future collision time. It then uses the estimates of the obstacle's dimensions and velocity together with a social constraint to update the costmap, which eventually may be used by, for example, an A* planner to find the shortest path. The costmap update process is iterative. It is executed at a frequency *f*, high enough to ensure that constant velocity assumption is valid for each iteration.

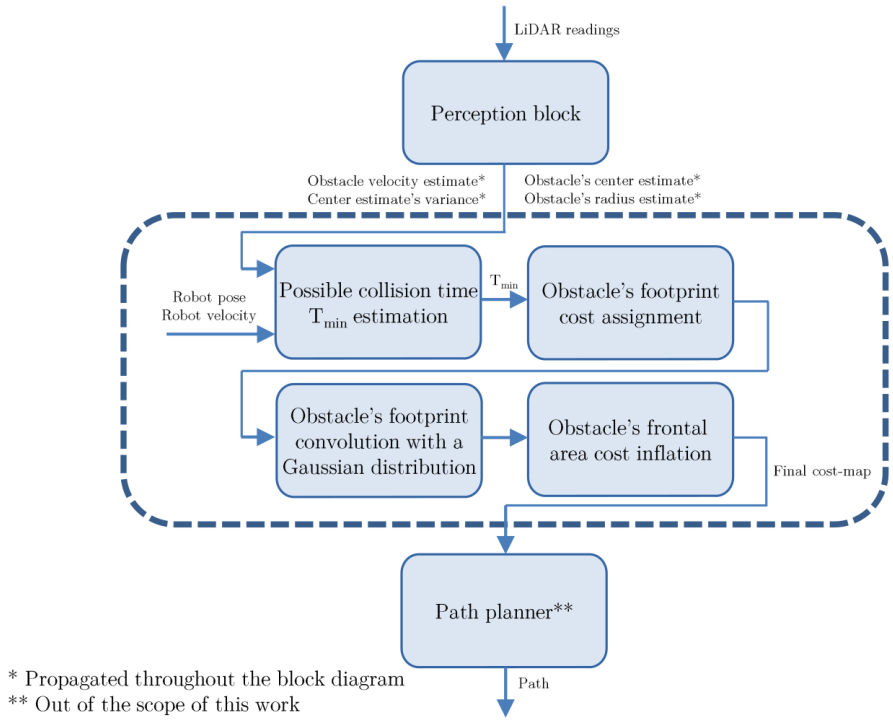


Figure 3.4: Block diagram describing one iteration of the cost-map update process

3.3.1 Determining possible collision time

Let agent A be the robot and agent B be a dynamic obstacle, eventually a human. Let $\vec{P}_k(t)$ and $\vec{v}_k$ respectively denote the 2D position and speed of agent k at time t. Therefore, assuming constant speed for a small duration of time $1/f$, the positions of the robot and the dynamic obstacle at time t_1, which is close to t_0, can be estimated with the following equations:

$$\vec{P}_A(t_1) = \vec{P}_A(t_0) + (t_1 - t_0) \cdot \vec{v}_A \tag{3.20}$$

$$\vec{P}_B(t_1) = \vec{P}_B(t_0) + (t_1 - t_0) \cdot \vec{v}_B \tag{3.21}$$

Taking into account that the agents have a non-zero footprint, collision between agents A and B means their trajectories get closer than a threshold in the future. This can be determined by checking the evolution of the Euclidean distance, $d_{\text{AB}}(t) = \|\vec{P}_A - \vec{P}_B\|$, between the two. Therefore, given equations (3.20) and (3.21), one can determine the instant t from an initial time t_0 when the two agents will be at their closest, denoted by t_{min}, by solving $\frac{\partial d_{AB}(t)}{\partial t} = 0$ for t, which leads to the following equation:

$$t_{min} = -\frac{(P_{A_x}(t_0) - P_{B_x}(t_0)) \cdot (v_{A_x} - v_{B_x}) + (P_{A_y}(t_0) - P_{B_y}(t_0)) \cdot (v_{A_y} - v_{B_y})}{(v_{A_x} - v_{B_x})^2 + (v_{A_y} - v_{B_y})^2} \tag{3.22}$$

The previously presented condition does not ensure that there will, in fact, occur a collision between the robot and the obstacle at time $t_{\min}$, but, in any case, allows the robot to keep its distance from the obstacles found during navigation. Situations may arise where $t_{\min}$ does not assume valid values (e.g. when A and B are moving away from each other, $t_{\min}$ will be negative). In those cases the obstacle is to be represented in the cost-map at its current position, so we set $t_{\min}$ to 0. It is also desirable to define a temporal threshold on $t_{\min}$ so as to limit how far into the future we are willing to look for possible collisions. We set this threshold to 10 seconds.

Generally, t_{min} will be much larger than $1/f$. However, t_{min} is prone to change when $\vec{v}_A$ or $\vec{v}_B$ change, therefore it is calculated at a constant rate f, to ensure a cost assignment that is consistent with the most recent and only information available regarding the state of the agents.

3.3.2 Cost assignments in the costmap

The cost assignment process takes into consideration an estimate of the obstacle's position and associated uncertainty, as well as its size and social constraints. In light of this, we divide the cost assignment into three different stages.

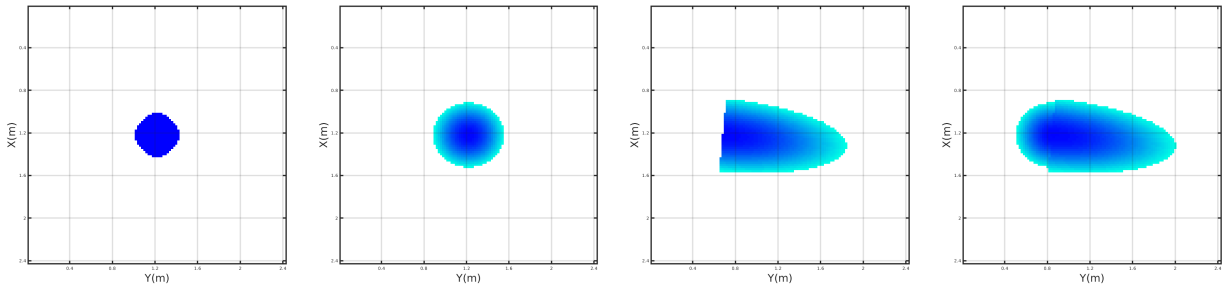
1) Obstacle footprint cost assignment: With $t_{\min}$ found using (3.22), $\overrightarrow{P_{\mathrm{B}}}(t_{\min}))$ can be determined using (3.21). Then, a circle centered on $\overrightarrow{P_{\mathrm{B}}}(t_{\min})$ with the estimated obstacle radius is marked on the cost-map, by assigning the maximum cost possible to all of the cells that belong to it (Fig. 3.5a).

2) Enlarging the obstacle's footprint using a Gaussian distribution: Let $\sigma^2_{P_{B_x}}$ and $\sigma^2_{P_{B_y}}$ denote the variances of the x and y components of the obstacle's current position estimate. A Gaussian probability distribution of $\overrightarrow{P_{\mathrm{B}}}(t_{\min})$ can be established with the following probability density function:

$$f(x,y) = \exp\left(- \frac{[x - P_{B_x}(t_{min})]^2}{2\sigma^2_{P_{B_x}}} - \frac{[y - P_{B_y}(t_{min})]^2}{2\sigma^2_{P_{B_y}}} \right) \tag{3.23}$$

where (x,y) are the coordinates of a point C in the map reference frame. This Gaussian distribution is convolved with the obstacle's footprint, leading to a cost-map that conveys the uncertainty associated with the estimated $\overrightarrow{P_{\mathrm{B}}}(t_{\min})$ (Fig. 3.5b).

3) Cost inflation of the obstacle's frontal area: The area immediately in front of a moving



(a) **(b)** **(c)** **(d)**

Figure 3.5: Evolution of the cost-map of the obstacle using the proposed approach (a) Obstacle's footprint. (b) Obstacle's footprint after applying the Gaussian in equation 3.23. (c) Obstacle's footprint after inflating the frontal area (equation 3.25) (d) The final costmap assignment, resulting from fusion of b and c.

obstacle should be avoided both to reduce the awkward feeling it may cause in a human, and to avoid the possibility of the human reacting to an obstacle passing through the front. So no paths planned by the robot should cross that region. A way to force the robot's path planner to not consider this forbidden space in its planning process is to assign its cells a high cost in the cost-map. To attain this, we use a Gaussian function based on the formulation introduced by Yang et al. [68], with the difference that we modify the base (preset) variances based on the estimated velocity of the obstacle rather than a fixed value, in an approach very much alike the one adopted by De Rose et al. [75]. This only applies to situations where a collision is bound to happen. Otherwise, the planned path will be whatever the planner generates, without considering whether to pass through the front or the back of the obstacle.

Consider a point C in the map reference frame, whose displacement vector from $\overrightarrow{P_{\mathrm{B}}}(t_{\min})$ is $\overrightarrow{BC} = (BC_x, BC_y)$ (Fig. 3.6). It is easily perceivable that if the absolute value of the angle θ formed between the $\overrightarrow{BC}$ and $\vec{v}_B$ is equal to or less than 90°, i.e., $\cos\theta \geq 0$, then C is situated in the obstacle's frontal area. The value of $\cos\theta$ can be determined through the concept of vector dot product, leading to the following equation:

$$\cos\theta = \frac{v_{B_x} \cdot BC_x + v_{B_y} \cdot BC_y}{\|\vec{v}_B\| \cdot \|\overrightarrow{BC}\|} \tag{3.24}$$

The formulation adopted for the Gaussian that returns the cost to be assigned to an arbitrary point C is the following:

$$f(C, P_B(t_{min})) = A \exp\left(-\frac{\left(\|\overrightarrow{BC}\| \cdot \cos\theta\right)^2}{2\sigma_x^2} - \frac{\left(\|\overrightarrow{BC}\| \cdot \sin\theta\right)^2}{2\sigma_y^2}\right) \tag{3.25}$$

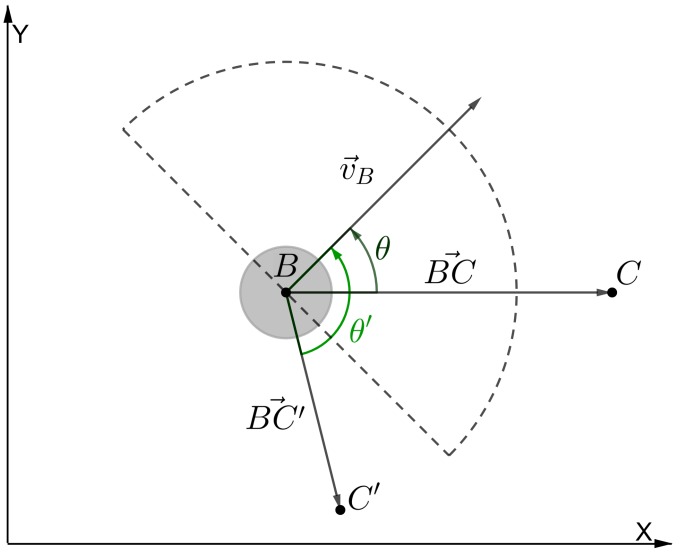


Figure 3.6: A diagram showing the frontal area of the dynamic obstacle. B is the obstacle's position and the grey circle represents its footprint. The frontal area corresponds to the dashed semicircle, whose radius would theoretically be infinite. Point C is situated in the obstacle's frontal area, while C' is not

where A is an amplitude parameter that corresponds to the maximum possible cost on the costmap. σ_x^2 and σ_y^2 are the diagonal entries of the Gaussian's covariance matrix Σ and essentially control the shape formed by the inflated costmap cells in the context of a local reference frame centered at $\overrightarrow{P_{\mathrm{B}}}(t_{\min})$ whose X-axis is oriented in the direction of $\vec{v}_B$ and whose Y-axis can be determined through the application of the right hand rule, knowing that its Z-axis points outwards from the costmap.

The modification of σ_x^2 and σ_y^2 is guided by the following heuristic:

$$\begin{cases} \sigma_x^2 = (1+r)\sigma_{x_{base}}^2 \\ \sigma_y^2 = (1-\frac{r}{2})\sigma_{y_{base}}^2 \end{cases} \tag{3.26}$$

where $\sigma_{x_{base}}^2$ and $\sigma_{y_{base}}^2$ are preset values that help define the shape assumed by the costmap markings and r is a ratio between the magnitude of $\vec{v}_B$ and a predefined maximum speed, $v_{\max}$, that allows control of the dimensions of the forbidden area according to the magnitude of obstacle's velocity:

$$r = \frac{\|\vec{v}_B\|}{v_{\max}} \tag{3.27}$$

$\sigma_{x_{base}}^2$ should be set to a significantly larger value than $\sigma_{y_{base}}^2$ in order achieve higher costs in the cells situated in the direction of the obstacle's motion.

3.4 Implementation

The methods presented in this chapter were implemented for use with the ROS Noetic distribution[1]. The costmap update block is integrated into ROS *Navigation Stack* in the form of a costmap layer plugin for the **costmap_2d** package. The perception block is implemented as a standalone node, which subscribes the topic to which the LiDAR readings are published to and then publishes the information relative to the detected obstacles to the **/Obstacles** topic. In order to be able to publish the obstacle information to the **/Obstacles** topic two different message types were created:

- **obstacle** - This message carries information regarding an obstacle, namely its current position estimate and associated variance, radius, velocity estimate and an unique ID;
- **obstacleArray** - This message is comprised of an array of **obstacle** messages.

For every other aspect of robot navigation we rely on the default ROS nodes mentioned in 2.2. It should, however, be noted that we do not use the **move_base** package, but rather the **move_base_flex**[2] package, which is an enhanced and more flexible version of **move_base**. One of the biggest advantages of the **move_base_flex** package over **move_base** is the power it gives to the user for controlling the navigation process. In particular, it allows the user to trigger the global path planning process when desired, instead of relying on a fixed-rate planning. This factor led to the decision of using the **move_base_flex** package, because being able to replan when convenient would allow us to mitigate "indecisiveness" in the planned paths, as well as reduce computational load. In order to determine when a path would need to be replanned we developed the **plannerTrigger** node. This node evaluates the number of the detected obstacles in each **obstacleArray** message, as well as their predicted future positions. If the number of detected obstacles changes or if the predicted future position for one obstacle changes more than a preset distance threshold, $change_{max}$, the global planner is triggered. The $change_{max}$ parameter is extremely important, because if it assumes a very small value the path will constantly be replanned and if it assumes a very large value the path won't be replanned as frequently as it should. Through multiple experiments we found that the ideal value for this parameter is 0.2 m.

[1] http://wiki.ros.org/noetic
[2] http://wiki.ros.org/move_base_flex

4 Experimental Work and Discussion of Results

4.1 Metrics

In order to evaluate our work and compare its performance to that of other approaches, some clear and relevant metrics must be defined. We use three different metrics for this purpose:

- ***Occurrence of collision*** *(Coll.)* - Whether or not the robot collided with any of the obstacles. The most critical of the three adopted metrics;
- ***Minimum distance to obstacles*** *(d_{min})* - The minimum distance, in meters, that separated the center of the robot from the center of any of the obstacles present in the scenario during its traversal from its starting position to its goal. The second most important metric, as the robot passing too close to a person will reduce the human's sense of comfort;
- ***Elapsed Time*** (T) - The time, in seconds, it took for the robot to go from its starting position to its goal (not considering the in-place rotation performed by the robot after achieving the goal position in order to attain its goal pose). The least important of three metrics here presented, but still relevant as it is not desirable for the robot to take extremely long to reach its goals.

4.2 Simulations

In this section we present and analyze some simulation results obtained using the proposed approach on three different scenarios with a growing difficulty level. These results are compared to the robot behavior observed when using the ROS *Navigation Stack* default

configuration (using ROS Obstacle Layer), which in every iteration of the costmap update process assumes obstacle staticity, i.e, the obstacles are marked in the costmap exactly where they are in each instant without considering their motion.

4.2.1 Setup

All simulations were conducted using the open-source simulation platform Gazebo and ROS Noetic. As previously explained, our approach is integrated into the ROS *Navigation Stack* in the form of a costmap plugin. Additionally, we use the default Global Planner (using the A* algorithm) and the DWA Local Planner.

The robotic platform adopted for these tests was the Pioneer 3-DX, a differential drive robot with an approximate radius of 20 cm. It is equipped with a 2D LiDAR device, with a range of 10 m, an aperture angle of 260° and a resolution of 1°.

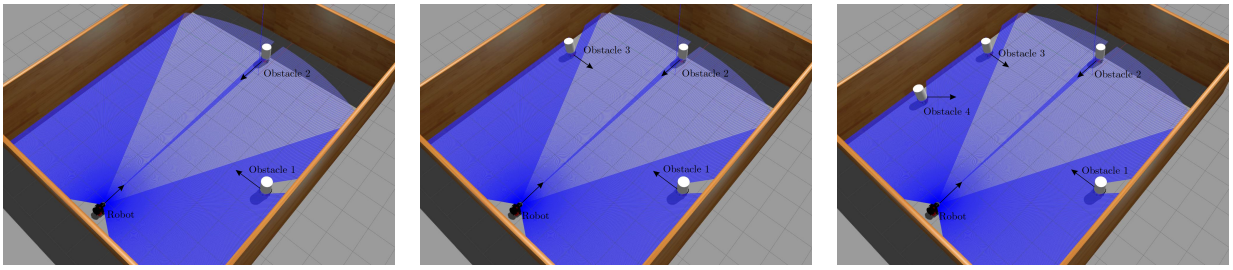
Three scenarios are considered (Fig. 4.1), with the number of dynamic obstacles performing pre-recorded straight-line trajectories without any sort of obstacle avoidance, i.e., acting as dynamic obstacles, increasing from the first to the third scenario. In the first scenario, the robot faces two dynamic obstacles. The number of obstacles increases to three in the second scenario and to four in the third scenario, with each obstacle moving at an average speed of approximately 0.76 m/s and reaching a top speed of 0.85 m/s. Across all scenarios, the point the robot tries to reach is the starting position of obstacle 2, which is presented in Fig. 4.1.

All three scenarios were tested 15 times using both the proposed approach and the one adopted by ROS *Navigation Stack* default configuration, in order to obtain a set of results that cover a wide array of possible outcomes. When applying our approach, global path planning was triggered by changes in the obstacles' predicted future poses as explained in 3.4. With the *Navigation Stack* default configuration (using ROS Obstacle Layer), the planning was triggered at a constant rate of 8 Hz.

4.2.2 Results and Discussion

First Scenario

The results presented in Table 4.1 reveal that the adoption of a costmap construction strategy where the motion profile of the dynamic obstacles is not taken into account is hardly adequate for a scenario bearing any resemblance to what a robot may face in the real



(a) First scenario **(b)** Second scenario **(c)** Third scenario

Figure 4.1: The three testing scenarios used in the simulations. The arrows indicate the direction of motion of the obstacles.

world. Even though this scenario is not particularly complex or challenging, the usage of ROS standard approach to costmap construction led to a 100% collision rate on the 15 tests executed for this scenario, while our approach assured that the robot never crashed into any obstacle.

However, looking at the minimum distances registered in each execution present in 4.1a, and considering that the footprint of the robot platform used can be approximated by a circle with a radius of roughly 0.2 m, one can conclude that our approach does not always keep a distance from obstacles that can be considered socially comfortable or acceptable. This is particularly critical in the eleventh test, where the robot passed an obstacle at a center-to-center distance of 0.4145 m, meaning that a collision was extremely close to happening. These instances where the performance is weaker are mostly due to the perception part of the method proposed in this dissertation. Particularly, inaccuracies in the velocity estimates of the obstacles can cause significant changes in their predicted critical future poses, which in turn leads to excessive path replanning. These constant changes in the path to follow cause the robot to not actively avoid the obstacles and show some "indecisiveness" until the planned path stabilizes. However, when that happens, the distance of the robot to an obstacle may already be smaller than what would be required for the creation of a path with the desired properties. That is the case with test number 11. In terms of the desired behavior, that is, the robot's ability to perform a socially acceptable navigation by not passing through the front of obstacle 1, which moves perpendicularly to the robot, our approach produced to positive results. Out of all the tests performed in this scenario, only in the first scenario did the robot pass through the front of obstacle 1, although keeping a safe distance thanks to the inflation of the costs of the cells located in the moving obstacle's frontal area. In every other instance, the robot passed obstacle 1 through its back and then passed obstacle 2 through the right.

In figure 4.2 we present the key frames of run number 10 using our approach. Figures 4.2a

and 4.2e portray the beginning of the simulation scenario. The path planned (the green line in fig. 4.2e) assumes a shape close to a straight line because the obstacles are not yet moving. In the phase of the simulation presented in figures 4.2b and 4.2f, the velocity estimates of the moving obstacles have become stable and accurate enough, as visible in the costmap cost assignment (Fig. 4.2f) relative to each of the two obstacles. This allows the path planned to both avoid the obstacles and pass through the back of obstacle 1, as desired. In figures 4.2c and 4.2g, the path is slightly readjusted due to some changes in the predicted future critical poses of obstacles 1 and 2, but largely maintains its previous shape. Finally, in figures 4.2d and 4.2h the robot has already successfully avoided the moving obstacles and is free to head toward its goal.

Run	$T(s)$	$d_{min}(m)$	$Collision$
1	13.738	0.5289	
2	15.260	0.5736	
3	16.239	0.8270	
4	14.748	0.8363	
5	13.615	1.0078	
6	14.401	0.6501	
7	17.253	0.7750	
8	17.008	0.9096	
9	16.355	0.6075	
10	15.248	1.0032	
11	14.504	0.4145	
12	16.385	1.0249	
13	15.893	0.8918	
14	14.110	1.1551	
15	14.767	0.5535	
Average	15.368	0.7839	—
σ	1.256	0.2194	—
Coll. %	—	—	0

(a) Results using the proposed approach

Run	$T(s)$	$d_{min}(m)$	$Collision$
1	18.912	0.3628	**X**
2	20.365	0.3598	**X**
3	19.612	0.3615	**X**
4	41.850	0.3481	**X**
5	44.294	0.3238	**X**
6	34.996	0.3516	**X**
7	21.094	0.3576	**X**
8	20.860	0.3569	**X**
9	20.356	0.3720	**X**
10	21.413	0.3628	**X**
11	20.854	0.3348	**X**
12	19.601	1.3660	**X**
13	33.982	0.3485	**X**
14	45.495	0.3394	**X**
15	22.237	0.3623	**X**
Average	27.061	0.3539	—
σ	10.005	0.0130	—
Coll. %	—	—	100

(b) Results using ROS default configuration

Table 4.1: Simulation results obtained on the first scenario

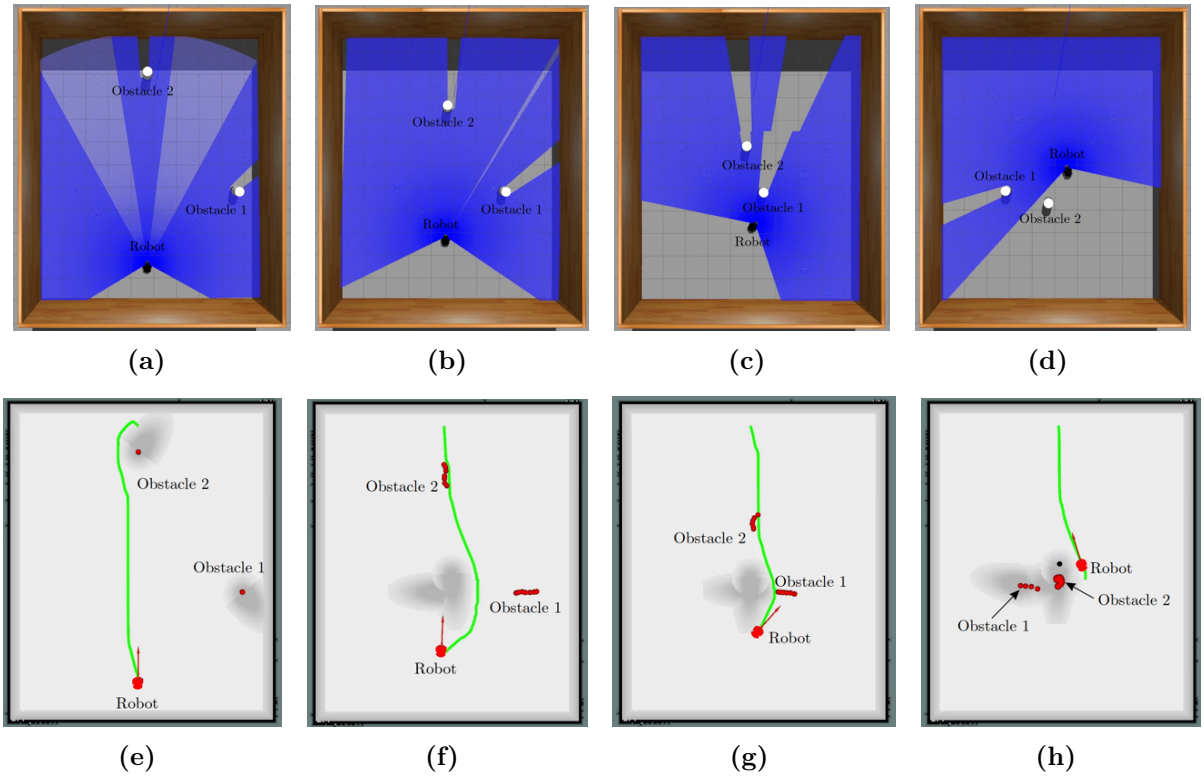


Figure 4.2: Screenshots of the key frames of test number 10 on the first simulation scenario using the proposed approach. On top are screenshots from Gazebo, while the bottom ones are from RViz

Second Scenario

Analysis of the results presented in Table 4.2 reveal that the general paradigm identified in the results from the first scenario is still present here: costmap construction assuming that all obstacles are static does not perform well in environments populated with dynamic obstacles. Once again, with ROS *Navigation Stack* default configuration, we get a 100% collision rate. However, with our approach, we only got 2 collisions in 15 repetitions of the scenario, which translates to a collision rate of 13.33%. This is a downgrade from the performance registered in the first scenario, owing to the increased difficulty of this one.

The minimum center-to-center distances registered in each execution do not always assume values as large as would be desirable. The cause of this is, as concluded in the first scenario, the perception method, which tends to take a bit longer than it ideally should to determine a stable and correct velocity estimate for the obstacles. As a consequence, excessive path replanning tends to occur, creating a navigational behavior marked by an "indecisiveness" of sorts regarding the path to follow, e.g., whether to go through the left or the right. The larger number of obstacles in this scenario makes it all the more challenging for the perception process due to the obstacle occlusions that inevitably happen.

Considering solely the actual behavior of the robot, one can safely state that, in this

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	16.208	1.0518	
2	15.210	0.6064	
3	15.507	0.5082	
4	13.746	1.1215	
5	20.535	0.3942	**X**
6	15.748	0.4157	
7	15.733	0.5011	
8	15.607	0.9125	
9	15.001	0.7925	
10	14.480	0.8967	
11	14.900	0.7453	
12	22.392	0.3986	**X**
13	19.119	0.9067	
14	14.986	0.5468	
15	15.540	0.7078	
Average	16.369	0.7003	—
σ	2.497	0.2409	—
Coll. %	—	—	13.33

(a) Results using the proposed approach

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	57.385	0.3587	**X**
2	46.356	0.3521	**X**
3	25.483	0.3538	**X**
4	15.979	0.3865	**X**
5	15.997	0.3887	**X**
6	25.305	0.3454	**X**
7	44.979	0.3508	**X**
8	16.872	0.3864	**X**
9	24.270	0.3488	**X**
10	47.552	0.3355	**X**
11	42.994	0.3655	**X**
12	58.574	0.3364	**X**
13	20.004	0.3683	**X**
14	34.771	0.3380	**X**
15	22.618	0.3363	**X**
Average	32.582	0.3567	—
σ	15.419	0.0187	—
Coll. %	—	—	100

(b) Results using ROS default configuration

Table 4.2: Simulation results obtained on the second scenario

scenario, it is both satisfactory and in accordance with what was desired and expected. Except for the two tests where collisions happened, the robot always passed through the back of both obstacles 1 and 3, although sometimes a bit closer than it should (run number 6).

In figure 4.3 we present the key frames of run number 1 using our approach. Figures 4.3a and 4.3e portray the beginning of the simulation scenario. The path planned (the green line in fig. 4.3e) assumes a shape close to a straight line because the obstacles are not yet moving. In the phase of the simulation presented in figures 4.3b and 4.3f, it is clear that the velocity estimates of the moving obstacles, while not as accurate as would be desirable, are good enough to condition the path planning process into collision avoidance. The black circles in figures 4.3c and 4.3g correspond to previous instances where the perception block momentarily lost track of obstacles 2 and 3, though it was able to quickly recover. Finally, in figures 4.3d and 4.3h the robot has already successfully avoided obstacles 1 and 2 and has

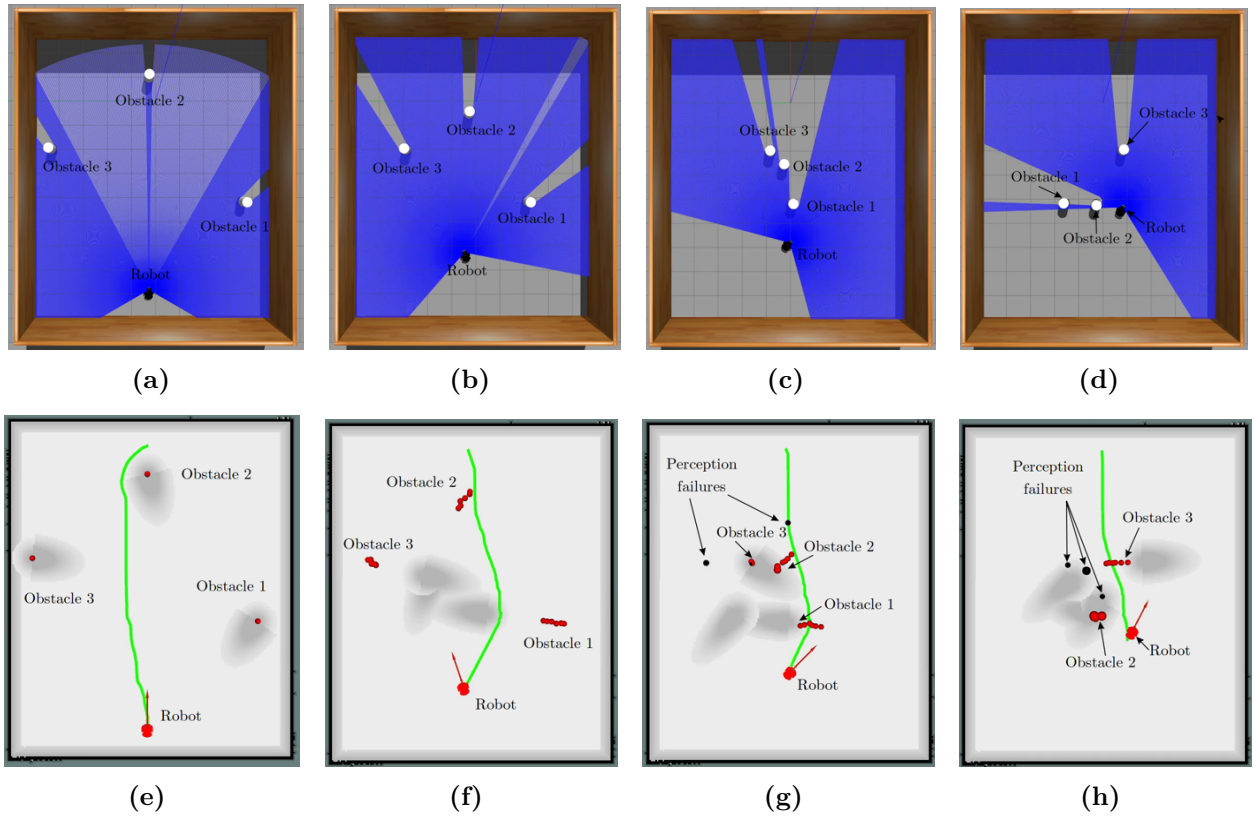


Figure 4.3: Screenshots of the key frames of test number 1 on the second simulation scenario using the proposed approach. On top are screenshots from Gazebo, while the bottom ones are from RViz

a stable path guiding it through the back of obstacle 3 as intended, beingfree to head toward its goal. Once again, the black dots correspond to moments when the perception block failed its task. Particularly, its difficulties handling occluded obstacles become abundantly clear. Obstacle 1 is occluded by obstacle 2 and, therefore, completely lost track of, its presence in the costmap being represented at its formerly predicted critical future position.

Third Scenario

In terms of the occurrence of collisions, this scenario brings similar results to the ones presented for the first and second scenarios. The robot's behavior, however, becomes somewhat degraded in this case. Although in most of the tests using our approach the robot did not collide with any obstacles, it frequently suffered from more severe "indecisiveness" in comparison to the two previous scenarios. This means that it was not uncommon for the robot to perform an in place rotation (one of ROS predefined recovery behaviors) right after starting its movement due to the instability of the generated paths.

The large number of obstacles in this scenario makes it all the more challenging for the perception process due to the obstacle occlusions that inevitably happen. Predicting an

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	17.114	0.8435	
2	31.323	0.3602	**X**
3	33.648	0.3549	**X**
4	22.007	0.4568	
5	21.416	0.4378	
6	19.236	1.0093	
7	17.991	0.9410	
8	35.065	0.3791	**X**
9	16.276	0.8381	
10	18.622	0.7545	
11	16.509	0.7225	
12	17.888	1.2506	
13	17.129	1.3056	
14	16.710	0.7853	
15	16.124	0.6963	
Average	21.137	0.7424	—
σ	6.587	0.3062	—
Coll. %	—	—	20

(a) Results using the proposed approach

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	20.245	0.3580	**X**
2	15.245	0.3623	**X**
3	16.627	0.3855	**X**
4	45.988	0.3152	**X**
5	16.853	0.3698	**X**
6	23.598	0.3554	**X**
7	22.214	0.3370	**X**
8	31.933	0.3228	**X**
9	15.351	0.3562	**X**
10	29.104	0.3631	**X**
11	30.238	0.3232	**X**
12	19.743	0.3741	**X**
13	16.978	0.3716	**X**
14	26.986	0.3673	**X**
15	17.222	0.3615	**X**
Average	23.222	0.3549	—
σ	8.447	0.0208	—
Coll. %	—	—	100

(b) Results using ROS default configuration

Table 4.3: Simulation results obtained on the third scenario

obstacle's current position when it is not visible can be done relying solely on its previous position and the velocity estimate. Given that the velocity estimate can be far from perfect, this prediction is bound to be quite inaccurate. This creates a significant problem when the occluded obstacle becomes visible again, as its predicted and real positions can be largely different, thus making the system unable to match the observed obstacle to a previously known one. This means that the costmap may at times have the same obstacle marked twice in different places, something that indeed happened frequently in this scenario and hindered the path planning process. In figure 4.4 we present the key frames of run number 13 using our approach. Given that the first planned path is similar across all three scenarios, we decided to not include it here. Figures 4.4a and 4.4e represent the instant where a second path is planned. The planning of this new path is triggered by the changes in the obstacles' predicted future critical positions, which happen due to the improvement of the obstacles' velocity estimates. Figure 4.4f reveals a lack of accuracy in the direction of obstacle 2's

velocity estimate. There, the area of the costmap corresponding to the future position of obstacle 2 is marked near obstacle 1's current position, when the correct placement for it would be roughly the same as in figure 4.4e (by the center of the scenario, pointing downwards). Despite this, the planned path is safe for the robot, although it would not pass obstacle 3 through its back. In figures 4.4c and 4.4g a very different path is planned from the ones present in figures 4.4e and 4.4f. This is a case of the "indecisiveness" of the robot's behavior that was mentioned earlier, causing the robot to stop for some brief moments. Finally, figures 4.4d and 4.4h portray the moment when a new, correct path is planned, which the robot follows to its goal. The difference between figures 4.4g and 4.4h lies essentially on the new predicted critical position for obstacle 3. Since the robot had stopped for some moments, it gave time for obstacle 3 to travel a far enough distance so that a path could be planned for the robot to go through its back.

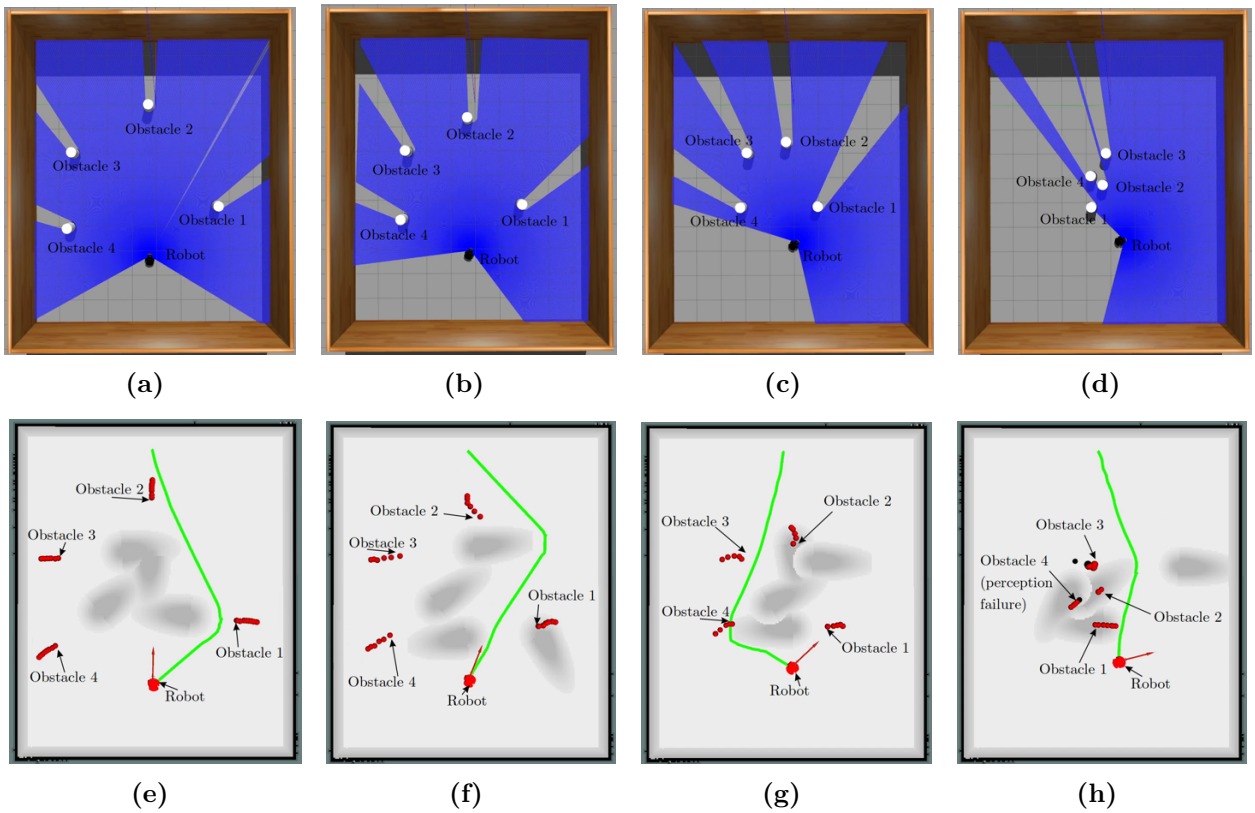


Figure 4.4: Screenshots of the key frames of test number 13 on the third simulation scenario using the proposed approach. On top are screenshots from Gazebo, while the bottom ones are from RViz

4.3 Real-World Experiments

4.3.1 Setup

The differential drive robotic platform used for these experiments was a modified Nomadic Technologies Scout robot, with a radius of approximately 20 cm (fig. 4.5a) equipped with an an OrangePi computer running Linux. In terms of sensory devices, except for proprioceptive sensors such as wheel encoders, it is equipped with only a SICK TIM551-2050001 2D LiDAR (fig. 4.5b) with an aperture angle of 270°, an angular resolution of 1°, a working range of 0.05 m to 10 m and a scanning frequency of 15 Hz. The OrangePi is used solely for running the ROS nodes of the drivers for the robot's motors and for the LiDAR. All the other computationally heavier nodes, such as the ones related to the work developed in this dissertation and the default ones used by ROS for robot navigation are executed on a laptop equipped with an AMD Ryzen™ 4600H CPU (6 cores, 12 threads at a base clock of 3.0 GHz) and 16 GB of RAM.

Three scenarios, repeated 5 times each, were tested using our approach, all of them involving the robot and a human who performs a simple trajectory without trying to avoid the robot. The objective is, naturally, for the robot to avoid a collision and keep a comfortable distance from the human. The first scenario is a *collision at a crossing scenario*, where the robot is faced with a human moving perpendicularly to the path the robot is following. The second scenario is a *face to face collision* scenario, where the robot and the human are moving in opposite directions. The third scenario is an *overtaking* scenario, where the robot finds a human moving slowly in the same direction of the robot and must overtake it. All tests were performed in the second floor of the DEEC, in the area between the S and R towers (visible behind the robot in figure 4.5a).

4.3.2 Results and Discussion

The metrics used here are still the same ones presented in 4.1. However, it must be taken into account that, unlikely what happened in the simulations, in the real-world experiments we have no ground truth data regarding the obstacles' positions. This means that in order to calculate the minimum distance between the robot and an obstacle we had to rely on the estimated positions of the obstacles. Therefore, this metric will certainly incorporate some level of uncertainty due to the imperfections in the obstacles' position estimate. Furthermore,

(a) Differential drive Scout robot **(b)** SICK TIM551-2050001 2D LiDAR

Figure 4.5: Pictures of the hardware used for the real-world experiments

the elapsed time metric may also lose some relevance here, due to the difficulty of perfectly repeating the starting and goal positions of the robot in every test. As such, the more relevant metrics here are certainly the minimal distance to obstacle, d_{min} and the occurrence of a collision.

First Scenario

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	19.854	0.6166	
2	25.639	0.7389	
3	25.218	0.7665	
4	22.774	0.6185	
5	22.615	0.7563	
Average	23.220	0.6890	—
σ	2.331	0.0837	—
Coll. %	—	—	0

Table 4.4: Results obtained on the first real-world scenario

The results presented in table 4.4 reveal that the desired behavior was achieved, avoiding a collision while also keeping a socially comfortable distance from the human. It should be

noted that, while not perceivable based on the results presented on the table, in an initial stage of each test there was some indecisiveness in the planned paths due to fluctuations of the predicted critical future pose of the human. However, unlike what happened in the simulations, it did not hinder the robot's motion nor did it force it to stop, because the predicted critical future pose quickly stabilized. The biggest difficulties with this scenario were being able to time correctly the beginning of the human's motion and figuring out the speed at which the person should walk, due to the limited space available in the area used for testing. Starting to move too early or too late, as well as walking too briskly or too slowly, would cause the predicted critical position to lie outside the robot's path, thus not creating the need for the robot to avoid a collision.

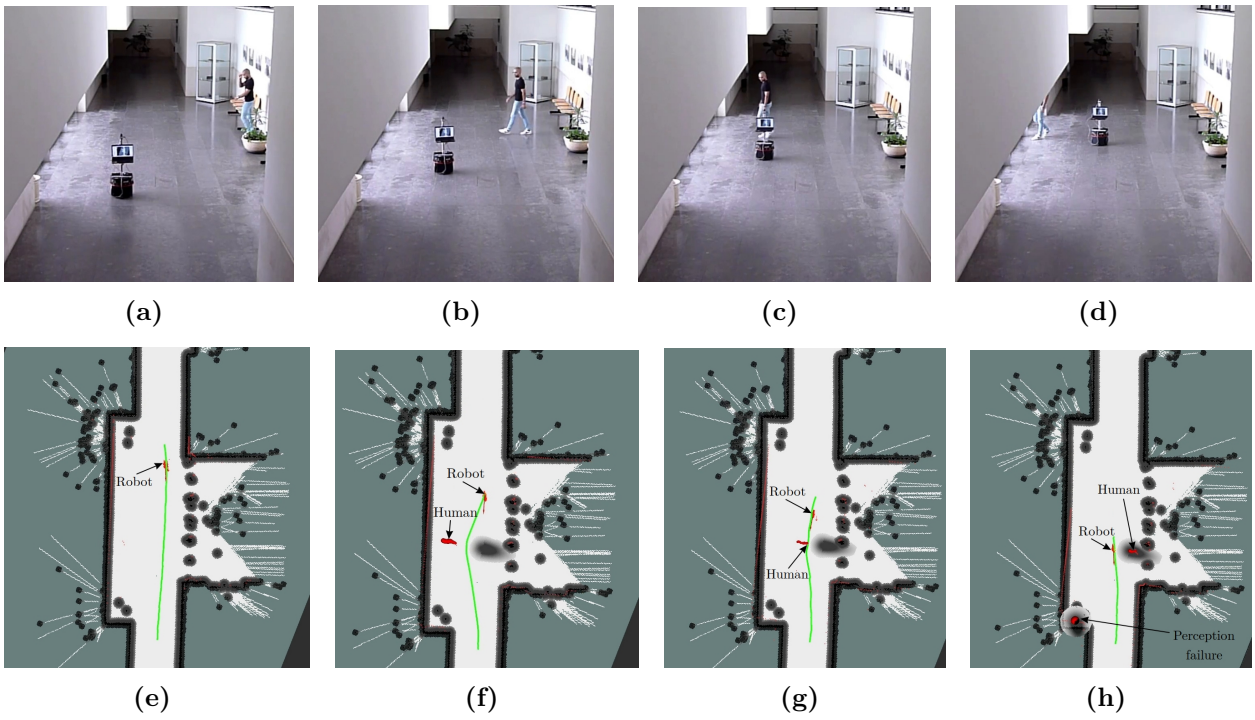


Figure 4.6: Screenshots of the key frames of test number 2 on the first real-world scenario using the proposed approach. On top are screenshots from the recorded video, while the bottom ones are from RViz

Figure 4.6 portrays the key moments of test number 2 in this scenario. In figures 4.6a and 4.6e, the human is extremely close to the wall and is yet to start moving. Therefore, the planned path goes almost straight to the robot's goal. Figures 4.6b and 4.6f show the moment when the human starts moving and its critical pose is predicted, triggering path replanning. In figures 4.6c and 4.6g the human's velocity estimate has been improved and stabilized, thus altering the predicted critical pose, which in turn causes a new path to be planned. In figures 4.6d and 4.6h the moment when the robot is at its closest to the person and free to reach its goal is portrayed. Comparing it to 4.6c and 4.6g one concludes that the predicted

Run	$T(s)$	$d_{min}(m)$	$Collision$
1	24.318	0.5650	
2	22.370	0.6394	
3	22.565	0.4235	
4	23.656	0.4579	
5	25.234	0.5073	
$Average$	23.365	0.5186	—
σ	1.205	0.0860	—
$Coll.\,\%$	—	—	0

Table 4.5: Results obtained on the second real-world scenario

future position of the human was correct. These figures showcase the previously explained difficulty of correctly timing the start of human motion, because although the original path would eventually pass too close to the human, a collision wouldn't happen. Figure 4.6h also reveals an erroneously detected obstacle. This happened due to the considerable error in the robot's localization, particularly in its orientation, that was verifiable by the end of the test.

Second Scenario

The results presented in table 4.5 reveal, in comparison with the ones in table 4.4, that in this scenario the robot had a tendency to pass closer to the human than in the first one. This is something that we noticed during the experiments as well. Considering that the robot has a radius of approximately 20 cm, this means that in the worst of the runs, the minimum distance to the human was of approximately 22 cm. This is a safe distance but not a particularly comfortable one, even though these points of minimal distance occur when the robot is already side-by-side with the human and not on its front anymore. Nevertheless, the path replanning when the robot detects the human and estimates its speed happens in a timely manner.

In figure 4.7 we present some of the key frames of run number 1 using our approach. In figures 4.7a and 4.7e, there are not enough LiDAR readings of the human for him to be considered an obstacle. Therefore, the planned path goes almost straight to the robot's goal. Figures 4.7b and 4.7f show the moment when the human is detected as an obstacle. Since there isn't a velocity estimate yet, the human's costmap assignment is done on his current position and a new path is calculated. In figures 4.7c and 4.7g we can see that the future critical position of the human has been determined and thus path replanning

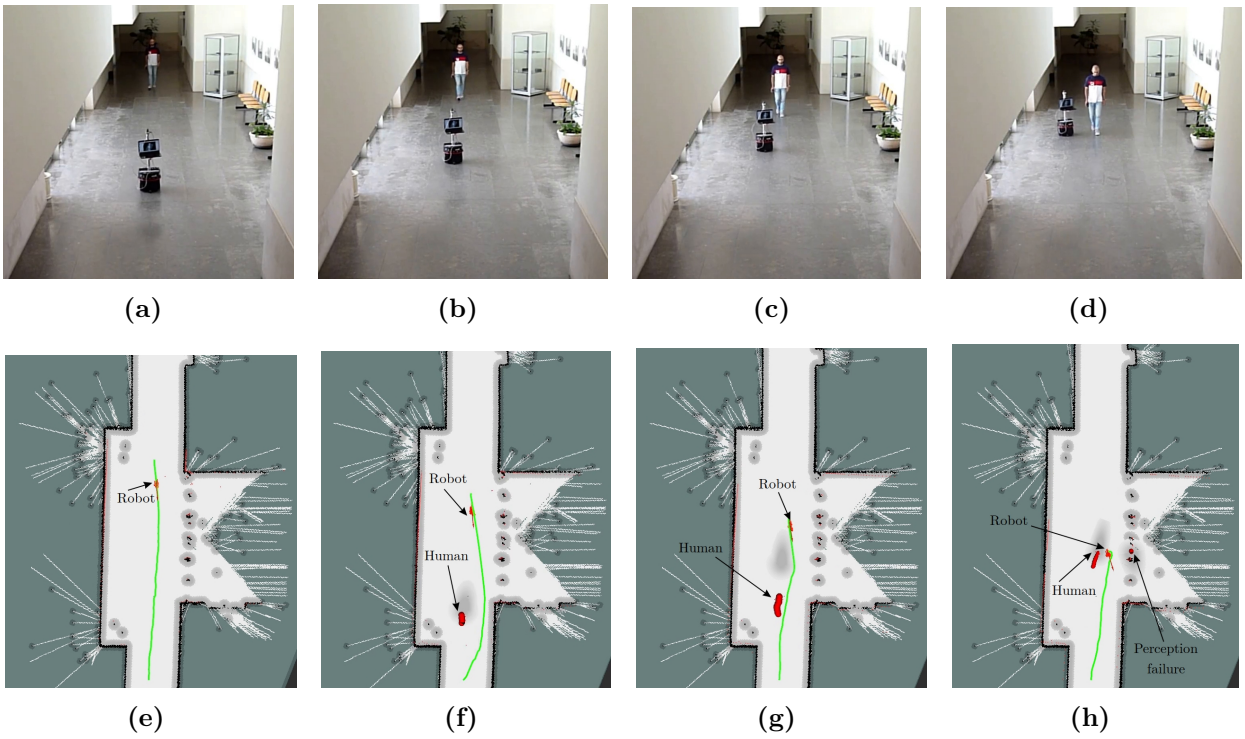


Figure 4.7: Screenshots of the key frames of test number 1 on the second real-world scenario using the proposed approach. On top are screenshots from the recorded video, while the bottom ones are from RViz

occurred. Figures 4.7d and 4.7h portray the instant when the robot and the human are side by side. It is clear that human's position estimate introduced in figure 4.7g was accurate, for the human's cost assignment covers the same as it did in figure 4.7g. It is also perceivable in figure 4.7h that, as in the previous scenario, a nonexistent obstacle is detected due to the problems in the localization of the robot.

Third Scenario

Run	$T(s)$	$d_{min}(m)$	*Collision*
1	27.391	0.6778	
2	29.656	0.5335	
3	27.972	0.4861	
4	27.774	0.6070	
5	30.312	0.6411	
Average	28.929	0.5891	—
σ	1.250	0.078	—
Coll. %	—	—	0

Table 4.6: Results obtained on the third real-world scenario

The paradigm in this scenario's results doesn't differ much from what was observed in the first two scenarios. It should be noted that the elapsed time increases in comparison to both of the other two scenarios, because in the beginning, before overtaking, the robot moves behind the slowly walking human, causing it to take more time to reach the goal.

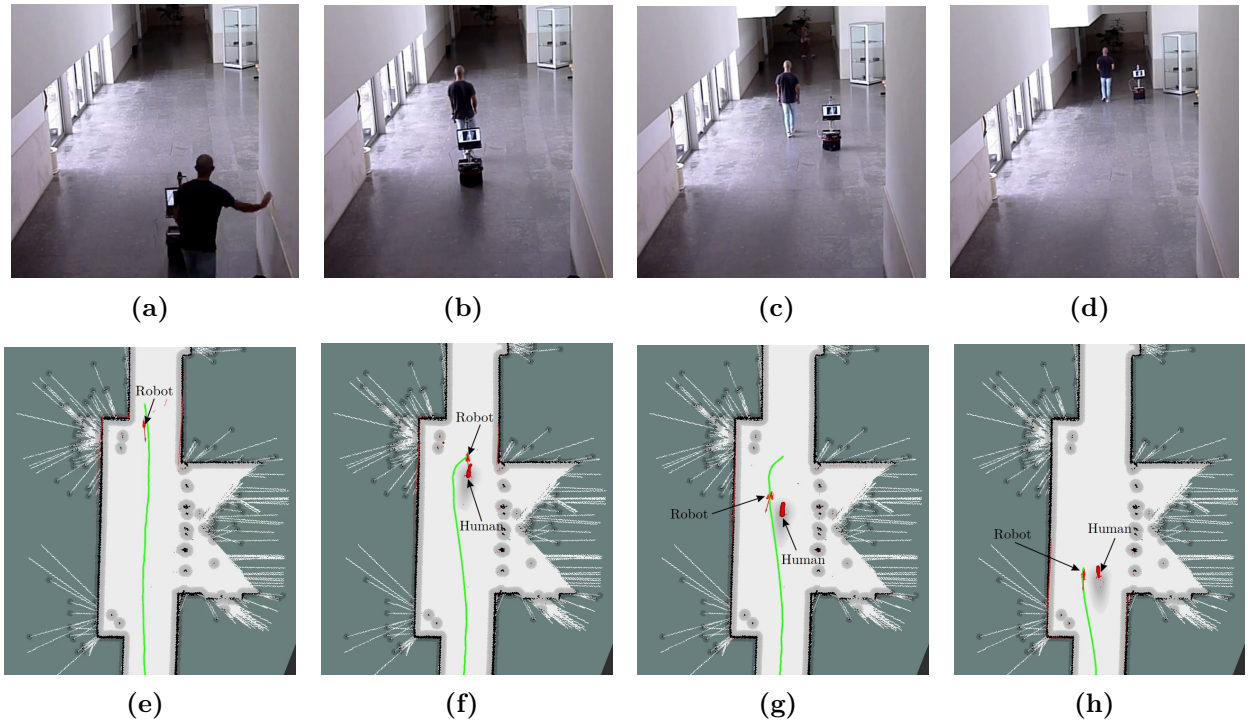


Figure 4.8: Screenshots of the key frames of test number 1 on the third real-world scenario using the proposed approach. On top are screenshots from the recorded video, while the bottom ones are from RViz

In figure 4.8 we present a set of 4 crucial frames from both a recorded video of run number 1 and a screen recording of RViz. Just as in the results presented for the first scenarios, the first pair of images, figures 4.8a and 4.8e, portray a moment when the robot's path is still unimpeded by the human. In figures 4.8b and 4.8f is already in front of the robot, walking at a very slow speed, and a path has been planned that contemplates an overtaking maneuver by the robot. Figures 4.8c and 4.8g intend to show the robot following the previously planned path and thus overtaking the human at a safe lateral distance. Finally, in figures 4.8d and 4.8h the robot is in the final stages of the overtaking process. A slightly different path has been planned since figures 4.8c and 4.8g, one which guarantees that the robot will very progressively pass to the front of the human, instead of immediately moving to that area.

5 Conclusions and Future Work

This dissertation presents an approach to costmap based robot navigation in dynamic environments. Specifically, our technique takes into account the motion of an obstacle to predict the area that the robot should avoid going through in the future in order to prevent a collision, instead of assuming a static environment. Our approach also incorporates the uncertainty in the estimate of the obstacles' motion into the cost assigned to the costmap. We ensure that the paths generated by the path planner of choice do not hinder the obstacle's motion and lead the robot through a socially more acceptable trajectory through the inflation of the costs of the costmap cells located along the obstacle's moving direction. The proposed method was integrated into the ROS *Navigation Stack* as a costmap plugin, i.e, a costmap layer. Because our approach requires specific information regarding the obstacles, such as their estimated position and its associated variance, the approximate radius of their footprint and an estimate of its velocity, we also present a perception solution tailored for use with LiDAR devices.

The results presented in this dissertation lead us to determine that cost assignment approaches that do not take into account the motion of dynamic obstacles are not adequate for real-world scenarios, given their 100% collision rate in the experiments we performed. In both the simulations and the real-world experiments our approach was able to, most of the times, avoid collisions, although the distance kept from the obstacles was not always as large as would be desirable. We also found that our approach consistently produced the desired navigational behavior, i.e., the planned paths tended to pass through the back of the obstacles when appropriate.

The work developed in this dissertation leaves some room for improvement. The perception block revealed itself to be the weakest link of this work and could be largely enhanced through the use of a camera. In fact, the use of a camera would not only improve the perception block but also provide the cost assignment block with a richer set of information regarding the obstacles, especially considering that humans are the dynamic obstacles most

likely to be encountered by a robot. This richer set of information could also be useful for the incorporation of more social constraints onto the cost assignment process, which could result in a navigational behavior more compliant with social etiquette.

6 Bibliography

[1] Marina Kollmitz, Kaijen Hsiao, Johannes Gaa, and Wolfram Burgard. Time Dependent Planning on a Layered Social Cost Map for Human-Aware Robot Navigation. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2015.

[2] Kuanqi Cai, Chaoqun Wang, Jiyu Cheng, Clarence W. De Silva, and Max Q.-H. Meng. Mobile Robot Path Planning in Dynamic Environments: A Survey. 2021.

[3] Amin Ghorbani, Saeed Shiry, and Ali Nodehi. Using Genetic Algorithm for a Mobile Robot Path Planning. In *2009 International Conference on Future Computer and Communication*, pages 164–166. IEEE, 2009.

[4] Junjie Zeng, Rusheng Ju, Long Qin, Yue Hu, Quanjun Yin, and Cong Hu. Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning. *Sensors*, 19(18):3837, 2019.

[5] Md Arafat Hossain and Israt Ferdous. Autonomous Robot Path Planning in Dynamic Environment Using a New Optimization Technique Inspired by Bacterial Foraging Technique. *Robotics and Autonomous Systems*, 64:137–141, 2015.

[6] Marija Seder and Ivan Petrovic. Dynamic Window Based Approach to Mobile Robot Motion Control in the Presence of Moving Obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1986–1991. IEEE, 2007.

[7] Jiyu Cheng, Hu Cheng, Max Q.-H. Meng, and Hong Zhang. Autonomous Navigation by Mobile Robots in Human Environments: A Survey. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1981–1986. IEEE, December 2018.

[8] Carlos A. Silva, Sedat Dogru, and Lino Marques. (Accepted) Mobile Robot Navigation in Dynamic Environments Taking Into Account Obstacle Motion in Costmap Construction. In Danilo Tardioli, Vicente Matellán, Guillermo Heredía, Manuel F. Silva, and

Lino Marques, editors, *ROBOT 2022: Fifth Iberian Robotics Conference. ROBOT 2022. Advances in Intelligent Systems and Computing*. Springer.

[9] Maja J Mataric. *The Robotics Primer*. MIT Press, 2007.

[10] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2011.

[11] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A Review of Mobile Robots: Concepts, Methods, Theoretical Framework, and Applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

[12] John J Leonard and Hugh F Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[13] Cyrill Stachniss. *Robotic Mapping and Exploration*, volume 55. Springer, 2009.

[14] Yi Cheng and Gong Ye Wang. Mobile Robot Navigation Based on Lidar. In *2018 Chinese Control and Decision Conference (CCDC)*, pages 1243–1246. IEEE, 2018.

[15] Joydeep Biswas and Manuela Veloso. Depth Camera Based Indoor Mobile Robot Localization and Navigation. In *2012 IEEE International Conference on Robotics and Automation*, pages 1697–1702. IEEE, 2012.

[16] Sukkpranhachai Gatesichapakorn, Jun Takamatsu, and Miti 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)*, pages 151–154. IEEE, 2019.

[17] Sebastian Thrun et al. Robotic Mapping: A Survey. *Exploring Artificial Intelligence in the New Millennium*, 1(1-35):1, 2002.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[19] Alberto Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57, 1989.

[20] Kwangro Joo, Tae-Kyeong Lee, Sanghoon Baek, and Se-Young Oh. Generating Topological Map from Occupancy Grid-Map using Virtual Door Detection. In *IEEE Congress on Evolutionary Computation*, pages 1–6. IEEE, 2010.

[21] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[22] Kruno Lenac, Andrej Kitanov, Robert Cupec, and Ivan Petrović. Fast Planar Surface 3D SLAM using LiDAR. *Robotics and Autonomous Systems*, 92:197–220, 2017.

[23] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A Versatile Visual SLAM Framework. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 2292–2295, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[25] Mohammad O.A. Aqel, Mohammad H Marhaban, M. Iqbal Saripan, and Napsiah Bt. Ismail. Review of Visual Odometry: Types, Approaches, Challenges, and Applications. *SpringerPlus*, 5(1):1–26, 2016.

[26] Huangying Zhan, Chamara Saroj Weerasekera, Jia-Wang Bian, and Ian Reid. Visual Odometry Revisited: What Should Be Learnt? In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4203–4210, 2020.

[27] Yen-Cheng Kung, Chen-Chien J Hsu, and Wei-Yen Wang. Monte Carlo Localization Incorporating an Error Correction Vector for Mobile Robot. In *ICSSE*, pages 306–318, 2015.

[28] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for Mobile Robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 2, pages 1322–1328. IEEE, 1999.

[29] Maarten Speekenbrink. A Tutorial on Particle Filters. *Journal of Mathematical Psychology*, 73:140–152, 2016.

[30] B.K. Patle, Ganesh Babu L, Anish Pandey, D.R.K. Parhi, and A. Jagadeesh. A Review: On Path Planning Strategies for Navigation of Mobile Robot. *Defence Technology*, 15(4):582–606, 2019.

[31] Anis Koubaa, Hachemi Bennaceur, Imen Chaari, Sahar Trigui, Adel Ammar, Mohamed-Foued Sriti, Maram Alajlan, Omar Cheikhrouhou, and Yasir Javed. Introduction to Mobile Robot Path Planning. In *Robot Path Planning and Cooperation*, pages 3–12. Springer, 2018.

[32] Mohamed Elbanhawi and Milan Simic. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2:56–77, 2014.

[33] Edsger W Dijkstra et al. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[34] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[35] L. Kavraki and J.-C. Latombe. Randomized Preprocessing of Configuration Space for Path Planning: Articulated Robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 3, pages 1764–1771. IEEE, 1994.

[36] Nancy M Amato and Yan Wu. A Randomized Roadmap Method for Path and Manipulation Planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 113–120. IEEE, 1996.

[37] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[38] Steven M LaValle et al. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

[39] Sertac Karaman and Emilio Frazzoli. Sampling-Based Algorithms for Optimal Motion Planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[40] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch Informed Trees (BIT): Sampling-based Optimal Planning via the Heuristically Guided Search

of Implicit Random Geometric Graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[41] Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer. Regionally Accelerated Batch Informed Trees (RABIT*): A Framework to Integrate Local Information into Optimal Path Planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4207–4214. IEEE, 2016.

[42] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. Evolutionary Algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195, 2014.

[43] Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer Science & Business Media, 2010.

[44] Agoston E Eiben and James E Smith. What Is an Evolutionary Algorithm? In *Introduction to Evolutionary Computing*, pages 25–48. Springer, 2015.

[45] Yanrong Hu and Simon X Yang. A Knowledge Based Genetic Algorithm for Path Planning of a Mobile Robot. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04*, volume 5, pages 4350–4355. IEEE, 2004.

[46] Rui Wang, Jinguo Wang, and Na Wang. Robot Global Path Planning Based on Improved Ant Colony Algorithm. In *Proceedings of the 3rd International Conference on Material, Mechanical and Manufacturing Engineering*, pages 946–949. Atlantis Press, 2015.

[47] Hong Liu, Bin Xu, Dianjie Lu, and Guijuan Zhang. A Path Planning Approach for Crowd Evacuation in Buildings Based on Improved Artificial Bee Colony Algorithm. *Applied Soft Computing*, 68:360–376, 2018.

[48] Alaa Tharwat, Mohamed Elhoseny, Aboul Ella Hassanien, Thomas Gabel, and Arun Kumar. Intelligent Bézier Curve-Based Path Planning Model using Chaotic Particle Swarm Optimization Algorithm. *Cluster Computing*, 22(2):4745–4766, 2019.

[49] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal Path Planning of Mobile Robots: A Review. *International Journal of Physical Sciences*, 7(9):1314–1320, 2012.

[50] Johann Borenstein and Yoram Koren. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[51] Seyyed Mohammad Hosseini Rostami, Arun Kumar Sangaiah, Jin Wang, and Xiaozhu Liu. Obstacle Avoidance of Mobile Robots using Modified Artificial Potential Field Algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–19, 2019.

[52] Paolo Fiorini and Zvi Shiller. Motion Planning in Dynamic Environments Using Velocity Obstacles. *The International Journal of Robotics Research*, 17:760–772, July 1998.

[53] David Wilkie, Jur van den Berg, and Dinesh Manocha. Generalized Velocity Obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578, St. Louis, MO, USA, October 2009. IEEE.

[54] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots. In *Distributed Autonomous Robotic Systems*, pages 203–216. Springer, 2013.

[55] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[56] Piyapat Saranrittichai, Nattee Niparnan, and Attawith Sudsang. Robust Local Obstacle Avoidance for Mobile Robot Based on Dynamic Window Approach. In *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–4. IEEE, 2013.

[57] Mohammed Algabri, Hassan Mathkour, Hedjar Ramdane, and Mansour Alsulaiman. Comparative Study of Soft Computing Techniques for Mobile Robot Navigation in an Unknown Environment. *Computers in Human Behavior*, 50:42–56, September 2015.

[58] Amir Nasrinahar and Joon Huang Chuah. Intelligent Motion Planning of a Mobile Robot with Dynamic Obstacle Avoidance. *Journal on Vehicle Routing Algorithms*, 1:89–104, November 2018.

[59] Ngangbam Herojit Singh and Khelchandra Thongam. Neural Network-Based Approaches for Mobile Robot Navigation in Static and Moving Obstacles Environments. *Intelligent Service Robotics*, 12(1):55–67, 2019.

[60] Gregor Klancar, Andrej Zdesar, Saso Blazic, and Igor Skrjanc. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*. Butterworth-Heinemann, 2017.

[61] David V Lu, Dave Hershberger, and William D Smart. Layered Costmaps for Context-Sensitive Navigation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–715. IEEE, 2014.

[62] David D Fan, Ali-Akbar Agha-Mohammadi, and Evangelos A Theodorou. Learning Risk-Aware Costmaps for Traversability in Challenging Environments. *IEEE Robotics and Automation Letters*, 7(1):279–286, 2021.

[63] Rachel Kirby, Reid Simmons, and Jodi Forlizzi. COMPANION: A Constraint-Optimizing Method for Person-Acceptable Navigation. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 607–612. IEEE, 2009.

[64] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

[65] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: an Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, page 5. Kobe, Japan, 2009.

[66] Kaiyu Zheng. ROS Navigation Tuning Guide. In *Robot Operating System (ROS)*, pages 197–226. Springer, 2021.

[67] Setup and Configuration of the Navigation Stack on a Robot, 2018. [Online; Available at `http://wiki.ros.org/navigation/Tutorials/RobotSetup`; accessed 31-August-2022].

[68] Fangkai Yang and Christopher Peters. Social-Aware Navigation in Crowds with Static and Dynamic Groups. In *2019 11th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*, pages 1–4. IEEE, 2019.

[69] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[70] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceed-*

ings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), volume 96, pages 226–231. AAAI Press, 1996.

[71] Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[72] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems*, 42(3):1–21, July 2017.

[73] Ian D Coope. Circle Fitting by Linear and Nonlinear Least Squares. *Journal of Optimization Theory and Applications*, 76(2):381–388, 1993.

[74] Greg Welch, Gary Bishop, et al. An Introduction to the Kalman Filter. 1995.

[75] Matteo De Rose, Marina Indri, and Gianluca Prato. LiDAR-based Dynamic Path Planning of a Mobile Robot Adopting a Costmap Layer Approach in ROS2. Master's thesis, Polytechnic University of Turin, 2019.