Pedro Dinis
Bastos Tavares

NAVEGAÇÃO SOCIAL PARA DESINFEÇÃO DE ESPAÇOS PÚBLICOS

SOCIAL NAVIGATION FOR PUBLIC SPACE DISINFECTION

Universidade de Aveiro
2023

Pedro Dinis Bastos Tavares

NAVEGAÇÃO SOCIAL PARA DESINFEÇÃO DE ESPAÇOS PÚBLICOS

SOCIAL NAVIGATION FOR PUBLIC SPACE DISINFECTION

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à conclusão da unidade curricular Dissertação / Projeto / Estágio, condição necessária para obtenção do grau de Mestre em Robótica e Sistemas Inteligentes , realizada sob a orientação científica do Doutor Nuno Lau, Professor Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Eurico Pedrosa, Professor Investigador Doutorado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Vítor Manuel Ferreira dos Santos
Professor Associado C/ Agregação, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Luis Paulo Gonçalves dos Reis
Professor Associado, Universidade do Porto - Faculdade de Engenharia

Professor Doutor José Nuno Panelas Nunes Lau
Professor Associado, Universidade de Aveiro

Palavras Chave

robótica inteligente, navegação social, mapeamento, desinfeção, deteção de pessoas.

Resumo

A desinfeção de espaços públicos é uma tarefa vital para impedir o alastramento de doenças, e isso tornou-se ainda mais claro devido à pandemia associada ao COVID-19. O problema é que esta tarefa é executada por humanos, que se arriscam a ficar infetados por se exporem ao vírus. Uma possível solução para esse tipo de problemas seria tornar a desinfeção num processo automático através do uso de robôs móveis com dispositivos de desinfeção acoplados, quer por radiação (por exemplo radiação ultravioleta), quer por agentes químicos. Este tipo de solução requer uma baixa interação entre humanos e a doença, reduzindo o risco de contaminação dos mesmos. A implementação proposta deve ser capaz de realizar esta tarefa com humanos ao redor, de forma a não afetar as suas dinâmicas, daí precisar dum módulo de deteção e rastreamento dos movimentos das pessoas. O módulo de deteção de pessoas desenvolvido é baseado em LiDARs ao nível das pernas. Com a informação desses sensores é possível detetar as onde estão as pessoas que rodeiam o robô e deduzir a sua velocidade. Com esta informação, o robô pode executar trajetórias socialmente conscientes, ou seja, trajetórias que contem com as pessoas e seus movimentos de forma a não as perturbar. Com a este tipo de navegação implementado, o robô deve ser capaz de navegar em ambientes populados e após implementar navegação orientada à desinfeção, que está fora do contexto desta dissertação, já deve ser capaz de executar desinfeção de locais públicos com pessoas presentes.

Keywords intelligent robotics, social navigation, mapping, disinfection, people detection.

Abstract

Disinfecting public environments is a vital task to stop the spread of disease, and it recently became more relevant than ever before with the COVID-19 pandemic. The problem is that this task is performed by humans, putting them at risk of infection by being exposed to the virus. A possible solution for this type of problem is to make that disinfection automatic through a mobile robot carrying a disinfection device, either utilizing radiation, such as ultraviolet light, or chemicals. This type of solution would require little to no human intervention with the diseases, reducing the risk of infection. The proposed implementation should have the ability to perform the disinfection task with people in its surroundings without affecting its dynamics, hence the need for a module that detects and tracks people. The developed people detection and tracking module is based on shin-level LiDARs. With this sensor information, the robot can detect and track people and deduce their velocity. After gathering information regarding the people surrounding the robot, the robot is capable of navigating in a socially conscientious manner, meaning that it should be able to navigate without disturbing people and their movements. With this type of navigation implemented, the robot is capable of navigating in populated environments and when the disinfection-oriented navigation is implemented, which is out of the context of this dissertation, it should be capable of executing disinfection of public environments with people present.

Contents

List of Figures

List of Tables

Glossary

TEB	Timed-Elastic-Band
DWA	Dynamic Window Approach
DWB	Dynamic Window Approach B
ML	Machine Learning
AGV	Automated Guided Vehicle
ROS	Robotic Operating System
IMU	Inertial Measurement Unit
SFM	Social Force Model
GNN	Global Nearest Neighbor
AGV	Automated Guided Vehicles
LiDAR	Light Detection And Ranging
CNN	Convolutional Neural Network
NN	Neural Network
SVM	Support Vector Machine
ORCA	Obstacle Reciprocal Collision Avoidance

CHAPTER 1

Introduction

As the world was hit by the unprecedented health crisis triggered by the coronavirus, SARS-CoV-2, space disinfection has become a crucial process in everyone's lives. The problem with disinfection of such an infectious virus is that by performing disinfection, the people who perform those tasks are susceptible to being infected by the disease themselves.

To diminish the risk of infection and reduce the spread of diseases, an alternative to manual disinfection is needed. To reduce the risk and spread of diseases, this job should not be performed by humans. A possible way of diminishing the risks for humans who perform this task is through the use of robots instead.

To perform such a task, the robot needs to be capable of navigating through the environment safely and in a way that the environment is disinfected correctly. For simple, static, and unpopulated environments, it may be a relatively simple task, but if such a task is performed in public spaces where moving people are obstacles, the problem becomes significantly more complex, especially when the robot can compromise people's safety.

From the first mobile robots developed to the robots available today, there has been massive progress, both from the sensors available to aid in navigation to the computational power inherent to the robots and to the algorithms used to traverse the environments.

A simple solution, which is mainly used in factories, is the use of Automated Guided Vehicles (AGV)s. AGVs. They are guided by a predefined path that the robot is limited to. These AGVs can be guided through a variety of methods, from underground wires that emit a signal that the robot reads, to tapes attached to the floor and more. Some AGVs stop or can perform small adjustments to avoid possible obstacles in the way, but they lack movement freedom. These types of robots are sufficient when it comes to simple scenarios where the predefined paths are most likely not obstructed and in scenarios where there is little to no interaction with humans because humans can show unpredictable behavior and AGVs are not equipped to deal with such behavior.

The problem is that it is now more common than ever before for robots to work in collaboration or, at least, closely with humans. The pervasiveness of robots in people's lives

comes with safety problems that must be tackled adequately, and in later years that task has been addressed, which has resulted in several navigation methods that are capable of doing so.

Even if robots can navigate environments safely, doing so in a socially aware manner is an entirely different task that is becoming increasingly relevant as robots become more present in people's everyday lives. However, this task is more complex because it requires extensive processing and must consider a broader range of notions and concepts. The robot should think of people not only as obstacles but also as social entities. Some relevant metrics that should be considered when navigating alongside humans are personal spaces, which should not be crossed, the type of trajectory taken to reach its goals, and the traversal speed of the robot, among others. These are all metrics that should be considered when building a social navigation framework. If these metrics are respected, the results should already be satisfactory as the robot's behavior most likely will not interfere with the movements made by the people in its vicinity. But even though there is no interference in the social aspect, it may still be considered odd behavior. For that reason, another path to follow is to try mimicking human behavior when put into the scenarios that the robots will be inserted into. An example of common human behavior is to keep a certain distance from the people ahead, and if there is no intention to overtake, match the person's speed. Another example is to, just like on the roads, stay to the right of the path.

Even with the importance of the previously mentioned metrics, to achieve sociability, it is worth noting that there is no standardized way of evaluating how socially compliant these robots are [1]. With that in mind, shortcomings can still be pinpointed according to the environment in which the robot is used. Depending on the scenario, some aspects may be more relevant than others, which means that methodologies or algorithms can still be chosen based on the identified drawbacks.

1.1 Problem Description

Space disinfection has always been an important task, be it from viruses or bacteria. The coronavirus, SARS-CoV-2, pandemic took that importance a step further, as it became an indispensable task that had to be performed repeatedly and exhaustively to diminish the infection rate by the Coronavirus. This repetitive disinfection can be performed through chemicals or exposure to radiation. For example, ultraviolet radiation. These disinfection methods can be carried out in multiple ways, and are, in most cases, performed by humans. The problem is that if humans are inserted in these potentially infected environments, they are exposed to such diseases, and may get infected, harming the people performing those tasks. Furthermore, those people can now infect others. For those reasons, disinfection is a task that can be performed by robot agents by carrying disinfection equipment along the environment that is to be disinfected. Another improvement would be to perform this task without having to evacuate everyone from the area and without interfering with the environment that it is working on.

With this task in mind, the University of Aveiro is involved in the GermiRRad project whose objective is to use a mobile robot equipped with navigation skills to disinfect populated

areas without interfering with the normal dynamics of the environment. Besides not disturbing the expected behavior of the area, the agent should be able to correctly and thoroughly disinfect the area. To perform the disinfection task in its entirety, the agent should be able to navigate among humans, both moving and stationary.

1.1.1 Objectives

This dissertation aims to develop tools that allow the autonomous navigation of a robot in public indoor environments that are being shared with humans. To achieve this result, the static parts of the environment where the disinfection task will be performed must be mapped beforehand without people so that the agent can plan and navigate autonomously in that same environment when performing the disinfection task. During the disinfection, there is the possibility of humans being present. Therefore, the agent should be able to detect and track humans. This detection and tracking are crucial to navigating socially among people so that the robot can adapt its route and people do not feel threatened or worried. This process will be developed in the Robotic Operating System (ROS) tool, making both ROS and ROS2 eligible for development.

1.1.2 Expected Results

This thesis aims to develop a system that can serve as the groundwork for performing disinfection tasks in crowded indoor environments without disrupting the natural dynamics of the surroundings. The system should be able to identify individuals and adjust its actions accordingly to prevent any interference or collisions with obstacles and people.

1.2 Document Structure

In Chapter 2 of this document, there is an introduction to the tools used to develop this thesis and the background work on which this work is based. After introducing the technologies and methodologies currently used in the industry, the document begins to focus on the development process. The first chapter to focus on the thesis' development is Chapter 3. This chapter covers topics from how people detection is going to be performed, both in terms of tools and techniques that were explored. It starts by explaining the logic behind the tools used for people detection and leans briefly over how the people tracking algorithm works. Finally, there is a section that covers the experiments made along with the results gathered from such experiments. Following the chapter invested in exploring the people tracking processes, navigation starts being addressed. Chapter 4 tackles social navigation between global waypoints. This chapter is where the development of obstacle and people avoidance methodologies is addressed in simulation environments. Finally, Chapter 5, summarizes the generic ideas of the document along with notes about what could have been done differently and what the next steps in development would be.

CHAPTER 2

Background

Robotics and social navigation have been widely explored in recent years. Robotics has been investigated to perform tasks that can be high-precision, dangerous, or repetitive, whereas social navigation is a more recent concept in the field of robotics whose objective is to traverse an environment safely and in a way that does not interfere with human movements. From the exploration of social navigation in the context of robotics, a wide variety of tools, methodologies, and frameworks have been developed and have served as a foundation for this thesis and research in this area.

In the following sections, the main tools and the publications that served as groundwork for the development of this thesis will be highlighted.

2.1 Background Work

The first step in understanding the current state of the area being explored is, in this case, the social navigation area, along with the people detection and tracking that are needed for the robot to know where the people are so that they can be avoided. When it comes to the social navigation issues that will be tackled, some well-established concepts and methodologies and background work serve as a base for many new algorithms and solutions. These solutions depend on the tools used and the environment in which the tools will be applied. Independent of the tools used, Edward Hall's proxemic theory [2] is widely accepted and finds a simplified way of knowing whether or not something is interfering with a person's personal space and causing discomfort Fig. 2.1.

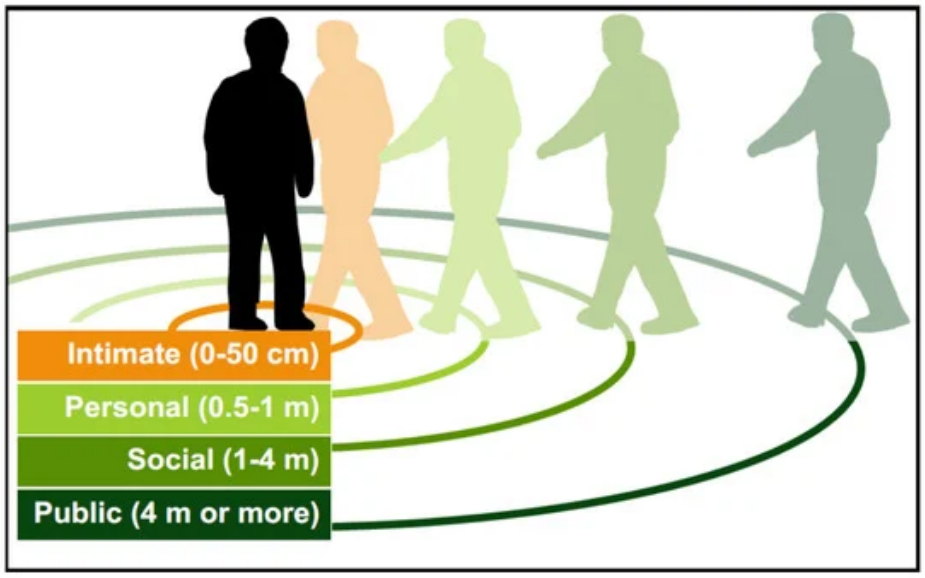


Figure 2.1: Representation of Edward Hall's interpersonal distances [3]

2.1.1 People Detection and Tracking

The basis for this thesis is the detection and tracking of people around the robot. To perform a person detection, which will lead to a person being tracked, it is crucial to determine how such detections will be performed. For this thesis, everything will be developed on an unmodified MiR100 robot [4], as it is the robot being used in the context of the GermiRRad project. The sensors built into the MiR100 that have enough range to help with people detection are the shin-level LiDARs and the 3D cameras. Both types of sensors have pros and cons that depend not only on the kind of sensors but also on how they are mounted onto the robot. The LiDAR configuration on the MiR100 has 360º degree coverage, which allows the detection of obstacles and people around the robot. Another argument for the LiDAR sensor is that LiDARs are not affected by lighting conditions. On the other hand, 3D cameras offer both depth perception and RGB images of the environment. The RGB data makes it possible to use computer vision techniques to determine where people are in the environment and the depth perception allows the robot to understand if it has any objects nearby in three dimensions, as opposed to the LiDAR, which only has two. Even with these advantages, 3D cameras have a critical problem. Considering how the cameras are configured, the cameras are only capable of covering 118º instead of 360º covered by the LiDAR configuration of the MiR100. Leg detection is crucial for this thesis because it informs the robot where people are so that it can adapt its behavior to their presence and activities. Otherwise, the robot would perceive people just like any other obstacle surrounding it. People detection can be performed in distinct ways and by different sensors.

With the characteristics of the sensors that were just mentioned, the first step is to decide which sensor or sensors to use. Both options can be explored, but it is preferable to explore one option at a time, understand if the results for such an option are satisfactory, and proceed according to the outcome. The sensors chosen to be explored were the LiDARs. This decision

was made because of the 360º coverage that the LiDARs provide in the context of the MiR100. This is indispensable, as the robot must be able to adapt its trajectory in case an obstacle is in the way. If the robot is not aware of what is surrounding it, it will not detect possible obstacles in its way and will not act appropriately. Another advantage of the LiDARs over the 3D cameras is that the data provided by the LiDAR is faster to read and process when compared to both the depth and RGB information from the 3D cameras. This slower reading and processing of sensor information could lead to processing delays, which would hinder people detection and tracking in the context of navigation because either information gets ignored to keep processing the latest data or the information is processed out of time. Combining all the reasons for and against each type of sensor information, it was decided that the LiDAR, on its own, would most likely show better results when compared to the 3D cameras.

After deciding that people detection and tracking would be based on leg detection, some exploration was carried out, mainly searching for algorithms that use leg detection with LiDAR information, but also not discarding the possibility of the 3D cameras as, even if they are not used in this dissertation, they may help in improving the solution for people detection and tracking.

People detection based on leg detection can be performed using various algorithms. Some algorithms use features associated with the clusters they create. Those features are fed to the Machine Learning (ML) models that determine whether the clusters represent people or not [5] [6]. In this context, a cluster represents a group of nearby laser readings. For a group of points to be considered a cluster, they must not exceed a certain distance threshold and the group must be composed of at least x points. Only with those criteria met is a group of points considered to be a cluster.

Other algorithms, instead of features, use a variety of heuristics [7] to determine whether or not clusters represent human legs. The clusters detected are put side-by-side with previously detected clusters and the values returned from the heuristics. The probability of being human gathered from the previous scan is considered to update a cluster's probability of being a human leg.

Another way presented uses sensor fusion between 2D LiDARs and images captured from a camera [8]. The LiDARs and camera are calibrated and from there, feature extraction is performed on both data sources independently to train two of their ML models. Upon classification from each model, both classifications are fused, considering the calibration, and the result determines whether it detects a person.

2.1.2 Social Robot Navigation

The social navigation task receives people's detections and tracks. It attempts to navigate accordingly through the environment, and there is not a single solution to this problem, as the understanding of this concept keeps improving.

One commonly mentioned algorithm is the Social Force Model (SFM) [9]. Attractive forces and repulsive forces define this algorithm. The navigation goal defines the attractive force, and the obstacles define the repulsive force. Yet, humans represent a different form

of obstruction with different properties. This is because a person's personal space must be respected.

Another possible algorithm for social navigation is the Elastic Band algorithm. The Elastic Band algorithm is similar to SFM, as it is also based on attractive and repulsive forces. The main difference is the way these forces are applied. In this algorithm, the path is updated by initially creating a path with abrupt changes in direction, and then the contractive and repulsive forces are applied to shape the rough trajectory. The contraction force is the same as the one applied to an elastic, which makes it so that the elastic follows the shortest path between the two contraction points. In this case, the start and the end of the path. After applying both types of forces to the path until the balance between forces is reached, the algorithm has found its final result. If a new dynamic obstacle is found, the path can be updated. A visual representation of this behavior is displayed in Fig. 2.2.

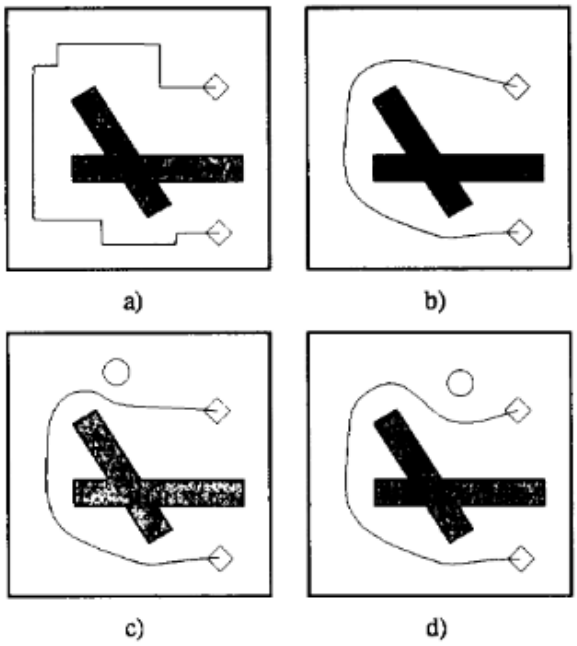


Figure 2.2: Example of path adaptation from the elastic band algorithm [10]. a) Path generated by a planner. b) Path after applying both contraction and repulsion force. c, d) Path as a dynamic object gets near the path.

Another approach, this time not based on forces, is the Dynamic Window Approach (DWA) [11] which is a completely different process. The first step in this approach is to understand which poses are reachable by the robot in a short time frame t while considering its movement limitations and through a velocity pair constituted by angular and linear velocities so that the search for the optimal path does not take too much time, which would hinder the robot's navigation quality. Besides the mentioned limitations, the angular and linear velocity pairs that lead the robot into an obstacle are also immediately discarded. This prediction is done by projecting the robot n up to the expected pose t seconds into the future.

An example of this process is displayed in Fig. 2.3. Where *Vs* represents "the space of possible velocities", *Va* is the space of admissible velocities, or, in other words, the space that

allows for movement that does not crash into any obstacles in its trajectory. The clearer area is admissible, whereas the darker area is not. *Vd* is the space within the dynamic window that surrounds the robot. From these areas, *Vr* is formed, which is the combination of all the previously mentioned areas.

$$Vr = Vs \cap Va \cap Vd$$

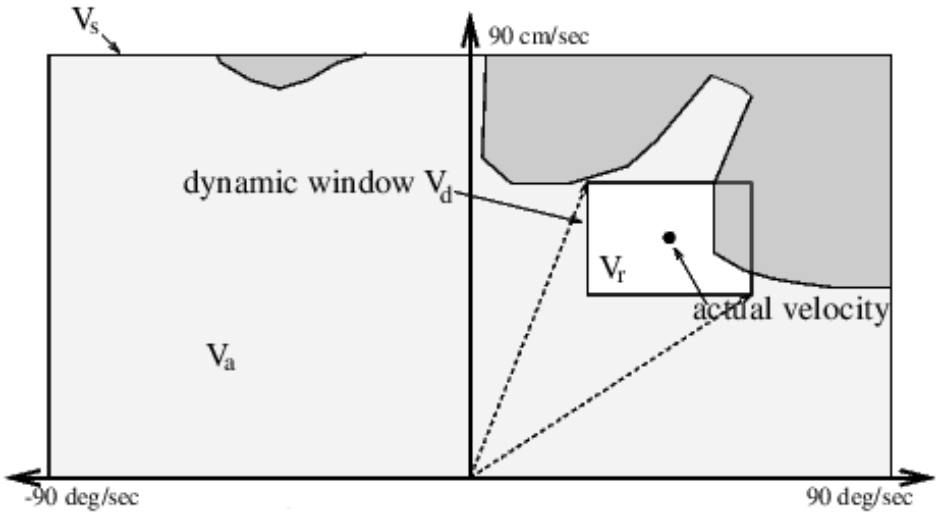


Figure 2.3: Example of space representation in DWA [12].

Upon defining the feasible velocity pairs (angular and linear) that can be used, an objective function is used to reach the ideal combination of velocities according to such objective functions. The parameters considered in the initial approach to this algorithm are *target heading*, *clearance*, and the robot's *velocity*. These parameters, when put together along with their respective weights, form the objective function that determines the best angular and linear velocities in a given instant.

Another way of reducing the social impact of robot navigation is to create a way for the robot to perform early adjustments to its behavior so that the people around it know the trajectory the robot means to take to reach the goal [13]. To create this effect, it is suggested that if a robot has humanoid characteristics, like sensors in a face-shaped format, the robot could adapt the trajectory and have gaze control to preemptively indicate its intentions and not constantly look at the human, which can be unsettling to some people. However, some robots do not have this capability due to their shape, as is the case with MiR100. Even without the cue from the gaze control, when using the social navigation algorithm that adapts early, the results favored only using the social navigation, instead of using it combined with the gaze control due to a limitation on the movement speed of the robot when implementing the cost map. Another relevant model is the Obstacle Reciprocal Collision Avoidance (ORCA) model [3]. It merely tries to avoid collisions, so it does not integrate Social Navigation for obstacle avoidance, yet it is mentioned often in this area.

Even with these methods, these types of environments can be too complex to be modeled in a way that is generalizable and robust [14]. Exactly with that argument in mind, and as

ML grows in popularity, ML algorithms are a replacement. One example of a ML algorithm is based on a Markov Decision Process (MDP), which is defined by states, the set of actions, the probability of getting to a state through an action, and the expected reward. This MDP is Trajectory Bayesian Inverse Reinforcement Learning (TBIRL) which involves gathering expert information to train the model and trying to limit the nearly infinite action possibilities to a finite amount, reducing the computational complexity of the algorithm. To train the model, six features are needed: the distances to obstacles, the goal, and the heading of the obstacles and goals.

2.2 State of the Art

Recently, social navigation and people detection and tracking were explored further with the implementation of new and more complex methodologies to reduce the drawbacks from previous works and create new methods so that the interactions between humans and robots can become as seamless as possible. Most importantly, the COVID-19 pandemic struck, creating new needs, which led to different human dynamics. These dynamics and needs are responsible for more complex tasks being assigned to robots, and because of that increase in complexity, robots have to adapt to perform them correctly.

2.2.1 People Detection and Tracking

When it comes to people detection and tracking, there are a variety of more recent methods. The method proposed by [15], instead of only using geometric features, also uses the relations between neighboring clusters. This information is relevant because people usually have two legs, meaning that for one leg to be detected, another should follow, and a set of features could be formed between both leg detections. Another possibility is to perform detections through sensor fusion, as proposed by [16] which, in this particular example, makes use of a 2D LiDAR and a camera. In this case, unlike the previously mentioned sensor fusion method [8], pedestrian detection is performed on the images through a MobileNet-SSD [17] followed by its tracking, which uses the SORT algorithm [18]. Finally, the pointcloud is compared to the image, and the points that match the constrained image pixels are extracted, allowing to position each detection in the environment. This method presents better results but is limited to the area the camera can cover. Building on previous papers that perform feature extraction [5] [6], another form of detection has been provided in a more recent study by [19]. This method assumes that the robot already knows the base map and the static elements of the map. From that base map, when compared to the pointcloud generated by the LiDAR readings, background extraction can be performed and only leave the segments that do not match the preconceived map, and clusters are created from those points. The clusters are then processed, feature extraction is performed, and upon categorization, a Kalman filter is applied to track these previously classified clusters as humans. However, this assumption can pose a problem if the map is not known a priori.

It is worth mentioning that after the detections are performed, most methods that include tracking have them performed through Kalman Filters [6] [20] [19].

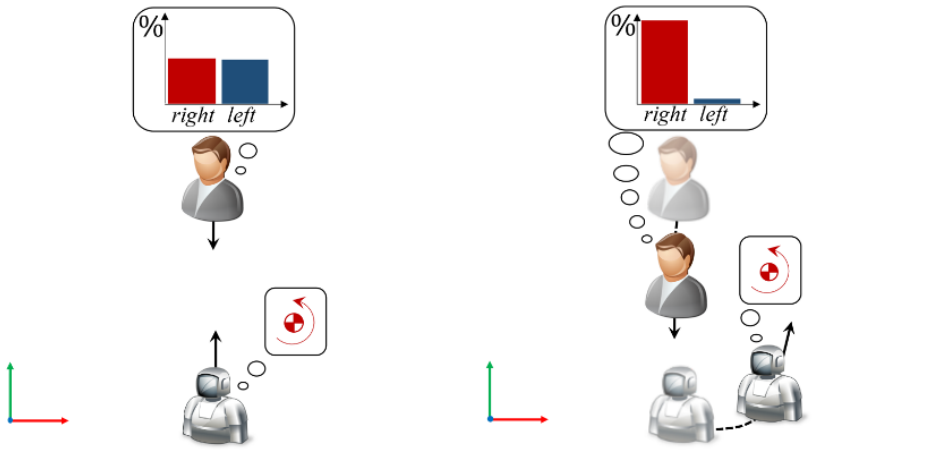
Most methods addressed were solely based on 2D LiDAR, but a few tools based on image input are highly influential and maintained to this day and can be used in sensor fusion. OpenPose [21] is a tool capable of detecting humans and creating a representation of the human skeleton, among other features, for multiple people in real time. OpenPose uses a bottom-up methodology, starting from Part Affinity Fields and reaching the result of the limb and feature representation for one or more individuals. Part Affinity Fields are a form of representing points of a limb so that, when connected, they can form the limbs of one or more people. Usually, the points used to describe a limb in an image would be represented as a point, but with the Part Affinity Field method, these points in the limbs are represented as vectors, so that the algorithm can account for occlusions in case of multiple people gathering in a single frame. Another tool available is YOLO [22], which is an object detection tool that can be adapted to detect humans instead of trying to detect body parts. This tool is used for object detection but may be used solely for human detection.

2.2.2 Social Robot Navigation

In recent years, social navigation methods have been explored more deeply because robots are becoming increasingly popular, especially around humans. With the increase in popularity of ML, learning-based methods and frameworks have been published to improve the existing ones. An example of a learning-based method is the Crowd-Robot Interaction (CRI) [23]. This work attempts to not only consider Human-Robot interactions, but also Human-Human interactions, even though it focuses on the first kind of interactions. These are then turned into weights, which will be fed into a Multilayer Perceptron and return an estimate of the possible state values based on the crowd, and the weights from the interactions. Afterward, there was an attempt at introducing different weight definitions into this model instead of the interaction weights [24]. This attempt used the previously mentioned SFM and ORCA, but it was not successful in its attempt.

Even though learning methods have started to get traction regarding social navigation, attempts at building mathematical models have not stopped. A model was published that considered proxemics and the grouping of people together to create a human-density function alongside space affordance [25]. Space affordance encompasses areas related to human activities, which means that people may gather around that area, resulting in potential areas to avoid. Another framework is called Social Momentum (SM) [26] [27]. This model tries to create pairwise relations between the agent and the people around it, and in case there is someone with conflicting paths when trying to reach the goal, the agent tries to detect the side that other actors try to take to not collide and reinforce that intention by adapting to that change. An example of a decision process of the SM algorithm is displayed in Fig. 2.4.

Another non-ML approach is Dynamic Window Approach B (DWB) [11]. This approach is the successor to the DWA mentioned in 2.1.2. The DWB, instead of using a closed set of weights along with their respective metrics, adopts a *critic* approach. A critic is responsible for calculating the performance of a given trajectory according to a certain constraint. Each critic has an associated weight, and the higher the weight of the critic, the more influence it



(a) Despite the ambiguity, the robot detects a slight inclination towards the right hand side.

(b) The robot's *legible* maneuver resolves the ambiguity and reinforces a right hand side passing.

Figure 2.4: Example of Social Momentum decision process [27].

will have on the robot's trajectory. This means that the core is the same, yet the architecture used in code is more flexible because it allows the developer to add critics that fit the scenarios that are being studied and remove the critics that are thought to be not as important for that same scenario.

Finally, the Timed-Elastic-Band (TEB) algorithm [28] is built on top of the elastic band algorithm, mentioned in 2.1.2. This new approach tries to find a balance between the compression force of the elastic band, which forces the path to be the shortest possible, and the repulsive force, which is generated by the obstacles and is responsible for keeping the path at a safe distance from the obstacles and the fastest path possible. It may not be clear at first the difference between the fastest and the shortest path, but in some cases, these can differ. A smaller path with a lot of changes in direction may be slower than a slightly longer path with a smaller number of turns. This phenomenon is displayed in Fig. 2.5 where the black sections are obstacles and the white area is free space.

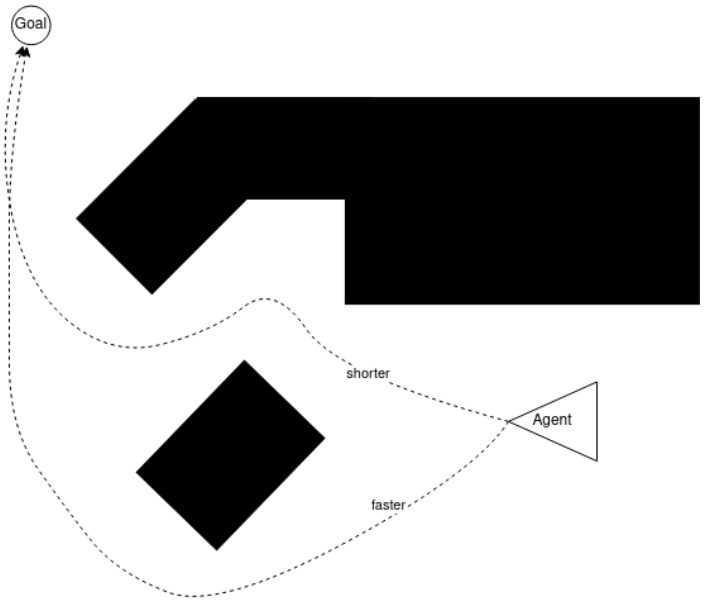


Figure 2.5: Example between two paths, one shorter and another that can be faster depending on the agent's characteristics.

2.2.3 Disinfection-Oriented Navigation

The disinfection-oriented navigation scenario has only recently increased in popularity because no previous emergency required that specific kind of navigation. The COVID-19 pandemic changed that scenario by creating a state of emergency. What sets this navigation task apart is that it is not limited to getting to goals, it also has to make sure that the disinfection process is correctly performed, affecting the global navigation process. At first glance, a mobile robot carrying UV lights can be obvious, but a variety of static UV disinfection lamps can be placed strategically to perform this task. A study [29] tested this possibility by setting a route for a robot to follow and comparing it with the static light at key points, and found that using a disinfection-oriented navigation algorithm performs better when compared to the static UV lights according to their benchmark. However, that test also points out the need for an improved navigation algorithm to tackle cluttered environments. To test the dosage output, markers were placed throughout the environment and the absorbed energy was measured, showing that dosages with the moving disinfection system had, on average, a higher energetic dosage with some exceptions being when the markers were close to the disinfection rig.

Ultra-Violet Disinfection

The light that is to be used in the context of the GermiRRad is a UV-C light, considered the most germicidal wavelength range Fig. 2.6. Even with the wavelength of the light used for disinfection, other variables play a role, such as the distance of the light source, the exposure time, and the dosage that the virus can withstand. The COVID-19 virus can take 10-20 mJ/cm^2 using a direct light of 254nm of wavelength according to [30], and 5 mJ/cm^2 according

to [31].

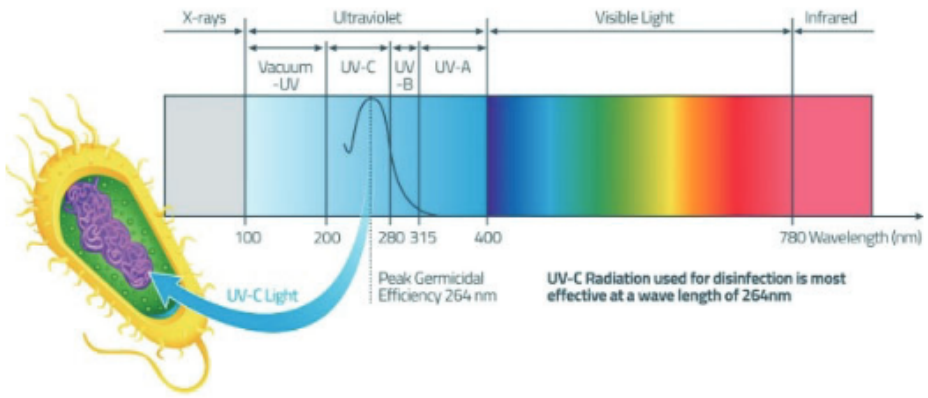


Figure 2.6: Light Spectrum and Respective germicidal wavelength [32].

Disinfection Scenarios Description

Scenarios will have different population densities throughout them. This creates a heterogeneous environment, not only when it comes to the map of the area but because some areas will, on average, be more populated, creating areas where a person is more prone to being infected. Also, if a region is occluded, that area will not be properly disinfected. This can be caused by obstacles, walls, or even people Fig. 2.7.

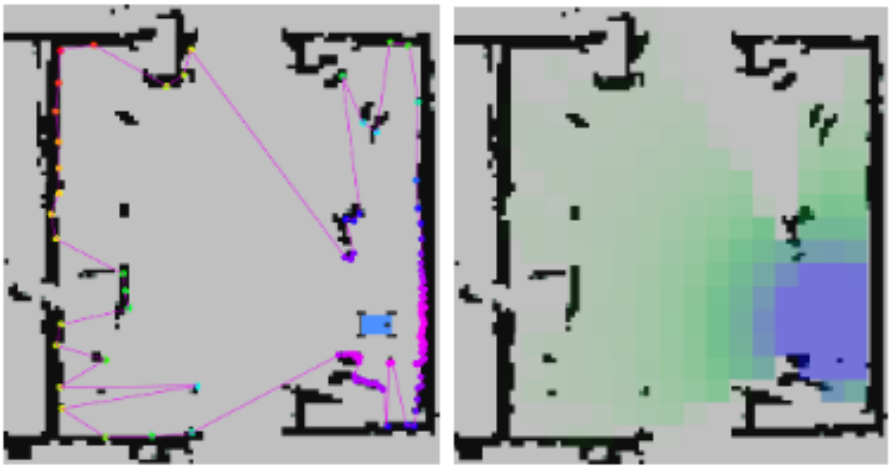


Figure 2.7: Example of visible points in the LiDARs (left) and respective disinfection dosage (right). [31].

A way of defining areas with the highest contamination probability is by using object/area affordance [33] [25] [34]. This method defines areas of interest based on human behavior concerning objects or specific sections, which can create areas that should be prioritized. Another way of doing so is by detecting humans close to each other and crowd density as the agent moves to the defined waypoints [35]. Upon this detection, a heatmap is generated, and the detections are represented as Gaussian functions. The Gaussian functions, depending on

the number of people, can represent a space with a high risk of contamination. Consequently, disinfection should be performed in areas with higher priority.

2.3 Tools

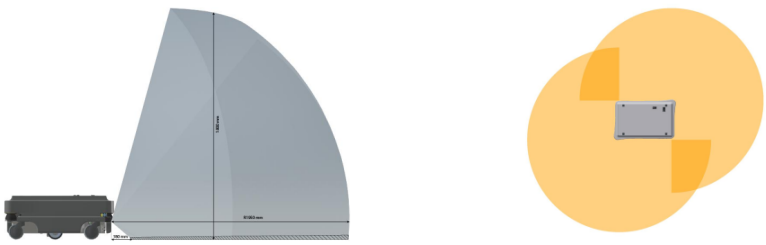
This thesis requires some base tools to ease the development process. These tools are the robot itself, the programming tool, and a driver that serves as a bridge between both the robot and the programming tool.

2.3.1 Robot Specifications

The MiR100 is a robot designed for indoor navigation capable of withstanding up to 100kg [4]. Sensor-wise, the Mir100 has two safety laser scanners, two 3D cameras in the front, and four ultrasound sensors. The robot is equipped with two 2D LiDARs with a 270º coverage angle. Combining both sensors makes it possible to get a 360º view of the robot's surroundings, as shown in Fig. 2.8b. The robot also has two 3D cameras in the front with a 118º angle horizontally. The cameras can detect objects at a distance of up to 1950mm ahead of the robot and 1800mm up and are strictly for navigation purposes, as they are not involved in the safety system built into the robot. The information from the 3D cameras returns 3D pointclouds. Besides laser scanners and 3D cameras, the robot is also equipped with four ultrasound sensors. Two of the sensors are placed in the front of the robot, and the other two can be placed in the back or front. Depending on the location of the sensors, the sensor distance limits differ.

The robot can also locate itself on the map. This localization requires the robot's initial position, which is used as a reference point, and the data from both the Inertial Measurement Unit (IMU) and encoder to determine the position and speed of the robot relative to the starting position. Finally, with all this data and the laser scan data, the MiR100 uses a Particle Filter to determine the likelihood of the robot's position by comparing the laser information with the mapped nearby walls.

Regarding navigation, the robot is equipped with a safety obstacle avoidance system. This system is based on laser sensors. These laser sensors, along with the robot's speed based on the data from the encoders, form Protective Fields, and if the lasers detect the presence of an obstacle within the defined Protective Fields, the robot stops.



(a) 3D camera sensor ranges. **(b)** Placement and angles of LiDAR sensors.

Figure 2.8: Representation of built-in sensor ranges. [4]

2.3.2 Robot Operating System (ROS)

ROS is a widely used open-source tool that intends to simplify and ease the development of robotic agents [36]. This tool presents a way to deal with the asynchronous nature of sensor data in real-world environments by providing the developers with a "middleware" between the raw data that is provided by the robot's sensors and the developer. ROS is a tool that uses a publish/subscribe pattern. This pattern promotes modular code since each node will only treat data from nodes that they are subscribed to, which makes maintenance and contributions a simpler process. It also makes reusing code a simple task, as this "middleware" allows for the creation of isolated code sections that work independently from the system and can be reused in different projects. The nodes and topics that were just mentioned are core components of ROS. Nodes are the blocks that process information and contribute to the actions that the agents perform. To process information, the information has to be fed to the nodes. This data input is performed through topics. Topics are the connections between the nodes and the structure that is responsible for making information to the nodes that need it. For topics to share information, nodes publish messages through them, and the nodes that are subscribed to that same topic receive those messages. These messages are structured pieces of information, as each type of message is formed by a set of fields that are published to and received from the ROS topics. An example is the *pose* message from the *geometry_msgs* module, which is composed of a *position* and an*orientation*, which are both associated with their respective ROS message. However, topics are not the only form of communication. ROS also uses services. These services are structures that accept a certain message type, and after processing the data, they return an answer. Services follow a request/response structure, which means that every request that the service receives produces a response that is guaranteed to be relative to the correct request.

Besides the structural features from ROS, ROS also has another key component for debugging and testing. To help with that, ROS has a module that allows one to record data and replay that same data with little to no changes consistently, which allows the replication of scenarios that may only occur sparsely. In cases where work depends, for example, on data from sensors, a rosbag can be used to test if the program is working properly without having to spend time connecting to the robot itself since test data has already been recorded in a file. This data is stored in and played from rosbags, or files with the *.bag* extension, which is a format specific to ROS.

Rviz

Rviz is a ROS-based visualization tool that has access to the ROS topic information among other information, such as the robot transformations, and is capable of displaying such information in a 3D user interface. Topic information extrapolates beyond what is displayed in the simulator. The ROS topics publish sensor data, paths, and waypoints, among other types of data. This is useful for the researcher because it allows them to perceive the scenario considering other pieces of information that are not available either in the real world or in simulated environments. When dealing with sensor data, even if the data being analyzed

corresponds to a short interval, because of the high refresh rate and because each datum has hundreds or thousands of values, depending on the type of data, analyzing such data solely through logs becomes a tedious and challenging task. Thanks to the visualization capabilities Rviz offers, it becomes easier to debug robot code when compared to debugging based solely on logs because it allows one to display invisible information in both real-life and simulation scenarios visually. Besides being able to display information, Rviz is also capable of publishing to ROS topics. An example is navigation goals. Rviz allows the publication of navigation goals that the robot tries to follow, which makes it a versatile tool for tests.

rqt

rqt is a tool within ROS and is built in Qt, which is a tool used to develop graphical interfaces. The rqt visualization tool allows the use of a wide range of tools in one place. Some can help understand the architecture of the project or just a section of that same architecture, while others allow one to change components' configurations in real-time or even send instructions to the robotic agents being tested. The rqt tool, by itself, does not allow one to perform any of the actions mentioned above, but by associating plug-ins, such functions become possible because of the core program that is rqt.

2.3.3 MiR package for ROS

MiR does not provide a native API in ROS, yet it provides a REST API to connect devices to the MiR100 robot. For this reason, to use the MiR100 robot, a third-party driver is needed to map the REST API requests to the topics in ROS. To perform this task, the *mir_robot* package is used [37] allowing us to explore the robot's functionalities in the ROS environment.

2.3.4 Gazebo

Gazebo is a 3D simulation tool that is built with open-source libraries that can serve multiple purposes, from the simulation itself to a physics engine, and is mainly used by designers and developers. This is the case because Gazebo can use 3D models, which are useful to create environment representations in detail which is usually enough for design purposes as the 3D representation of the environment makes visualization easier.

In the context of this dissertation, Gazebo is used as a simulation tool because of its simulation capabilities. Gazebo supports a set of physics engines that simulate real-world physics in detail, which is crucial when considering that the focus of this dissertation is the navigation of a robot that must consider the robot's dynamics and behavior and be able to simulate the behavior of the sensors in the robot according to their location and the obstacles that surround them. Lastly, it is compatible with ROS, which is the core of the dissertation's implementation. Just like ROS, Gazebo is an open-source tool developed and maintained, for the most part, by Open Robotics.

CHAPTER 3

People Detection and Tracking Based on LiDAR data

As mentioned in Chapter 2.1, because of the MiR100's sensor configuration, the people detection and tracking will be performed based on leg information gathered from the LiDARs. After the decision relative to the sensors that were used, the first step is to understand how the LiDAR data is published to ROS. The LiDARs are placed at a 20cm height and have a perception of 360º around the MiR100 and publish the sensor information to two different ROS topics, not necessarily, simultaneously. Because of the height of the sensors, if there are people around it, the LiDAR will detect people's legs. For that reason, the people detection process must be based on leg shape, which should allow us to perceive where people are in the environment. To adapt the people detection algorithm to the LiDAR configuration, research was made to find a method that would suit this configuration.

When searching for people detection and tracking algorithms, a ROS package built in the context of the ICRA 2015 conference [6] was tested. The package was tested by using the datasets that came with the package itself, and the results were promising since accurate detections and tracks were formed when visually analyzing the data in Rviz. To determine if the package was suitable for other sensor configurations, the developers of the package suggest training the model again with new data, and, if needed, adapting the parameters according to the sensor characteristics.

To train and test this package, the following steps must be followed:

- Label and extract positive and negative data from datasets.
- Train a model with this labeled data.
- Test the performance of the model on different data.

Fig. 3.1 highlights the tool's structure for human detection and tracking.

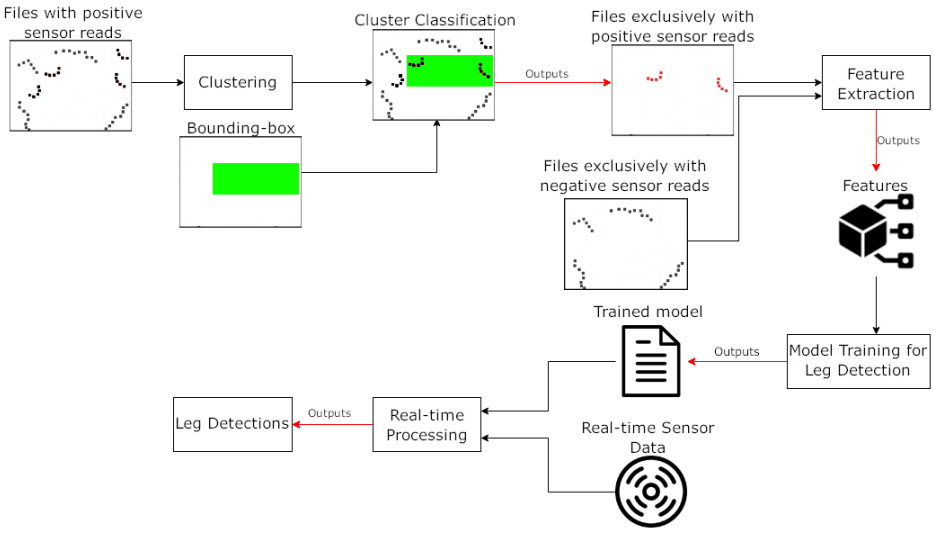


Figure 3.1: Detection process when using the leg detection tool [6].

3.1 Implementation Using Default Machine Learning Models

3.1.1 Datasets

The data classification task is performed on rosbag files, but different robots have different sensor characteristics, and researchers may use different tools that use other file formats. Before training the model with the data from the MiR, other datasets were tested [38]. To test these datasets, they had to be converted to rosbags using a package [39]. After converting these datasets, it was found that they were not suitable because there was no data without people in the scenario to train the model. Another issue with the data is related to the bounding methods that the package offers to establish the boundaries between the positive and negative data. This issue is addressed in the next Section (3.1.2). For these reasons, to test if the algorithm works correctly with the robot's specifications, bag files were recorded with the MiR to train and test the algorithm using data recorded by the actual robot. Initially, three bag files were recorded to train the model, one without people and the robot in motion, and two with people around the robot and the robot stationary. Later, more files were recorded to balance the data and increase the variety for each label, so that the ML models are trained with data that encloses a greater variety of information.

When recording the data from the MiR100 sensors, the LiDAR data was published by two different sensors in two different ROS topics, which creates an issue when using the *leg_tracker* package, since the package only accepts one topic as input both for training and after training in live scenarios. To bridge this gap, the *ira_laser_tools* [40] is used. This package allows the merging of two topics into one topic with a distinct name so that there are no name collisions between active topics. To merge the laser scans, the origin of the lasers is slightly changed so that all points of the scan have the same origin point. Because of this change, some points from the resulting topic may not be exactly in the same position as the original scans, but the difference is small enough to be negligible.

3.1.2 Data extraction and labeling for model training

To extract labeled data from the bag files, the authors of the article [6] suggest using scenarios where the robot is not moving and with people around it to extract data labeled as positive, and to record data without people around to extract negative examples with no leg detections. The next step is to process the data so that it is labeled properly. The original code from the package in charge of extracting positive data from the bag files would only extract the clusters from within a bounding box that the user had set.

The original package implementation was found to have two issues. The first is relative to the bounding box because it could only be defined by the maximum and minimum values of the x and y axes or through the angles of the points in the scan, with both being limiting ways of extracting labeled data. By using the option to provide the minimum and maximum x and y coordinates, the area from which the clusters labeled as positive for training and testing would necessarily be a rectangle. Additionally, the rectangle would not be adjustable angle-wise. This is not ideal because it is challenging to create a bag file where only legs are detected within that bounding box. The alternative built-in method is to provide angles where only legs are present in the scans, which may be even harder due to the gaps between legs, which could be considered legs but are not. To overcome these data labeling problems, another bounding method was implemented. The new method accepts x and y coordinates to form a polygon and check if the clusters identified in the scan are inside or outside that polygon. This method is more flexible than both bounding methods built into the package.

The logic behind the algorithm used to understand whether a point is within the defined polygon is: A straight horizontal line is drawn through the point being classified. From there, the algorithm only considers the intersections to the left or right of that point. If the number of times this line intercepts the sides of the polygon is odd, that point is inside the polygon. This is highlighted in Fig. 3.2. This also works if the line being drawn is vertical or in any other direction, but the easiest way to perform the calculations is by using horizontal or vertical lines.

The code used to implement the algorithm is adapted from W.R. Franklin [41]. The configuration below shows how to configure the extraction node when using the new bounding method. The difference is that to use the new bounding method, the program receives the parameters *polygon_x* and *polygon_y*, which are the coordinates that define the vertices of the bounding polygon.

```
<node pkg="leg_tracker" type="extract_positive_training_clusters"
name="placeholder_name" output="screen">
    <param name="load_bag_file"
    value="$(find leg_tracker)/file_location.bag"/>
    <param name="save_bag_file"
    value="$(find leg_tracker)/file_location.bag"/>
    <param name="scan_topic" value="/topic_name"/>
    <param name="laser_frame" value="example_link"/>
```

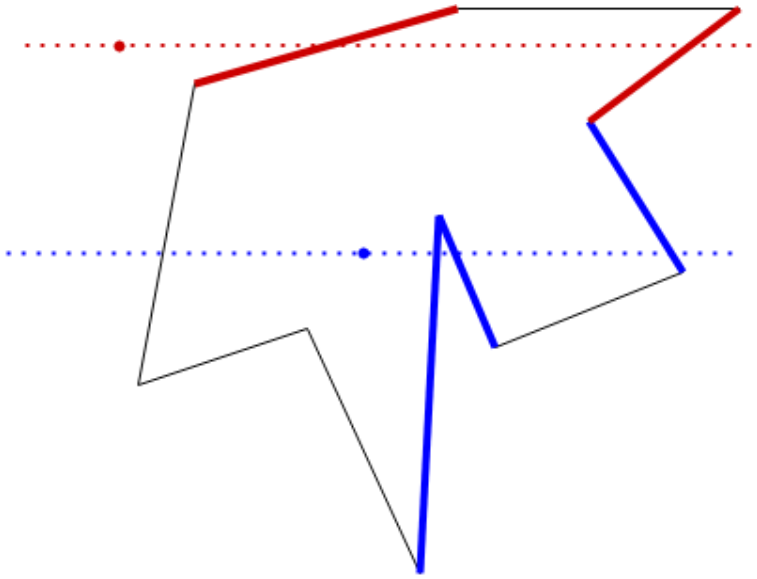


Figure 3.2: The red elements show an example of a point outside the figure, and the blue elements show a point inside the figure.

```
    <param name="tf_data" value="/tf"/>
    <rosparam param="polygon_x"> [6,-0.7,-7.5,1]</rosparam>
    <rosparam param="polygon_y"> [-1.8,-10,-2,6.5]</rosparam>
</node>
```

The second problem may not be as evident, but, depending on the granularity and the bag file itself, it may hinder the training of the detection model. These problems become more prominent, especially when working with the new bounding method because it can define more detailed areas. The problem is that, when considering large clusters, their centroids can be far from the points themselves, and, therefore, when creating the bounding box. the cluster may be mislabeled, which affects the detection's performance. As displayed in Fig. 3.3, the cluster's centroid is distant from the points that originate it and because the bounding box is defined close to the points gathered from the scans, the cluster is classified as a positive cluster because its centroid is within the bounding box even though not a single point from the scan is within the bounding box. If instead of using clusters the scan points are considered, the classification for scenarios like the one in Fig. 3.3 becomes correct.

To solve this problem, instead of clustering and, only then, deciding if the clusters are inside the bounding box or not, the new algorithm goes through each point from the scan and decides whether or not that point is within the bounding box. After iterating through all points, the positive clusters are gathered. The points in clusters and within the bounding box are saved along with the cluster's centroid, just like in the original version. As a result of these changes to the bounding-box definition, the data extraction becomes more accurate compared to the original work.

To capitalize on this change, besides using the files with exclusively non-leg detections as a data source for negatively labeled data, negative data can also be extracted from the files

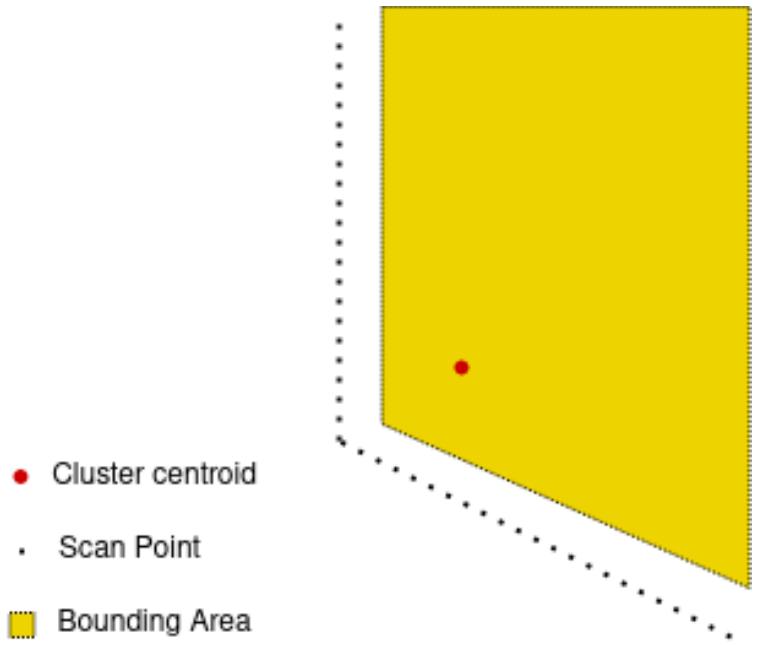


Figure 3.3: Representation of bounding problem relative to clusters with many points.

from which the leg detections are extracted. When extracting leg and non-leg clusters from the file, the extracted scans are stored in the same rosbag in their separate topics so they can be distinguished when training and testing the models. These files with negative examples can be used to train the model alongside the bags of the moving robot with no people nearby, as suggested by the developers of the original package.

With the changes performed to the structure of the leg detection algorithm, the architecture changes from what is displayed in Fig. 3.1 to the one evidenced in Fig. 3.4

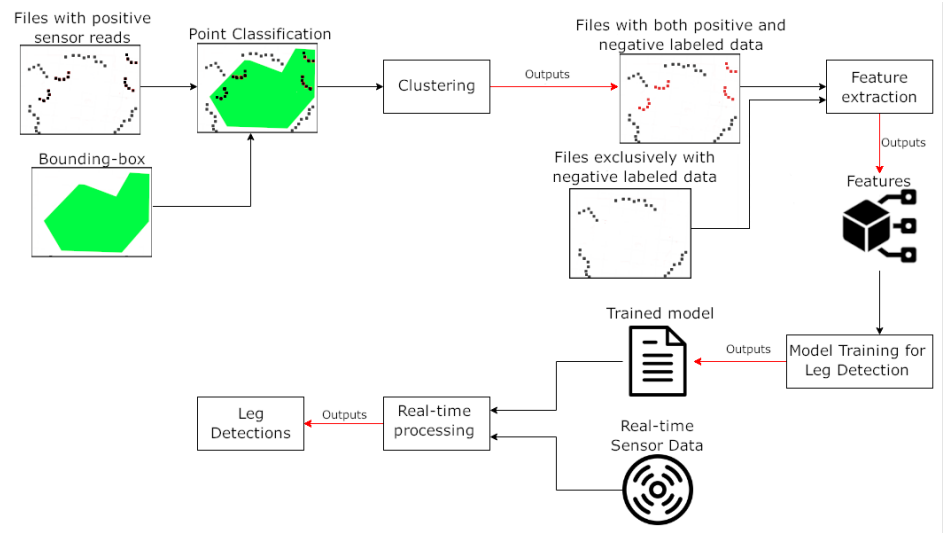


Figure 3.4: Leg detection process after adaptations made to the leg detection tool [6].

3.1.3 Default Model Training

After the data extraction and processing, this data needs to be fed to the ML model that is to be trained. The first step is to provide the default model with the positive cluster's positions and the scans, both positive and negative. To do so, the rosbags are passed along with the ROS topics to the script that trains the ML model as arguments. The rosbag, along with the topics, is passed after one of the following arguments: *–pos*, *–neg*, *–test_neg* or *–test_pos*. These arguments are used to understand if the data from the topic is positive or negative and if it is supposed to be used for training or testing. Upon receiving the data, the entire bag is used according to the associated argument. The data from the bag is processed to extract geometric features, which are then used to train or test the leg detection model. The leg detection model used in the original implementation is a Random Forest Finally, the ML model is stored in a *.yaml* file so that it can be used to perform people detection and tracking in real time. When the original code from the *leg_tracker* package is used, the leg classification is performed directly in that same file, which ensures that there is little delay regarding the classifications being made.

3.2 Implementation Using Alternative Machine Learning Models

Even though the examples present in the package have satisfactory results, there may be alternative models that result in better outcomes. To understand if that is the case, other methods were tested. The following sections address the changes needed to test such ML models. These changes are highlighted alongside the alternative models being tested.

To feed the data from the rosbags to these ML models being tested, the script responsible for cluster classification was adapted to save the data from the rosbag into a CSV file that has the data organized and already labeled. This process makes it easy to manipulate the data used in the training and testing of the rest of the models.

3.2.1 Alternative Machine Learning Model Training

Even though not much work is needed to use the original implementation, there may be better solutions for this classification problem. With that being considered, some alternatives were explored to understand if any would perform better than the original. Before starting the development, the first question to be addressed was which tools to use and what type of model should be implemented. There is a wide variety of tools commonly used for data classification using ML, such as TensorFlow, Scikit-Learn, and PyTorch. In theory, when using the same parameters for the models, the difference between results by implementing one tool in favor of another is negligible. From these tools, TensorFlow was used due to familiarity, and Scikit-Learn was also used because it is a high-level tool for ML development. One of Scikit-Learn's main objectives is to be accessible to the general public and for that reason, it did not require much insight in the development of ML to use when compared with the other solutions. The methods were developed in an attempt to improve the original work. The different types of ML models developed besides the already implemented Random Forest are: CNN, NN, and SVM.

For these ML models to be used, some changes have to be performed because ML libraries are not ready to receive the same type of data as the original implementation. To feed the data from the rosbags to these ML models, the script responsible for cluster classification was adapted to save the data from the rosbags into a CSV file that has the data organized and already labeled. This process makes it easy to manipulate the data used in the training and testing of the rest of the models.

CNN (Convolutional Neural Network)

CNNs are a type of Neural Network that usually receives raw data as input and is most commonly used in problems that require image-related processing. CNNs are composed of an input layer, an output layer, and at least one convolutional layer and are commonly followed by pooling and normalization layers. Convolutional layers work by setting up a kernel that iterates through each datum to extract features that will be used to classify the datum. In the example displayed in Fig. 3.5, the convolutional layer is processed by a 3x3 kernel that iterates the entire input data to generate the output.

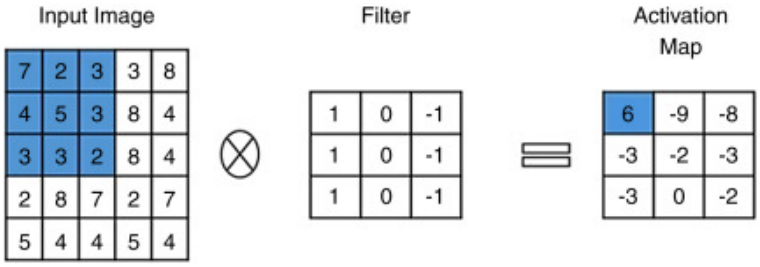


Figure 3.5: Example of a Convolutional process [42].

The CNN was tested with the cluster features as training data, which means that the CNN is extracting features from the arbitrary features extracted from the data. The main issue with this approach is that the features extracted from each cluster are not ordered in any specific manner. Even if they were, the features do not have an inherent order to be arranged in, which means that by swapping the order of the features in the dataset the results could change the outcome drastically. Each feature relates to the other because it refers to the same cluster, yet there is no way of knowing if the feature order is optimal.

Even though theoretically CNNs may not be suitable for the problem being tackled, the method was not discarded. Instead, it was tested to see if the practical results would converge with the theoretical when it comes to the relationship between neighboring data.

NN (Neural Network)

The Neural Network is the basis of Deep Learning. It has at least three layers: the input, the output, and at least one hidden layer. This hidden layer can be fully connected or partially connected. In some cases, it could also be a pooling or normalization layer, the most common being the first two examples. In Fig.3.6 The Neural Network model uses Dense Layers to attempt to connect every neuron from the previous layer to the current layer. The number of neurons depends on the problem's complexity and the network's architecture. Just like CNNs,

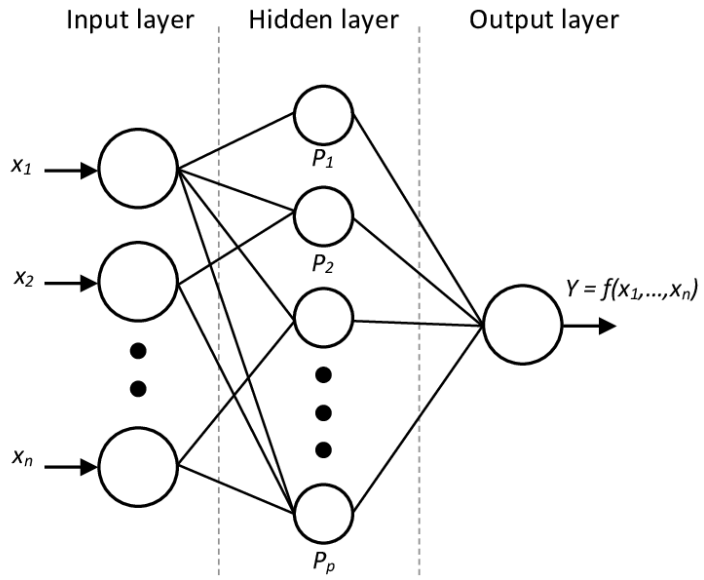


Figure 3.6: Example of Neural Network with dense layer [43].

neural networks are sometimes complemented with pooling layers, normalization layers, or both.

SVM (Support Vector Machine)

The last method tested was the SVM model. A Support Vector Machine is a model that may be linear or nonlinear that identifies the optimal boundary to separate data into distinct classes by generating a maximized margin between these classes. To achieve this, the data is transformed to be defined by a higher-dimensional environment that allows the classification of the data linearly or through a hyperplane. The classification depends on the dimensions of the environment.

To improve the SVM model, three hyperparameters can influence the model's results: the kernel type, the C value, and the *gamma* value. The kernel type changes the formula that will perform the classification. The C and *gamma* values must be adapted to reach the best possible level without losing the ability to generalize the classification in case new and different data is to be classified. If the data does not present a clear division, a smaller C value is better because regularization will increase, instead of relying solely on what the data shows. If the opposite is true, the C value should be higher to allow the results to be more influenced by the dataset. Another relevant parameter when optimizing SVMs is the *gamma* value. The *gamma* value determines how many pieces of data will influence the training of the SVM. A visual representation of the results when using different *gamma* and C values is shown in Fig. 3.7.

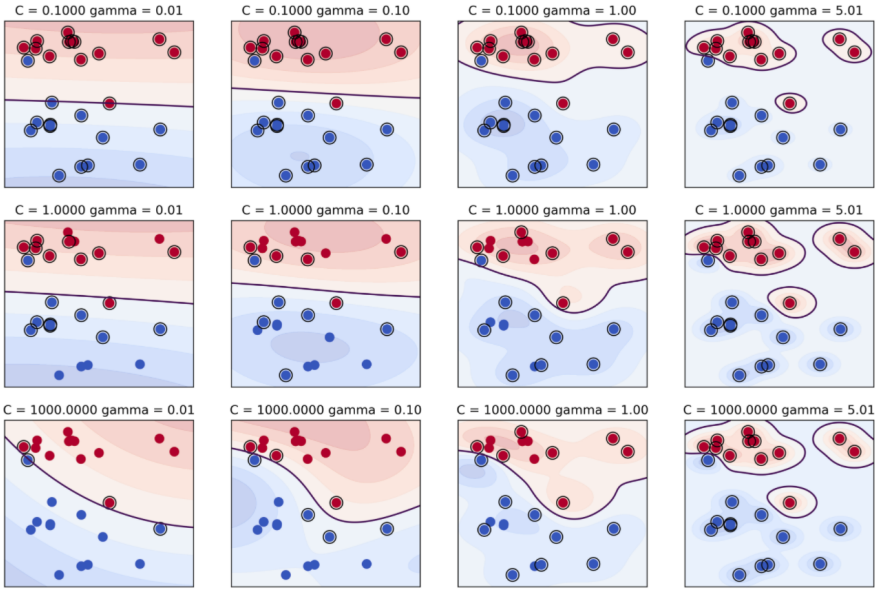


Figure 3.7: Example of how the *gamma* and C values affect SVM results. [44]

3.2.2 People Tracking

After detecting the different people, or, in this case, the people's legs, the next step for building a social navigation framework is people tracking. In robot navigation, people tracking consists of understanding how a person or multiple people evolve in the environment where the robot has been inserted. This may include a person's position but also considering that the person is moving, the movement it is performing, and, if possible, estimating where the person will move afterward. This information is relevant to the robot because it helps decide which movements are most appropriate in social scenarios.

The people tracking code section is closely coupled with the leg detections mentioned in Chapter 3 and alongside the mapping information. In Fig. 3.8, the architecture of nodes and topics needed to perform people tracking is evidenced. As it is shown in Fig. 3.8 the topics */map* and */detected_leg_clusters* are the ones that are provided to the *joint_leg_tracker* node, which is responsible for people tracking based on the legs detected, with the information needed for it to operate.

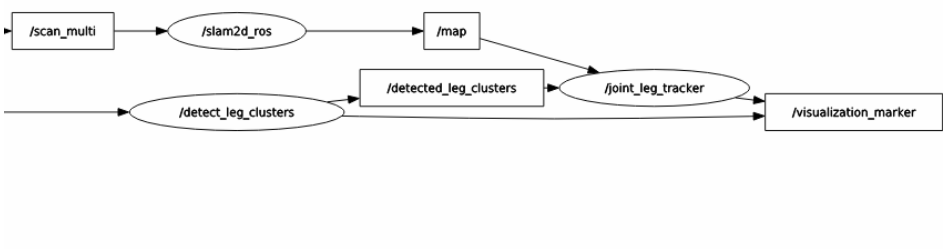


Figure 3.8: Nodes and topics directly related to the tracking process (rectangles are topics whereas ellipses are nodes.

For the implementation, the package used for leg detection [6] also has built-in people tracking. This section of the implementation does not rely directly on sensor input or in ML models that must be trained. For these reasons, and since the communication between these two sections of the package is already made, there was no need to adapt the code relative to the connection between the leg detection and the people tracker or the algorithm for people tracking itself.

3.3 Adaptations to communication modules

For the tracks to be available in the navigation module, data needs to be published by the people tracker in a different way because of a discrepancy between the tools used regarding the type of message that the people tracking algorithm publishes and the navigation package [45], which will be explored in Chapter 4, receives. The navigation package receives a *People* message that is not being published by the people tracking module by default. To fix this problem, *Person* messages from the *people_msgs* package [46] began being published so that the navigation section could use the data from the people that surround the robot. Both types of ROS messages are composed of the person's position and velocity, but there are crucial differences. The one from *people_msgs* has a field for the reliability of the person's track and uses the *Point* ROS message from that same set of messages both for position and velocity. The default ROS message published from people tracking uses the *Pose* message from *people_msgs* that uses a *Point* message for the position and a *Quaternion* message for the orientation, which in this case translates to velocity because there is no way of knowing where a person is facing only using a LiDAR. Both ROS messages are published from the node responsible for managing people's tracks onto the respective ROS topics.

Another issue related to communication is that there are two ROS messages within the project's environment with the same name. Both are named *Person*. One of them is from the *leg_tracker* package, whereas the other is needed for the navigation package that is present in *people_msgs* [46] under the *people_msgs/msgs* folder. For that reason, the name of the ROS message from *people_msgs* was changed to *PersonProxemic*, since it will only be used for local planning and because the naming convention continues to make sense even with this change.

3.4 Tracking Algorithm

The implementation [6], when it comes to people tracking, receives as input the leg positions and the map of the environment. The first step is to track every cluster in the environment because it may constitute a leg cluster. Each cluster is tracked with the use of a Kalman Filter which is performed based on its position in x and y along with their velocities through both x and y axes. The person track is also registered in the same way as the clusters. The new tracks are considered static since the laser scan does not provide velocity-related information. After the first track is added, in the second reading, the velocity is updated considering the constant velocity motion model, which estimates the velocity based on sensor frequency and the distance between successive readings of the same track.

To understand if a track from time *t* corresponds to another track in time *t-1*, a Global Nearest Neighbor (GNN) algorithm is used. The GNN algorithm is performed with the aid of the Munkres algorithm, which is used to calculate the optimal combination of propagations and current clusters so that the total distance between correspondences is minimal. A visual example of propagation is displayed in Fig. 3.9.

The GNN algorithm starts by propagating the tracks from time *t-1* to time *t*. This is done through the use of the velocity and the position from time *t-1* to estimate the position it would occupy at time *t* based on the data from time *t-1*. Then, the data from the estimation is put against the sensor readings. The readings are compared using the Mahalanobis distance, and then the corresponding tracks are calculated through the Munkres algorithm, which minimizes the total distance between all projections and all clusters received in the time *t*.

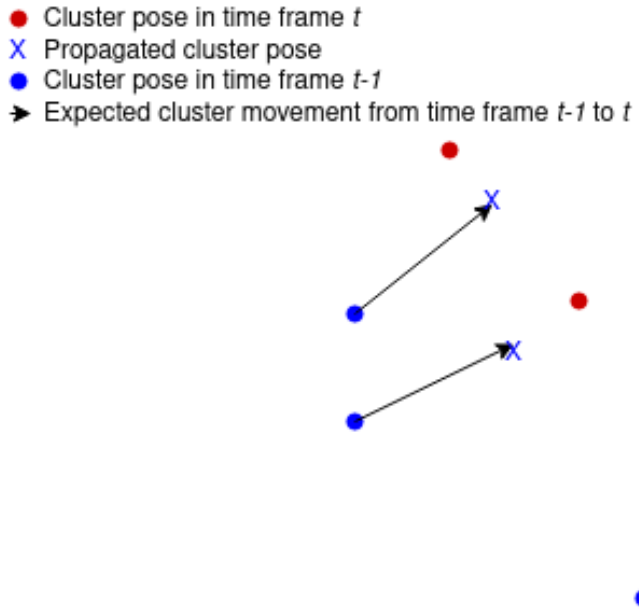


Figure 3.9: Propagation process over which the GNN algorithm is going to be performed for cluster association.

The use of the Mahalanobis distance is important as it calculates the distance between cluster and projection in variance. If the distance between the projection and cluster is null, the Mahalanobis distance is 0, if it corresponds to the variance, it is 1. This means that this method is scale-invariant and can be adapted by adapting the variance of the algorithm.

To create a person track, the detections must follow a set of criteria. The first criterion that needs to be met is the detection of two tracks that have moved at least 0.5m without drifting apart. After that first criterion is fulfilled, both tracks must maintain a level of confidence over a certain threshold. Finally, both tracks must be in *freespace*[6]. *Freespace* is a metric based on the local occupancy grid map. The local occupancy is set to *freespace* until there is a non-human observation.

3.5 Experiments and Result Analysis

To understand which ML model is best, two experimentation phases were performed. Theoretical tests, whose results come from the statistics related to the models, such as accuracy, precision, recall, and F1 score. The second test is the empirical/analytic test, which is used to corroborate the information gathered by the theoretical tests.

In both tests and for all the ML models tested, the files with the features used for training and testing had 17 features, but one of the features would always be 0, so that feature was removed. Besides that flawed feature, other features had issues, such as infinite values and values represented as *NaN*. Because of those flaws, the data that had those problems was removed before the dataset was used for training and testing.

3.5.1 Random Forest

The Random Forest model was already built into the *leg_tracker* package. Even though the method was already available, the authors recommend testing other parameters for the Random Forest because, as sensor characteristics change, the results may vary and benefit from other parameters for the model. The best results were achieved with few changes to the original implementation, namely the maximum categories, which were reduced from 1000 to the default ten from the *OpenCV* package. The other change made was the number of trees in the Random Forest. The number of trees is used as a stop condition for tree generation. This means that if this condition is fulfilled, no more trees are generated [47]. Initially, the number of trees needed to stop tree generation was 100 and became 200, which means that only when 200 trees are formed does the Random Forest stop generating new trees to get closer to the optimal results.

3.5.2 CNN

After cleaning the data, CNNs were tested. The initial tests with a CNN were made to understand how many epochs were needed to maximize the ML model results. Tests were made from 50 epochs up to 200. When using up to 100 epochs, accuracy levels were still increasing much slower than compared to the initial epochs, so more epochs were tested. The number of epochs that got the best results is 150, since when testing with 200, besides taking more time, there was little to no improvement in the accuracy compared to the 150 that got selected. When testing the CNNs, one crucial change was to perform data normalization since a wide range of values can differ in the order of magnitude, from a -16 to a 3.

The best result gathered from using the CNNs in the test set is the one in Table 3.1 and was gathered with the architecture displayed in Fig. 3.10.

Layer (type)	Output Shape	Param #
conv1d_18 (Conv1D)	(None, 12, 64)	384
conv1d_19 (Conv1D)	(None, 8, 64)	20544
flatten_9 (Flatten)	(None, 512)	0
dense_18 (Dense)	(None, 32)	16416
dense_19 (Dense)	(None, 2)	66

Total params: 37,410
Trainable params: 37,410
Non-trainable params: 0

Figure 3.10: Final CNN final model architecture.

3.5.3 NN

The data fed to the NN model was the same as the CNN's and got the same processing except for the normalization. Initially, since the code for data preprocessing was adapted from the CNN to the NN, the process was the same. Yet, tests of the NN revealed that using batch normalization in the NN resulted in a better outcome. Multiple architectures were explored, with more or less dense layers as well as units. Since there are a considerable number of geometric features being given as input, it was thought that increasing the number of units in a dense layer would most likely be the best option. With this in mind, initial tests were performed with a lower number of neurons in each layer, starting with 16 up to 96, and going through 32, and 64. Dropout rates from 0.15 to 0.3, with an increase of 0.05 were tested, but since overfitting was not apparent, no dropout layers were used in the final NN. After testing various combinations of dense layer units, the best results were gathered through an architecture with increasing units in the dense layers with all *leaky-relu* activation functions.

From all the NNs tested, the best result is displayed in Table 3.1 and was gathered with the architecture present in Fig. 3.11.

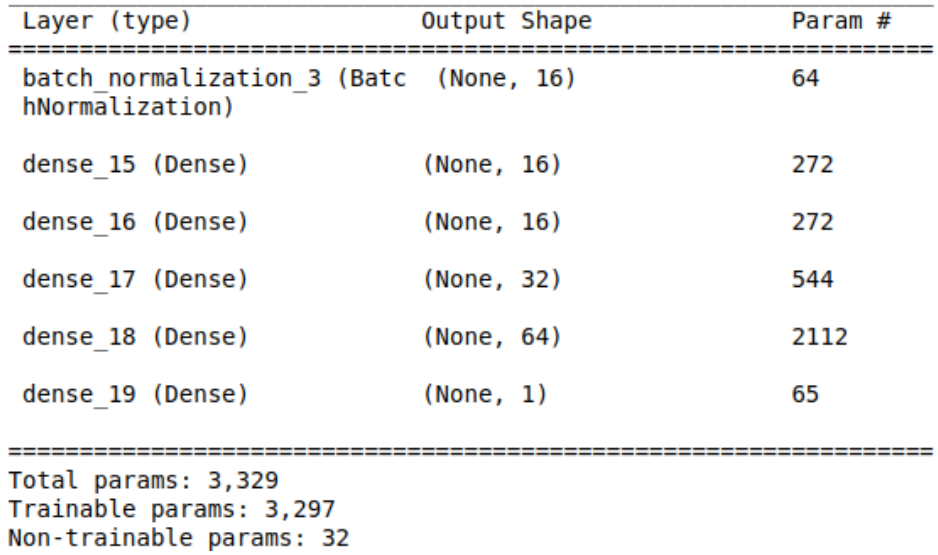


Figure 3.11: Final NN final model architecture.

3.5.4 SVM (Support Vector Machine)

Just like when CNNs are used, the SVM model got the data in the same format and processed it in the same way. The first step is to choose the SVM type. To make this decision, we have to consider that the problem to be solved is a binary classification, and for that reason, SVC was picked. From the different SVCs available in Scikit-Learn, SVC, NuSVC, and LinearSVC, the SVC was picked because LinearSVC is indicated for larger datasets because it is less computationally intensive and, as per Scikit-Learn's documentation, SVC and NuSVC display similar results overall [48].

Upon choosing the ideal SVC, its parameters had to be decided. The C value was the parameter that was explored next, as its value is one of the parameters that influence the results of the SVC the most. The C value is used for data regularization. For Scikit-Learn, the default C value is 1, but as the value was tested, it became clear that such a value was too small. Lower values of C were tested with worse results, and from there tests were performed with higher values of C, starting with C as 1000 and going as high as 3000 in intervals of 500. Out of the tests performed, the best C value found was 2500. The kernel type was chosen based on performance. The performance of the other kernels available in Scikit-Learn was tested with the C as 2500 just like the RBF where the RBF kernel was the one with the best accuracy, as demonstrated in Table 3.1, and the other kernel that got the closest to that value was the polynomial kernel with 96.56% accuracy.

3.5.5 Theoretical Experiments and Results

To understand which ML model is best, out of the ones mentioned, all models were optimized individually. After improving the results from the ML models mentioned previously, the results were gathered and introduced into Table 3.1, so that they could easily be compared. The comparison was made based on which model had better accuracy. However, if the results are similar, other metrics were considered to decide which model was best. The results

displayed are regarding the values obtained in the test sets.

Table 3.1: Scenario with moving people

	Accuracy	Precision	Recall	F1 Score
Random Forest	**99.10**	**99.61**	**99.61**	**99.38**
CNN	96.47	96.47	96.47	96.47
NN	96.94	96.95	96.94	96.94
SVM	96.70	96.70	96.70	96.66

As displayed in Table 3.1, from the models tested, the results from the Random Forest implementation were the best out of the ML models tested with an accuracy of 99.10%. One difference that can be seen when comparing the results from the Random Forest to the other ML models is that this value differs considerably from all other metrics. That may be the case because the data is not balanced, meaning that the data gathered has more instances from one label than the other, which is the result of the data not being treated in the same manner as the remaining models.

The Random Forest model uses the entirety of the topic for its respective functions (training and testing data, which can be labeled as positive or negative). The data from other models is treated differently when compared to the Random Forest. This difference causes the data to be split differently, which may affect the comparison of the results between the Random Forest and the remaining models.

3.5.6 Empirical Experiments and Results

To tackle the issue relative to dataset discrepancies, a test was performed to check how the theoretical results would translate into the real world. To perform this task, another rosbag was recorded. This rosbag was not used in the training or testing of any model. Instead of testing with all the models, only the NN model and the Random Forest were tested because the Random Forest was the one that got the best results and the NN is the model that got the best results out of the models that used a different data split.

This comes with a choice: "What language should I load the NN model in?" This is important not only for testing purposes but also because the NN could, in practice, be a better answer to the problem. This means that not only did it have to work, but it had to be optimized so that the leg classification could be performed within a reasonable amount of time.

The first decision to be made is the language since both Python and C++ could be used. In C++, Tensorflow or PyTorch would need to be imported into the file where cluster classification is performed, to load the model and classify the data according to the features. This would be preferable, but the ML libraries would have to be inside the ROS scope to become accessible, some are not stable, and there was no previous experience with these tools in C++, which may result in complications when implementing the classification models.

The second option is to use Python. By using Python, the C++ program where cluster classification is performed in the original code must connect to the Python code. The solution

was to create a ROS Service that receives the cluster features, classifies the data, and returns the results. When using the ROS Service, a connection must be created and the messages sent get an answer specifically to that message. That way, the service always returns the value corresponding to the request, which is not the case in the regular ROS publish-subscribe architecture. This comes with the problem that the code that calls the classification section needs to wait for that process, which takes time. By maintaining the code structure as it originally was and calling the classifier for each cluster individually, the classifier would run through around 70 scans in 100 seconds, meaning an average of a scan every 1.4 seconds. These and all results were achieved with an Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz. The flow of the service calls needed to perform leg classification using the original configuration is evidenced in Fig. 3.12. Assuming that a person walks on average at 3.5km/h (0.972m/s), a person could move around 1.4 meters from point a to point b, and the scan is still classified at point a, which is not ideal.

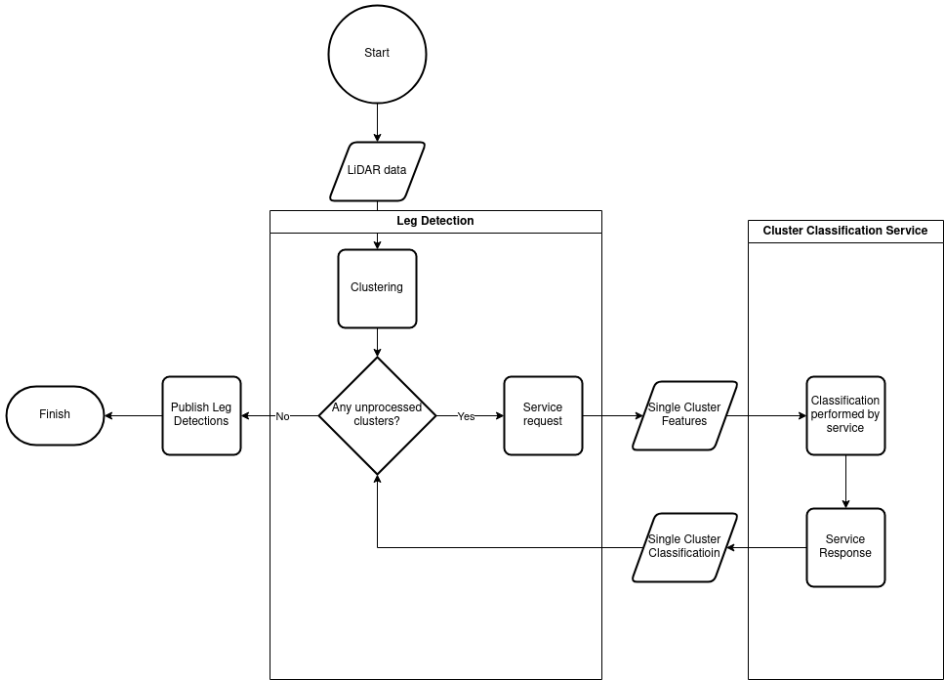


Figure 3.12: Service interaction using original flow.

The solution to the problem is to adapt the code so that the service is called once per scan, sends the features of every cluster from that scan at once, and creates a single connection to the service at the start of the program. Instead of creating a connection for each scan, the processing times became significantly lower, being able to process most of the scans from the MiR100's real LiDARs, which is published at around 12.7Hz, due to the reduction of service calls. With the original implementation, it was already possible to only form a single connection, and for that reason, in both Figs. 3.12 and 3.13 the connection formation is not represented. The flow between the main code section and the ROS service after optimizing the service calls is displayed in Fig. 3.13.

To test the developed ML methods using Tensorflow, the models and the training parame-

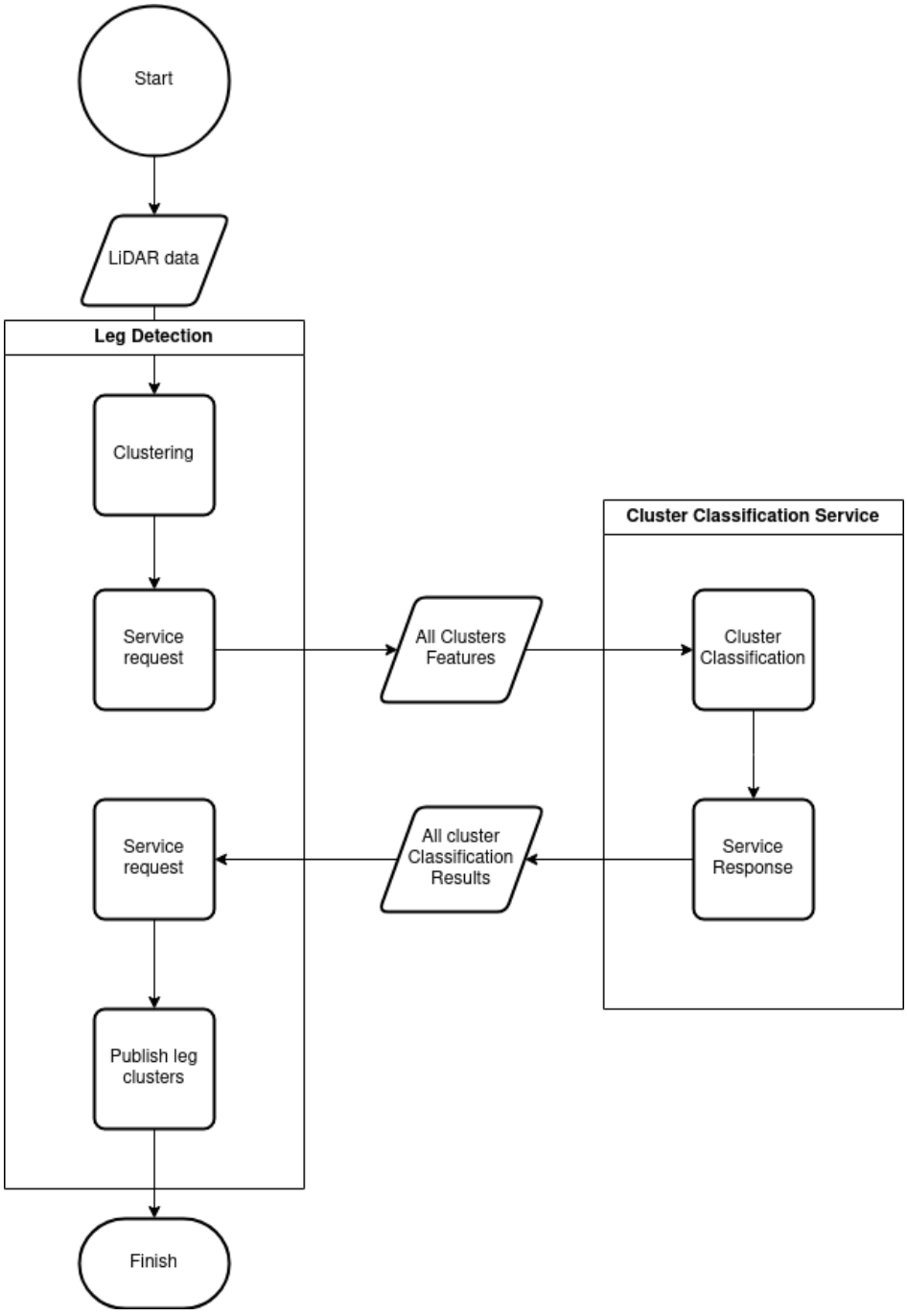


Figure 3.13: Service interaction after changes made to code.

ters were saved after being trained so that they could be loaded to use in run-time.

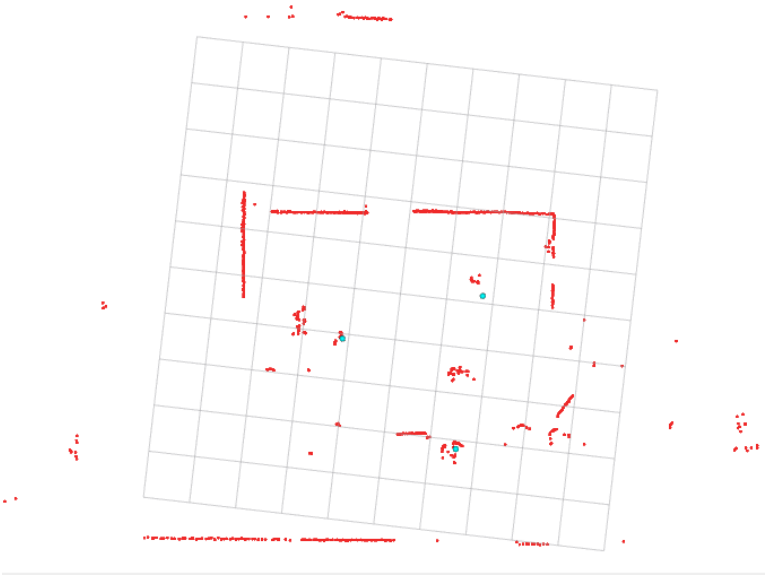


Figure 3.14: Snapshot of leg classification using NN model.

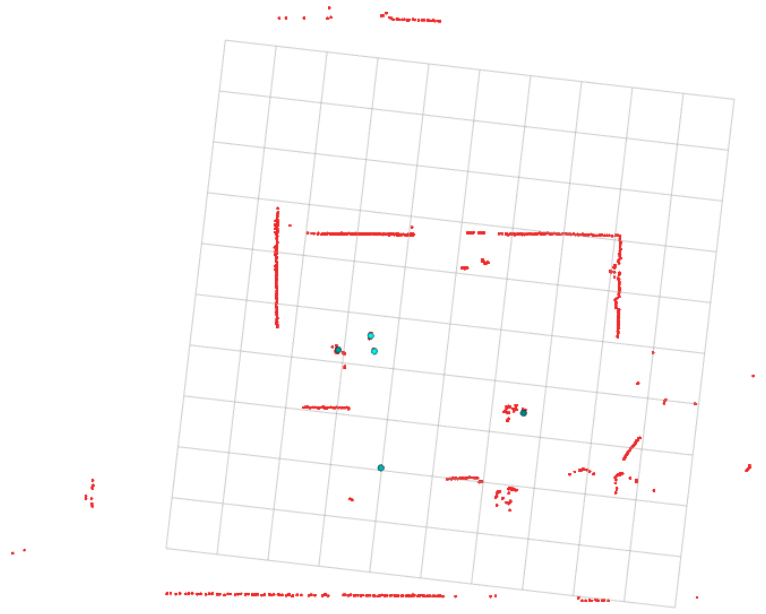


Figure 3.15: Snapshot of leg classification using Random Forest model.

When testing both the NN and the Random Forest with the new rosbag, the results from the theoretical tests were corroborated, with the Random Forest presenting better classification results when compared to the saved NN. An example of both models can be seen in Figs. 3.14,3.15. Even though the differences may not be noticeable when checking the RVIZ visualizations presented, the results displayed in Table 3.2 confirm the information gathered through visualization. This means that the Random Forest classifier, even though

Table 3.2: Model performance on test split

	Accuracy	Precision	Recall	F1 Score
Random Forest	**94.37**	**98.35**	**94.21**	**96.23**
NN	94.12	98.13	94.12	94.24

the results were worse than the ones in Table 3.1, was still slightly better than the best NN configuration.

An important difference between the results from the theoretical and empiric tests is that both reduced their accuracy significantly, with the Random Forest being the one that was affected the most, which means that even though the results were better for the Random Forest, the NN may be capable of a better generalization. However, it still performed better and did not require the use of service calls for leg classification. In this stage of development, it can classify mostly, if not all scans that are supposed to be classified, but as the navigation stack increases, this may impact the stack's performance negatively. If the classifier cannot handle the scans properly at one thing and the Random Forest at the other, it is essential to understand which is more important. Correctly classifying leg clusters or correctly classifying non-leg clusters.

The next experiment that was performed is related to the tracking algorithm. The experiment performed to test the tracking algorithm is a mere extension of the previous test. The difference is that in this test the people tracks were also displayed to understand if they would adapt accordingly to the detections.

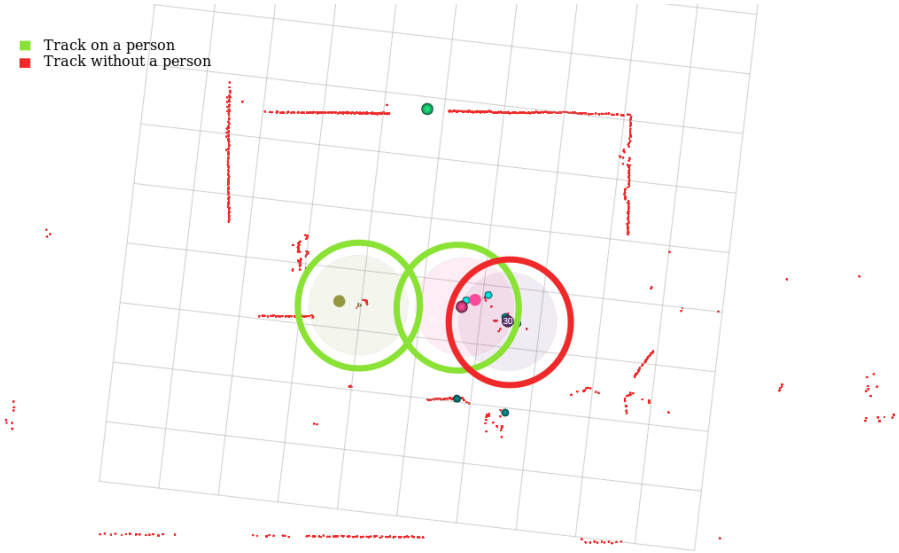


Figure 3.16: Example of tracks created

This tracking method presents satisfactory results when it comes to the position of the human track since it uses the leg positions to decide where the person is at a given moment, as is displayed in Fig. 3.16. Furthermore, the tracks visible in Fig. 3.16 are generated along with detections not too far from one another. Unfortunately, one of the tracks is generated on detections that do not correspond to actual legs, but in that case, the leg detection module is

at fault, not necessarily the tracker.

Even though the positioning of the person is good, the method for velocity estimation has flaws since it uses the constant velocity model, which implies that the person is moving at a constant velocity, which can occur occasionally, but it does not adapt well when the movement is inconsistent, for example, Fig. 3.17, when a person is moving in a curve-like trajectory, the constant velocity model does not perform well. This is true, but even though people are mostly predictable, unpredictable behaviors, like a sudden change in direction, are the ones that can cause the most problems in this velocity model. The issue is that other models would struggle in the same way because more complex models are not capable of dealing with such unpredictability either. This can become an important problem if the robotic agent is moving close to the person because if the robot is moving close to the person, it would have to adapt quickly to that sudden change. After all, if it is not able to react, it depends on humans to adapt its behavior according to the robot's movement, which is unintended. That is where this motion model can be better than any other. Because the constant velocity method is one of the simplest, if not the simplest, motion models, this model will update quickly in response to abrupt changes and therefore have better chances at updating the person's movement fast enough for the agent to adapt its behavior accordingly.

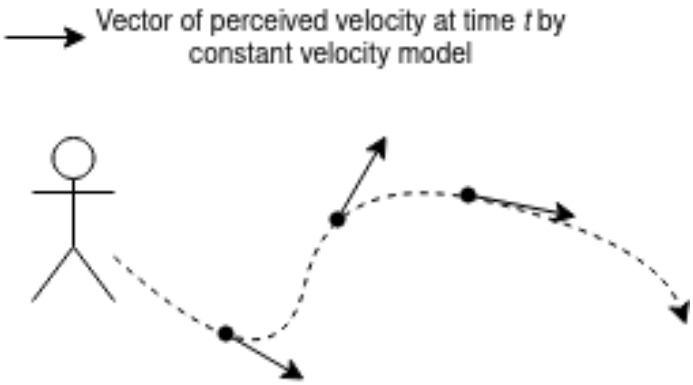


Figure 3.17: Visual representation of what the constant velocity model considers the velocity at a set of waypoints.

CHAPTER 4

Proposed Social Navigation Algorithm

This chapter of the dissertation leans over the information relative to navigation throughout different environments with and without people to understand which approach is best when it comes to social navigation. To perform a good enough job at social navigation, global and local planners and costmaps must be configured, and to understand if the results are satisfactory, a variety of metrics are involved. These metrics range from the distances to people to the total distance of the path and the time it takes for the path to be executed. The development of the social navigation stack is based on the *mir_robot* package [37]. The configuration files for the navigation stack were copied to the dissertation package where all the adaptations will be performed since its code has not been changed.

For the robot to understand its behavior within the environment a SLAM algorithm is required. Most related work uses a variation of Gmapping, but this thesis was developed with the help of the *iris_lama_ros* package, which is a wrapper to ROS [49] of the work from the same authors of LaMa [50]. This package provides a SLAM algorithm with two variants, one based on a Particle Filter whereas the other option is based on a method that tackles the localization problem independently by using only the previous positions and the Dynamic Likelihood Field to obtain its results. From the research conducted, the accuracy of this approach is comparable to other commonly used algorithms with low computational complexity. Since the accuracy results are similar to other SLAM algorithms, the one that uses the Dynamic Likelihood Field was adopted.

Upon deciding what the base for navigation will be, the navigation stack has to be developed. The navigation stack can be divided into two parts. The first is environment representation, and the second is the planning section.

4.1 Costmaps

The first step explored was environment representation through the use of costmaps. A costmap is a method that allows one to determine the state of the environment where the agent has been inserted based on its occupancy. With the help of a costmap, it is possible to understand what regions are empty, occupied, or to avoid. These regions to avoid may not necessarily be close to obstacles, they may represent an area that, even though it is currently accessible, should not be accessed. Depending on the costmap's configuration, we can represent in the costmap different types of obstacles according to the robot's goal and the expected surroundings of the agent. Even though the goal affects the costmap configurations, in most cases, two costmaps are used simultaneously and in a complementary way. The first kind of costmap is the global costmap. Global costmaps will be used by the global planner to set a route to the active goals. This path represents a guideline for the ideal route to reach its goals. However, in non-static scenarios, it can only be a guideline because, generally, it does not consider sensor data. The second map is the local map, which, unlike global maps, uses sensor information to account for changes in different elements of the environment. Another difference is that the global map variant is supposed to represent the map in its entirety, whereas the local map only extends within a given configurable distance beyond the agent, so that the local planner does not consume as many resources.

Even though both local and global costmaps have different objectives, they are formed by overlapping and merging layers. The layers used can change according to the situation, and each layer can be configured to suit the robot's needs.

4.1.1 Costmap Layers

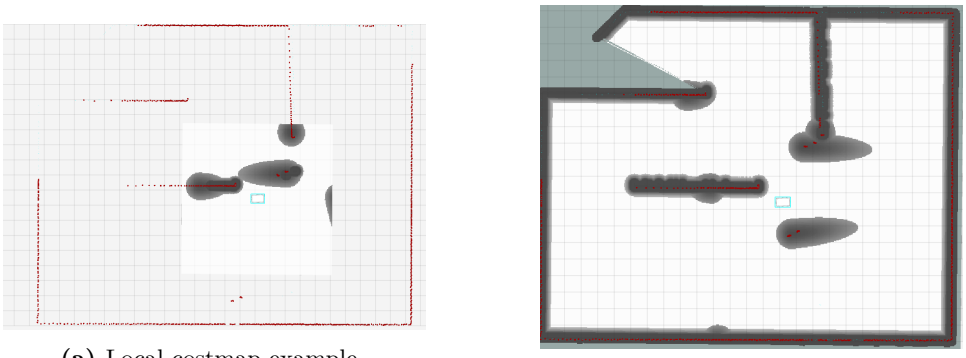
Costmaps are generated through the use of layers. These costmap layers are what allow obstacles to be distinguished.

- Inflation Layer. The Inflation Layer is responsible for indicating that the area relative to the obstacle is not empty.
- Static Layer. This layer is responsible for mapping the obstacles that are, as the name of the layer suggests, static, just like walls and tables. It also creates a region around the obstacle that is likely to be occupied, so that there is a safety margin between the area represented as empty and the obstacle.
- Obstacle/Voxel Layer. Both Obstacle and Voxel layers are similar, with the difference being that the Obstacle layer has two dimensions, while the Voxel Layer has three. These layers are generated through the detection of obstacles in real time through the use of sensors.

These layers are usually enough to represent and navigate environments without people in movement. However, in the context of this dissertation, because of the social navigation aspect, the costmap must also have a way to ensure that people are properly represented to allow for a safe and adequate coexistence of the robotic agent and the people surrounding it.

In the scope of this dissertation, it is fundamental to avoid obstacles and to maintain socially compliant behavior. For those reasons, the costmap should be able to represent regular obstacles and people in a way that people have a custom representation, so that the robot can consider people's personal spaces and, in case of movement, consider the movement being performed, so that the people surrounding the robotic agent can move freely without feeling bothered by the agent's presence and movement. The *navigation_layers* ROS package includes two extra layers, which, in this case, help with people's representation in the environment. These layers receive a ROS message with the position and velocity information relative to the people. This message contains the pose and velocity of each person, which will be used to adapt the costmap to the expected people's movement. These layers are as follows:

- Proxemic Layer, which is responsible for adapting the costmap directly to people's movements. This is relevant because people generally become uncomfortable when an object or another person gets too close. For this reason, the ideal Proxemic Layer should be able to describe which areas should be restricted to avoid placing people in uncomfortable situations. For this reason, this layer attempts to exclude regions around people and adapt it depending on the person's movement. Examples of local and global costmaps with a proxemic layer are displayed in Figs. 4.1a, 4.1b.

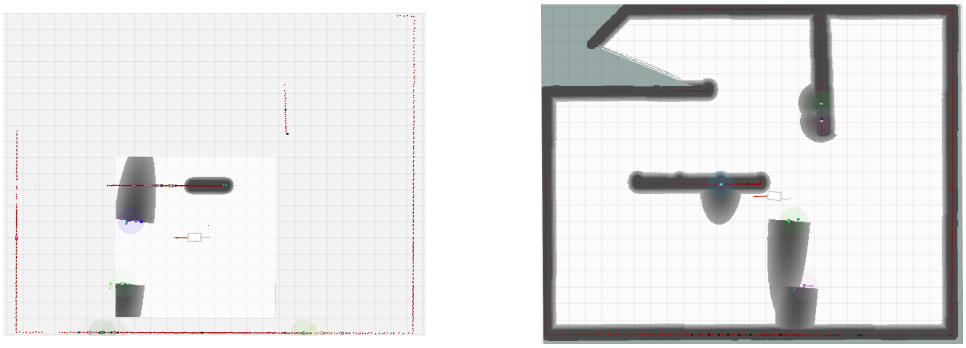


(a) Local costmap example.

(b) Global costmap example.

Figure 4.1: Examples of costmap with Proxemic Layer.

- Passing Layer, is a layer related to social conventions. This layer receives the people's movement as input and increases the values of the costmap to the right side of each individual. The layer decides this based on the movement. If a camera was being used, the orientation could be determined through image information, but since the sensors being used are LiDARs there is no other method to determine which way each person is facing. This layer might help because in most countries people drive on the right side of the road and overtake through the left. For that reason, people from those countries commonly favor that behavior even when they are not driving. This layer is an attempt to condition the robot's behavior to adapt to such norms. Examples of local and global costmaps with a passing layer are displayed in Figs. 4.2a, 4.2b. From looking at Figs. 4.2a, 4.2b, it is visible that according to the movement of the people in the environment, costmap regions will become dark strictly to the right of the people detections performed as well as their perceived movement.



(a) Local costmap.

(b) Global costmap.

Figure 4.2: Examples of costmap with Passing Layer.

All these Layers have configurable parameters, which, when changed, may present significantly different results.

4.2 Global Planner

The global planner is responsible for defining an outline of what the path to the desired goals should be according to the environment. The global planner is dependent on the global costmap to represent the environment. In this case, the global planner needs to gather the people that surround the robot so that the robot's global path already considers the presence of people. It may not seem relevant at first since the local planner will perform adaptations to the global path, but it is. It is important because some local planners, like DWA and DWB may have metrics that compare the generated local path against the global one, as to not deviate too much from it. With that in mind, if the global path already considers the people in the environment, the adjustments will most likely be reduced.

4.3 Local Planner

The local planner is responsible for performing the adaptations needed to overcome the changes that occur in the environment, which is essential for social navigation because the robot must be able to avoid colliding with and disturbing people who might be moving. The local planners depend mostly on the local costmap to perform their job properly, because if that is not the case, the local planner would not be able to adapt, since it does not have any information to work with. However, in some cases, they also depend on the global path. Tests for local navigation were performed with two different navigation algorithms: DWB (Section 2.2.2) andTEB (Section 2.2.2). Because the DWB algorithm, as previously mentioned, is an improved version of DWA in terms of architecture and adaptability, DWA will not be tested. DWB could be used to test DWA, if the critics used are limited to the ones used in DWA.

4.4 Experiments and Results

To be able to navigate in a socially adequate manner, the robot must be able to properly represent humans around itself through the use of costmaps, and after settling on how to

perform a person's representation, the robotic agent should be able to act according to the people surrounding it.

4.4.1 Planner Experiments and Results in Simulation

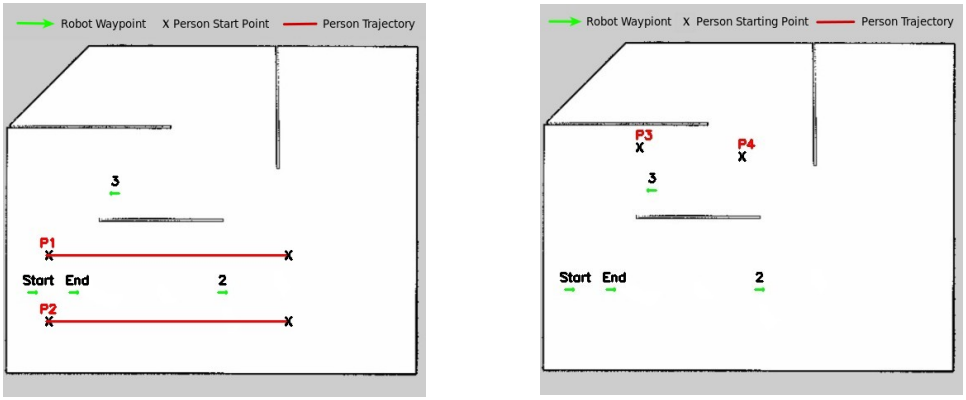
After configuring what will be the basis for the experiments regarding the local planners' performance, they, along with their respective configurations, are tested. To get the best configuration for each planner and what planner is best, the first step is to perform navigation tasks through simulation because the planners, initially, are suboptimal and generate dangerous behavior that can lead to accidents. Simulation is a crucial starting point for the tests in real-world environments because it reduces the risk of accidents and are faster to perform. Even though the simulation tests are not capable of perfectly replicating what happens in real life, they are a crucial step in safely developing a solution.

Configurations in common between planners being testing

This section is responsible for noting the configuration in which the local planner experiments are performed. This is crucial so that all planners are evaluated under the same conditions and that an impartial evaluation is performed.

One of the configurations that the planner depends on is the costmaps' configuration.

To understand which costmap configurations were most suitable for the use case, simulation scenarios were created. One of the scenarios only had people moving, and another had static people. These scenarios are evidenced in Figs. 4.3a, 4.3b.



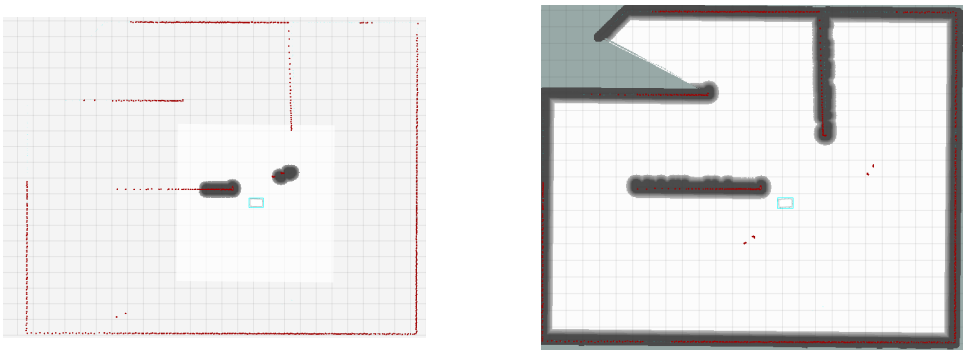
(a) Environment created for experiments with people moving through the environment.

(b) Environment created for experiments of the environment with static people.

Figure 4.3: Simulation environments used to test the costmap layers and parameters.

Regarding the global costmap, the *mir_robot* package starts by default with a Static layer, a Voxel, and an Inflation layer. This combination of layers does not accurately represent the social characteristics of the environment in the costmap. This means that the costmap strictly provides the representation needed to not crash into people only when they are static. Fig. 4.4a displays how the costmap is by default in the *mir_robot* package. It represents areas surrounding walls and people, even though there is no special consideration for the social

boundaries associated with humans. Fig. 4.4b shows an example of the results from the initial layers and configurations of the global map.



(a) Local costmap.

(b) Global costmap.

Figure 4.4: Examples of costmap with default configurations.

To overcome the shortcomings of social constraints, both Proxemic and Passing from the *navigation_layers* package were added to the global and local maps in an attempt to comply with social distances and social norms. To adjust the parameters of the layers, *rqt_reconfigure* was used so that the results could be seen in runtime, speeding up the testing process. This reconfiguration, even if saved, does not save directly for later iterations, so this tool was only used to make the testing process faster. If a better configuration is found, the costmaps are adapted to use such values.

With both the Proxemic and Passing layers activated and configured, there were cases in which the Passing layer, perpendicular to the person's movement, would block routes that do not interfere with people's actions. For that reason, the Passing layer was removed, leaving only the Proxemic layer, which, with parameter tweaking, can create a safe area surrounding each individual according to its movement. Another factor is that, when a false positive occurs, the track will also be represented in the costmap by its Passing layer, which may cause unnecessary hassles to the robot's planner. In Figs. 4.1b,4.1a, examples of the Local and Global maps with the tweaked Proxemic layer are displayed. The Proxemic layer was added to both costmaps so that the global plan could be formed considering the people present when forming the global plan. This is relevant because most local planners try to stay close to the global path, and if the global path is generated and that same path crosses a busy area of the environment, the resulting local plan may be negatively affected by the global plan going through such areas since it attracts the local plan.

The costmap configurations that resulted from the exploration performed before the navigation tests are bound to change. The main reason behind the potential need to adapt the values from the costmaps is that the costmaps are being adjusted based on the default planner configurations that are suboptimal for social scenarios. For that reason, the robot with the default navigation configuration is not expected to perform well under social scenarios, which means that the default costamp configurations will be picked solely through visualization, which means that the configuration used may be adapted when testing different local planner

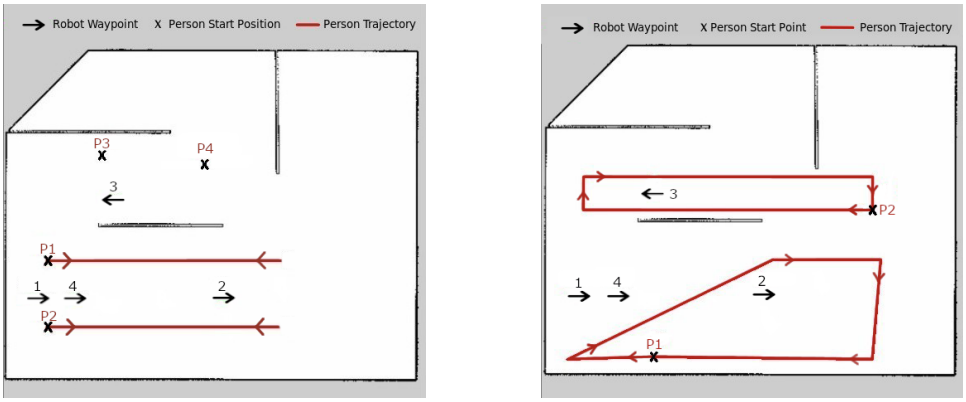
configurations.

The global planner being used is the SBPL [51] from the ROS package *sbpl_lattice_planner* [52] with the ARA* graph search, which is a variation of the A* algorithm. SBPL considers the robot's motion model and limitations to generate a graph of poses the robot can move to and through the use of ARA* it decides which combinations are suitable to reach the goal. For it to have access to that information, the motion models available in the *mir_robot* ROS package were used, so that the global planner could adapt to the MiR100's motion characteristics.

Another relevant configuration needed for the navigation process is the update frequency of the costmap and the costmap publish frequency. By default, the costmap update frequency is 5Hz and the publishing rate is 1Hz. The maximum robot speed is set to 0.8m/s and the expected walking speed of a person can go up to 1.34m/s meaning that if the robot is going in opposite directions to a person, in one second, the distance could change up to 2.16m, which is a significant distance and may become a bottleneck for navigation. To diminish this issue, the publishing frequency was changed to 3Hz, which instead of up to 2.16m distance differential between publications becomes a maximum differential of 0.72m. This significantly increases the quality of the data used by the local planners, which theoretically should improve their performances.

The metrics used for planner classification are the time needed to perform a task, the minimum distance to people, and the number of collisions. Besides these metrics, the paths taken to perform such tasks are recorded to visualize the path and have a subjective yet more intuitive way to look at the performance. For the metrics that result from the tests to be as representative as possible, the global planner was kept the same for all the tests performed, just like the used cost map layers. It is worth mentioning that, since the local planners are highly dependent on the configurations, the computational power required for each local planner can vary. This means that a planner that requires more computational power could affect the results since the simulation rate could differ from planner to planner. This is important because, besides avoiding collisions and not disturbing the human activities surrounding the robot, it is important that the robot perform these tasks efficiently and quickly. Some studies [13] consider the speed of the robot when passing near people as a performance statistic. In this case, the speed is replaced by the time spent performing the whole task, and this is where the performance comes in. To avoid time biases due to the simulation rate, the time recorded for the test results is the simulation time, which theoretically is not affected by the performance differences of the planners.

Since the planners have a variety of configurable parameters, it is also crucial to improve the local planners individually and, only then, make comparisons between them, just like with the ML models in Chapter 3. These individual tests were performed in the environments highlighted in Figs. 4.5a,4.5b.



(a) Simulation environment with both moving and static people

(b) Environment with movement without sudden changes.

Figure 4.5: Environments used for local planner testing.

Experiments and Results using DWB planner in Simulation

The first parameter being tested is the lookahead time, or, in other words, how much time the planner will consider past the current point in time to generate a trajectory. This is a relevant factor because, if the value is too small, the adjustments the robot would make would be performed too late. Following that logic, the higher the lookahead time, the better. The problem is that by increasing it, the computational complexity can increase, which translates into slower movement from the robot. Considering both factors, a balanced lookahead time must be found so that it can adapt and be quick. To test which lookahead time is best, different lookahead times ranging from 0.8s to 1.5s were tested in four instances each in environment 4.5a to understand how it would affect the navigation relative to the time spent on a given task. It is expected that the times will increase along with the lookahead times.

Table 4.1: Tests on how simulation/lookahead time affects path completion time

Lookahead time *(s)*	**Minimum completion time** *(s)*	**Maximum completion time** *(s)*	**Average completion time** *(s)*
0.8 ***(default)***	45.7	46.1	45.9
1.1	48.9	49.1	49.0
1.2	49.3	50.9	50.1
1.3	50.7	58.4	53.3
1.4	53.4	83.5	68.7
1.5	69.0	108.9	83.5

The results from the previously mentioned test are displayed in Table 4.1 where it is confirmed that as the lookahead time increases, the average completion time when using the same scenario, in this case, scenario 4.5a increases alongside the lookahead time. Not only that but as the lookahead time increases, the consistency between times decreases, which creates disparities in the time needed to finish the task. Small time disparities are expected since there are random factors affecting the robot's navigation, but, besides being slower,

with a lookahead time of 1.3 seconds, the difference between the fastest and slowest times recorded with the same lookahead time is 7.7s which corresponds to over 15% time required to complete the same navigation task. For that reason, the following tests with DWB will only be performed on lookahead times of up to 1.2 seconds.

The next tests performed were meant to understand if the robot was capable of avoiding collisions while performing navigation tasks. The tests were made with a lookahead time of 1.1 seconds so that the lookahead time was enough for it to have enough effect without compromising the performance, and by adapting the weight of the critic relative to the distance to obstacles, to understand how it affects the safety and compliance of the navigation around people according to the Edward Hall social space representation [2]. The DWA and DWB planners do not distinguish between common obstacles and people, at least with the used critics, so to avoid people, they base their path planning solely on what they receive from the costmap.

To get the results from Table 4.2, for each weight, 10 attempts were performed, and with the recorded data, the metrics presented in the other columns were calculated. The results

Table 4.2: Tests with different weights for the distance to obstacle critic

Weight - Lookahead Time	Time/trajectory	Min. dist. person	Collisions
0.01 (default) - 1.1	58.4	0.49	3
8 - 1.1	64.1	0.53	1
16 - 1.1	**65.1**	**0.56**	**0**
32 - 1.1	61.6	0.55	1
16 - 1.2	62.5	0.65	1

in the columns *Time/trajectory* and *Min. dist. person* from Table 4.2 do not use data from the attempts where the robot collided. These values are not considered because if there is a collision, the attempt is stopped immediately, which creates outliers both in the times per attempt and for the minimum distances to a person.

When analyzing the values from Table 4.2, it is possible to understand that the default Obstacle Critic weight is not suitable for this type of scenario. This is evident because of the number of times the robot collided with an obstacle. The other weights performed better since the number of collisions was lower than the original. Out of the non-default Object Critic weights tested, the best results were gathered when the Object Critic's weight was 16. Because the best results out of the DWB were gathered with the Objet Critic weight set to 16, the lookahead time was increased from 1.1s to 1.2s, because according to Table 4.1. does not have a big enough computational impact to be discarded.

Experiments and Results using TEB planner in Simulation

When using the TEB algorithm, there is a set of parameters regarding dynamic obstacles. These dynamic obstacles are displayed in both global and local costmaps through the use of the Proxemic Layer. Even though these dynamic obstacles are represented in the costmaps, these obstacles are not seen by the classifiers as being dynamic since, besides the odd shape

of the costmap cells that surround the obstacle, there is no information in the costmap to inform the planner that the obstacle is a moving obstacle. For that reason, in the same way that the Proxemic Layer is fed with the dynamic obstacles, the TEB planner [53], to be able to use the parameters, the dynamic obstacles must be published to the */move_base_node/TebLocalPlannerROS/obstacles* topic with the correct message type, so that the planner can consider this information when creating the robot's path. TEB must be used for this information to be published since the critics used in the DWB planner do not use dynamic obstacles. If another planner is used, this information is not published since there would be no advantage to it, which would translate to the program running slower since there would be more processing required to process the dynamic object's data and unnecessary messages would be generated. This information is published along with the data published for the Proxemic Layer. Ideally, both would receive information from the same source and with the same message format by default, but keeping the packages "as is" can also be beneficial because it creates fewer problems related to project versions.

Table 4.3: Parameters changed throughout TEB navigation test configurations.

Parameter Names	Configurations						
	Default	1	2	3	4	5	6
min obstacle dist	0.4	0.4	0.5	0.5	0.5	0.5	0.5
dynamic obstacle inflation dist	0.6	0.6	0.7	0.8	0.8	0.8	0.8
include dynamic obstacles	false	true	true	true	true	true	true
obstacle association force inclusion factor	1.5	1.5	1.5	1.5	2.0	1.5	1.5
weight obstacle	50.0	50.0	50.0	50.0	50.0	25.0	50.0
weight dynamic obstacle	50.0	50.0	50.0	50.0	50.0	50.0	25.0

The TEB algorithm is highly parameterized, and, for that reason, to make sure the results obtained from it are satisfactory, these parameters must be tweaked to adapt to social scenarios. If the parameters were not tweaked, the results would most likely not present good results and could potentially result in the choice of a worse algorithm, leading to worse performance overall. The configurations used for the tests in Table 4.4 are displayed in Table 4.3. The results displayed in Table 4.4 were gathered in the same way as the results from Table 4.2, where 10 tests were performed for each configuration, and the data extraction was also performed using the same principles.

Table 4.4: Scenario with moving people

Configuration	Time/trajectory *(s)*	Min. dist. person *(m)*	Collisions
Default Config	48.8	0.55	3
Config. 1	45.2	0.64	1
Config. 2	55.8	0.44	1
Config. 3	**53.7**	**0.73**	**0**
Config. 4	51.5	0.61	1
Config. 5	52.4	0.73	1
Config. 6	**54.3**	**0.47**	**0**

A conclusion that can be drawn from the results just by looking at the first two rows of Table 4.4 is that when TEB is used considering all obstacles to be static, it creates an incomplete algorithm, even if the costmap is adapted to represent a person's movement, which is the case. This can be observed by comparing the results from the Default Configuration and Configuration 1 in Table 4.4 since the only change between those configurations is that in the Default Configuration, dynamic obstacles are treated just like the static ones. This difference alone is critical because the algorithm can somewhat understand where the moving people intend to move and adapt its behavior accordingly. When TEB only considers the costmap information, the costmap is adapted, whereas the weight of the cells around these obstacles may not be given their due importance, which may lead to risky behavior.

4.4.2 Results from Simulation Planner Experiments

To conclude the social navigation Chapter it is necessary to observe the results and decide which approaches and respective configurations should be considered. The best planners and respective configurations are decided based on the results available in Tables 4.2 and 4.4 and based on what behaviors are considered to be more suitable when it comes to avoiding collisions with both people and static obstacles. If no solution has a clear advantage over the others, a set of options should be carried over for real-world testing. The tests were performed with simulated people who do not have social awareness, which means that if the robot was being followed by the person in the simulation and the person was moving quicker, the person would not consider the robot's presence and crash into it. Even though this happens in some scenarios, in most cases, it is avoidable. For that reason, these scenarios were not separated into different metrics and are counted as crashes when they occur in the navigation process. Besides the number of times the robot crashed into an obstacle, be it a wall or a person, the most relevant metric is the minimum distance to people because it is preferable to compromise the task's completion speed instead of performing risky behavior to avoid people's discomfort.

The following tables represent the best five results gathered from the planners and configurations tested. The planners present in Table 4.5 are the best planner configurations out of all the tests the the planner's performance. The configurations for the TEB configurations are displayed in Table 4.3 whereas the first number in DWB configurations is the weight for obstacle avoidance critic, and the second is the time that DWB will simulate into the future. On the other hand, the results from Table 4.6 are used to confirm whether the results from

Table 4.5: Compilation of best results from planners tested in the scenario represented in Fig. 4.5b.

Configuration	Time/trajectory *(s)*	Min. dist. person *(m)*	Collisions
TEB Config. 3	53.7	0.73	0
TEB Config. 5	52.4	0.73	1
TEB Config. 6	54.3	0.47	0
DWB 16 1.1	65.1	0.56	0
DWB 16 1.2	62.5	0.64	1

Table 4.5 hold up to different scenarios with different movement patterns.

Table 4.6: Compilation of best results from planners tested in the scenario represented in Fig. 4.5a.

Configuration	Time/trajectory *(s)*	Min. dist. person *(m)*	Collisions
TEB Config. 3	49.583	1.052	0
TEB Config. 5	48.555	0.563	0
TEB Config. 6	49.912	0.614	0
DWB 16 1.1	53.504	0.886	1
DWB 16 1.2	59.701	0.689	0

Considering the previously mentioned metrics gathered, the DWB and TEB algorithms, both with the proper set of criteria, have completed the tests without crashing a single time. Looking back into Edward Hall's proxemic theory [2], the rough transition distance between a person's intimate and personal space is about 50cm. This is not a completely accurate depiction because of cultural and personal preferences, but serves as a guideline for what is an acceptable minimum distance between the agent and a person while performing a task.

Considering the 0.50m threshold as the threshold for what is an acceptable distance to a person, it can be observed in Table 4.5 that, besides the crashes, most planners from the ones compiled in Table 4.5 can keep the robot at a safe distance relative to the surrounding people. As can be seen in Table 4.5, after performing the same test in the same environment with similar people movements, both DWB and TEB have configurations that managed to perform the task 10 times with slight variations in people's movement without crashing, which, for that metric, is the ideal result. Some other configurations led to satisfactory results with a single collision in all 10 iterations of that test. The next most important metric to decide which configuration is the best is the minimum distance to a person. In this field, configuration number three of the TEB algorithm is the one that has the highest value of "minimum distance to a person" out of the configurations that did not result in any collisions throughout all test iterations.

All optimizations made to the planning algorithms were made based on the behavior the robot would present for the environment in Fig.4.5b. Further tests were performed to evaluate if the local planners could adapt to change and new environments. For that reason, using the scenario from Fig. 4.5a, a new set of 10 iterations for each of the local planners from Table 4.5. As can be deduced by analyzing both Tables 4.5 and 4.6, the algorithms can adapt to new scenarios where people display different movement patterns. These patterns may be

considered unpredictable behavior since people in the environment represented in Fig. 4.5a reach a point where they turn around, which may cause problems because of the constant velocity motion model that is being used by the *leg_tracker*.

Besides confirming that the planners can adapt to different environments, when the results from both Tables 4.5 and 4.6 are combined, it becomes clear that Configurations 3 and 6 from the TEB algorithm are the best because they are the only configurations out of the five best that had a total of 0 crashes throughout the entirety of the tests performed. Out of these 2 configurations, Configuration 3 is better, as the minimum distances captured from this configuration, are much higher without only a slight variation in the average times per task.

CHAPTER 5

Conclusion

In this final chapter, the global results of the research made in this dissertation's context will be discussed. It also explains what could have been done differently throughout the dissertation that could have led to better results. The main contributions made by this dissertation to the field of social navigation will come after this discussion. Finally, the shortcomings of this dissertation are discussed, and some possible future solutions are presented.

5.1 Discussion and Alternative Approaches

This dissertation presents a framework for social navigation whose initial objective is to be used in disinfection scenarios. The framework explores a variety of Machine Learning models in attempts to improve People Detection and Tracking algorithms based on leg detection. The gathered results for people detection and tracking were positive, even though the model tested that got the best results for leg classification was a variation of the original. The accuracy of this model with different tests was never below 94%.

When it comes to social navigation, even though it is hard to affirm in concrete terms what constitutes a successful social navigation model, it is safe to say that, according to the experiments performed in simulation and the metrics used to determine whether a trajectory is socially compliant, the robot's movement is socially compliant. This is the result of using the configured costmap layers to adapt to social scenarios in which the robotic agent may be inserted.

The current costmap layers are capable of representing obstacles based on the environment's map and sensor information, they also have a layer responsible for representing a region around obstacles to increase safety. Apart from these layers, there is also a layer that is responsible for the representation of the regions that may be dangerous because of the presence and movement of people.

5.2 Conclusion

To summarize, this dissertation provides a framework for social navigation. This framework is based on LiDAR people detection that culminates in a social navigation framework. Considering that the results of the people detection using LiDAR information are positive means that the social planner has a solid ground for development. To accomplish such results, alterations had to be performed to help with the data extraction so that the data was correctly labeled for proper training and testing. After the people detection and tracking problem was tackled, the social navigation module was explored. This module was explored by using different costmap layers and configurations as well as different planners with different configurations. The costmap and planner were changed to adapt to people's behavior. The costmap and planner system was not tested in real-life scenarios, however, the results gathered in simulation environments are good indicators that the robot would be able to adapt to such scenarios since the system performs tasks swiftly and can maintain a safe distance from people.

5.3 Future Work

For this project to be used in a real-life scenario for space disinfection, extra development and exploration would need to be performed. When it comes to the social navigation aspect, the navigation framework has only been tested in simulation. Even though the results achieved after the simulation experiments were positive, to know that the framework works in real-life conditions, that framework must be tested in real-life scenarios. With that done and with successful results, the social navigation framework would be ready to use. However, even with the framework being ready to use, it would need to be complemented with an algorithm that creates sets of waypoints in the environment according to the disinfection needs and the people in the environment.

A possible improvement for the social navigation framework is to add the information gathered from the 3D cameras to the people detection and tracking so that obstacles that are not at the height of the LiDAR can also be detected, which would help with the detection of possible obstacles at different heights and, with the aid of sensor fusion, create better human detections. This could, depending on the types of obstacles in the environment, improve the results for the robot's navigation in general and not only social navigation, because this would help in the detection of people and obstacle perception.

References

[1] Y. Gao and C.-M. Huang, "Evaluation of socially-aware robot navigation," *Frontiers in Robotics and AI*, vol. 8, 2022, ISSN: 2296-9144. DOI: 10.3389/frobt.2021.721317. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2021.721317.

[2] N. Brown, "Edward t. hall, proxemic theory.," 1966.

[3] M. Daza, D. Barrios-Aranibar, J. Diaz-Amado, Y. Cardinale, and J. Vilasboas, "An approach of social navigation based on proxemics for crowded environments of humans and robots," *Micromachines*, vol. 12, 2 2021, ISSN: 2072666X. DOI: 10.3390/mi12020193.

[4] M. I. R. A. (MiR), *Mir100 user guide 3rd ed.* Mobile Industrial Robots A/S (MiR), Odense, Denmark, 2020. (visited on 2023).

[5] K. O. Arras, O. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2d range data," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3402–3407. DOI: 10.1109/ROBOT.2007.363998.

[6] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang, "Person tracking and following with 2d laser scanners," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 726–733. DOI: 10.1109/ICRA.2015.7139259.

[7] T. Taipalus and J. Ahtiainen, "Human detection and tracking with knee-high mobile 2d lidar," in *2011 IEEE International Conference on Robotics and Biomimetics*, 2011, pp. 1672–1677. DOI: 10.1109/ROBIO.2011.6181529.

[8] L. Spinello and R. Siegwart, "Human detection using multimodal and multidimensional features," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 3264–3269. DOI: 10.1109/ROBOT.2008.4543708.

[9] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, pp. 4282–4286, 5 1995, ISSN: 1063651X. DOI: 10.1103/PhysRevE.51.4282. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.51.4282.

[10] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, 1993, pp. 802–807. DOI: 10.1109/ROBOT.1993.291936.

[11] D. V. L. (Robotics), "Fundamentals of local planning," in *ROSCon Vancouver 2017*, Open Robotics, Oct. 2017. DOI: 10.36288/ROSCon2017-900782. [Online]. Available: https://doi.org/10.36288/ROSCon2017-900782.

[12] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997. DOI: 10.1109/100.580977.

[13] D. V. Lu and W. D. Smart, "Towards more efficient navigation for robots and humans," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1707–1713. DOI: 10.1109/IROS.2013.6696579.

[14] B. Okal and K. O. Arras, "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016. DOI: 10.1109/ICRA.2016.7487452.

[15] D. Li, L. Li, Y. Li, F. Yang, and X. Zuo, "A multi-type features method for leg detection in 2-d laser range data," *IEEE Sensors Journal*, vol. 18, pp. 1675–1684, 4 2018, ISSN: 1530437X. DOI: 10.1109/JSEN.2017.2784900.

[16] C. Gu, Q. Gao, and Z. Xing, "Pedestrian localization based on image plane constraint," in *2021 International Conference on Control Science and Electric Power Systems (CSEPS)*, 2021, pp. 103–106. DOI: 10.1109/CSEPS53726.2021.00028.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, [Online]. Available: http://arxiv.org/abs/1512.02325.

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," vol. 2016-August, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[19] J. Chen, P. Ye, and Z. Sun, "Pedestrian detection and tracking based on 2d lidar," 2019, pp. 421–426. DOI: 10.1109/ICSAI48974.2019.9010202.

[20] C. Álvarez-Aparicio, Á. M. Guerrero-Higueras, F. J. Rodríguez-Lera, J. G. Clavero, F. M. Rico, and V. Matellán, "People detection and tracking using lidar sensors," *Robotics*, vol. 8, 3 2019, ISSN: 22186581. DOI: 10.3390/robotics8030075.

[21] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. [Online]. Available: http://arxiv.org/abs/1812.08008.

[22] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767.

[23] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, 2019, pp. 6015–6022. DOI: 10.1109/ICRA.2019.8794134.

[24] Ó. Gil and A. Sanfeliu, "Effects of a social force model reward in robot navigation based on deep reinforcement learning," vol. 1093 AISC, 2020. DOI: 10.1007/978-3-030-36150-1_18.

[25] A. Vega, L. J. Manso, D. G. Macharet, P. Bustos, and P. Núñez, "Socially aware robot navigation system in human-populated and interactive environments based on an adaptive spatial density function and space affordances," *Pattern Recognition Letters*, vol. 118, pp. 72–84, 2019, ISSN: 01678655. DOI: 10.1016/j.patrec.2018.07.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865518303052.

[26] C. I. Mavrogiannis, W. B. Thomason, and R. A. Knepper, "Social momentum: A framework for legible navigation in dynamic multi-agent environments," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 361–369. DOI: 10.1145/3171221.3171255. [Online]. Available: https://doi.org/10.1145/3171221.3171255.

[27] C. Mavrogiannis, P. Alves-Oliveira, W. Thomason, and R. A. Knepper, "Social momentum: Design and evaluation of a framework for socially competent robot navigation," *ACM Transactions on Human-Robot Interaction*, vol. 11, no. 2, 2022, ISSN: 25739522. DOI: 10.1145/3495244. [Online]. Available: https://doi.org/10.1145/3495244.

[28] C. Rösmann, F. Hoffmann, and T. Bertram, "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control," in *2015 European Control Conference (ECC)*, 2015, pp. 3352–3357. DOI: 10.1109/ECC.2015.7331052.

[29] L. Tiseni, D. Chiaradia, M. Gabardi, M. Solazzi, D. Leonardis, and A. Frisoli, "Uv-c mobile robots with optimized path planning: Algorithm design and on-field measurements to improve surface disinfection against sars-cov-2," *IEEE Robotics and Automation Magazine*, vol. 28, 1 2021, ISSN: 1558223X. DOI: 10.1109/MRA.2020.3045069.

[30] IUVA, *Iuva covid-19 faq*. [Online]. Available: https://iuva.org/iuva-covid-19-faq.

[31] A. Pierson, J. W. Romanishin, H. Hansen, L. Z. Yanez, and D. Rus, "Designing and deploying a mobile uvc disinfection robot," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6700–6707. DOI: 10.1109/IROS51168.2021.9636260.

[32] P. Chanprakon, T. Sae-Oung, T. Treebupachatsakul, P. Hannanta-Anan, and W. Piyawattanametha, "An ultra-violet sterilization robot for disinfection," in *2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, 2019, pp. 1–4. DOI: 10.1109/ICEAST.2019.8802528.

[33] D. Hu, H. Zhong, S. Li, J. Tan, and Q. He, "Segmenting areas of potential contamination for adaptive robotic disinfection in built environments," *Building and Environment*, vol. 184, p. 107 226, 2020, ISSN: 03601323. DOI: 10.1016/j.buildenv.2020.107226. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360132320305977.

[34] D. I. Kim and E. Martinson, "Human centric spatial affordances for improving human activity recognition," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2016-November, 2016, pp. 725–730. DOI: 10.1109/IROS.2016.7759132.

[35] C. Tamantini, F. S. D. Luzio, F. Cordella, G. Pascarella, F. E. Agro, and L. Zollo, "A robotic healthcare assistant for covid-19 emergency: A proposed solution for logistics and disinfection in a hospital environment," *IEEE Robotics and Automation Magazine*, vol. 28, pp. 71–81, 1 2021, ISSN: 1558223X. DOI: 10.1109/MRA.2020.3044953. [Online]. Available: https://ieeexplore.ieee.org/document/9340010.

[36] ROS, *Ecosystem*, 2021. [Online]. Available: https://www.ros.org/blog/ecosystem/.

[37] D. RIC, *Mir_robot*, 2023. [Online]. Available: https://github.com/dfki-ric/mir_robot.

[38] B. Lau, K. O. Arras, W. Burgard, B. Lau, · . W. Burgard, W. Burgard, and K. O. Arras, "Multi-model hypothesis group tracking and group size estimation," *Int J Soc Robot*, vol. 2, pp. 19–30, 2010. DOI: 10.1007/s12369-009-0036-0. [Online]. Available: https://link.springer.com/article/10.1007/s12369-009-0036-0.

[39] J. Deray, *Carmen_publisher*, https://github.com/artivis/carmen_publisher, commit = e23a779a8a534d6b56af94712df25a0caaae73b8, email = deray.jeremie@gmail.com, 2021.

[40] F. D. Augusto Ballardini Pietro Colombo, *Ira_laser_tools*, https://wiki.ros.org/ira_laser_tools, 2021.

[41] W. R. Franklin, *Plpoly*. [Online]. Available: https://wrfranklin.org/Research/Short_Notes/pnpoly.html.

[42] S. Mostafa and F.-X. Wu, "Chapter 3 - diagnosis of autism spectrum disorder with convolutional autoencoder and structural mri images," in *Neural Engineering Techniques for Autism Spectrum Disorder*, A. S. El-Baz and J. S. Suri, Eds., Academic Press, 2021, pp. 23–38, ISBN: 978-0-12-822822-7. DOI: https://doi.org/10.1016/B978-0-12-822822-7.00003-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012822822700003X.

[43] H. Brooks and N. Tucker, "Electrospinning predictions using artificial neural networks," *Polymer*, vol. 58, pp. 22–29, 2014. DOI: 10.1016/j.polymer.2014.12.046. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003238611401146X.

[44] A. C. Mueller, *Support vector machines*, https://github.com/amueller/COMS4995-s19/blob/master/slides/unused-aml-09-support-vector-machines/images/img_4.png, 2018.

[45] D. V. Lu, *Navigation_layers*, https://github.com/DLu/navigation_layers, commit = 8899b6e9504165469468cc966f9030305ed5d827, 2021.

[46] D. V. Lu, *People_msgs*, https://github.com/wg-perception/people/tree/noetic, commit = 852851cbf0b8a879114c31e657b83a06b7b149b5, 2021.

[47] OpenCV, *Rtrees class reference*, OpenCV, 2023. [Online]. Available: https://docs.opencv.org/3.4/d0/d65/classcv_1_1ml_1_1RTrees.html#a4f833672045ab3be645ca0cba0fc7a89.

[48] Scikit-Learn, *Nusvc*, Scikit-Learn, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC.

[49] E. Pedrosa, *Iris_lama_ros*, https://github.com/iris-ua/iris_lama_ros, commit = d6c004ab7bd228f347cf26d01d60447f006cc0fa, 2022.

[50] E. Pedrosa, A. Pereira, and N. Lau, "A non-linear least squares approach to slam using a dynamic likelihood field," *Journal of Intelligent & Robotic Systems*, vol. 93, pp. 519–532, Mar. 2019. DOI: 10.1007/s10846-017-0763-7.

[51] M. Likhachev, *Search-based planning with motion primitives*, http://www.ros.org/wiki/Events/CoTeSys-ROS-School?action=AttachFile&do=get&target=robschooltutorial_oct10.pdf, 2010.

[52] M. Phillips, *Sbpl_lattice_planner*, http://wiki.ros.org/sbpl_lattice_planner, 2018.

[53] C. Rösmann, *Teb_local_planner*, https://github.com/rst-tu-dortmund/teb_local_planner.git, 2021.