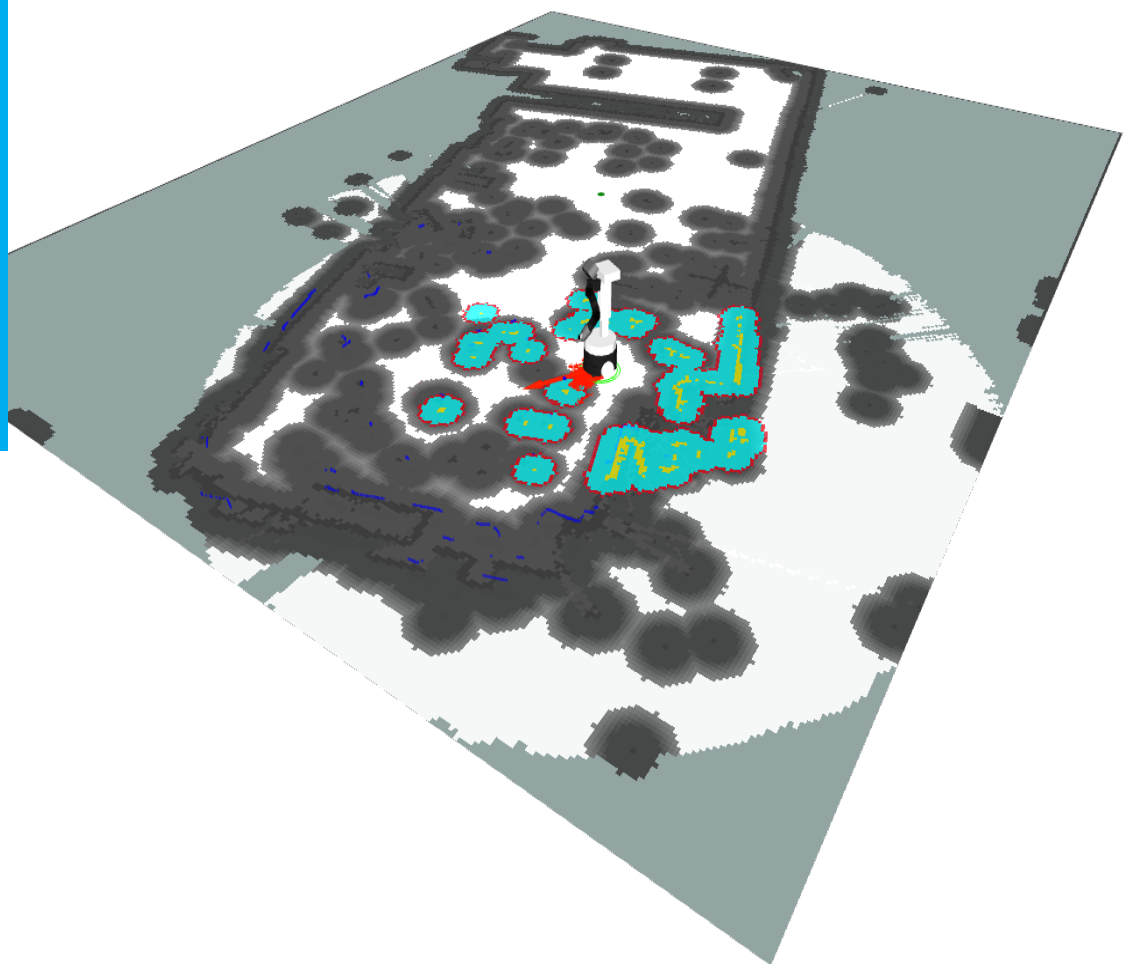
Human-Aware Autonomous Navigation of a Care Robot in Domestic Environments

Dimitrios Karageorgos

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Human-Aware Autonomous Navigation of a Care Robot in Domestic Environments

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Dimitrios Karageorgos

September 27, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was conducted on behalf of, and carried out in cooperation with Heemskerk Innovative Technology (HiT). Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Human-Aware Autonomous Navigation of a Care Robot in Domestic Environments

by

Dimitrios Karageorgos

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: September 27, 2017

Supervisor(s):

ir. Nicky Mol

prof. dr. ir. Robert Babuška

Reader(s):

Dr. Javier Alonso-Mora

Dr. ir. Tamas Keviczky

Dr. Riccardo Ferrari

Acknowledgements

I would like to express my special gratitude to ir. Nicky Mol, for his daily guidance and support for the completion of this MSc thesis. Especially, I appreciate the patience he showed on helping me with technical and operational problems. He gave me precious feedback both on technical and academical matters and he helped me many times organize my thoughts. Secondly, I am grateful to dr. ir. C.J.M. Heemskerk, who gave me the opportunity to work at HIT, on such an exciting and challenging project. He constantly provided me with meaningful advice and feedback on my work and helped me improve my communicational and presentation skills.

I would also like to thank my academic supervisor prof. dr. ir. Robert Babuška, for his key observations and remarks, that helped me shape my work and add more value to my contributions.

Moreover I would like to thank dr. Javier Alonso-Mora, dr. ir. Tamas Keviczky and dr. Riccardo Ferrari for being the reviewers of my thesis.

I would not be able to accomplish anything without the sincere love and support of my parents, Lampros and Spyridoula and my brother Lazaros, who always motivated me to follow my dreams and they helped me with all their strength to do so. Mom and dad thank you for everything!

I would like to thank Enrico with whom I have been working both at the university and at HIT, for the past two years. We helped each other many times and most importantly we formed a good friendship. Thank you Ciccio! Furthermore, I am thankful to my friends Maya, Marina, Kallirroi, Anastasia, Nadia, Haris, Alex, Aris and Dimitris for making my stay at Delft such a pleasant experience!

Finally and most importantly, I would like to thank my beloved Natasa, for her patience to my grumbling about the thesis and her strong support to every step and decision of mine. Thank you for making me a better person!

Abstract

A major societal challenge in occidental countries nowadays, is the continuously increasing aging population. Due to this aging society, solutions are needed to alleviate the limited care personnel and the ever increasing healthcare costs. To this end, service robots are proposed to meet this demand by assisting elderly with their daily activities and providing them with a more independent life.

When people are present, new behavioral concepts are necessary to assure acceptance of robots as daily assistants. Robot motion must take safety, as well as human comfort into account. This is the goal of Human-Aware Navigation (HAN). Most current HAN approaches focus on maintaining appropriate interaction spatial distances between the robot and the human, aiming at respecting their proximity space. However, these approaches are effective only when the human lies in the close vicinity of the robot. Predictive methods, on the other hand, aim at foreseeing how humans will move in the environment and plan a navigational behavior based on this anticipated motion. These methods though, usually fail to produce socially acceptable robot motion as they do not account for proximity constraints.

This thesis is concerned with the implementation of a human-aware navigation framework that incorporates the proximity space around a human as well as human intention, as layers of a layered 2D costmap architecture, used for navigation. The proximity space is modeled as a Gaussian cost function around the person. Human intention is defined by the probability that a person will move to a specific destination within the environment and is used as a means of human motion prediction model. In order to infer the path that the human will follow to reach the destination, a simplistic path planner is implemented. Then, the intention cost model is assigned along this expected path.

The proposed framework is benchmarked against two current state-of-the-art navigation methods. The contestants are: the navigation method used in the ubiquitous Robot Operating System (ROS) and a social navigation method that accounts only for the proximity space around humans. The goal of the thesis is to show that the incorporation of human intention in the robot path planning process is able to produce friendlier motion and increase human comfort. Simulated experiments have been conducted and metrics have been defined to evaluate the methods in terms of human comfort and navigation performance. Results

demonstrate that the proposed navigation framework outperformed the other two methods, proving to be able to produce friendlier robot motion while exhibiting similar navigation performance.

Table of Contents

List of Figures

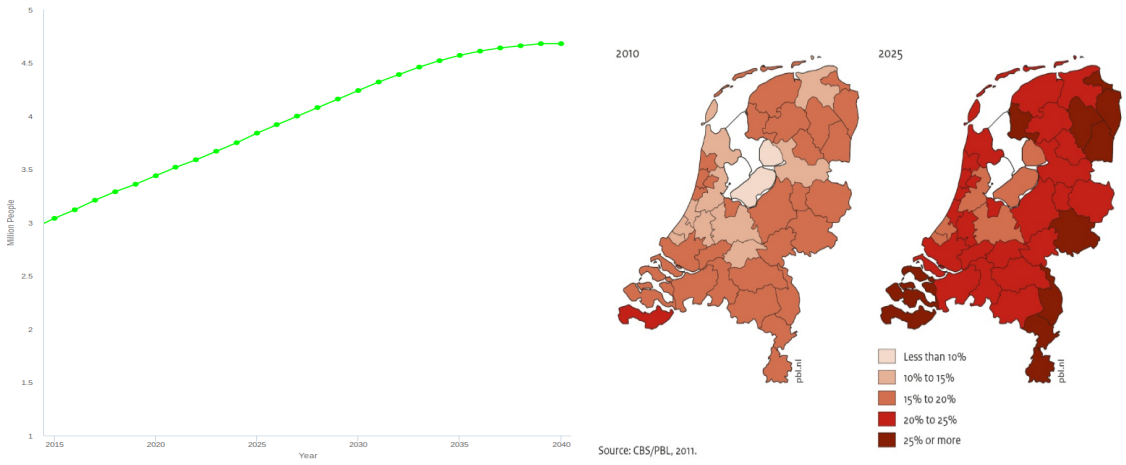
List of Tables

Chapter 1

Introduction

1-1 Motivation

A major societal challenge in Europe and particularly in the Netherlands, nowadays, is the continuously increasing aging population. Studies have shown that within the next 20 years, the aging population in the Netherlands will be nearly doubled [1].



(a) Aging population in the Netherlands, over the next 25 years[2].

(b) Proportion of population aged 65 and over in the Netherlands (From [1]).

Figure 1-1: Predictions about the aging population in the Netherlands.

This phenomenon imposes several challenges on the healthcare domain. The workload on care facilities becomes higher while at the same time there is limited personnel and increased nursing costs. The lack of healthcare professionals results in poor quality of care services and

[1] http://www.zorgvoorbeter.nl/ouderenzorg/hervorming-zorg-cijfers-vergrijzing.html
[2] http://www.ifs.du.edu

in absence of support for the elderly and physically handicapped people. Since these people are highly dependent on care professionals due to their inability to perform on their own, basic routine tasks such as eating, moving to get distant objects and cleaning the table, living in hospitals or nursing facilities is inevitable and often perceived as a degradation of quality of life.

A promising idea is to employ mobile home service robots in order to help people with their Activities of Daily Living (ADLs)[3]. This solution arises from the great recent advances in robotics which forebode that robots will soon be part of our home environment and share the same workspace with humans. Healthcare can be significantly benefited from this evolution. Care service robots will be able to alleviate the limited care personnel and the high costs and most importantly can provide people with a more independent life.

In this context, Heemskerk Innovative Technology (HiT) [4] aims to develop a semi-autonomous care robot solution in the Semi Autonomous Care Robot (SACRO) project. In this project, the goal is to develop a mobile care robot, intended to assist elderly and physically disabled people on their daily life activities. The principal idea of SACRO project is the implementation of a low cost, semi-autonomous and robust commercial home care robot that combines the use of a hardware platform which has limited autonomy, with the eminent abilities of a remote human operator. The robot is expected to perform basic and generic tasks autonomously, whereas more complex tasks can be performed by the operator via haptic tele-manipulation using a remote cockpit, as can be seen in Figure 1-2.

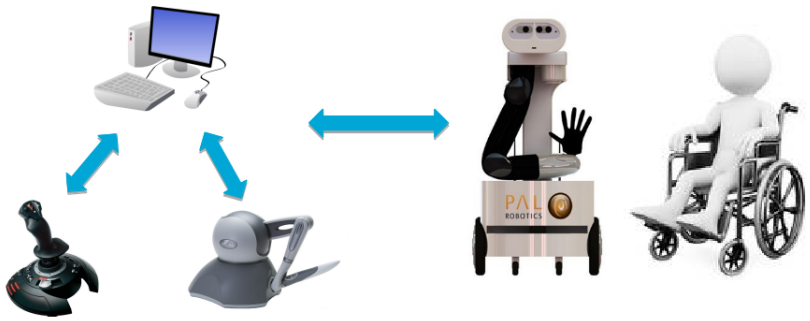
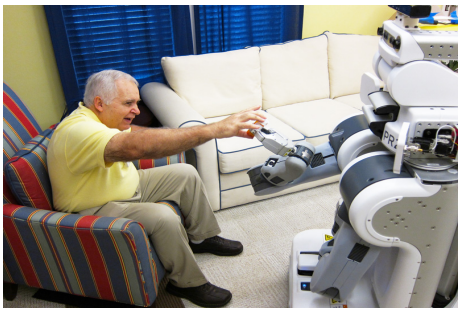


Figure 1-2: Architecture of the SACRO project. A remote operator will be able, by using a remote cockpit (left), to control the care robot (right) via haptic tele-manipulation. The robot will be able to perform most of the tasks autonomously and in case of failure of its autonomous capabilities, the control will be transferred to the operator. The cockpit consists of a joystick for controlling the base and the torso of the robot, a haptic device for controlling the arm and a screen for visual feedback.

Coexistence and collaboration between humans and robots is not a new concept. For several years robots have been used in industry to perform multiple tasks. However, an industrial environment is quite different from a domestic one, in the sense that it is perfectly known and properly structured for the operation of robots in it, as shown in Figure 1-3a. Robots are mounted on suitable bases and restricted to work inside a well-defined workspace. This spatial isolation ensures the physical safety of humans moving and working inside the environment.

[3] https://en.wikipedia.org/wiki/Activities_of_daily_living
[4] http://www.heemskerk-innovative.nl/

(a) Multiple robots working in an industrial environment [5].



(b) A care robot in a household environment [6].

Figure 1-3: Examples where robots coexist with humans to accomplish a task. In Figure 1-3a, interaction with humans is limited since the robots act within a restricted workspace. Contrariwise in Figure 1-3b, the robot closely cooperates with humans for jointly performing a hand-over task.

On the contrary, a domestic environment is a highly dynamic environment with a lot of uncertainties, where humans are constantly moving in an arbitrary manner and in the vicinity of the robot. Both its motion and its actions, e.g. arm manipulation, must ensure the safety of objects and humans. A care robot in a domestic environment is shown in Figure 1-3b.

One of the key attributes of a service robot assistant is its ability to navigate autonomously towards its goal. In a domestic environment, the human-robot interaction is immediate, hence the robot should be able to detect and avoid humans as well as static objects efficiently, while at the same time navigating towards a specified target position to perform a certain task. Several solutions have been proposed in the last 25 years (surveys can be found in [2], [3] and [4]), which constitute the robot able to avoid obstacles and humans with safety. Nevertheless, the resulting robot motion is not friendly to humans. As a consequence, humans and especially the ones that are not familiar with robotics, as the majority of the elderly population, feel annoyed and intimidated by the presence of the robot in their proximity. Therefore they are skeptical towards a care robot existing and operating inside their homes. Hence, when humans are present, extra behavioral attributes are necessary for acceptable robot motion. This reluctance of humans towards robot motion is demonstrated by an example in Section 1-1-1.

1-1-1 A motivational example

As mentioned above, state of the art autonomous navigation algorithms are able to produce a safe and efficient robot motion, though not human friendly. Unfriendly robot motion prevents people from accepting the robot as a social companion. Feeling comfortable with the presence and motion of the robot is a high predominant for coexisting with it every day. An example of an unfriendly robot motion is illustrated in Figure 1-4.

[5] http://www.nytimes.com/2012/08/19/business/new-wave-of-adept-robots-is-changing-global-industry.html

[6] http://robohub.org

(a) A robot approaches a human.

(b) The robot avoids but comes too close to the human.

(c) The robot passed and the human felt awkward.

Figure 1-4: The Figure shows the resulting robot motion of a conventional autonomous navigation algorithm. The robot avoids the human with safety but at the same time it does not account for the feelings and comfort of the human. Although there was no collision, the human still might have felt irritated by the behavior of the robot [7].

Although the robot avoids the human, at the same time it enters the person's intimate zone causing him negative feelings such as annoyance, anxiety and even fear. Such a robot motion is *unaware* of humans, since the robot treats them as mere moving obstacles, disregarding their feelings and comfort.

1-2 Problem statement

Autonomous navigation is a well studied problem in the robotics domain and many solutions have been proposed. Advanced navigation algorithms ([6], [7], [8]) provide safe, collision free paths for the robot, even in highly uncertain and dynamic environments. A domestic environment belongs to this category. In addition, a domestic environment has a crucial oddity, which is the presence of humans. Albeit, state of the art approaches enable mobile robots to move in a safe, collision-free manner among people, nonetheless they fail to produce a robot motion which apart from being safe, is also *perceived* as safe.

In order to allow robots to collaborate and coexist with humans in domestic environments, humans, during motion, must be considered as social entities and not simply as dynamic obstacles. Thus, apart from physical safety, additional constraints such as comfort, anxiety, fear and surprise must explicitly be taken into account in the motion planning process. These constraints have been defined as '*social*' in the literature ([9], [10], [11], [12]). Autonomous navigation solutions so far, do not consider these social constraints, leading the robot to exhibit lack of human-awareness. Satisfaction of these constraints will yield the acceptance of the robot as an assistant in everyday activities. The **problem** of the thesis can hence be formalized as:

[7] [5] https://www.youtube.com/watch?v=evWTA8FcYJo

Treating humans as mere moving obstacles during robot navigation, results in an antisocial behavior that averts the robot to be accepted as a reliable daily companion.

Addressing this problem is the primary objective of Human-Aware Navigation (HAN) ([13], [9]). This research domain, which gains increasing interest as robots make their way into our everyday lives, introduces additional behavioral concepts in motion planning which aim to satisfy social constraints. However, as will be discussed in Section 1-3, most of the proposed approaches still lack the necessary characteristics that will lead in complete acceptance of the robot motion.

1-3 Related work

Human-Aware Navigation (HAN) is the intersection between Human-Robot Interaction (HRI) and motion planning. HRI is a sub-domain of robotics and, as stated in [14]: "is the interdisciplinary study of interaction dynamics between humans and robots". A form of interaction dynamics is the navigation of the robot among humans. The presence of humans imposes new constraints in robot motion and these constraints are purely related to human comfort and social rules. Robots need to navigate beyond simple collision avoidance in order to ensure that humans will not feel intimidated or annoyed by their presence.

Conventional navigation approaches aim at finding a collision-free and short path, by a combination of global and local path planning. A cost model is assigned to the environment, where static and dynamic obstacles are considered as prohibited areas (with very high cost) and by means of a Dijkstra or A* algorithm [7], the shortest collision free path to a destination is found. Moving obstacles and humans are taken into account by constantly replanning fast enough and by assuming some mutual action in order to avoid collision.

On the contrary, HAN does not intend to find the shortest or the most energy efficient path, but the one that adheres as much as possible to the social constraints, and most importantly to human comfort. The majority of HAN approaches attempt to satisfy social constraints by maintaining certain spatial distances between the robot and the humans. These distances have been defined by the *proxemics* theory [15], introduced by Hall in the early sixties. Proxemics describe the interaction distances that humans keep from each other in their daily life. Several studies ([16], [17]) have shown that maintaining these distances is also preferable in human-robot interaction and stress their importance in accepting the robot as a social companion.

Various HAN approaches use a social cost model or potential fields to take into account social constraints during robot navigation. Sisbot et al. [13] used a social cost model that aims to satisfy four social constraints, namely safety, comfort, visibility and hidden zones. Kirby et al. [18] used a mixture of Gaussian cost functions to model the personal space around a human and the tendency of people to pass a person on the right side of a hallway. However, both the aforementioned approaches do not account for human motion in the environment. Hence, the robot has a false assumption of a static environment, meaning that it thinks that humans remain still. Therefore it has to replan its avoidance behavior once it spots the human in a different position. This may result in unnatural and therefore unfriendly motion. Kruse et al.

([19], [20]), modified the cost model of Sisbot et al. to consider human motion by adding a cost function that represents the space in front of a moving human, assuming though that the human is always moving forward. Lu et al. [21] used the model of Kirby et al. to represent the proximity space of a human as a layer of a multilayered cost model of the environment (this work constitutes the base for the implementation of the proposed framework of this thesis and will be discussed extensively in Chapter 2). Recently, Kollmitz et al. [22] proposed a Gaussian social cost model that consists of a cascade of linearly decay cost functions capturing the uncertainty of the future human trajectory in space. Although the model of Kollmitz et al. is able to cope with moving humans, it assumes only straightforward motion. There are also approaches that utilize the Social Force Model ([23], [24]) for navigation among people, but these approaches might not be able to solve complex interaction in highly cluttered and confined domestic environments.

The aforementioned approaches aim to respect the desired distances in human-robot interaction and although some of them reason about the motion of humans on a local level, however they do not aspire to predict how the human is going to move inside the environment. As a result, these methods simply wait for the human to approach enough and only then order the robot to react. These methods can be pigeonholed as *reactive* methods.

On the contrary, there is rich literature on methods that explicitly aim to predict the human trajectory and use this information to design a collision avoidance strategy. These can be described as *proactive* methods, since they command the robot to move according to an expected human motion. These methods usually employ machine learning to learn motion patterns, either from observed datasets of real human trajectories ([25], [26]) or from demonstration examples [5]. Also, work has been done that is purely based on spatial reasoning between the human and the robot and use geometry concepts to infer how the human will move ([27], [28]). Other approaches, mostly used in autonomous driving, try to estimate human motion based on the human posture by employing visual feedback from cameras [29]. Albeit proactive approaches in general are able to accurately predict how the human will move, in their vast majority they fail to produce human friendly motion as they do not account for local social constraints, such as the proximity space around a person.

In this thesis, it is believed that a combination of reactive and proactive behavior is able to overcome the limitations of each category and produce socially acceptable robot motion in realistic domestic scenarios. In Section 1-4 the research questions of this thesis are formulated. In section 1-5, the differentiation of the proposed approach from the rest of the literature is presented and in Section 1-6, the requirements and goals of the thesis are set. Based on the requirements is also explained, why some powerful human-aware navigation solutions are not applicable in the context of this thesis.

1-4 Research questions of the thesis

As mentioned in Section 1-1, the goal of SACRO project is to develop a daily service robot assistant. Such a robot assistant must incorporate in its motion the social constraints mentioned above, in order to be accepted by people and deployed in populated domestic environments and healthcare facilities.

Literature revealed that respecting the social constraints alone, without accounting for human

motion, may not be sufficient for the acceptance of service robots. Aiming purely to predict how humans are going to move, on the other hand, produces a robot motion that neglects the way people feel with the robot in their proximity. Therefore, the **first research question** of this thesis is:

How could a human motion prediction model be incorporated, along with the social constraints in the motion planning process, in order to produce more friendly robot motion?

Prediction of human motion in domestic environments is rather a cumbersome task, as it will be explained in Section 1-5. Hence, instead of trying to predict *how* the human will move, this thesis aims to infer *where* the human will move to, and utilize this knowledge to design a human-aware robot motion. Relative to the definition given by Nagariya et al. [30], we define as ***human intention*** , the probability that a human will move towards a specific destination. We argue that, if the robot is able to infer human intention, i.e. to understand where the human wants to go, it can plan its avoidance behavior earlier without having to approach the intimate or even the personal zone of the human. Thus, close interaction with the human can be completely avoided, leading to a more socially acceptable robot motion. Hence, the **second research question** of this thesis is formulated as:

How does the knowledge of human intention contribute to the increase of friendliness of robot motion?

In Section 1-5 it is discussed what are the main contributions of this thesis, in order to answer the aforementioned research questions.

1-5 Contribution of the thesis

Modeling social constraints as costmap layers (Section 2-1-3) is a recent and novel approach ([20], [21], [22]). However most of these approaches are only efficient when the human resides in the vicinity of the robot. The work of Kollmitz et al. [22], additionally, attempts to capture the uncertainty of a human trajectory in time, but assumes only straightforward, obstacle-free motion. Their main focus is to develop a polite robot behavior that also waits for the human to pass first. For this purpose, they proposed a time-lattice planner [31] that searches for a human-friendly path in time. However, this approach could lead in confusing motion in some cases, due to its exploratory nature. Yet, this work is proved capable to generate paths that respect human comfort more than conventional approaches.

Knowing how the human will move could be, in general, of substantial importance for designing efficient human-aware robot motion. However, this task is particularly troublesome in cases where human motion is rather random or arbitrary. Such an example is human motion in domestic environments. People moving inside their home do not follow certain motion patterns and often change their direction either due to some object blocking their way or simply because they changed their mind. Nevertheless, it can be argued that there are certain places inside their house that humans visit quite often or with an increased certainty. An example is the entrance of a room. It is highly likely that humans heading and walking towards a

door, are actually planning to go to that door. Another representative example is the sink in the kitchen. A person entering the kitchen is quite probable to go to the sink for doing the dishes. Such places inside the environment, that people use to visit a lot are termed as *places of interest*. If a care robot, has knowledge of these places, it could use it to plan its behavior accordingly, i.e. it would try not to interfere with a human that is heading towards the entrance, by blocking his path, but rather choose to pass behind him. This reasoning is even more evident in healthcare facilities, which is also a launching application for the robot concerned in this thesis. Nurses visit specific places during their work, such as the patients' beds, docking stations with medical tools or places where they store useful equipment. A robot operating in such an environment should avoid interfering with their paths, hindering their work, otherwise it will not be accepted as an assistant. Human intention has been utilized by Nagariya et al. [30] in order to design collision avoidance maneuvers, by adapting the robot's velocity and trajectory. Their method computes the most probable destination (point of interest) for a walking human and optimizes over the velocity space to find a collision free velocity for the robot. Though, their method does not explicitly consider the personal space of the human and in their experiments, they assumed an obstacle free environment.

We differentiate from the aforementioned works ***by introducing human intention as an additional social constraint***. In the proposed approach, human intention is used as a proactive criterion in order to develop a reactive collision avoidance behavior. In simple words, by knowing where the human intents to go, the robot could plan a path that does not obstruct the human path and additionally respects the comfort zone of the human.

The assumption that a human moves straight to the desired destination, in an environment without obstructions, is valid but rather conservative. In presence of static or moving obstacles, which is usually the case in domestic environments, the human might deviate from the direct path towards his destination. Employing the robot with the ability to perform local path planning for the human, in order to find a collision-free path between him and his destination would increase chances that the robot will not block his way.

According to the two research questions, this thesis aims to satisfy the social constraints, as well as to infer human intention as a means of human motion prediction model, in order to design a socially acceptable motion behavior. Hence, the **contribution** of the thesis is twofold:

1. **The implementation of a cost model that considers human intention as a social constraint. The cost model is incorporated, as a layer of a layered costmap, in consonance with the costmap architecture of Lu et al. [21]. The *Human Intention Layer* is built on top of a *Proximity Layer* that accounts for the comfort zone around humans.**
2. **Incorporation of a simplistic, local path planning approach, using the Dijkstra algorithm, that given the detected position of a human in the environment, calculates a collision-free path towards the destination. This path is the shortest, collision-free path and the one that the human is expected to follow.**

It is argued that the proposed framework, to the best of the author's knowledge, is the first one that models human intention as a costmap layer and the first one that attempts to perform human-aware navigation by considering a human path in the presence of obstacles.

1-6 Goals and requirements

The proposed approach is in consistency with the goal and the requirements of the SACRO project regarding the operation of the service robot in domestic environments. The **goal** of this thesis is the following:

> **Implementation of a safe, socially acceptable robot motion in human-populated, domestic environments.**

Pertaining to this goal several requirements are set:

1. The robot has to autonomously navigate in dynamic environments with static and moving objects, as well as humans.
2. The robot has to respect the personal space of humans, as much as possible and not interfere with their path.
3. Human detection and tracking as well as localization of the robot, must be purely rely on onboard sensors. Installation of overhead cameras or any other modification of the environment is not preferable.
4. The robot should be able to replan its path in real time, as changes in the environment occur, i.e. obstacles in different positions or moving humans.
5. Decision making of the robot for collision avoidance must be performed online and without any prior training using motion datasets.

Ideally, the service robot should be able to navigate also to previously unknown environments. However, the use of places of interests for inferring human intention (as discussed in Section 1-5) is not consistent with this requirement. As, inferring intention has a substantial role in improving sociability of the robot, this last requirement was loosed. Determination of the places of interest is done manually a priori, as will be explained in Chapter 2. Therefore, the environment or at least a rough description of it must be known. The proposed framework is independent of the approach by which a (static) map of the environment is obtained. Advanced Simultaneous Localization and Mapping (SLAM) approaches ([32], [33], [34]) are able to build online a map of the environment. Hence, instead of global places of interest, temporal ones (midpoints) could be defined while the map is constructed. However, this solution is out of the scope of this thesis.

A second comment regards requirement 5. Navigation of the service robot is desired to be an out of the box process. The robot should be able to start navigating autonomously as soon as it is launched, yet after having acquired a map of the environment. This rules out any machine learning solutions that require a particular amount and period of training. Besides, as already discussed, human motion patterning in domestic environments is cumbersome due to the abstract motion. As a result, learning approaches such as [35], although efficient in public scenes, might be insufficient in cluttered domestic environments.

1-7 Approach

In order to implement and validate the proposed framework, the approach which is followed is described in this Section.

Firstly, a set of probable destinations (places of interest) inside the environment is manually selected. The robot detects a person and computes the probability to go to each one of these destinations. The highest probability, denoting the most probable destination is determined and a social cost model in the form of a multilayered costmap is assigned along the path that the human is expected to follow in order to reach the destination. In case that no obstacles are present between the human and the destination, a straight path is assumed. Otherwise, a simplistic path planning algorithm is performed to determine a collision free path that the human will take to reach his destination. This path is determined under the assumption that it is the shortest collision-free path.

To prove the efficiency of our approach, three different scenarios have been selected in which we compare the proposed approach with two state-of-the art navigation approaches. The contestants are the standard navigation approach used in Robot Operating System (ROS) navigation stack (see Appendix B) and the social navigation proposed by Lu et al. [21]. The testing scenarios have been determined based on HAN literature. Simulation studies in all three scenarios are conducted and the three algorithms are compared using several metrics, in terms of comfort, smoothness in motion and navigation performance.

Chapter 2

The proposed Human-Aware Navigation framework

In this Chapter, the proposed Human-Aware Navigation (HAN) framework is presented and the mathematical principles that govern each component are provided. Section 2-1 provides some basic information about the costmap architecture, used for robot navigation. This knowledge is considered essential for understanding the implementation of the framework of this thesis. In Section 2-2 the structure of the proposed framework is introduced and the individual components are discussed. A mathematical description of the concepts implemented, is provided. Section 2-3 recapitulates the presented approach.

Each of the components presented in this Chapter, were implemented in the open-source Robot Operating System (ROS) framework (further information can be found in Appendix B), to work with the real robot platform (discussed in Section 2-2-2) used in the Semi Autonomous Care Robot (SACRO) project.

2-1 Background Information

2-1-1 Navigating within a costmap

A costmap is a two-dimensional, grid-based representation of the environment, used for robot navigation. Costmaps are the descendants of the occupancy grids ([36], [37]). In traditional occupancy grids, each cell is assigned a value, denoting the probability that an obstacle is located in that particular cell. As the robot moves within the grid, it updates the grid by a straightforward application of the Bayesian rule. In grid-based costmaps, the value of each cell, referred to as the *cost*, represents the knowledge of the robot about the presence of an obstacle as a numerical value, rather than a probability. This means that the value of each cell is easier to tacked, whereas in case of probabilistic grids, the value of the cell is determined stochastically (i.e. using a sonar). An example of an occupancy grid is shown in Figure 2-1 [1].

[1] http://www.innovolition.com/backgroundrationale1.html

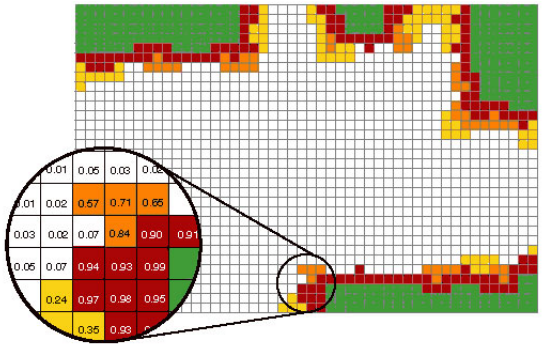


Figure 2-1: An occupancy grid of an indoor environment. In green, the areas that are not identified by the robot's sensors. From white to red, the increasing value of the probabilities that a cell is occupied by an obstacle, is denoted. Practically, the red cells represent the walls of the room.

In their original form, grid-based costmaps had a binary nature, meaning that each cell was either empty or occupied. In recent years, more complicated costs have been added with values varying between empty and occupied. These costs represent more complex semantic information about the obstacle's presence. A robot needs at least three levels of knowledge about the obstacles in order to plan a collision-free path in space. These are [38]:

- Lethal cost : The cell is occupied by an obstacle.
- Zero cost : The cell is free.
- Unknown cost : No information about the cell.

These levels can be further segregated in cost values that represent even more complex information as follows:

- cost $= 0$: Free space.
- $0 <$ cost $<$ `lethal_cost`: Some cost, but no obstacle is present
- cost $\geq$ `lethal_cost`: Either obstacle or higher cost, in any case the robot should not traverse this cell.
- Unknown cost: No information about the cell.

The primary concern during navigation is the robot to avoid areas with lethal (or higher) cost, as traversing through these areas will lead to certain collision. If the robot is allowed to traverse areas with unknown cost or not, in other words if the robot is allowed to explore, is defined by the user. For instance, two paths lead to a certain goal. One path might be shorter but passes behind a wall. If the robot is allowed to cross the (unknown) area behind the wall, in order to reach the goal sooner, is up to the user to decide. Furthermore, the intermediate costs represent the so-called *soft constraints* [21]. These constraints concern desired navigational behaviors that the robot should exhibit. Passing a human always from the right side in a hallway, is an example of such a soft constraint. In autonomous navigation parlance, these intermediate costs are called non-lethal costs. An example of a costmap with both lethal and non-lethal cost cells is shown in Figure 2-2 [2].

A key challenge in costmap-based navigation is the updating of the map. As the robot moves inside the environment, new data are constantly obtained by its sensors. For example,

[2] http://wiki.ros.org/costmap_2d

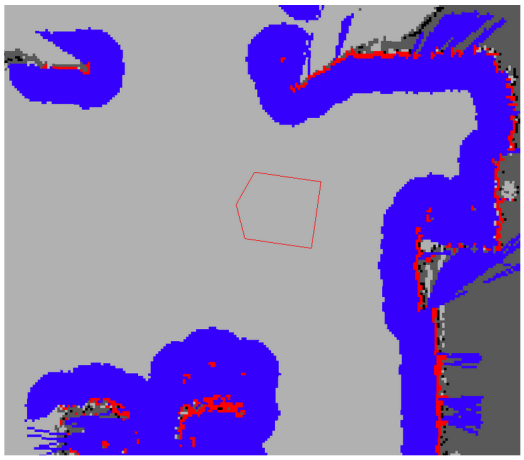


Figure 2-2: An example of a 2D grid-based costmap with various costs. The red polygon represents the mobile robot. Red cells represent obstacles and have a lethal cost. Blue cells represent areas around obstacles and have a non-lethal cost. In general these areas are defined for extra safety. Finally, grey areas are unknown. In order to avoid collision, the robot, namely the red polygon, should never intersect with a red cell and the center of the polygon should never cross a blue cell.

obstacles in new positions or previously occupied areas that now are free, must be taken into account in the new instance of the costmap. Combining these cost-contextualized data is a tricky endeavor and plays a vital role in efficient navigation. The result of the updated information has a different semantic meaning than the previous one and explicitly determines the behavior of the robot. In the next Sections, two different costmaps architectures are discussed, the *monolithic* and the *layered* costmap and the superiority of the latter regarding the updating process is evinced.

2-1-2 The Monolithic Costmap

In its traditional form, a costmap stores all its information in a single grid of values. Such a costmap has been termed as *monolithic* [38] . This data structure was popular due to its simplicity, since values were written to and read from a single place. Path planning within a monolithic costmap is achieved by computing the shortest collision-free path. However, the monolithic costmap suffers from serious limitations, particularly regarding its update process.

Storing and reading values from the same place can lead to problematic situations. Let us assume a simple example where the values stored in the costmap indicate the presence of an obstacle in a particular area, whereas the sensor data reveal that the area is clear. The right way to update the costmap is to account for the origin of the data and some additional semantics that describe the type of the obstacle. For example, the obstacle might be a human that moved, so the considered previously occupied area in the costmap, should be overwritten by the new non-lethal costs. However, another case could be that in the considered area, a glass wall is present and some specific sensors, such as a laser scanner, are unable to detect it. Therefore, the initial lethal costs in the costmap, declaring the wall, should remain as they were. From this example, it is evident that without additional semantic information it is impossible to identify which situation is true. This is the core problem of the monolithic

costmap, since due to its simplistic nature, there is limited information for correctly updating the costmap. In view of the fact that there is only a singe value stored in each cell, any kind of semantic information obtained from the sensors is pinned down to this single value. Another limitation is that the monolithic costmap is unable to accurately represent three-dimensional obstacles. Imagine a table, where the legs are detected by the robot via a laser scanner and the surface through an RGB-D camera. Then, there is danger that since the surface of the table is at a different height, to be incorrectly removed in the monolithic costmap by the laser scans. This problem was partially solved by the ROS authors, using voxel-grids [39], but could not be easily generalized. The aforementioned example reveals also the inability of the monolithic costmap to deal with sensor precedence, namely which sensor should be appreciated more in each case.

Another problem arises from the different kinds of semantic information that the non-lethal costs represent. Let us consider a scenario where a robot has to pass between a wall and a human. The costs in the intermediate area, are the sum of the costs of the inflated area around the wall (blue areas in Figure 2-2) and the costs for passing close to the human. A change in one of the two kinds of costs would require recalculation of both, since only their sum is stored in the monolithic costmap. An extension to this problem is that the monolithic costmap cannot determine how long a cost is present in the costmap. Hence, it is unable to specify how much of the costmap has recently been updated or not. A non sophisticated solution is to demarcate a specific area of the costmap (roughly 6m by 6m around the robot) that could have been updated, no matter how much of this area has actually been updated. This is the approach that ROS uses and the potentially updated area is known as the *local costmap*. Respectively, the overall area that the robot has a priori knowledge of, is termed the *global costmap*. Evidently, the local costmap is a submap of the global costmap.

A severe limitation of the monolithic costmap is that once the values are stored in it, they are stripped from their semantic meaning. This means that all the information about the origin of the cost and what it represents, is lost. Implementation-wise, this also obstructs adding non-lethal costs in the standard ROS costmap implementation, since the monolithic ROS costmap only accepts binary data (either there is an obstacle or the area is free). As a result, more complex motion behaviors that are desired cannot be implemented in the monolithic architecture.

2-1-3 Layered Costmap

In order to eliminate the limitations of the monolithic costmap and to enable the implementation of more complex behaviors of robot motion, Lu et al. [21] introduced a novel architecture in which the overall, or as the creators named, the *master* costmap consists of multiple layers. Each layer is associated to a specific, desired behavior and is updated individually without requiring recalculation of the other layers. The layers are then merged into the master costmap. Navigation in the master costmap exhibits then, all the desired characteristics specified in each layer. The new architecture is called *layered costmap* and the updated phase is composed of two steps, each one performed by a core function. To better illustrate the idea of the layered costmap let us consider a scenario in which the robot apart from the static environment (static costmap), detects an obstacle in the middle of the room. The master costmap in this case consists of three layers, namely the static layer which includes the walls of the room, the

obstacle layer which includes the cells occupied by the obstacle and the inflation layer which includes cells around the obstacle with some non-lethal costs for extra safety (as explained in Figure 2-2). An illustration of such a simplified layered costmap is shown in Figure 2-3.

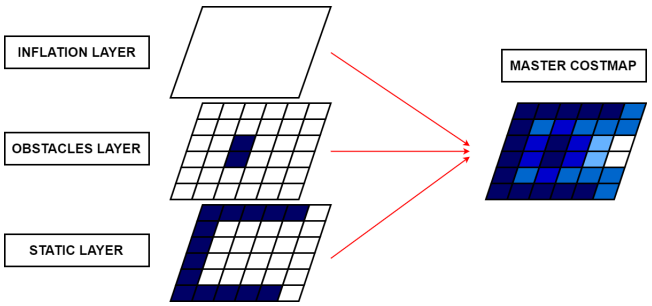


Figure 2-3: A layered costmap consisting of three layers: the static layer, the obstacles layer and the inflation layer.

As already mentioned, the update process of the layered costmap is performed in two steps. In the first step, the amount of the costmap that needs to be updated according to each layer is determined. This work is done by the `updateBounds` method. This method provides each layer with a bounding box that defines the area to be updated. The bounding box can be expanded by a layer if necessary. Initially the bounding box provided to the first layer is empty. The layers are iterated over and over and the resulting bounding box specifies the area of the master costmap to be updated. In the second step, takes place the update of the costs inside the area that needs to be updated. The `updateCosts` method updates the cost values in each consecutive layer within the bounding box of the master costmap. This two step update process is illustrated in Figure 2-4.

The principal advantage of the layered costmap is that it replicates the functionality of the monolithic costmap, but overcomes its limitations by enabling storage of different types of information that maintain their semantic meaning. This is achieved since each layer acts on a private costmap of equal size to the master costmap. Hence, no other layers have access to this private costmap. This architecture prevents overwriting previous information stored by another layer and furthermore simplifies the update process, since the costs have to be recalculated only on the layer that injects new values.

The layered costmap offers significant advantages in comparison with the traditional monolithic costmap. First of all, the layered costmap can store different types of semantic information, as each type can be stored in a separate layer. For example, if a robot is equipped with a laser scanner and a sonar, laser-detected obstacles can be stored in a different layer than sonar-detected obstacles maintaining both the quantitative and semantic information of each data type. In this way, also conflict between sensory data can be prevented for the reason that each layer only updates the information relative to the data type that it stores. In addition, updating in general becomes more efficient, as layers that remain static for relatively long time are not considered frequently in the update of the master costmap and the update method proceeds to the next layers. The layered architecture facilitates the creation and debugging of new layers from the user. The user can insert one layer at a time and test its performance, without concerning for the rest of the layers.

The update of the master costmap is further benefited from the layered structure in the sense

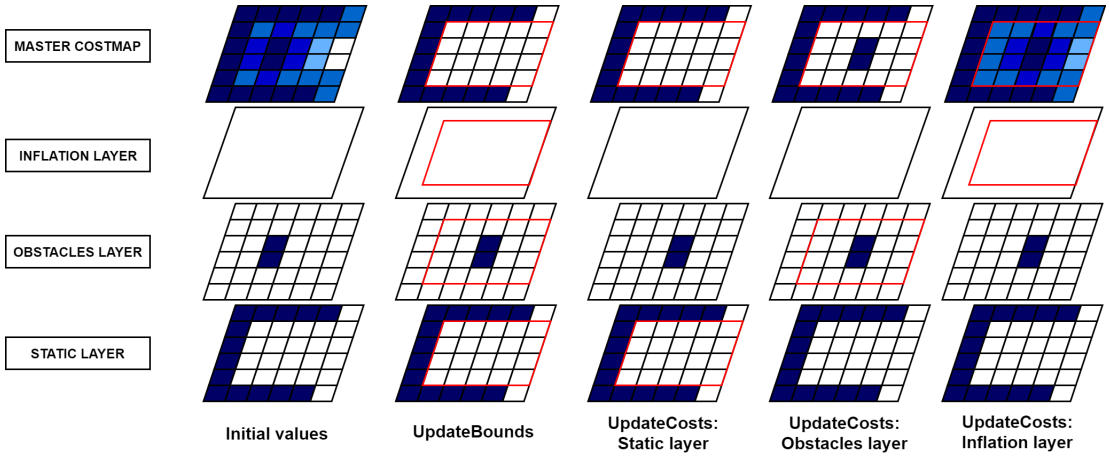


Figure 2-4: Update process of the layered costmap. In the first column the initial values at each layer are shown. The second column depicts the `updateBounds` method which assigns a bounding box to each layer including the area to be updated. The last three columns show the effect of the `updateCosts` method which assigns the new cost values at each layer according to the new data obtained by the sensors. The static layer does not change the master costmap since it has not changed. The obstacle layer updates the costs in the new cells that the obstacle is detected. Lastly, the inflation layer inflates the area around the new position of the obstacle. The updated master costmap is shown in the last column. (Adapted from [21]).

that the update method only considers the demarcated area specified by each individual layer, namely the bounding box of each layer. Therefore, the master costmap becomes more robust since only values within the bounded box are updated. The efficiency of the update process is enhanced by the fact that smaller portions of the costmap are updated.

The layered costmap offers a well-defined and ordered updating strategy. In the example of Figure 2-1-3, the inflation layer will update values both from the static and the obstacle layer as it succeeds them in the stack. Moreover, the result of combining the values between the layers can be mathematically defined as the maximum, minimum or the result of some other function.

The substantial importance of the layered costmap is its reconfigurability as an arbitrary number of layers can be introduced. Each of the new layers can implement a different desired behavior for the motion of the robot. Hence, more complex costs with richer semantics can be added and properly updated. Last but not least, this architecture can be combined with probabilistic occupancy grids which can reside in their own individual layer.

It is obvious that the layered costmap is superior with comparison to the monolithic costmap. An experimental comparison between the two architectures was conducted in [21], proving that the layered costmap, although more complex in structure, is equivalent in performance. Moreover, the reconfigurability that it exhibits and its apparent advantages with respect to its ancestor, constitute it the default data structure used in the ubiquitous ROS navigation stack (See Appendix B-1). The ROS Navigation stack is a navigation framework that most of the state-of-the-art robotic platforms use nowadays. For these reasons the layered costmap is the architecture used in this thesis, where the main contribution is the addition of a layer, which models human intention. The implementation of the layered costmap is done in C++ and the costmap is a class of type `costmap_2d::layered_costmap`. The functionalities of

this class enable the addition of layers which likewise are defined as classes of type `costmap_2d::Layer`. A more detailed explanation on how the costmap and the layers are implemented in C++, can be found on Appendix B.

2-2 Structure of the proposed framework

This Section provides an analytical description of the structure and the components of the proposed framework, including the mathematical principles that govern each component. Additional concepts such as robot localization and human detection, although they are not in the main scope of the thesis, are concisely reported and explained, since they are fundamental components in the implementation of the framework.

2-2-1 Overview

In general, for navigation in human-populated environments, apart from the static world and the location of the obstacles that is conceived by the sensors of the robot, necessary information is the pose of the robot and the location of the humans. The pose of the robot, namely the position and orientation, is obtained through a *localization module*. This module employs probabilistic approaches to estimate where the robot is in the costmap, based on sensor measurements. The localization module is presented in Section 2-2-3. The location of each human in the costmap is determined by a *human detection module* based on a technique for detecting legs. For enhancing certainty that indeed the identified object is a human, an upper body detector can also be incorporated. The human detection module is discussed in Section 2-2-4. The location of the human is utilized to define the personal space or the *proximity zone* around the human. For this purpose costs are assigned in an area around the human. By commanding the robot not to intrude this area, namely planning a least cost path that stays outside the proximity zone as much as possible, we aim to satisfy the social constraints and maintain human comfort. The proximity zone is one of the layers of the layered costmap.

The second introduced layer concerns human intention. As discussed in Section 1-5, some places of interest are specified in the environment, in order to compute the intention of a human to visit them. These places of interest, or the *set of destinations*, which is the term used in this thesis, can be manually set or determined based on learning techniques, as stated in Section 2-2-5. The set of destinations is used as input to an algorithm that calculates the intentions, based on the location and orientation of the human and the location of each destination. The algorithm will extensively be explained in Section 2-2-8. The calculated intentions are employed in order to implement the human intention layer, that is a cost model assigned to the direction of human intention. However, this approach assumes that the human is walking straight to his destination, without any obstacles obstructing his way. In real domestic scenes though, this is hardly ever the case. In order to account for obstacles, that would cause the human to deviate from his path and perform maneuvers to reach his destination, a human path planning module computes the shortest, free path between the human and the most probable destination, taking into account the computed intentions. Hence, the intention cost model is assigned along this path, discouraging the robot from

intersecting with it. The human path planning module is discussed in Section 2-2-9. In Figure 2-5, the structure of the entire proposed framework is shown.

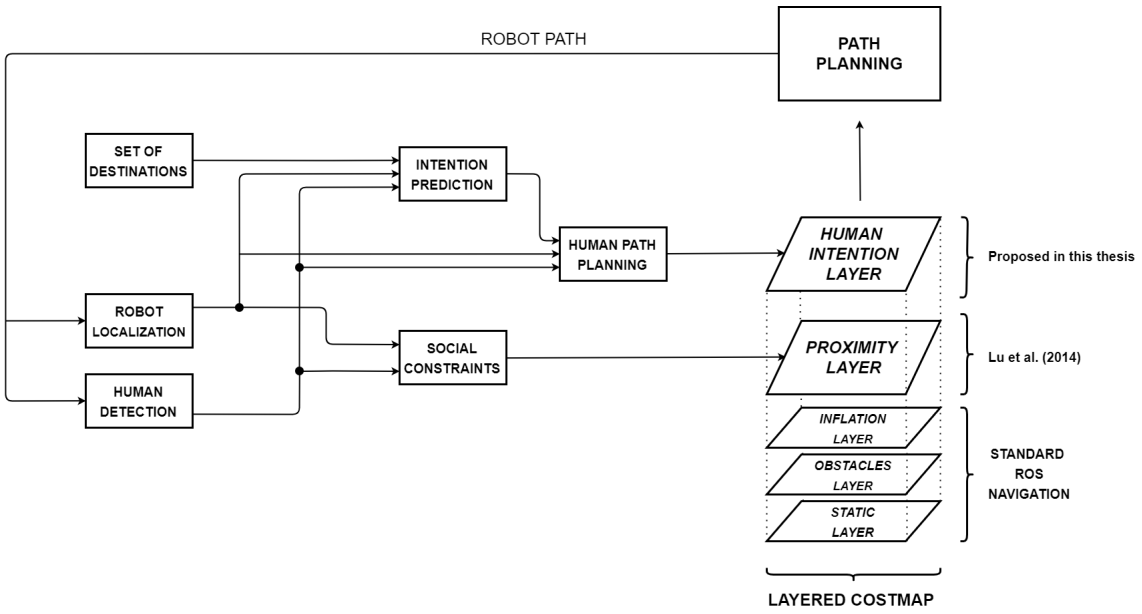


Figure 2-5: The structure of the proposed framework. The robot localization module estimates the pose of the robot. The human detection module provides the position of each human in the environment. Based on this information the personal area of the human is defined by the proximity layer. Using a predetermined set of probable destinations, an intention prediction algorithm computes human intentions and based on the expected human path provided by the human path planning module, the intention layer defines a costmap along that expected path. The proximity and human intention layers are accumulated with the standard navigation layers into the layered costmap which defines the area for safe and human-friendly navigation.

The two additional layers, to wit the proximity and the human intention layer, are accumulated with the rest omnipresent layers into the overall layered costmap. The latter are the static layer, that represents the rarely changing static world (e.g. walls of the room), the obstacle layer which accounts for mere static or moving objects, and the inflation layer that defines a safety zone around the obstacles. These layers are considered by default in the standard layered costmap implementation in ROS. Although they can be removed by preference, they are necessary for proper navigational behavior. More information on these layers can be found in Appendix B. The layered costmap, or more precisely, the master costmap, defines the area that the robot is advised not to enter in order to move in a safe, as well as socially acceptable manner. By avoiding the areas defined by the layers, the robot is able to exhibit the desired behaviors, each one defined by an individual layer. The next sections discuss each of the components of the framework in more detail.

2-2-2 The Robot

The work reported in this thesis was conducted on Marco robot (Figure 2-6a) which is based on the TIAGo robotic platform (Figure 2-6b) developed by PAL Robotics [1] . Marco is a mobile manipulator that consists of a differential-drive base, a lifting torso and an upper

body on which a 7 Degrees of Freedom (DOF) manipulator arm is mounted. The robot has a front laser scanner on the base. The upper body is equipped with an RGB-D camera which also gives the appearance of the head of the robot and a stereo microphone speaker. Marco runs Linux Ubuntu and ROS. It was used as the robot model both in simulations and real world trials. For the purposes of this thesis, the arm kept in the folded position during the experiments.

(a) Marco is the robot used in this thesis.

(b) TIAGo robot.

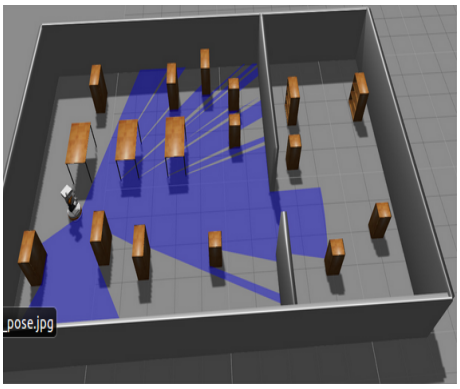
Figure 2-6: Marco and TIAGo robots manufactured by PAL Robotics. Marco is a prototype of TIAGo and used as the experimental setup in this thesis. TIAGo is the commercial version of Marco. Both robots are kinematically the same.

2-2-3 Robot localization

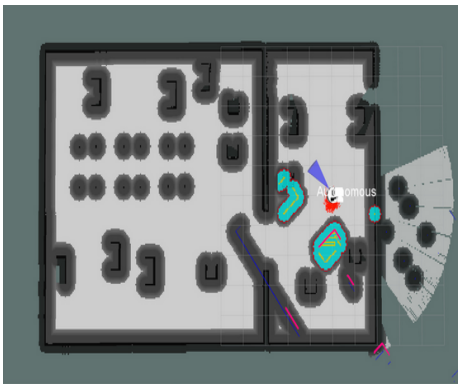
For motion in the proximity of humans, in confined domestic environments, accurate localization is of paramount importance, as it determines to a great extent if the navigation will be successful or not. The robot must be able to determine its pose at real time, in order to plan its moving actions. In this thesis, the localization method used is the Adaptive Monte Carlo Localization (AMCL) method, introduced by Dieter Fox [40] and explained in the book Probabilistic Robotics by Thrun et al [2]. AMCL is a probabilistic localization method for mobile robots moving in 2D and uses particle filters to estimate the pose of the robot in a known environment.

Let us assume that the robot wakes up at a random location inside a known map, as shown in Figures 2-7a-2-7b. The robot needs to understand where it is with respect to a global position, for instance the origin of the map. For localizing the robot, the localization algorithm spreads particles all over the known map (Figure 2-7c). The robot has to match the new obtained sensor measurements with the spread particles in order to estimate its actual location. By rotating in place, more measurements are obtained and matched to the known map. Hence, the fallacious particles are removed and the filter converges to the real location (Figures 2-7d-2-7e). The AMCL method is able to provide sufficiently accurate pose estimates for the purposes of this thesis. However, for navigation in previously unknown environments a Simultaneous Localization and Mapping (SLAM) approach would be more appropriate. Though, this is not in the scope of this thesis.

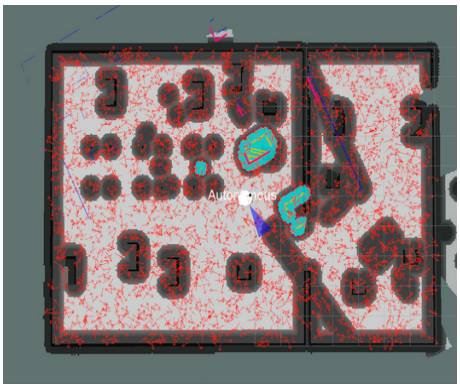
[1] http://tiago.pal-robotics.com/
[2] http://www.probabilistic-robotics.org/



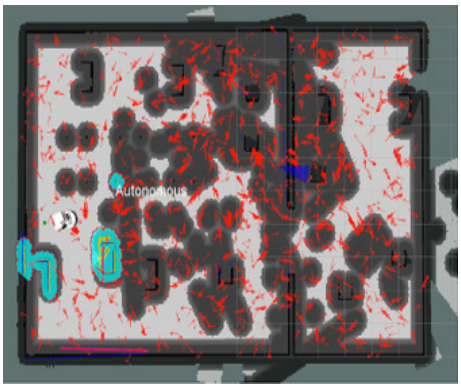
(a) The robot wakes up at a random location.



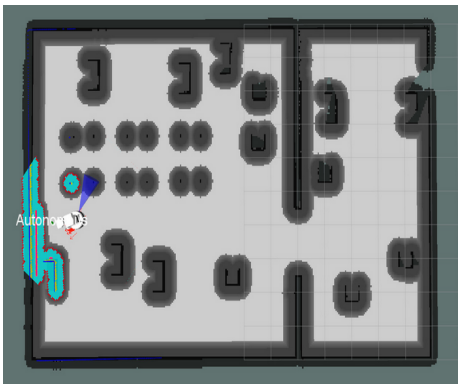
(b) At first, the robot thinks that it is at the origin of the map.



(c) The algorithm spreads particles in the map.



(d) The invalid particles are removed.



(e) The robot localizes itself correctly.

Figure 2-7: Localization within a known map performed by the AMCL method. The algorithm spreads particles all over the map. The robot rotates in place and matches new observations with the known map. The invalid particles are removed and the filter converges to the real location.

2-2-4 Human Detection

In order for the robot to be able to understand people's intentions and actions, at first it has to determine where those people are. The human detection module aims to provide information about the location and velocity of every person in the environment. Knowing the location

and the velocity of a person enables the construction of the cost models around it. The human detection module is an independent component of the proposed framework meaning that different approaches can be used without affecting the functionality of other components. The only thing needed are the data yielding the position of the human. Therefore, depending on the available sensor equipment and the computational abilities of the robot system used, several prominent methods can be utilized. Human detection is usually boiled down to the problem of detecting legs, using laser scans. Broadly used approaches, such as the approach of Arras et al. [41], employ low level classifiers to determine if a sequence of laser scans is a leg or not. The classifier selects several measurement features (such as circularity, size, convexity etc.) that represent human legs, and matches the incoming data to these features to determine if the new data are indeed human legs or not. The features are extracted using a popular machine learning algorithm [1], from several obtained laser reading from human legs. Exemplary readings are shown in Figure 2-8.

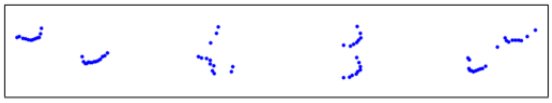


Figure 2-8: Exemplary laser readings from human legs. Machine learning algorithms can be employed to extract common features from such readings. The extracted features are used to pair obtained measurements and determine if they actually represent human legs. (From [41]).

Apart from leg detection through laser scans, there are more advanced approaches that aim to infer the presence of a human by detecting the upper body. Munaro et al. ([42], [43]) used RGB-D sensors to extract human features and detect people using a clustering method, based on the assumption that people move on a ground plane. Recently, Linder et al. [44] have presented a framework of several detectors that enables human detection and tracking, also in public scenes, using a combination of laser, RGB-D sensors and advanced computer vision feature extraction algorithms. Approaches that detect faces, have also been proposed. The method by Pantofaru [2] detects faces in image data obtained by a stereo camera. It uses the OpenCV face detector [3] to extract face features and acquire an initial set of detections. Then, depth information is employed to estimate the real size of the detected face and decide if it really is a human face. This idea is illustrated in Figure 2-9.

Nevertheless, the more advanced is the detection method, the more demanding is in processing effort and sensory equipment. For the purposes of this thesis, the implementation of Pantufaru developed at Willow Garage [3], is considered as a viable and easily implemented method for the Marco robot. Some real-life experimental trials tested this method with promising results. However, the sensory data obtained were quite noisy as will be stated in Chapter 4-1. This implementation is based on the method from Arras et al. [41] described above and is chosen because of the ease of installation on the real robot. Nevertheless, other approaches can also be incorporated, by enhancing the software architecture on the TIAGo robotic platform and by installing additional libraries, but this is not in the scope of this thesis.

[1] https://en.wikipedia.org/wiki/AdaBoost
[2] http://wiki.ros.org/face_detector
[3] http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

Figure 2-9: Face detection by use of feature detector and depth information. In the left image, the detector returns two possible faces, denoted by the bounding boxes. In the right image the depth information discards the blue box as a face candidate due to poor depth information (low illumination). It also qualifies another candidate (red box) which is discarded due to unrealistic size for a human head. In the end, only the green box is regarded as a human face which is the true case.

2-2-5 Selection of Destinations

The set of destinations is the set of places of interest that humans use to visit a lot. These places are environment-dependent and can be determined either by hand or by learning techniques. In the first case, the destinations are merely selected manually, based on our experience from daily living. To this end, salient positions for humans inside a house are e.g. the entrance, the bathroom, the closet etc. As far as healthcare facilities are concerned, probable destinations are the patients' beds, the storage rooms, places to keep medical equipment. The locations of these destinations, namely their x, y coordinates can be hard-coded into the proposed framework. Indeed, this approach is followed for the experiments of this thesis due to its simplicity. A more sophisticated, yet costly way, is to define the set of destinations based on real observed human trajectories. Using, for instance, overhead cameras, numerous trajectories can be tracked and collected. Then using clusterization techniques [45], the entry/exit points of these trajectories can be extracted, denoting prominent destinations. This method has proven to be successful in public scenes, however it is not advised for domestic environments mainly because of the need of installation of surveillance systems.

2-2-6 Standard Navigation Layers

In this Section, the layers that are ubiquitous in the layered costmap architecture are briefly discussed. These layers can be deactivated but this will most likely lead in erroneous navigation since their context determine safety and ensure collision avoidance.

Static Layer The static layer represents the static world, namely the walls and static obstacles, from a global point of view. This means that the robot has a perception (a map) of its entire environment, beyond its sensors range. The static map can be obtained a priori through a mapping or SLAM algorithm or created using an architectural floor plan. Regarding the update process, the static layer does not change and therefore it constitutes the bottom layer of the master costmap. The `updateBounds` method returns a bounding box or the entire static map. However, since the static layer does not change, it will not increase the

size of the bounding box in following iterations of the update process. Furthermore, it copies its values directly into the master, since no other layers will have written any values before it. The static layer is used by the path planning algorithm for global planning, that is creating a path by possessing only information about the static world. Therefore it is often termed as global costmap.

Obstacles Layer This layer gathers data, obtained by the senors of the robot and places them into the master costmap. The sensors can be laser scanners or RGB-D cameras. The area, namely the cells, between the sensor and the sensor reading (i.e. reflexion of the laser beam) is marked as empty space, and the location of the sensor reading is marked as occupied. The `updateBounds` method places newly obtained data into the costmap by expanding the bounding box to include them. The way in which the new data are combined with the data from the static layer differs according to the desired level of trust. If the static map is not very accurate then usually the data of the obstacle layer overwrite the ones from the static layer. However, if the static map is trustful then the obstacle layer can be configured to incorporate only newly observed lethal obstacles in the master costmap.

Inflation Layer As already mentioned, the inflation layer defines a safety zone around the obstacles (static and moving). Cells that are located exactly on the obstacles are assigned a lethal cost. Cells that are located in the immediate areas around the obstacles are assigned slightly smaller costs following a decay manner, as explained in Appendix B-1-7. The existence of this inflation zone ensures that the robot will not attempt to get too close to the obstacles and hence avoid collision with lethal obstacles. Regarding the update process, the `updateBounds` method increases the previous bounding box, inserted by the obstacles layer, in order to make sure that newly detected obstacles will be inflated. In addition, if some old obstacles, after being inflated enter the bounding box, are also inflated.

2-2-7 The Proximity Layer

The Proximity Layer represents the personal zone of a human, which is the space that people attempt to keep from others in their daily life. This space has been described by the proxemics theory [15], but varies between cultures, social groups and environments. In domestic environments where usually the level of familiarity with other people is higher, the personal space tends to shrink slightly. People tend to keep similar distances also in human-robot interaction scenarios. The personal space can be modeled by a combination of Gaussian functions centered at the human as proposed by Kirby et al. [18]. The resulted cost function has an oval shape and consists of an elliptical Gaussian to the front of the person, pointed at the direction of motion, and a symmetrical Gaussian at the back of the person. The size of the function is scaled according to the velocity of the human. This representation makes paths that pass closer to people, to have a higher cost. Hence, the robot prefers paths that do not cross the personal space, respecting people's concerns about their proximity. The *proximity cost function* is illustrated in Figure 2-10.

The cost values are assigned following a Gaussian distribution around the human. The 2D Gaussian distribution is defined as:

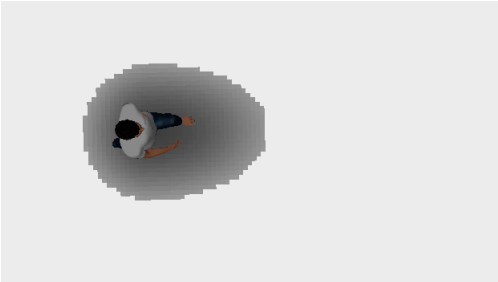


Figure 2-10: For a person moving in 2D, the proximity cost function is a mixture of Gaussians. An elliptical Gaussian in the front of the human and a symmetric one in the back.

$$f_{Gauss}(x,y) = Aexp\left(-\left(\frac{(x-x_0)^2}{2{\sigma_x}^2} + \frac{(y-y_0)^2}{2{\sigma_y}^2}\right)\right) \tag{2-1}$$

where f_{Gauss} denotes the Gaussian function, A is the amplitude, σ_x is the covariance in the x direction, σ_y is the covariance in the y direction, x, y the coordinates of a point in 2D and x_0, y_0 the coordinates of the center of the Gaussian.

Let us assume a cell $c = (x_c, y_c)$ inside the Gaussian region. The cost value of this cell is computed as:

$$\text{Cost}_c = Aexp\left(-\left(F_1 + F_2\right)\right) \tag{2-2}$$

$$\text{where} \quad F_1 = \frac{\left(cos(\phi-\theta) * \sqrt{(x-x_0)^2 + (y-y_0)^2}\right)^2}{2\sigma_x} \tag{2-3}$$

$$F_2 = \frac{\left(sin(\phi-\theta) * \sqrt{(x-x_0)^2 + (y-y_0)^2}\right)^2}{2\sigma_y} \tag{2-4}$$

$$\phi = atan\left(\frac{y-y_0}{x-x_0}\right) \tag{2-5}$$

and θ is the orientation of the Gaussian. The orientation of the Gaussian is determined by the pose of the human, which is obtained by the sensors of the robot. Given the pose of the human, the Gaussian cost model that represents the proximity zone of the human is shown in Figure 2-11.

Cells that are close to the human have very high costs (nearly lethal) and cells located further from the human have decreasing (non-lethal) costs. The Gaussian in the front of the human has a scaled covariance in the x direction, which is basically the direction of motion, according to the velocity of the human:

$$\sigma_x = \sigma \cdot factor \tag{2-6}$$

where σ and $factor$ are design parameters for the covariance and the scaling of the velocity respectively and are user-defined. Simply, the higher the velocity, the more pointy the frontal Gaussian. Respectively, the rear Gaussian becomes wider for higher velocities. This will lead the robot to respect sooner the proximity zone of the human, since he moves faster. In Figure

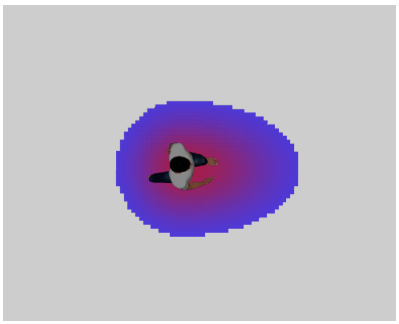
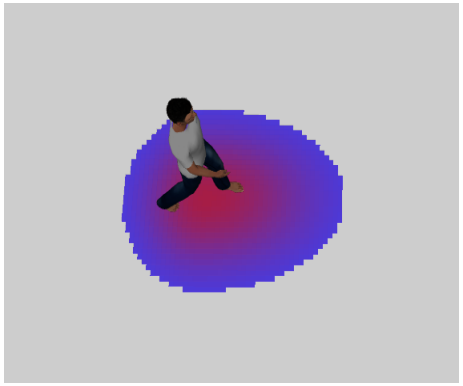
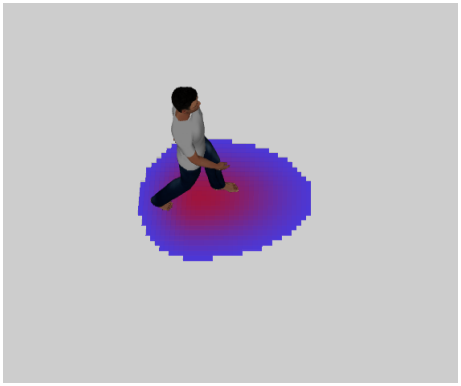


Figure 2-11: The proximity cost model with the assigned cost values. Lethal costs are assigned to the cells close to the human. Non-lethal decreasing costs are assigned to the surrounding cells as we move further from the human.



(a) Gaussian parameters: $\sigma = 0.2, factor = 5.$



(b) Gaussian parameters: $\sigma = 0.1, factor = 7.$

Figure 2-12: Visualization of the proximity layer which models the personal space, as a Gaussian costmap centered at the human. Different values for the covariance and the scaling factor result in a different size and shape of the Gaussian. The selection of the parameters is user-defined and depends on the structure of the environment and the desired behavior. In both cases a human velocity of $0.4m/s$ is selected.

2-12, two proximity costmaps are shown for different values of the covariance and the scaling factor.

It can be argued that, by increasing the aforementioned design parameters, namely the covariance and the scaling factor, paths can be moved further away from the human, increasing comfort. Nevertheless, it is mathematically proven [46], that there is a limit on how much these values can be increased before the optimal path chosen by the robot becomes the shortest one instead of the friendliest one. Path planning algorithms, in general, try to find the optimal path by balancing between distance and least cost. By increasing the covariance and/or the scaling factor too much, the cost of taking the shortest path outperforms the cost for moving far from humans, thus resulting in the selection of the shortest path that passes very close to the human. This means that an attempt to further increase human comfort would have the exact opposite results. Another limitation is that improvident increase of the proximity zone, would lead to elimination of the feasible friendly paths that the robot can follow, especially in confined environments such as a narrow corridor. As shown in Figure

2-13, increasing the variance of the proximity costmap, does not let free space for the robot to plan a path. In its attempt to find a feasible path, the robot would waver a lot, exhibiting an erratic and confusing behavior.

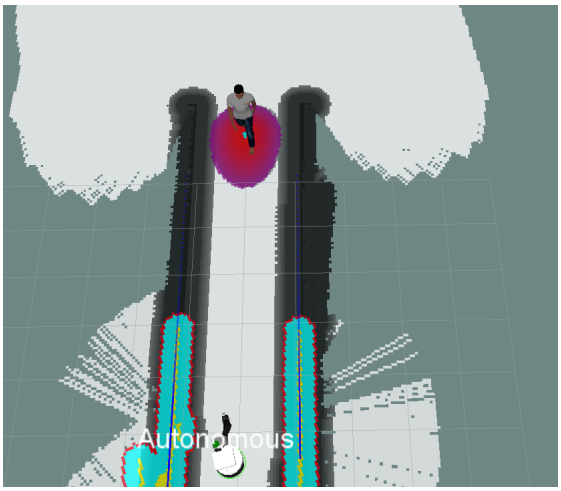


Figure 2-13: Increasing thoughtlessly the covariance and scaling factor of the proximity layer, could lead to situations where the robot is unable to find a feasible human-friendly path. This reasoning is evident especially in confined spaces such as the corridor depicted in this Figure. It is obvious that the proximity space blocks almost entirely the hallway.

From these examples, becomes evident that the selection of design parameters for the Gaussian proximity costmap is a troublesome task and highly dependent on the environmental structure. There is no general strategy to tune the Gaussian costmap. A mathematical description on how the parameters affect the resulting path and therefore the behavior of the robot is given in [46]. For the purposes of this thesis, the parameters where selected after manual tuning and based on each scenario. However, the restrictions defined in [46] were not violated and the robot's behavior was judged acceptable as will be discussed in Chapter 3.

2-2-8 Human Intention Layer

Although the introduction of the proximity layer improved significantly the motion behavior of the robot when humans are present, this approach does not take any prediction qualities into account and simply waits for the human to interact with the robot. Incorporating some form of prediction model for the motion of the human, can impel the robot to act sooner and plan a path that would require minimum interaction with the human. This is the principle idea behind the addition of the human intention layer. The purpose of this layer is to identify where the human intents to go and model this intention as a costmap that will lead the robot to plan a least cost path outside this area, as much as possible. To this end, in order to construct this layer, first the human intentions have to be calculated. For this purpose, a geometrical algorithm calculates the probability that a human heads toward a specific destination, based on the orientation of the human and the traversed trajectory up to that point. After the intentions have been calculated and the most probable destination has been extracted, a Gaussian cost model is assigned to the path between the human and that particular destination. The cost model accounts for future human positions and for

uncertainty over the trajectory. In the following, the mathematical description of the approach used to construct the cost model is presented.

Computation of intentions

We define as human intention the probability that a human h will move towards a specific destination d_i. Let us assume a set of probable destinations $D = \{d_1, d_2 \dots d_m\}$ obtained as explained in Section 2-2-5. The set D actually represents the overall set of intentions of the human. This means that, it is assumed that the human always moves to one of the destinations with a higher probability than to the others. The sum of probabilities is equal to 1.

Let T be a set of human detections along a certain trajectory. Then, this trajectory can be defined as:

$$\mathbf{X}_n = \{\mathbf{x}_n(1), \mathbf{x}_n(2) \cdots \mathbf{x}_n(T)\} \tag{2-7}$$

where $\mathbf{X}_n$ is the n observed trajectory of a human, and $\mathbf{x}_n(t)= [x(t), y(t)]_n$ is the detection at time t of the human along the trajectory n, with respect to a global frame, namely the world frame. The computation of a human intention at time t is the computation of the probability that a human that moves along the trajectory $\mathbf{X}_n$ and at time t is at $\mathbf{x}_n(t) = [x(t), y(t)]_n$, is heading towards the destination d_i. The method used in this thesis in order to compute the aforementioned probability is called Bayesian Human Motion Intentionality Prediction (BHMIP) and is introduced by Ferrer and Sanfeliu in [47].

Bayesian Human Motion Intentionality Prediction Let us assume a human that moves along a trajectory $\mathbf{X}_n$. At time t the human is at position $\mathbf{x}_n(t) = [x(t), y(t)]_n$, as shown in Figure 2-14. The orientation of the human is given by the first derivative of the followed trajectory, as a function of the current position $\mathbf{x}_n(t)$ and the previous position $\mathbf{x}_n(t-1)$. Let P_{Int_i} denote the probability that a person will pursue destination d_i. P_{Int_i} denotes then the *intention* of the human to move towards d_i. u_{ni} defines the angle between the current orientation of the human and destination d_i. Authors in [47] claim that there is a high similarity between the u angle Posterior Density Function (PDF) and a Gaussian function. Hence, the probability of pursuing destination d_i while moving along the trajectory $\mathbf{X}_n$ is given by:

$$P\left(\mathbf{x}_n(t) | \mathbf{x}_n(t-1), d_i\right) = \mathcal{N}\left(u; 0, \sigma_u\right) \tag{2-8}$$

where $\mathcal{N}\left(u; 0, \sigma_u\right)$, is a Gaussian distribution of zero mean and covariance σ.

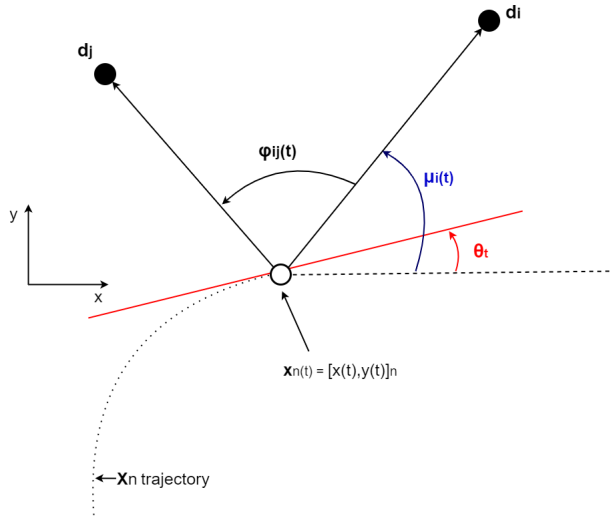


Figure 2-14: A person, moving along a trajectory $\mathbf{X}_n$, at time t is at $\mathbf{x}_n(t) = [x(t), y(t)]_n$ with an orientation θ_t. The angle between two probable destinations is $\phi_{ij}(t)$. $\mu_i(t)$ is the angle between the x axis and the destination d_i.

However, as shown in Figure 2-14, the angle u can be delineated by the relative difference of the angles θ_t and $\mu_i(t)$. More precisely, $\mu_i(t)$ is a measure relative to the destination d_i and θ_t is a global measure of the target orientation. Therefore the probability 2-8 can be given as:

$$P\left(\theta_t | d_i\right) = \mathcal{N}\left(\theta_t | \mu_i(t), \sigma_\theta\right) \tag{2-9}$$

where $P_{\theta_t|d_i}$ denotes the probability that a human that moves towards d_i has the orientation θ.

As stated in BHMIP paper [47], the problem of finding the most probable destination, namely finding the probability 2-9, boils down to a sequential data classification problem, where the decision of choosing a specific destination is taken at each time instant as the person walks. This problem is formulated as a Bayesian classifier as shown in Figure 2-15.

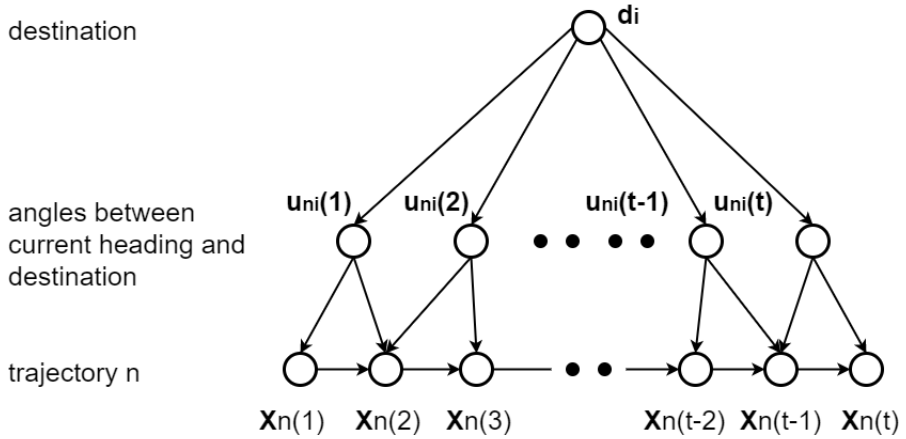


Figure 2-15: The graphical model of the Bayesian classifier. The relation of the destination d_i at time t is defined by the relation of the angle u_{ni} between the destination and the human orientation and the relation of the human positions of the trajectory n. (Adapted from [47]).

Using the Bayes theorem, the probability of pursuing the destination d_i, given that the human moves along the trajectory $\mathbf{X}_n$ is:

$$P\left(d_i|\mathbf{X}_n(t)\right) = \frac{P\left(\mathbf{X}_n(t)|d_i\right)P(d_i)}{P\left(\mathbf{X}_n(t)\right)} \tag{2-10}$$

or since the orientation of the human is a function of the followed trajectory. the above probability is equivalent to:

$$P\left(d_i|\theta_t\right) = \frac{P\left(\theta_t|d_i\right)P(d_i)}{P\left(\theta_t\right)} \tag{2-11}$$

After having formulated mathematically the problem of finding the most probable destination, or equivalently estimating the human intention, the next task is to actually calculate probability 2-11. The problem of calculating these probabilities is set down as a Hidden Markov Models (HMM) problem as described in [30].

Hidden Markov Models Let Equation 2-9 be redefined as:

$$b_i\left(\theta_t\right) = P\left(\theta_t|d_i\right) = \mathcal{N}\left(\theta_t|\mu_i(t), \sigma_\theta\right) \tag{2-12}$$

then if the angle θ_t is close to $\mu_i(t)$, we get a high value of $b_i\left(\theta_t\right)$,yielding high likeness that the human is pursuing destination d_i. Instead, if $b_i\left(\theta_t\right)$ is low, the destination d_i is hardly probable. Taking into consideration that a human might change his mind and decide to head towards another destination d_j, the *transition probability* $\alpha_{ij}(t)$ is defined as:

$$\alpha_{ij} = P\left(d_j|d_i\right) = \eta\mathcal{N}\left(\phi_{ij}(t)|0, \sigma_\alpha\right) \tag{2-13}$$

where $\phi_{ij}(t)$ is the angle between two destinations d_i and d_j as shown in Figure 2-14, and η is a normalization factor defined as:

$$\eta = \sum_{j=1}^{m} \alpha_{ij}(t) \tag{2-14}$$

where m is the number of destinations within the set of destinations D. The higher the angle ϕ_{ij} between d_i and d_j, the less likely that the person will change his mind to move to d_j instead of d_i at time t and hence, the lower the transition probability.

The desired probability $P\left(d_i|\theta_t\right)$ which denotes the probability that a human will head towards d_i at time t, can be expressed in HMM terms as:

$$P = (d_i|\theta_{1:T}, \lambda) \tag{2-15}$$

where T is the total number of observations up to that time, $\theta_{1:T}$= $\{\theta_1, \theta_2 \cdots \theta_T\}$ is the set of observation up to that time and λ is the set of HMM parameters. In HMM parlance the desired probability is referred to as $\gamma_t(i)$ and in order to find it we make use of forward and backward algorithms. A nice reference for how HMM work, can be found in [48]. In HMM the forward parameter $a_t(i)$ is defined as:

$$a_t(i) = P\left(d_i, \theta_{1:t}|\lambda\right) \tag{2-16}$$

and can be expressed as:

$$a_t(i) = \left[\sum_{i=1}^{m} a_{t-1}(j)a_{ji}\right] b_i(\theta_t) \tag{2-17}$$

The backward parameter $\beta_t(i)$ is defined as:

$$\beta_t(i) = P\left(\theta_{t+1:T}|d_i, \lambda\right) \tag{2-18}$$

and can further be expressed as:

$$\beta_t(i) = \sum_{i=1}^{m} \alpha_{ij}(t) b_j(\theta_{t+1}) \beta_{t+1}(j) \tag{2-19}$$

Using the forward and backward parameters and following the approach in [30], the probability $P = (d_i|\theta_{1:T}, \lambda)$ is obtained by:

$$P\left(d_i|\theta_{1:T},\lambda\right) = \frac{P\left(d_i,\theta_{1:T},\lambda\right)}{P\left(\theta_{1:T},\lambda\right)} \tag{2-20}$$

$$= \frac{P\left(d_i,\theta_{1:T}|\lambda\right)}{P\left(\theta_{1:T}|\lambda\right)} \tag{2-21}$$

$$= \frac{P\left(d_i,\theta_{1:t},\theta_{t+1:T}|\lambda\right)}{P\left(\theta_{1:T}|\lambda\right)} \tag{2-22}$$

$$= \frac{P\left(\theta_{t+1:T}|d_i,\theta_{1:t},\lambda\right)P\left(d_i,\theta_{1:t}|\lambda\right)}{P\left(\theta_{1:T}|\lambda\right)} \tag{2-23}$$

$$= \frac{a_t(i)\beta_t(i)}{\zeta} \tag{2-24}$$

where ζ is given by:

$$\zeta = \sum_{i=1}^{m} a_t(i)\beta_t(i) \tag{2-25}$$

Then the desired probability is given by:

$$P = (d_i|\theta_{1:T},\lambda) = \gamma_t(i) = \frac{a_t(i)\beta_t(i)}{\sum_{i=1}^{m} a_t(i)\beta_t(i)} \tag{2-26}$$

Probability 2-26 denotes the most probable destination of a human that moves along $\mathbf{X}_n$ and at time t is at position $[x(t), y(t)]_n$ with an orientation θ_t. Having extracted the most probable destination for the human, the next task is to construct and assign a cost function along the path that the human is expected to take to reach this destination.

Construction of the costmap

Let us assume for now that there are no obstacles or walls between the person and his destination. A straightforward assumption is that the person will walk directly towards the destination without making any circumventions because this is simply the most easy and convenient way. Hence, the path that the person will follow is known. Therefore, we can construct a cost model along this path, in order to discourage the robot from intersecting with it. To this end, the well known Gaussian cost function, discussed in Section 2-2-7, is employed.

However, simply by pointing the Gaussian function to the direction of the most probable destination, adds no value to the proximity layer itself, since the robot is still allowed to approach the human at least to the boundary of his proximity zone. We need to make the robot aware of the person's presence earlier and encourage it to plan a path that moves further away from it sooner. The robot by following this path, indicates to the human that it is aware of him and that will try to maneuver.

For this purpose we endow the robot with some sort of predictive behavior by expanding the Gaussian cost model along the human path for a number of prediction steps. The number

of prediction steps is determined based on the relative distance between the robot and the human and is scaled by the velocity of the robot as:

$$\texttt{num\_of\_steps} = \frac{\texttt{relative\_distance}}{\texttt{magnitude} * f}$$

where `num_of_steps` is the number of prediction steps, `relative_distance` is the relative distance between the robot and the human, `magnitude` is the magnitude of the human velocity and f is a factor for scaling the intention costmap and is a user preference.

For expanding the cost model, consecutive Gaussian functions are computed, one for each prediction step. Each Gaussian f_{Gauss_i} has an amplitude A_i and variances σ_{xi} and σ_{y_i} in the x and y axis respectively. In order to account for uncertainty over the human path, the Gaussians have an increasing variance in the y direction (σ_y) proportional to the prediction step. If σ_y is the variance of the first Gaussian in y direction, belonging to the proximity layer, then the variance in y of the ith Gaussian will be:

$$\sigma_{y_i} = \sigma_y \cdot (1 + i \cdot fc)$$

where fc is a scaling factor and is defined by the user. In Figure 2-16 the Gaussian cost model of the intention layer is shown. The cost model is assigned towards the destination corresponding to probability 2-15 or equivalently 2-26.

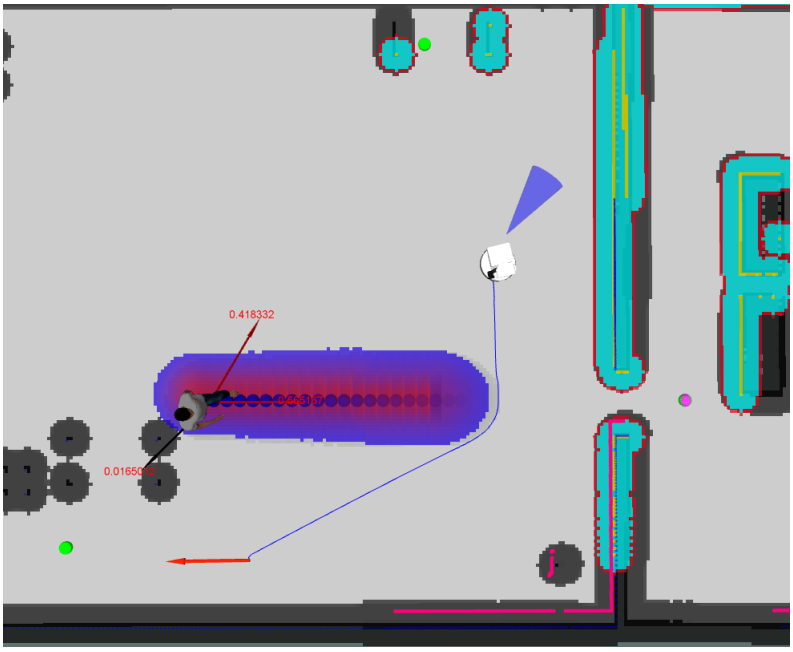


Figure 2-16: The human intention layer which constitutes of consecutive Gaussians with a slightly increasing variance in y. The human is heading towards the most probable destination following a straight path and the costmap is assigned along this path. The red arrows show the intentions (probabilities) for reaching the destinations. The light red in the most probable intention. The blue spheres denote the prediction steps. The robot tries to find a path that does not intersect with the future human positions.

2-2-9 Human path planning

Knowing the intention of the human provides some form of prediction capabilities to the robot and it constitutes it able to know the most probable future positions of the human and thus avoid interfering with his path. Nevertheless, the described method for constructing the human intention layer assumes an obstacle-free environment and straightforward motion towards the destination. Yet, in real domestic environments, static and moving obstacles are ubiquitous and humans have to maneuver amidst them to go where they want. If the robot was able to predict the circumvented path that a person is likely to follow to reach the destination, then assigning the cost model along this path would yield better results in inferring human intention in cluttered environments. This reasoning is illustrated in Figure 2-17. The human, in order to reach point d has to move around the table. The straight path A is obviously not feasible, so instead of assigning the cost model to this path, it would be more reasonable to assign it along the circumvented path B, which is a very likely path to choose. Again by slightly expanding the cost model, we can account for some uncertainty.

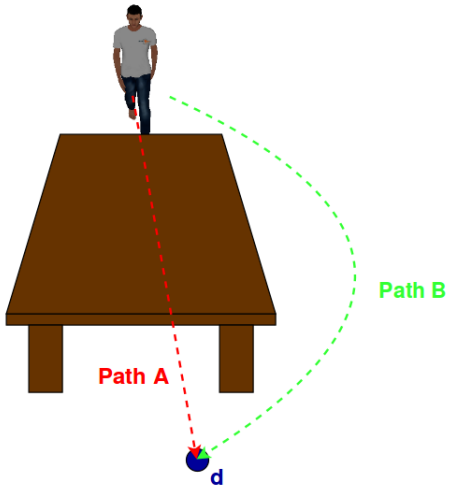
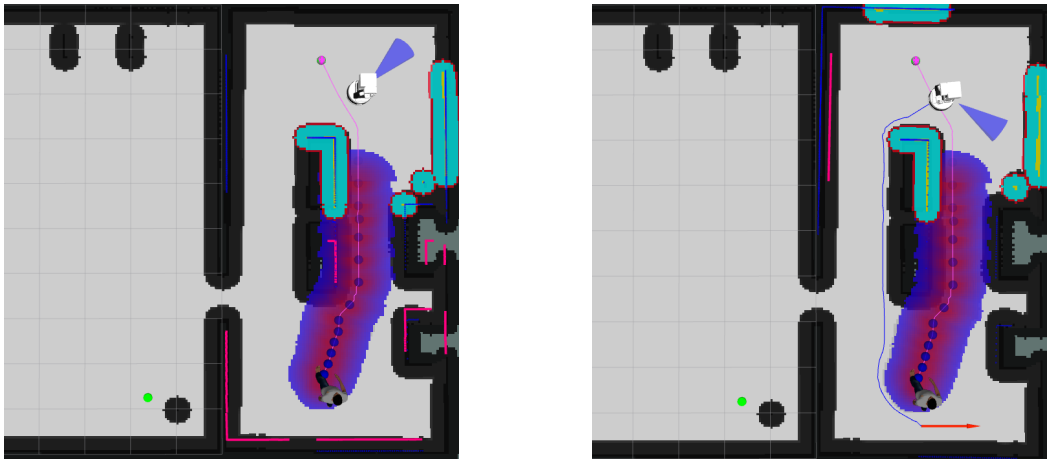


Figure 2-17: The human walks towards a destination d. However, the way is obstructed by a table. Hence, the straight path A towards d is not feasible and assigning the cost model to it is impractical. However, the human is expected to circumvent the table by choosing the free path B. Although the person could choose the other side of the table, it is argued that it would follow the shortest path.

This form of human path planning can be performed by the robot in real time for the human, as it would make for itself. For this purpose, what is needed is the current location of the human, the destination location and a local map of the environment. Then by use of a conventional planner (such as Dijkstra) the shortest collision-free path between the human and the destination can be computed, by looking for the least cost path inside the local costmap. Necessary condition is that both the human and the destination should be included at each time within the local costmap. This local costmap is the one that the robot uses for local path planning. This decision is made based on the fact that the robot should infer a path for the human based on its own perception of the environment, namely what it sees through its sensors. Depending on how far ahead we want the robot to be able to infer the human

path, the local costmap of the robot can be expanded accordingly with a cost of increasing computational complexity, as a bigger portion of the environment has to be updated. Though, in domestic environments where the lookahead of the robot is by default limited, this should not create any problems. After all, the robot does not need to know the human path beyond its sensors range since not even humans know how other persons move if they are not able to see them.

Figure 2-18 illustrates the idea of the intention costmap assigned along the most probable path. The human is anticipated to follow a circumvented path, similarly to the reasoning exhibited in Figure 2-17. By assigning the costmap along this path, the robot is forced to find a least-cost path which makes it pass from the other side of the obstacle without intersecting with the human, as shown in Figure 2-18b.

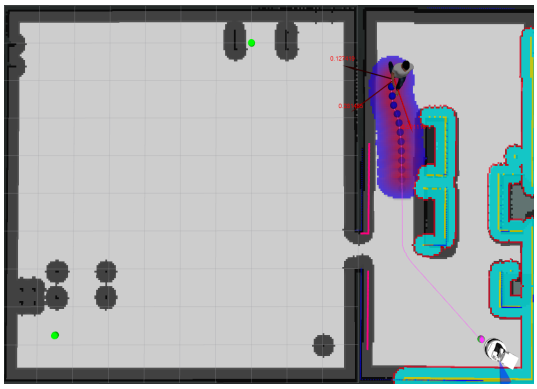


(a) The intention costmap along the expected path.

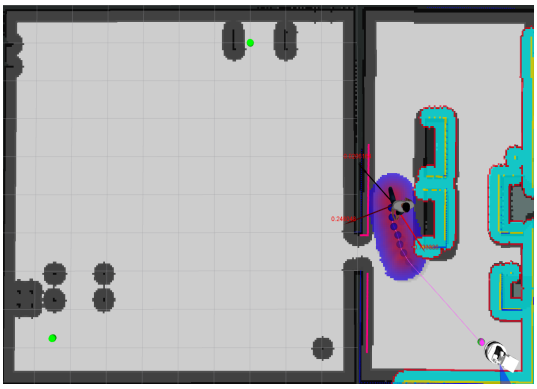
(b) The robot finds a path on the other side of the obstacle.

Figure 2-18: The intention costmap assigned along the expected human path. The shortest, collision-free path that the human is expected to follow in order to reach his destination is denoted by the pink path. The blue spheres denote future human positions. The other destinations in the environment are shown with green spheres and the most probable one is shown by the purple sphere. The robot is forced to find a way (blue path) on the other side of the obstacle.

A major advantage of the human path planning implementation is that the robot is able to replan the path online, as the human walks and changes positions. Hence, even if the human hesitates or performs strange movements, a collision free path to the destination would always be found (if exists) as long as both the human and the end destination lie within the local costmap of the robot. In Figure 2-19, the collision free path between a human moving in a cluttered domestic environment, and the most probable destination as pointed by probability 2-26 is illustrated. As the human walks toward the destination, the relative distance between him and the robot decreases, hence the number of prediction steps becomes smaller and the intention costmap shrinks.



(a) Intention costmap when the human is far from the robot.



(b) Intention costmap when the human is closer to the robot.

Figure 2-19: The shortest, collision-free path that the human is expected to follow in order to reach his destination is denoted by the pink path. The blue spheres denote future human positions. The red arrows show the probabilities for each intention. The other destinations in the environment are shown with green spheres and the most probable one is shown by the purple sphere. The path of the human circumvents among obstacles in the environment and the costmap is assigned along it. The number of prediction steps, is adjusted with respect to the relative distance and the human velocity.

2-3 Summary

In this Chapter the theoretical background and the mathematical principles of the proposed human-aware navigation framework were introduced.

In the beginning, the concept of the layered costmap was explained and its advantages comparing to the conventional monolithic costmap were presented. The numerous benefits that the layered architecture offers, constitute it a suitable choice for the development of our framework.

Consequently each component of the proposed framework was discussed. Emphasis was given to the additional layers that model the proximity zone around the human and the human intention.

At first, the proximity zone is modeled as a mixture of Gaussian cost functions, a symmetric in the back of the human and an elliptical in the front, scaled by the velocity of the human.

As far as the human intention layer is concerned, after having determined a set of destinations within the environment, the problem of finding the probability that a human will visit each one of these destinations, is formed as a Bayesian classifier and is solved by use of a recursive HMM algorithm. The algorithm computes the highest probability which denotes the most probable destination for the human. Afterwards, given the position of the human and the location of the most probable destination, the shortest collision-free path between these points is computed by means of the Dijkstra algorithm. The computed path is the one that is expected to be followed by the human in order to reach the destination.

The last step is to assign the intention costmap along this path. The intention costpmap consists of a cascade of Gaussians with increasing variance, in order to capture some uncertainty

for the future human positions. In the next Chapter, the proposed framework is tested and compared in simulated experiments to validate its effect on human comfort.

Chapter 3

Experimental Evaluation

In this Chapter, the proposed human-aware navigation framework is benchmarked against two state-of-the-art navigation methods in a series of simulated experiments. At first, a short description of the experimental setup and the sensors used, is given. Afterwards, the simulation studies and the obtained results are presented.

The purpose of the experiments is to evaluate how friendly the robot motion is, namely how comfortable a person feels with the motion of the robot, when apart from the proximity zone around humans, also human intention is taken into consideration. The proposed method is compared with the Robot Operating System (ROS) navigation stack and the social navigation introduced by Lu et al. [21]. As a main metric to evaluate human comfort, the relative distance that the robot keeps from the human during their interaction, is concerned. The key premise is that the bigger this distance is, the more comfortable the human feels with the robot.

Although the relative distance may be a good indicator of how much friendly a robot motion is, nevertheless distance alone is not adequate to render a motion as socially acceptable. Therefore, additional metrics are used for the comparison of the methods. The execution time of the robot motion, the length of the path between the robot's start position and its target as well as the smoothness of the motion are also addressed.

By these experiments, we intend to demonstrate that the proposed framework drives the robot to keep larger distances from the human and that it moves in a smoother way than with the currently used approaches, while at the same time exhibits comparable efficiency in terms of time and path length. This results in increasing human comfort without undermining navigation performance.

3-1 The Experimental Setup

3-1-1 Platform description

In the experiments, a simulated model of the Marco robotic platform, presented in Section 2-2-2, was used. The robot's mobile base has a diameter of 0.5 m and carries the motion actuators and the laser sensors. The robot model contains all the necessary transformations and the simulated dynamics and kinematics of the real robot. The real and the simulated robot are shown in Figure 3-1.

(a) The real robot.

(b) The simulated robot.

Figure 3-1: The real Marco robot and the simulated model. They both have the same sensor configuration, transformations, dynamics and kinematics. The real robot has some extra features (such as the screen on the top) that are not considered in the simulation studies.

The simulations were run on a HP ZBook 15 G2 Mobile Workstation with an Intel Core i7 2.5GHz processor and an onboard Intel Haswell Mobile graphics card. The laptop runs 64-bit Ubuntu 14.04 and ROS Indigo.

3-1-2 Sensors description

- The laser scanner is manufactured by SICK and is located at the front part of the mobile base. The sensor measures distances in the horizontal plane and is a key asset for mapping, navigation and obstacle detection. It has a range from 0.05 to 10m and a frequency of 15 Hz. The field of view is 180^o and the step angle is 0.33^o.
- The RGB-D sensor used is the Xtion Pro Live by ASUS. It is mounted inside the head of the robot and provides RGB images along with depth images, using an IR projector and an IR camera. From the depth image, the point cloud of the scene is obtained. The field of view of the sensor is 58^o horizontally, 45^o vertically and 70^o diagonally. It is connected to the robot's computer using a USB 2.0 interface. The depth range is from 0.4 to 8 m. The sensor can be used for 3D obstacle detection as well as human detection. In the experiments, the most essential use is to detect obstacles at different heights, such as the surface of a table.

3-1-3 Simulated environment

All the experiments were conducted in Gazebo simulator (Appendix A-2), where the entire robot with its sensors and actuators was simulated. The simulated sensors were configured to resemble the specifications and properties of the real sensors of the robot.

The Gazebo version in which the robot was simulated, does not support physics for simulating a moving human model. Therefore, a fake detection module was employed, which simulates a person walking along a predefined trajectory with constant velocity. The person is 'virtual', meaning that its legs are not actually detected by the laser sensor. The human position is published as a fake scan detection in the place where the physical person should be detected. An interactive marker is published at the location of the virtual person for visualization purposes.

3-2 Simulation studies

3-2-1 Approach and assumptions

In order to evaluate if the proposed framework is able to produce socially acceptable robot motion, we compared it with two state-of-the-art navigation methods.

The first competent is the standard ROS navigation provided by the ubiquitous ROS navigation stack (Appendix B-1) which is currently used in Marco (and TIAGo) robot, as well as in the majority of commercial robots worldwide. The standard ROS navigation treats humans as mere dynamic obstacles and simply commands the robot to stop before entering the inflation zone around them. Recent research has proven that the produced motion, although safe, lacks human-awareness and legibility and thus the robot fails to be accepted as a daily interactor. For ease of notation, the standard ROS navigation will be denoted as **Standard ROS** in the figures of this Chapter.

The second method is the social navigation method proposed by Lu et al. [21] in which the authors introduced the Social Layer on top of the standard navigation layers used in the ROS navigation stack. The Social Layer models the proximity zone around the human and dictates the robot to respect this area in order to fulfill the social constraints. This method is explained in Sections 2-1-3 and 2-2-7. The method has proven to be superior to standard ROS navigation and therefore it is considered as a strong baseline for comparison with the proposed framework. For ease of notation, the method of Lu et al. will be denoted as **PROX** in the figures of this Chapter.

The proposed human-aware navigation framework, builds on top of the work by Lu et al. and incorporates apart from the proximity layer, also the human intention layer. For this reason, it will be denoted as **PROX&HI** (Proximity & Human Intention) in the figures of this Chapter.

The purpose of the simulation studies is to show that the incorporation of the human intention layer is able to produce friendlier robot motion that outperforms both the standard ROS and the social navigation. For this reason, three interaction scenarios that are frequent in domestic environments, are investigated. In these scenarios we compare the three candidate methods under some assumptions and by use of different evaluation metrics. The methods

are evaluated in terms of human comfort, smoothness in motion and navigation performance as will be explained in the following.

Navigation planners

The ROS navigation stack provides a global and a local planner to find a collision free path within a costmap. The planners used in the simulation studies are the same for all three contestant methods. As global planner, the A* algorithm provided by the `global_planner` ROS package is used (further information can be found in Appendix B-1-4). As local planner a variation of the Vector Field Histogram (VFH) [49] is used. The variation was developed by PAL Robotics and uses the costmap instead of the laser scans directly.

The use of the same planners for all three navigation methods ensure that the motion of the robot strictly depends on the type of costmap used, which is the intended outcome of this thesis, since it does not propose a new planner but rather an additional context in navigation via the inclusion of human intention.

Defining different scenarios

Three different human-robot interaction scenarios, that are usual in a domestic environment, have been selected for evaluating the proposed framework.

The first scenario is the well-known *hallway passing*. This scenario has often been used in human-aware navigation literature ([50], [28], [51]) for performance evaluation. The robot enters a hallway and moves towards a target located at the other end of the hallway. At the same time, a human is approaching the robot coming from the opposite direction, as shown in Figure 3-2. The robot has to avoid collision with the human by moving to the right side of the corridor.

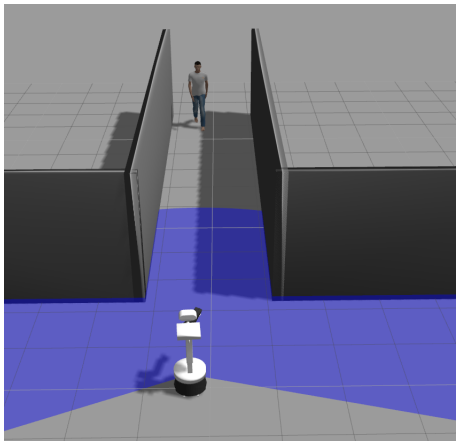


Figure 3-2: The hallway passing scenario. The robot moves in a hallway towards a target located at the far end of the hallway. At the same time, a human is approaching the robot coming from the opposite direction.

The second testing scenario is an *intersection* scenario ([52], [53], [22]). The robot moves to a target point B. A human walks in a direction, which crosses over the robot's direction,

towards a point A, and their future paths intersect at a point I. The scenario is illustrated in Figure 3-3.

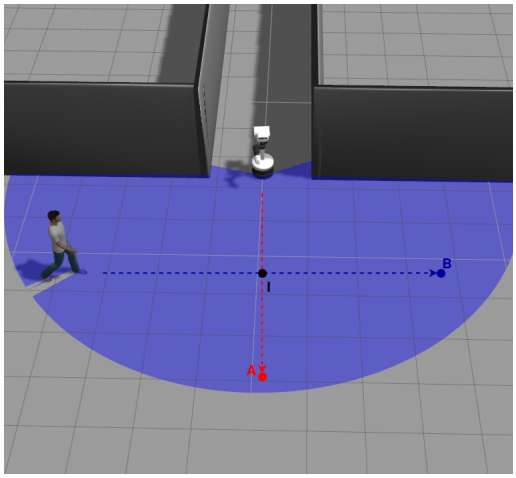


Figure 3-3: The intersection scenario. The robot moves to a target point B. A human walks perpendicularly to the robot's direction towards a point A, and their future paths intersect at a point I.

The last scenario depicts a *domestic scene* where the human and the robot move inside a room of a house. In the middle of the room there is a big obstacle (a large sofa). The human and the robot stand in two facing positions at each end of the sofa and intend to move to a position close to the other end. The robot intends to reach point R whereas the human plans to go to point D, as shown in Figure 3-4.

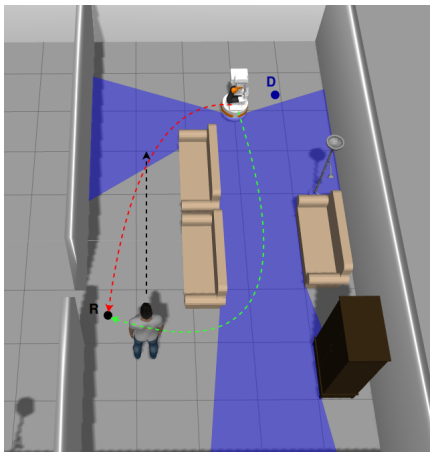


Figure 3-4: The domestic scene scenario. The robot's task is to go to point R and the human wants to go to point D. If the robot is able to infer the intention of the human, it should decide to follow the green path. Otherwise it will follow the red path (which is the shortest one) and will interact with the human.

The robot and the human will intersect if the robot follows the red path. However, if the robot is able to infer the intention of the human, it would choose a path that approximates the green path, avoiding interaction with the human at all. The domestic scene scenario aims to demonstrate the importance of the human path planning module.

Assumptions

The simulations were conducted under some assumptions. The main assumption is this of a 'dummy' person. A dummy person walks straight to its destination without stopping, with a constant velocity. This means that even if the robot is not able to avoid it on time, the dummy person will continue walking no matter what.

The second assumption is that the human position is always published by the fake detection mode. In real-life applications, the sensory data are usually noisy and quite often the readings are intermittent. In a simulated environment, the sensors are perfect and the incoming messages are always available.

The third assumption is relevant to the second. Since the human is virtual and continues walking under any circumstances, then it only bumps virtually on the robot. Since there are no physics regarding the human, there is only a visual collision between the interactive marker of the human and the corresponding marker of the robot model. Hence, there is no physical collision, but this does not undermine the credibility of our evaluation.

Finally, in the first two scenarios, the environment is obstacle free meaning that there is no obstruction between the human and his destination. Therefore, we assume that the human will move straightforward to the desired point, as this is the most legit behavior. In the domestic scene scenario the presence of the obstacle forces the human to deviate from the straight path. Then, we assume that he will pursue a path close to the shortest, collision-free path, with the latter being computed by the human path planning module.

Evaluation metrics

As already stated, the navigation methods are compared in terms of three criteria: **human comfort**, **smoothness** and **navigation performance**.

Human comfort Human comfort is a subjective criterion since each human may feel differently about the presence of the robot, as well as its motion. The level of comfort depends on numerous factors, such as the familiarity of the person with technology and machines, the professional and educational background, the age etc. Hence, human comfort is a so called qualitative metric. In order to quantify human comfort, mathematical values that measure comfort need to be employed. For this purpose, two metrics are defined. The first metric is the ***minimum relative distance*** measured between the human and the robot during their interaction. The measured relative distance is projected against the interaction zones specified by the proxemics theory [15], to judge the level of sociability that the robot exhibits, as shown in Figure 3-5. The relative distance is defined as:

$$d_{rel_{min}} = \min[d(t)], \qquad t \in [0, t_f] \tag{3-1}$$

where $d(t)$ is the measured relative distance over time and t_f is the moment that the robot reaches the target position.

The minimum relative distance denotes the closest point where the robot approaches the human. If that point lies within one of the zones of Figure 3-5, then it means that the robot

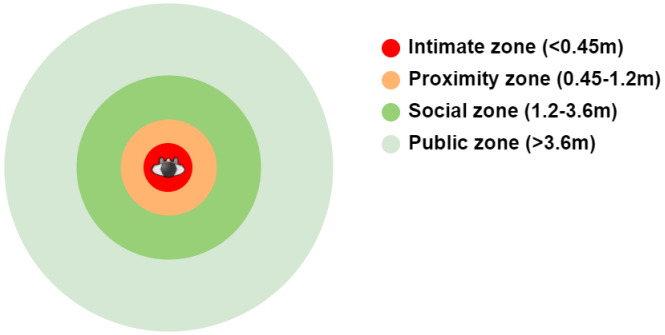
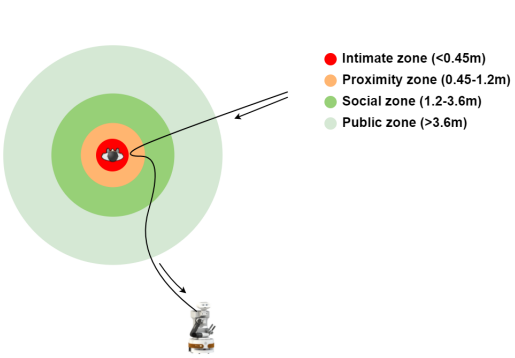


Figure 3-5: The interaction zones of a human, as specified by the proxemics theory.

has entered the corresponding interaction zone. Generally, in domestic environments, avoiding intrusion of the proximity zone can be considered as an acceptable behavior. Intrusion of the intimate zone yields not only unacceptable behavior, but also immediate danger for collision. Therefore, a robot motion that manages to keep the robot outside of the proximity zone (as much as possible) is considered friendly.

However, the minimum relative distance is not indicative enough for the friendliness of the motion. Figure 3-6a shows an example, in which the robot penetrates the proximity zone of the human, but only for a limited amount of time. The robot indeed approaches the human a lot (maybe due to a confined environment), but tries immediately to exit its proximity zone. On the contrary, Figure 3-6b illustrates a case where the robot stays for a long time within the proximity zone, even if it does not approach the human that much.



(a) The robot approaches the human a lot.

(b) The robot penetrates the proximity zone for a longer time.

Figure 3-6: An example that illustrates the importance of the integral of the relative distance, as a metric for quantifying human comfort. In Figure 3-6a, the robot penetrates the proximity zone of the human, but only for a limited amount of time. In Figure 3-6b the robot stays for a long time within the proximity zone, even if it does not approach the human that much.

From this example becomes evident that, an additional metric is needed that can quantify not only the distance, but also the way that the robot approaches the human as well as the amount of intrusion of its proximity zone. Therefore, we compute the ***integral of the relative distance*** from the moment that the robot enters a critical spatial area around the

human, till the moment that it exits it. The critical area is selected to be the double of the proximity zone ($\approx 2.5m$) around the human. Basically, when the robot enters the critical area, the person realizes that the robot is seeking interaction. The integral of the relative distance is equal to the area below the curve of the relative distance plotted over time as will be shown in Section 3-2-2. Higher values of this integral yield that the robot stayed less time inside the proximity zone or equivalently, approached less the human over time. The integral of relative distance is defined as:

$$I_{d_{rel}} = \int_{t_1}^{t_2} d_{rel}(t)dt \tag{3-2}$$

where $d_{rel}(t)$ is the measured relative distance between the robot and the human over time, t_1 is the moment that the robot enters the critical area and t_2 the moment that it exits it.

The last metric used for evaluating human comfort is the **number of failed runs**. A run has failed if the robot enters the intimate zone of the human. Entering the intimate zone is not only a paradigm of unfriendly motion, but also a dangerous situation, as there is immense risk of collision. Nevertheless, since the human in our experiments is virtual and there is no actual collision, it was decided to mark the simulation runs that were unsuccessful to provide acceptable motion, as failed. A failed run occurs if:

$$(x_r, y_r) \in Z_{int} \tag{3-3}$$

where x_r is the x position of the robot, y_r is the y position of the robot and Z_{int} denotes the intimate zone of the human.

Smoothness Distance alone is not a sufficient metric to characterize the robot motion as socially acceptable or not. Even if the robot manages to keep large distances, it can still move in a confusing way for the human, e.g. exhibiting a nervous or *erratic* motion with plenty of on spot rotations or by exhibiting hesitation in motion. Therefore, *smoothness* in motion is considered to be a second indicator of human friendliness. If the robot moves in a smooth and legible way, it is likely that the human will have a clearer view of where the robot intends to go. In order to measure the smoothness of the motion, the ***average integral of the angular and linear jerk*** of the robot is calculated for all three navigation methods. Higher values of the integral denote erratic and confusing motion, whereas the lower the integral of the jerk, the smoother the motion is. The reason for selecting the average integral (of a single simulated motion) is illustrated in Figure 3-7.

In the left part of Figure 3-7 the robot avoids the human by making an abrupt maneuver at a time at which it has already approached the human enough. In the right part of Figure 3-7, the motion of the robot is obviously friendlier as it makes an earlier maneuver and informs the human of its intentions in an earlier stage of the motion. Yet, it follows a larger path. The latter motion could yield higher cumulative absolute jerk value, since the robot moves for a longer time. Nevertheless, this is not indicative of an erratic motion, since the robot makes a smoother avoidance maneuver. By taking the average of the computed integral, we simply decouple the jerk value from time, constituting it a comparable metric which is independent from the time of the motion task. The average integral of the jerk (linear and angular) is given by:

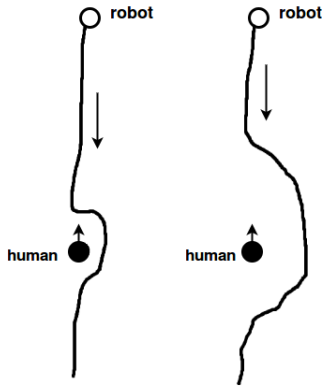


Figure 3-7: In the left image, the robot avoids the human by making an abrupt maneuver whereas in the right, the motion of the robot is obviously friendlier as it makes an earlier maneuver and keeps larger distance from the human. However, due to the larger path, the latter motion could yield higher cumulative absolute jerk value.

$$I_{j_{lin}} = \frac{\left(\int_{t_0}^{t_f} j_{lin}(t)dt\right)}{(t_f - t_0)} \tag{3-4}$$

$$I_{j_{ang}} = \frac{\left(\int_{t_0}^{t_f} j_{ang}(t)dt\right)}{(t_f - t_0)} \tag{3-5}$$

where j_{lin} is the linear jerk, j_{ang} is the angular jerk, t_0 is the starting time of the navigation run and t_f is the moment that the robot reaches at the target position.

Navigation performance For evaluating the performance of each navigation method, common quantitative metrics are employed. The first metric is the ***execution time***. The execution time is the time interval needed for the robot to reach its target position. Basically is the time needed to complete the navigation task. Higher execution times indicate either wandering of the robot or difficulty in avoiding humans and obstacles. Such behaviors denote poor performance of the navigation method. The second quantitative metric is the ***length of the path*** that the robot follows to reach its target position. It is merely the total traveled distance of the robot. Longer paths yield larger circumventions and lead to higher execution times. Though, a larger circumvention could mean a more efficient, as well as friendlier, collision avoidance behavior. Hence, larger paths do not necessarily mean degradation of navigation and should be evaluated together with the rest of the criteria to extract safe conclusions about the performance of the navigation method. Let us denote the total execution time of a navigation task as t_{task} and L the length of the traveled path.

In Table 3-1, a summary of the defined metrics and their meaning is presented.

Table 3-1: Metrics defined for the evaluation of the navigation methods.

Metric	Meaning	Formula
Human comfort		
Minimum relative distance	Minimum distance between robot and human.	$d_{rel_{min}} = \min[d(t)]$
Integral of relative distance	Distance kept over time within a critical region.	$I_{d_{rel}} = \int_{t_1}^{t_2} d_{rel}(t)dt$
Number of failed runs	Number of runs in which the robot entered the intimate zone of the human.	$(x_r, y_r) \in Z_{int}$
Smoothness		
Average integral of linear jerk	How much erratic is the translational motion.	$I_{j_{lin}} = \frac{\left(\int_{t_0}^{t_f} j_{lin}(t)dt\right)}{(t_f - t_0)}$
Average integral of angular jerk	How much erratic is the rotational motion.	$I_{j_{ang}} = \frac{\left(\int_{t_0}^{t_f} j_{ang}(t)dt\right)}{(t_f - t_0)}$
Navigation performance		
Execution time	Time that the robot needs to reach the target position.	t_{task}
Path length	Traveled distance until the robot reaches the target position.	L

Comments Before presenting the results of the simulation studies, a couple of comments are worth mentioned. The first comment concerns the specific coefficients of the optimization procedure (Appendix B-1-4) that is executed by the navigation planner to find the least-cost path. In all the experiments, the exact same values for the coefficients were used.

The second comment concerns the velocity of the robot. The velocity of the robot is dictated by the navigation planner. This means that the exhibited motion is a result of the dynamics of the navigation planner itself in its effort to find the path within the costmap. No velocity adaptation was performed. This could be an improvement as will be discussed in Section 4-1.

Finally and most importantly, both the global and the local navigation planners used in all the simulated experiments, are deterministic. This means that if all the configuration and

motion parameters are kept constant, then we should always obtain the same results. Hence, by perturbing one of the parameters of motion we can judge how this specific parameter affects the navigation results. In our simulated experiments, there is a huge number of parameters and variables that affect the navigation results. This great number arises mainly from the infinite choices of start and end positions for the robot and the human and from the numerous configuration parameters of each navigation method. Due to this vast amount of variables, it was decided to keep the costmap and planner parameters the same for all the simulations of each scenario. In addition, it was decided that the start and end position of the robot, as well as the start position of the human, in every scenario, would be the same. The only variable that is perturbed, in order to test each navigation method, is the velocity of the human. Hence, several simulation runs were executed for human velocities in a range of values that are legit for motion in domestic environments.

In this context, human velocity is the *independent variable* of our experiments, whereas the aforementioned metrics are the *dependent variables*. All the rest configuration and planning parameters are kept constant and it is argued that they do not have any effect on the results.

Configuration parameters Table 3-2 shows the values of some of the most important configuration parameters used in the path planning process. These parameters were kept the same in each scenario and for all the navigation methods. More information about the parameters can be found in Appendix B.

Table 3-2: General planning configuration parameters.

Type	Name	Parameter value
Global planner	`navfn`-A*	
		allow unknown area exploration: no
		cost factor: 3.0
		lethal cost: 253
		neutral cost: 50
		tolerance: 0.5
Local planner	PAL-VFH	
		x acceleration limit: 0.8 m/s^2
		y acceleration limit: 2.5 m/s^2
		maximum forward velocity: 0.3 m/s
		minimum forward velocity: 0.0 m/s
		maximum rotational velocity: 1.0 rad/s
		minimum rotational velocity: 0.0 rad/s
		weight for how much the local path should stay close to the global one: 0.6
		weight for how much the planner should attempt to reach its local goal: 0.8
		weight for how much the controller should attempt to avoid obstacles: 0.01
Global costmap		
		inflation radius: 0.45 m
		inflation cost factor: 2.0

		publish frequency: 1.0 Hz
		update frequency: 5.0 Hz
		resolution: 0.05
		cost for unknown areas: 255
Local costmap		
		inflation radius: 0.3 m
		inflation cost factor: 2.0
		publish frequency: 2.0 Hz
		update frequency: 4.0 Hz
		resolution: 0.05
		cost for unknown areas: 255

3-2-2 Results

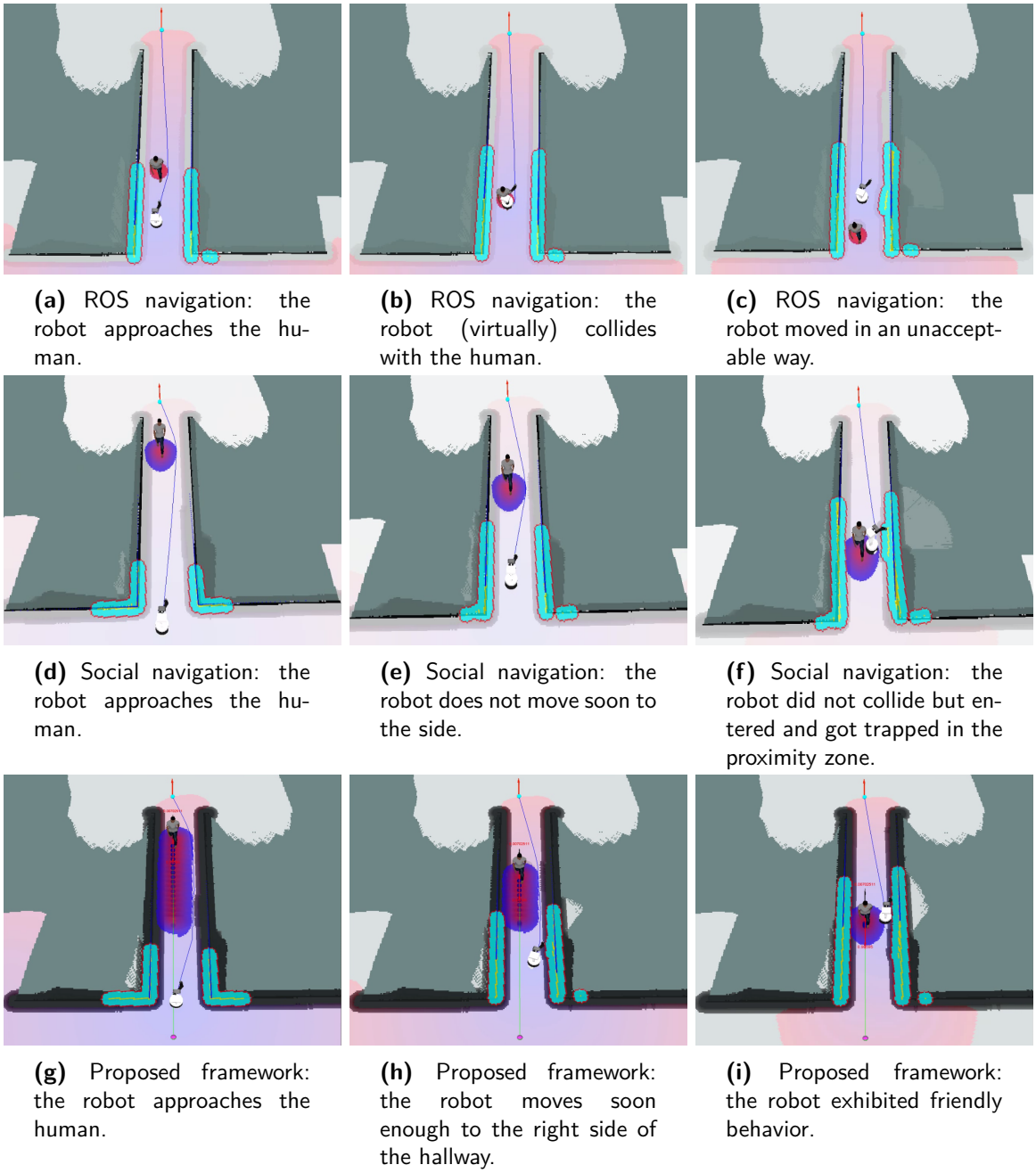
As already mentioned, it was decided to test the competent navigation methods under different human velocities within a range of values. In the following, the results of our evaluation in each scenario and for every navigation method are presented.

Scenario 1: Hallway passing

The first set of simulated experiments was conducted for the hallway passing scenario (Figure 3-2). The set contains 30 simulation runs at different human velocities. The human velocities were randomly generated by a simple Gaussian distribution with a mean $\mu = 0.4\ m/s$ and a standard deviation $\sigma = 0.1$. The resulting trajectories produced by each method are presented in the left column of Figure 3-9. The generated trajectory for every human velocity is drawn in gray color, whereas the generated trajectory of the robot and the corresponding human trajectory at the mean person velocity is highlighted and colorized to visualize the time dimension. Time is captured by the colorbar on the right of each plot. In the right column of Figure 3-9 the relative distance that the robot maintained from the human during their interaction, is plotted over time for every simulation. The relative distance is projected against the intimate and the proximity zone of the human. The relative distance measured at the mean human velocity is shown by the intense curve.

It is evident, that the standard ROS navigation made the robot to intrude the intimate zone of the human in every simulation run. If the human had actually a physical substance in our simulations, this motion would inevitably lead to collision. As can be seen from Figures 3-9b, 3-9d and 3-9f, the social navigation (PROX) and the proposed framework (PROXH&HI) managed to prevent the robot from entering the intimate zone in all the trials. Although the robot enters in many cases the proximity zone, due to the cost assigned to this area, the robot tries to exit it as soon as possible. In addition, the proximity costs avert the robot from further approaching the human. This behavior is also evident from Figure 3-8 which shows snapshots of the produced motion by each method, at the mean human velocity. As it can be seen, the standard ROS navigation method failed to produce acceptable motion and in fact, it led the robot to a virtual collision with the human. The social navigation method managed to avoid collision but the robot entered the proximity zone. On the other hand,

the proposed navigation framework drove the robot at the right side of the hallway early enough, by taking into account the intention costmap. As the robot approaches the human, the costmap decreases in size but it is able to keep the robot in a proper distance.



(a) ROS navigation: the robot approaches the human.

(b) ROS navigation: the robot (virtually) collides with the human.

(c) ROS navigation: the robot moved in an unacceptable way.

(d) Social navigation: the robot approaches the human.

(e) Social navigation: the robot does not move soon to the side.

(f) Social navigation: the robot did not collide but entered and got trapped in the proximity zone.

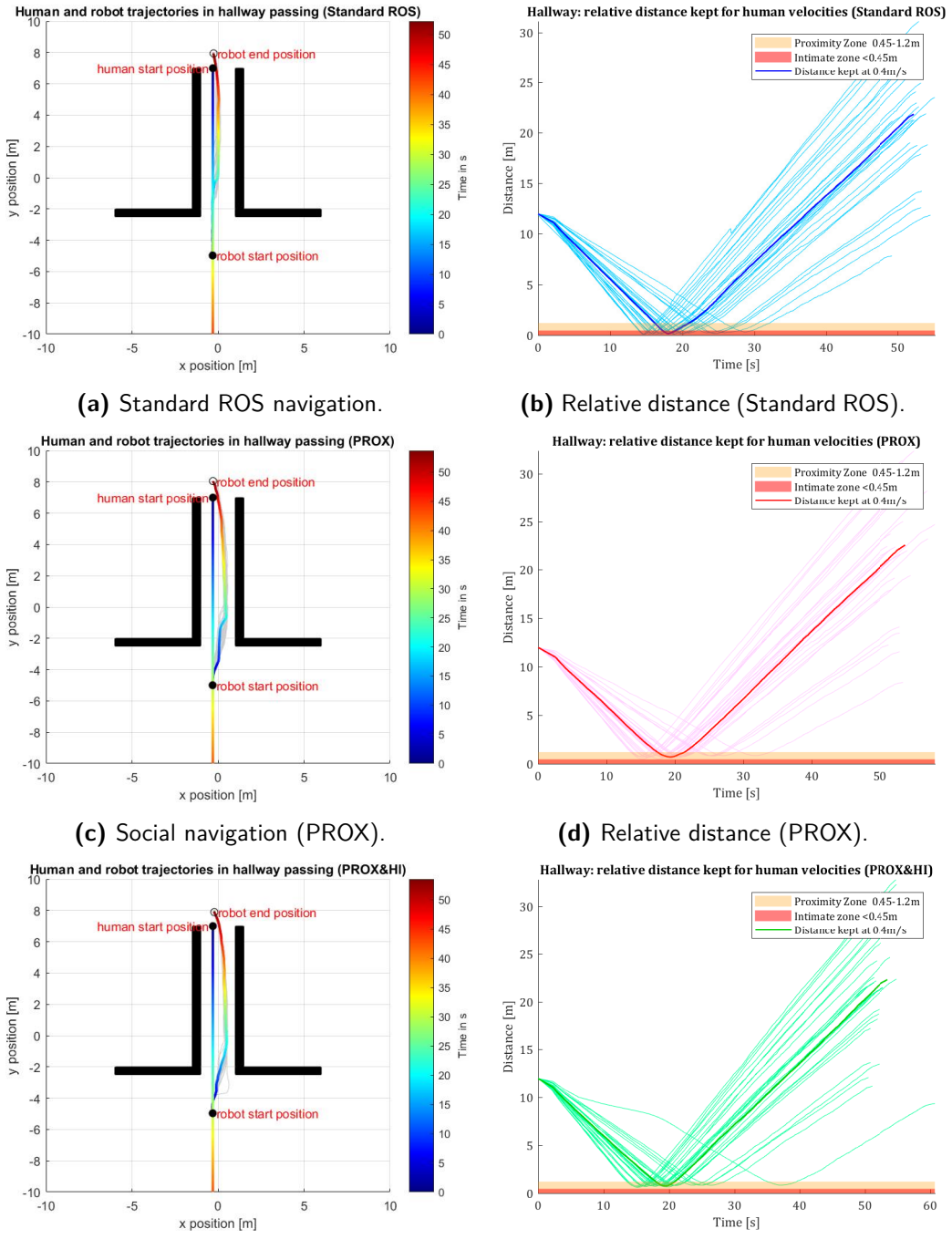
(g) Proposed framework: the robot approaches the human.

(h) Proposed framework: the robot moves soon enough to the right side of the hallway.

(i) Proposed framework: the robot exhibited friendly behavior.

Figure 3-8: Hallway passing. Illustration of the produced motion by each method at the mean human velocity. The proposed framework produced the most human-friendly motion comparing to the other two methods. At the bottom of Figures 3-8g-3-8i, the most probable destination (and the expected path) of the human is shown. The intentions are denoted by the red arrows.

In order to better evaluate the performance of each navigation method, the defined metrics are compared in the box plots in Figures 3-10, 3-11 and 3-12.



(a) Standard ROS navigation.

(b) Relative distance (Standard ROS).

(c) Social navigation (PROX).

(d) Relative distance (PROX).

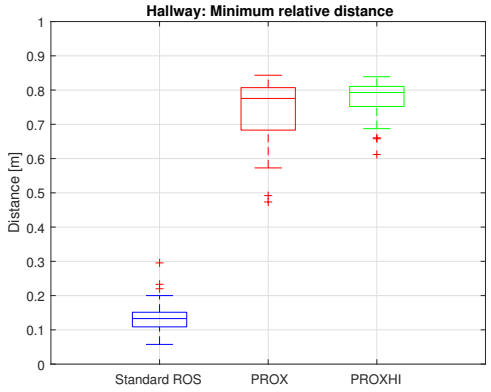
(e) Proposed framework (PROX&HI).

(f) Relative distance (PROX&HI).

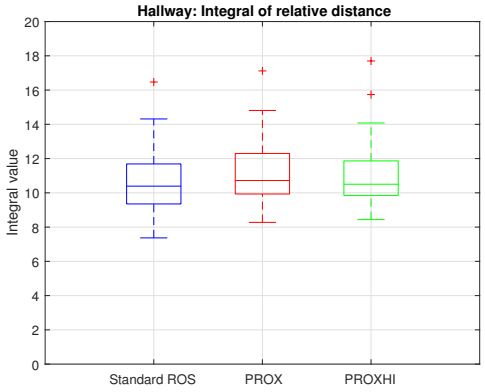
Figure 3-9: Hallway passing. Generated trajectories from each method and the relative distances kept for each simulation.

The central line in each box indicates the median value of the corresponding dataset. The bottom and top edges of each box denote the 25th and the 75th percentile respectively. The whiskers reach the extreme points that are not considered as outliers and the latter are denoted by the red crosses. Figure 3-10a shows the minimum relative distance kept by each navigation method. The social navigation and the proposed framework managed to keep the robot further from the human than the standard ROS navigation, which completely failed to produce safe navigation entering in all the trials in the intimate zone, as illustrated in Figure 3-9b. The medians of the two methods are close but there is a bit higher variation in the case of social navigation. From the integral of relative distance in Figure 3-10b, it can be seen that the robot approached the human in a similar way, for all the methods.

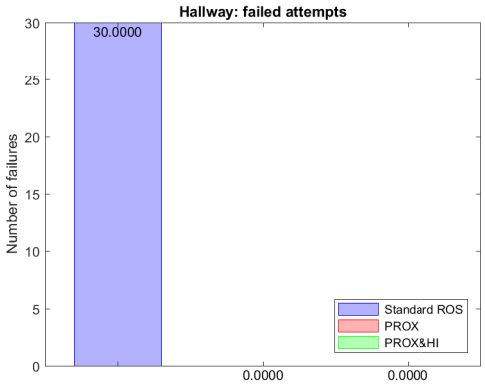
As far as smoothness is concerned, the results regarding the integral of the linear and angular jerk are shown in Figure 3-11. The standard ROS navigation exhibits the lowest values for both the linear and the angular jerk, but this result is not valid, as all the trials failed. The social navigation and the proposed framework demonstrate comparable results.



(a) Minimum relative distance.

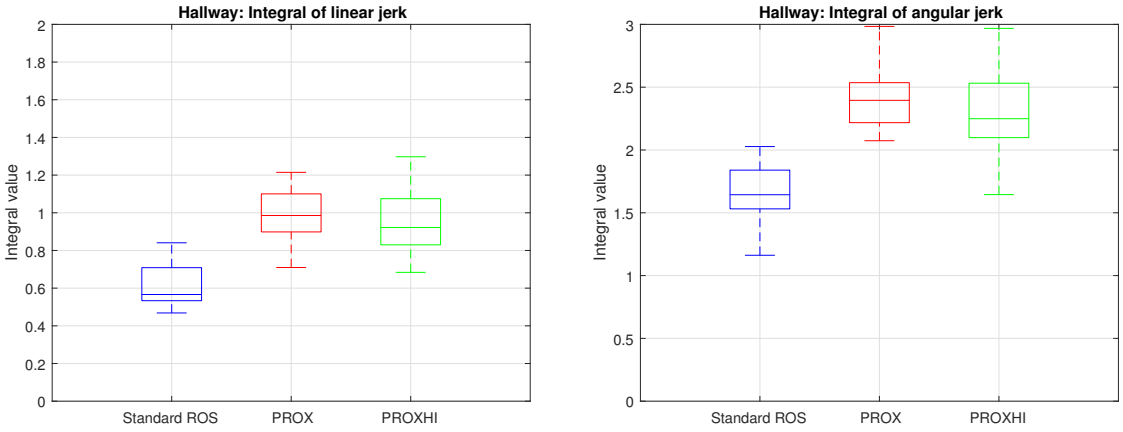


(b) Integral of relative distance.



(c) Average failed attempts.

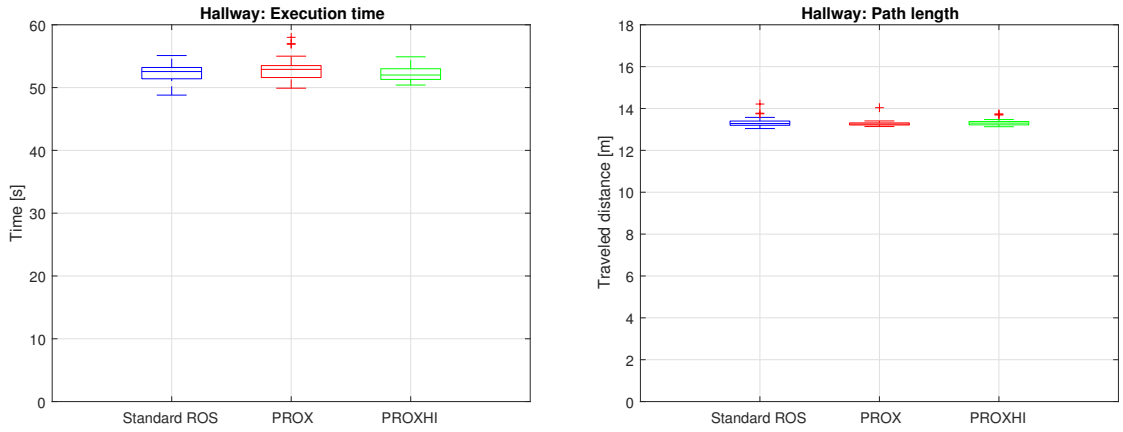
Figure 3-10: Hallway passing. Comparison of the metrics that evaluate human comfort. Figure 3-10a shows the minimum relative distance kept by each method. Figure 3-10b shows the integral of the relative distance. Figure 3-10c shows the average number of failed attempts.



(a) Integral of linear jerk. **(b)** Integral of angular jerk.

Figure 3-11: Hallway passing. Comparison of the integrals of the linear and angular jerk. Higher values of the integrals yield more erratic motion.

The last comparison concerns the navigation performance demonstrated by each method. Figure 3-12 shows the execution time and length path for each of the methods. As it can be seen, all methods exhibited similar execution times and the robot traveled approximately the same distance in all the cases.

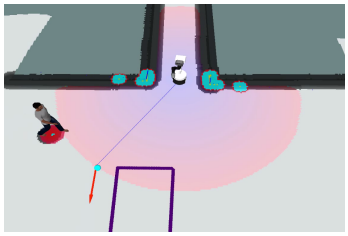


(a) Execution time. **(b)** Path length.

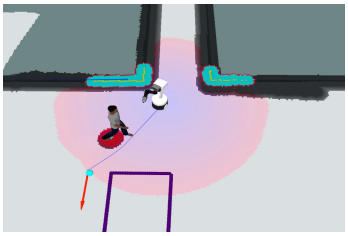
Figure 3-12: Hallway passing. Comparison of the execution time and path length for each navigation method.

Scenario 2: Intersection

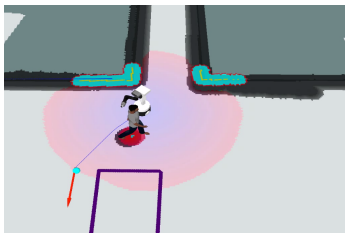
The second set of experiments tested two cases of an intersection scenario. The first case concerns a situation in which the human and the robot paths intersect under an angle. The experimental set consists of 30 simulation runs at different human velocities for each examined method. The human velocities were randomly generated by a simple Gaussian distribution with a mean $\mu = 0.4\ m/s$ and a standard deviation $\sigma = 0.1$.



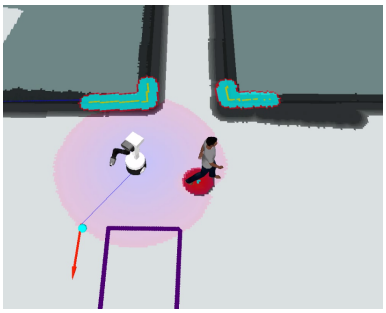
(a) The robot finds an initial path.



(b) The robot continues without showing intention to maneuver.



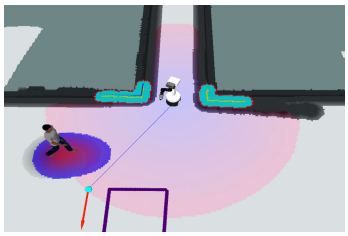
(c) The robot avoids the human because he managed to pass first.



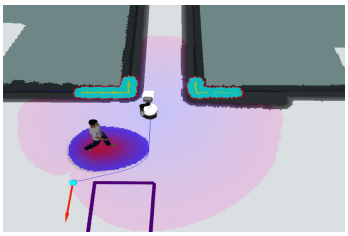
(d) The robot continues to the target.

Figure 3-13: Intersection - Standard ROS navigation. The robot did not collide with the human, since he managed to pass first.

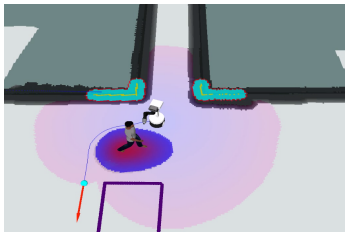
Figures 3-13-3-15 show the generated robot motion by each method at the mean human velocity. The standard ROS navigation method leads the robot in a straight path that intersects with the human path, as shown in Figure 3-13. Collision is avoided mainly due to the fact that the human forestalls the robot, but as will be shown later, this is not the case for all the human velocities. The social navigation (Figure 3-14) exhibits remarkable results, as the robot respects the proximity zone of the human and follows a path that communicates its intention to pass on the back. Similar behavior exhibits also the proposed framework (Figure 3-15), since the costmap decreases in size as the robot and the human approach each other, leading the intention costmap to have a similar shape to the proximity costmap.



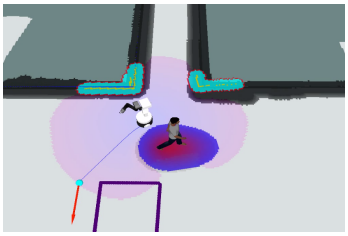
(a) The robot approaches the human.



(b) At first it tries to pass in front until it realizes the proximity constraint.

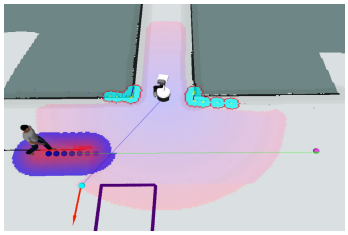


(c) The robot respects the proximity zone of the human and tries to pass in the back.

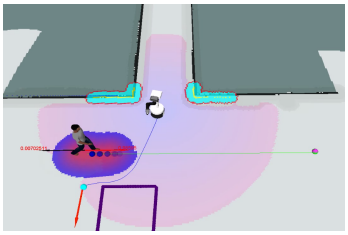


(d) The robot continues towards the target.

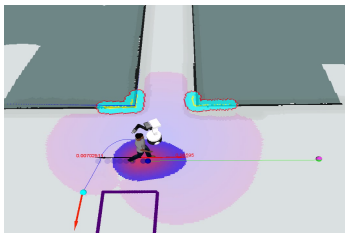
Figure 3-14: Intersection - Social navigation. The robot approaches the human while respecting the proximity zone. Then, it chooses to pass in the back of the human.



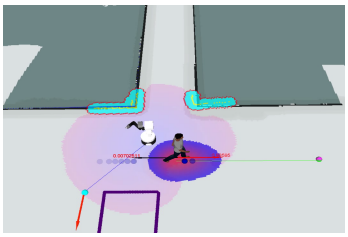
(a) The robot approaches the human.



(b) At first it tries to pass in front until it realizes the intention costmap.

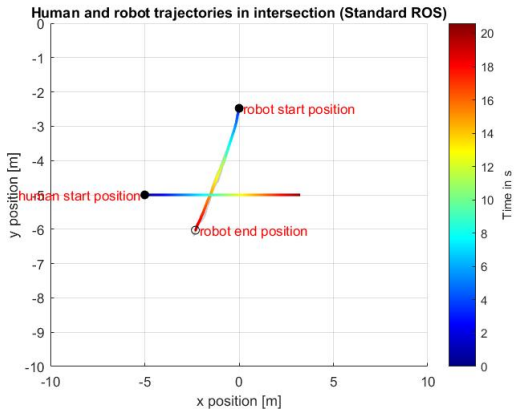


(c) The robot respects the comfort zone of the human and tries to pass in the back.

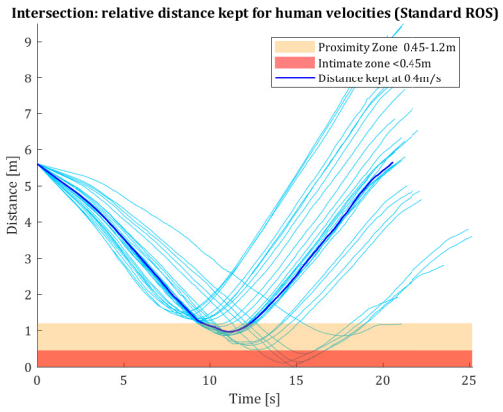


(d) The robot continues towards the target.

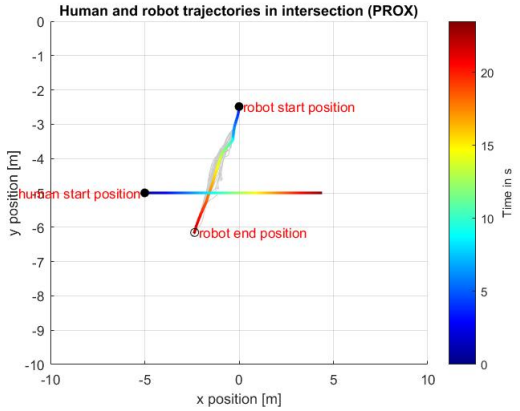
Figure 3-15: Intersection - Proposed framework. The robot approaches the human until it realizes his intention. Then, it chooses to pass in the back of the human. The method demonstrates similar behavior to the social navigation.



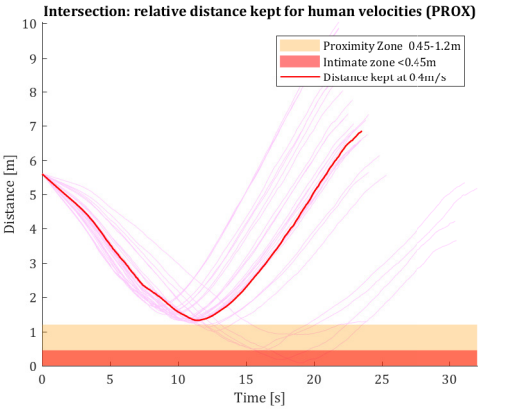
(a) Standard ROS navigation.



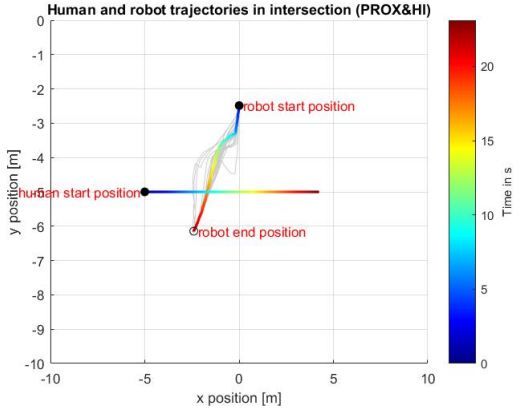
(b) Relative distance (Standard ROS).



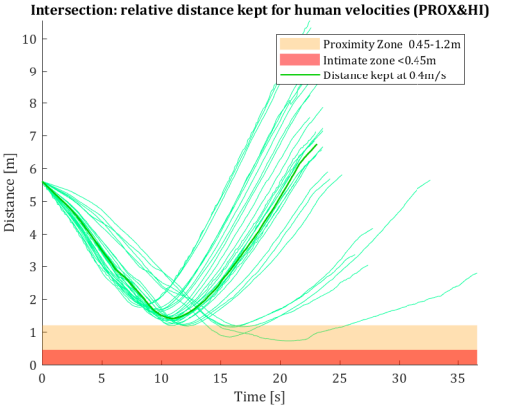
(c) Social navigation (PROX).



(d) Relative distance (PROX).



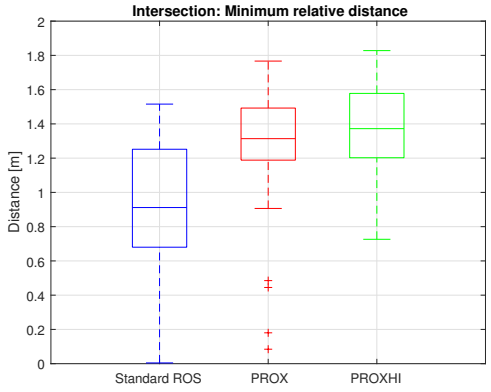
(e) Proposed framework (PROX&HI).



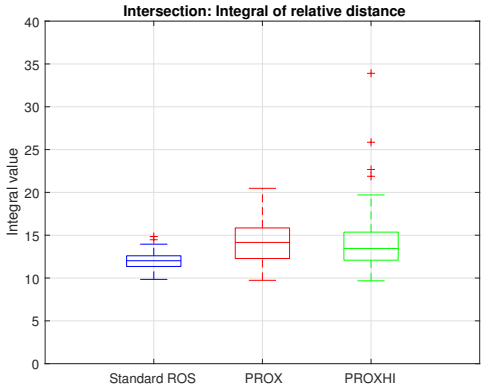
(f) Relative distance (PROX&HI).

Figure 3-16: Intersection (first case). Generated trajectories from each method and the relative distances kept for each simulation.

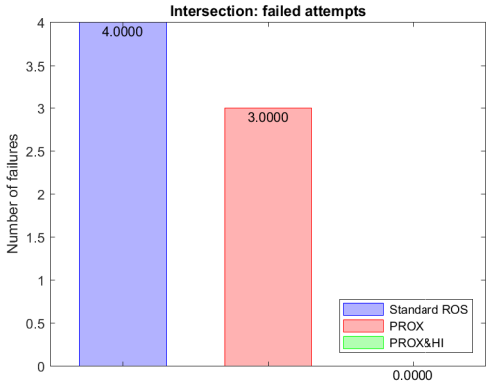
The overall results for all the simulations are shown in Figure 3-16. The generated trajectory for every human velocity is drawn in gray color, whereas the generated trajectory of the robot and the corresponding human trajectory at the mean person velocity is highlighted and colorized to visualize the time dimension. Time is captured by the colorbar on the right of each plot. In the right column of Figure 3-16 the relative distance that the robot maintained from the human during their interaction, is plotted over time for every simulation. The relative distance is projected against the intimate and the proximity zone of the human. The relative distance measured at the mean human velocity is shown by the intense curve.



(a) Minimum relative distance.



(b) Integral of relative distance.

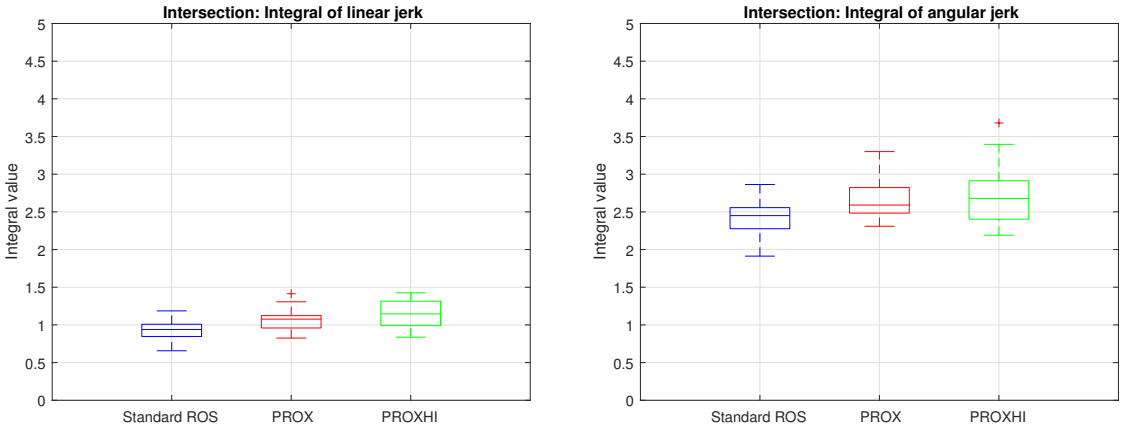


(c) Average failed attempts.

Figure 3-17: Intersection scenario (first case). Comparison of the metrics that evaluate human comfort. Figure 3-17a shows the minimum relative distance kept by each method. Figure 3-17b shows the integral of the relative distance. Figure 3-17c shows the average number of failed attempts.

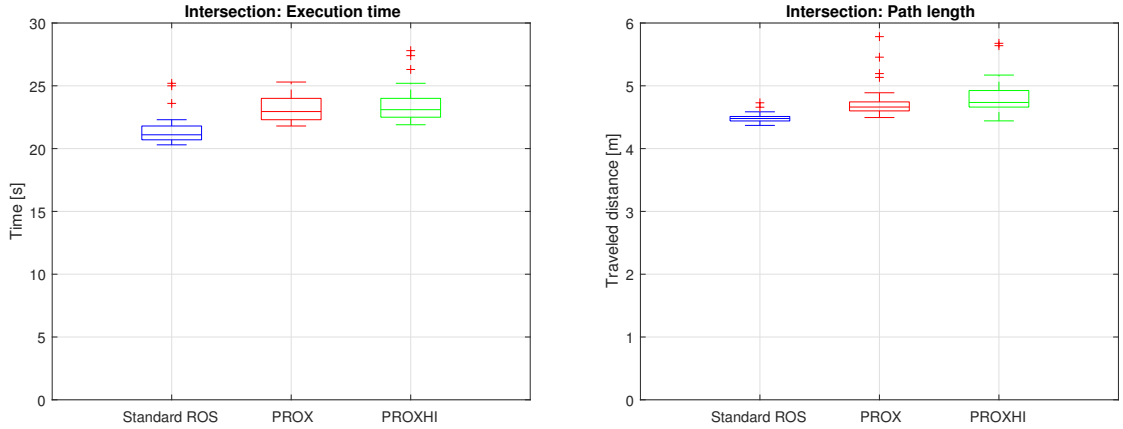
The figures reveal that the proposed framework managed to keep larger distance from the human, than the standard ROS navigation and similar distance to the social navigation. The slightly bigger variation in the data of the minimum distance in the case of standard ROS navigation, demonstrate the strong dependency of the behavior of the robot on the human velocity, since no costmap prevents the robot from approaching too much. It is worth to

mention that the proposed framework is the only method that managed to avoid dangerous situations, as it is denoted by the zero number of failed attempts. Regarding smoothness, the integral of the linear and angular jerk is illustrated in Figure 3-18. All methods produced comparable values, with the standard ROS navigation exhibiting slightly smaller values both for the linear and angular integral of jerk. Concerning the navigation performance, the standard ROS method produced slightly shorter execution times and shorter path lengths, as the robot followed in general a more direct path. The results between the social navigation and the proposed framework are similar.



(a) Integral of linear jerk. **(b)** Integral of angular jerk.

Figure 3-18: Intersection scenario (first case). Comparison of the integrals of the linear and angular jerk. Higher values of the integrals yield more erratic motion.

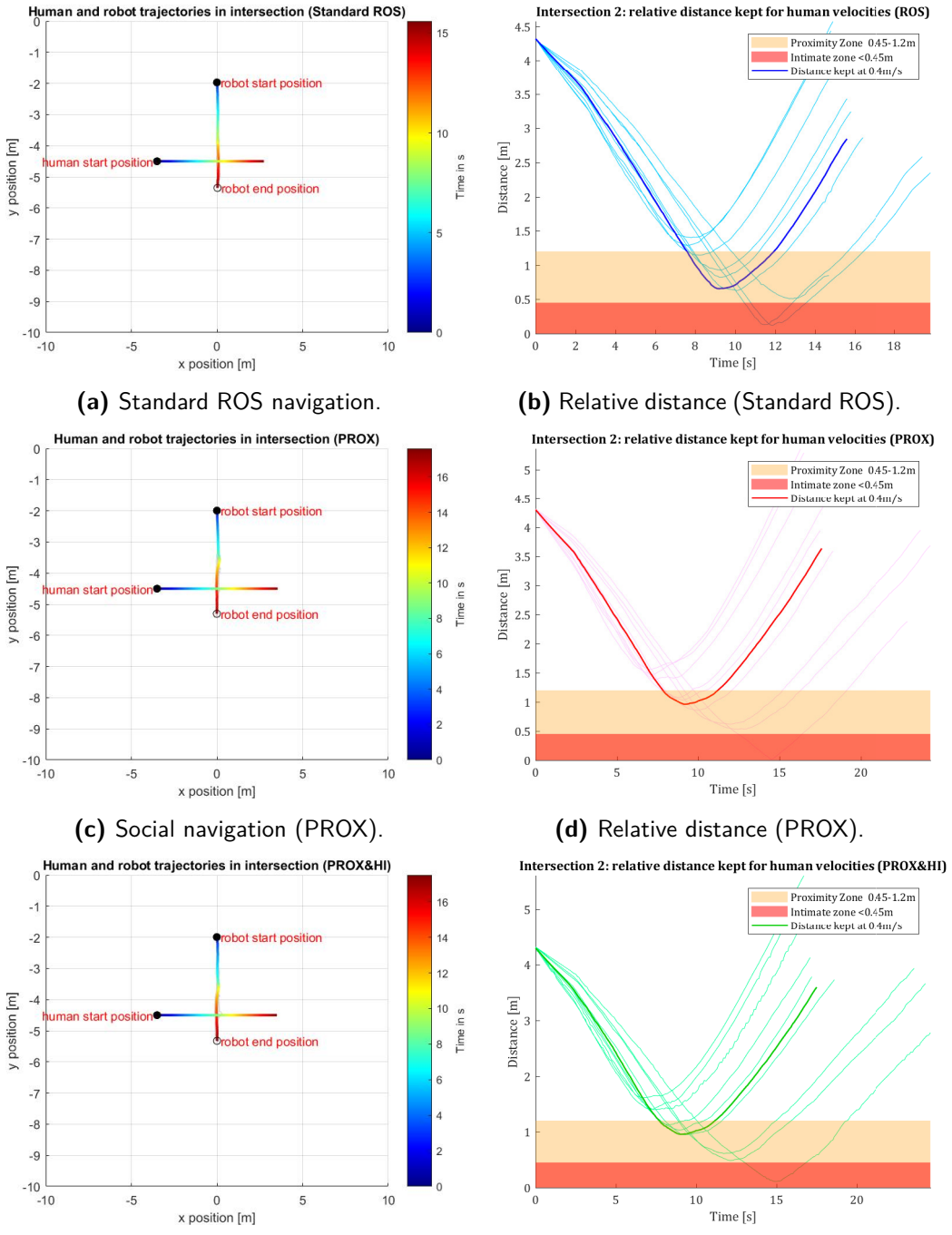


(a) Execution time. **(b)** Path length.

Figure 3-19: Intersection (first case). Comparison of the execution time and path length for each navigation method.

Nevertheless, there are cases where the proposed framework exhibits deteriorated performance mainly in terms of human comfort. To demonstrate this claim, a second intersection scenario is examined, in which the navigation methods are tested in a perpendicular crossing situation.

This set of experiments includes 10 simulation runs for each method and following the visualization approach of the previous cases, the generated trajectories and the relative distance, plotted over time, are shown in Figure 3-20.



(a) Standard ROS navigation.

(b) Relative distance (Standard ROS).

(c) Social navigation (PROX).

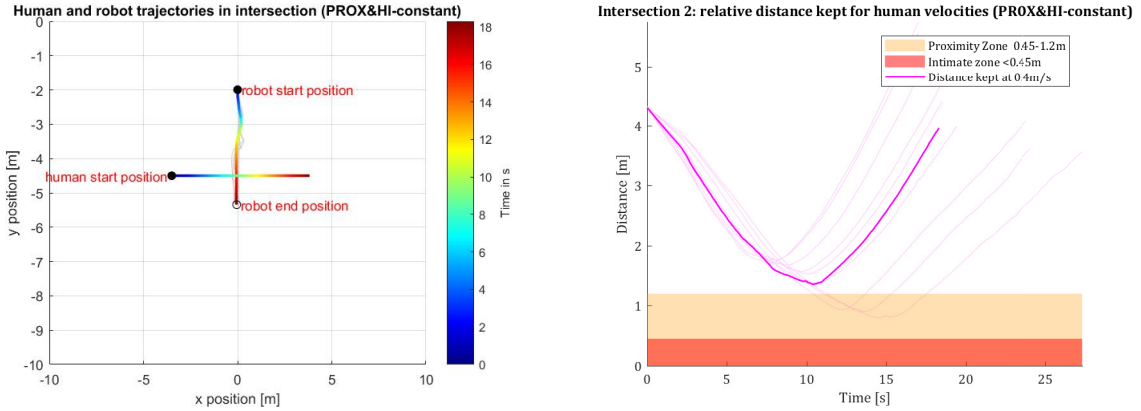
(d) Relative distance (PROX).

(e) Proposed framework (PROX&HI).

(f) Relative distance (PROX&HI).

Figure 3-20: Intersection (second case). Generated trajectories from each method and the relative distances kept for each simulation.

From the relative distance plots, it can be seen that in the crossing situation, the social navigation and the proposed framework have similar results and in fact, the proposed framework produced one failed run. It can be said that in the crossing scenario, the proposed framework approximates the social navigation. This approximation is strictly related to the adjustment of the intention costmap according to the relative distance. Let us keep the intention costmap constant and extended by a fixed number of prediction steps (10 steps), as described in Section 2-2-8. This means that the costmap remains unchangeable while the robot approaches the human. The generated robot trajectories and the relative distance in this case are illustrated in Figure 3-21



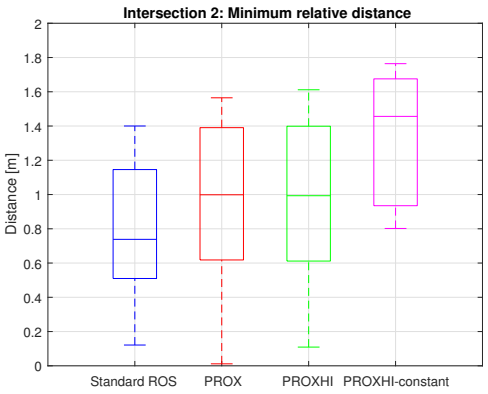
(a) Proposed framework with constant costmap (PROX&HI-constant).

(b) Relative distance.

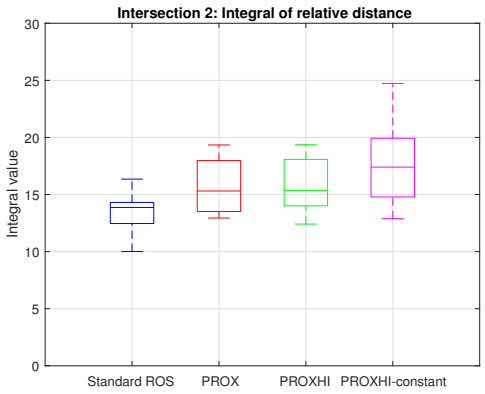
Figure 3-21: Intersection (second case). Generated trajectories and relative distance of the proposed framework with constant intention costmap.

Regarding comfort, Figure 3-22, shows that once more the standard ROS navigation method fails to keep acceptable interaction distances and created dangerous situations in 2 out of 10 simulations. The social navigation and the proposed framework with adjustable costmap have similar median minimum relative distance and one failure each. The constant intention costmap succeeds in maintaining higher distances and a zero number of failed attempts.

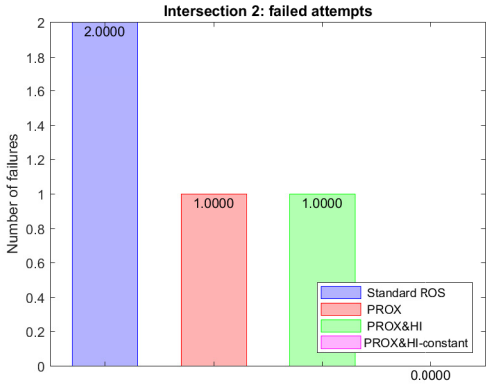
In terms of smoothness, in Figure 3-23 it is shown that the standard ROS navigation exhibits the lowest jerk. The social navigation and the proposed framework with adjustable costmap achieve similar results, whereas the proposed framework with constant costmap presents slightly higher values.



(a) Minimum relative distance.



(b) Integral of relative distance.

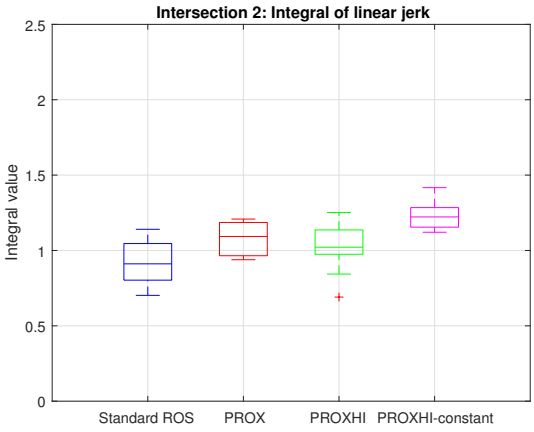


(c) Average failed attempts.

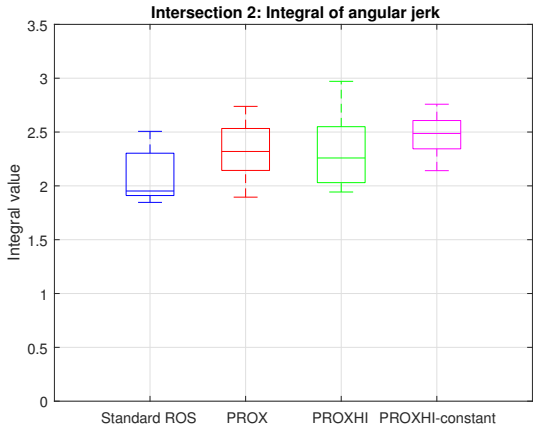
Figure 3-22: Intersection scenario (second case). Comparison of the metrics that evaluate human comfort. Figure 3-22a shows the minimum relative distance kept by each method. Figure 3-22b shows the integral of the relative distance. Figure 3-22c shows the number of failed attempts.

From Figures 3-22 and 3-23, becomes clear that keeping the intention costmap constant, prohibits the robot from approaching the human a lot, as captured by the higher values for the minimum relative distance and the integral of the relative distance. This increase in comfort comes with a trade-off in smoothness, since the constant intention costmap exhibits higher values for the jerk. From these results it can be concluded that, there are cases in which the adjustment of the intention costmap, based on the relative distance and the human velocity, is not a sufficient method to reduce erroneousness in motion. Others, more advanced methods, may be more adequate as will be discussed in Section 4-1.

Finally, Figure 3-24, demonstrates the results regarding the navigation performance of each method (including the constant intention costmap). Again, standard ROS navigation reveals the lowest execution time and path length. The social navigation and the proposed framework with adjustable costmap, exhibit equivalent performance, whereas the proposed framework with constant intention costmap, despite the small erroneousness, exhibits negligible differences both in terms of execution time and path length.



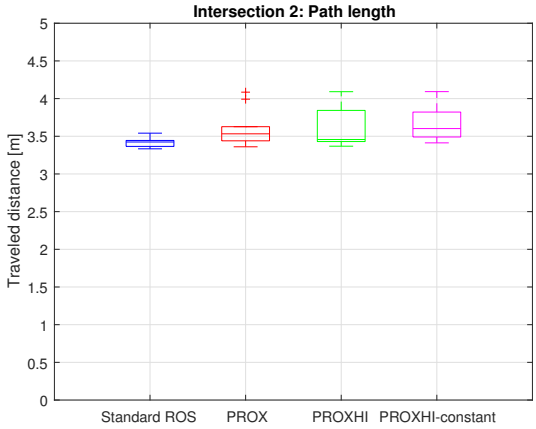
(a) Integral of linear jerk.



(b) Integral of angular jerk.

Figure 3-23: Intersection scenario (second case). Comparison of the integrals of the linear and angular jerk. Higher values of the integrals yield more erratic motion.

(a) Execution time.



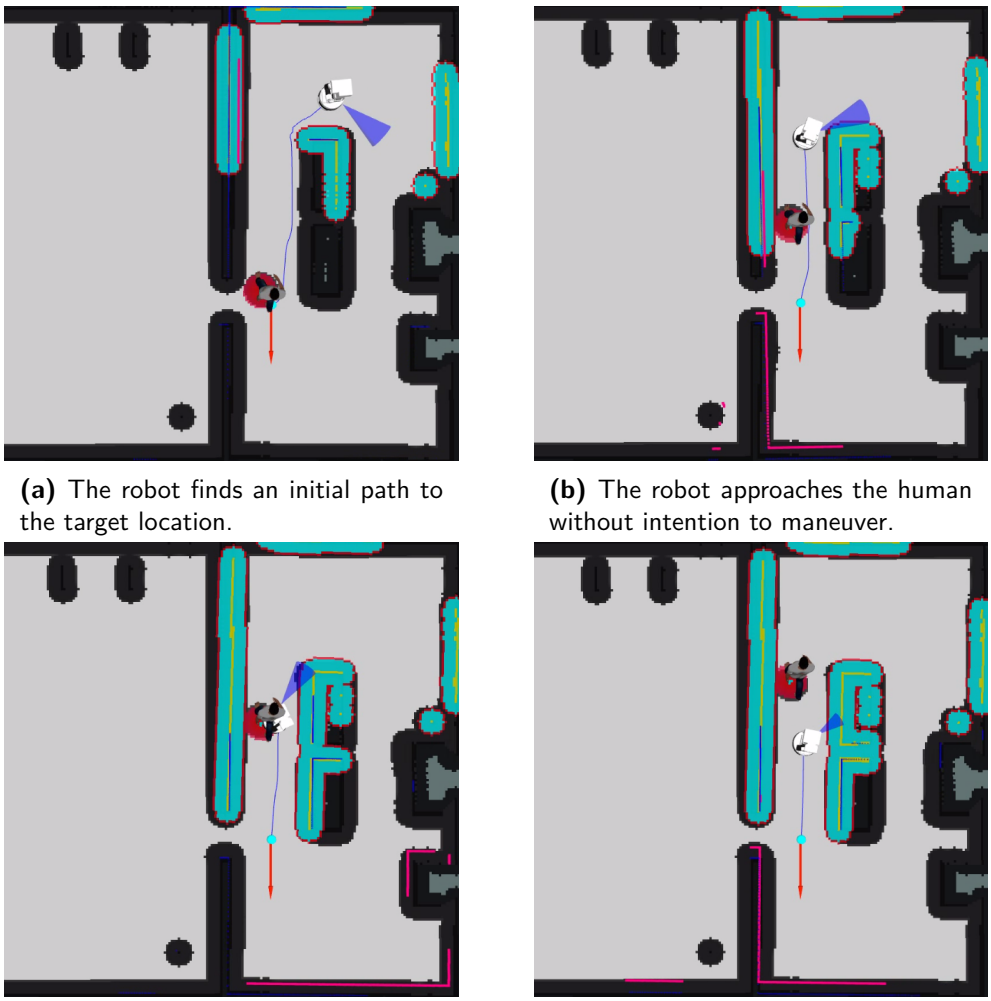
(b) Path length.

Figure 3-24: Intersection (second case). Comparison of the execution time and path length for each navigation method.

A last comment concerns the big variation observed in the datasets of Figures 3-22a and 3-24a. This variation is indicative of the effect that the human velocity has on the way that the robot and the human meet, in the case of the perpendicular crossing scenario. Since their directions are perpendicular, in the case of standard ROS navigation, the robot reacts just before it reaches the inflation zone of the human. The meeting point with the inflation zone differs according to the human velocity (the robot meets the zone in an earlier or latter stage of the motion). The same stands also for the other navigation methods but this time the robot meets the proximity or the intention costmap respectively. Based on the human velocity, the robot meets the costmap at its beginning, middle or end. Therefore a similar variation is observed for the social navigation and the two versions of the proposed framework.

Scenario 3: Domestic scene

The last set of experiments was tested in a domestic scenario, as shown in Figure 3-4. The domestic scene is a particular and peculiar case of interaction environment due to the presence of obstacles and the confined space. Hence, the robot even if it is willing to, it may be unable to keep proper distance from the human, simply due to lack of space. The experimental set includes 10 simulation runs at different human velocities, which were generated by a Gaussian distribution with mean $\mu = 0.4\ m/s$ and standard deviation $\sigma = 0.1$. Figure 3-25 illustrates the generated robot motion by the standard ROS navigation method, at the mean human velocity. The robot followed the shortest path to the target position, which intersects with the human one. As a result, it virtually collided with the human.



(a) The robot finds an initial path to the target location.

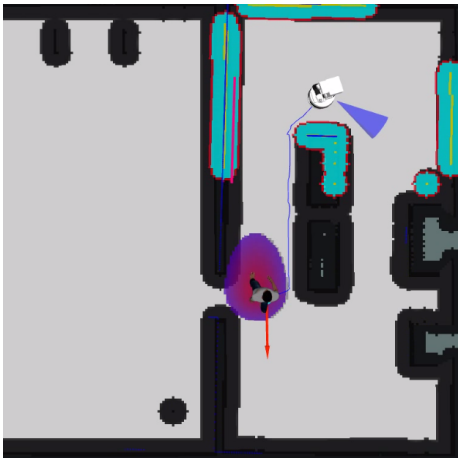
(b) The robot approaches the human without intention to maneuver.

(c) The robot (virtually) collides with the human, since it enters the intimate zone.

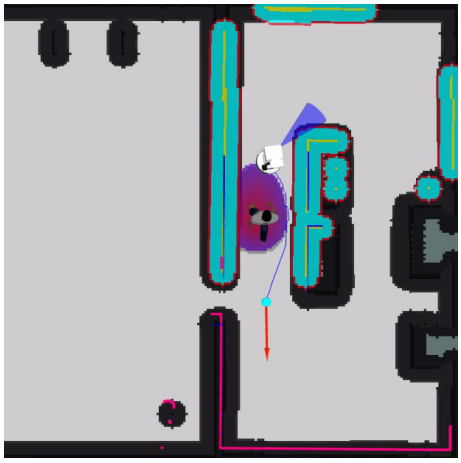
(d) The robot reaches the target.

Figure 3-25: Domestic scene. Generated motion by the standard ROS navigation method. As it can be seen the robot collided with the human.

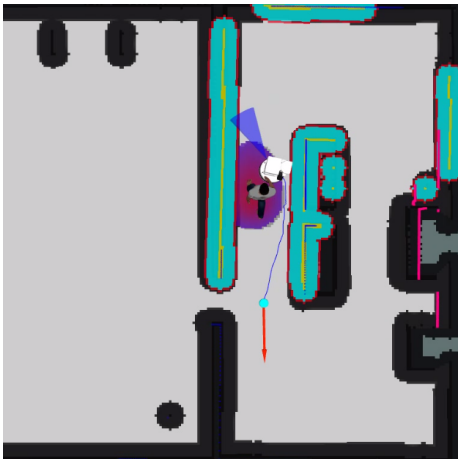
Figure 3-26 illustrates the generated robot motion by the social navigation method at the mean human velocity. It is obvious that the robot tried to maneuver in order to respect the proximity zone of the human, but due to the narrow and confined environment, it eventually got trapped inside the proximity costmap and inevitably, entered the intimate zone of the human.



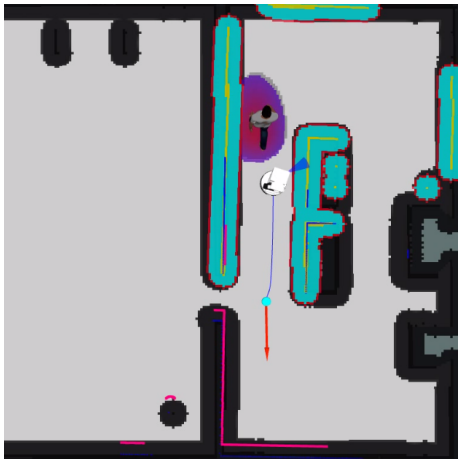
(a) The robot finds an initial path to the target location.



(b) The robot approaches the human showing its intention to maneuver.



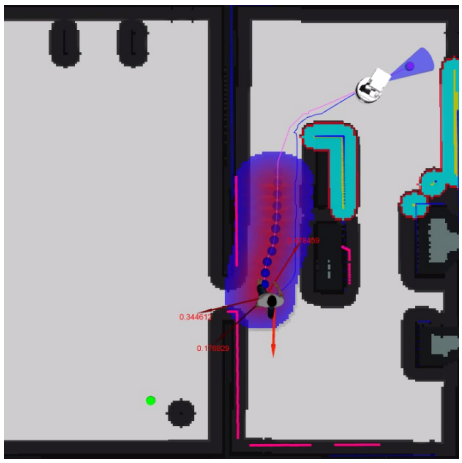
(c) The robot gets trapped in the costmap due to the confined space and enters the intimate zone of the human.



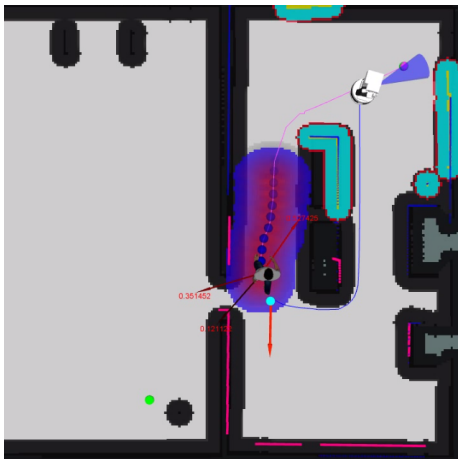
(d) The robot continues towards the target.

Figure 3-26: Domestic scene. Generated motion by the social navigation method. The robot tried to respect the proximity zone of the human, but due to the confined space, it got trapped in the proximity costmap and eventually entered the intimate zone, as well.

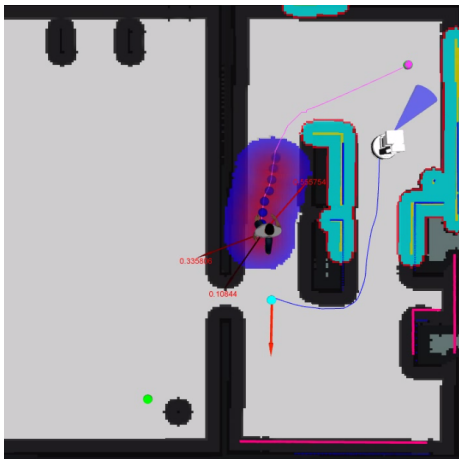
Figure 3-27 shows the generated motion produced by the proposed framework. The robot is able to infer the most destination of the human and by means of the human path planner, the anticipated path between the human and the most probable destination is designed. The intention costmap is assigned along this path and the robot takes into account the human intention by choosing a path on the other side of the obstacle.



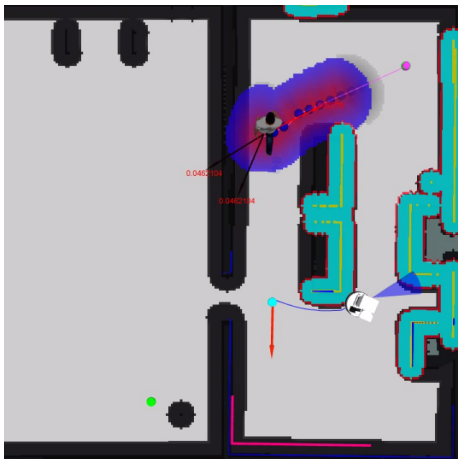
(a) The robot finds an initial path to the target location.



(b) The robot accounts for the intention costmap and chooses the longer path at the other side of the obstacle.



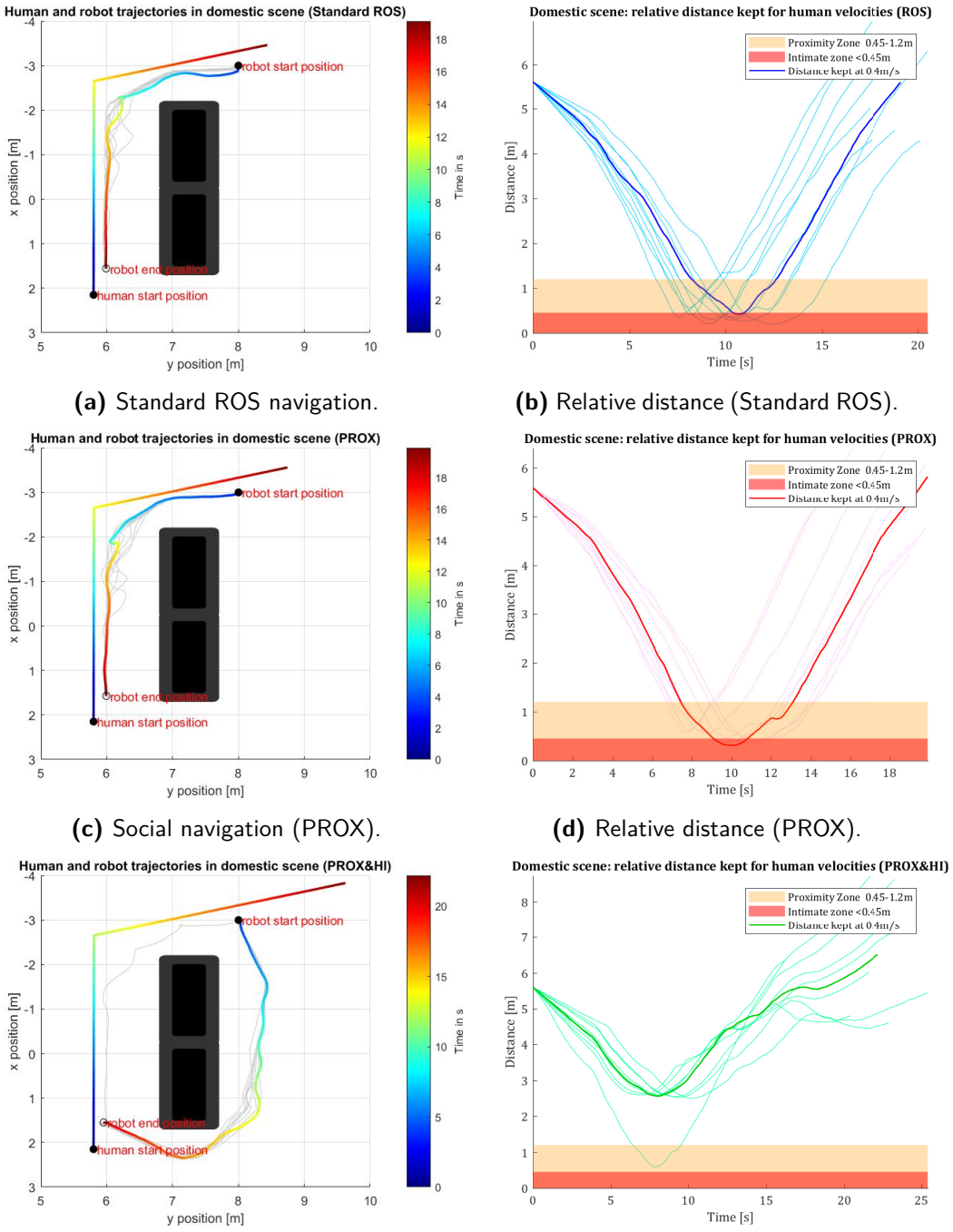
(c) The robot does not even meet the human on its way to the target.



(d) The robot continues towards the target.

Figure 3-27: Domestic scene. Generated motion by the proposed framework. The robot considers the intention of the human and prefers to follow a path that does not intersect with the human one. The most probable destination is denoted by the purple sphere and another destination in the environment is denoted by the green sphere.

As in the previous Sections, the overall generated trajectories by each method, are plotted for all simulations in grayscale and the run at the mean human velocity is colorized to capture the time dimension. The resulting trajectories and the relative distance kept for each simulation are shown in Figure 3-28.



(a) Standard ROS navigation.

(b) Relative distance (Standard ROS).

(c) Social navigation (PROX).

(d) Relative distance (PROX).

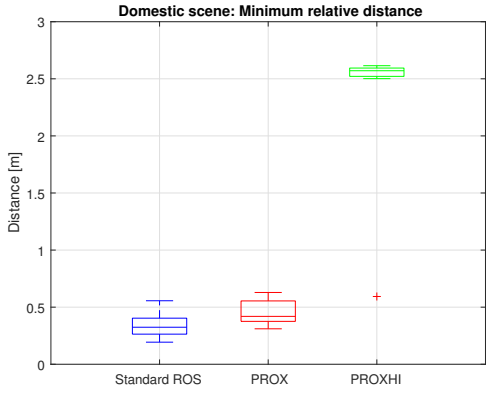
(e) Proposed framework (PROX&HI).

(f) Relative distance (PROX&HI).

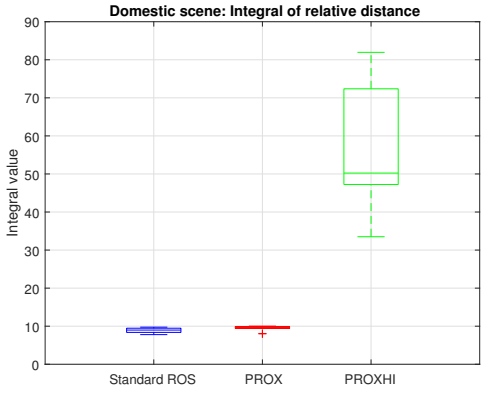
Figure 3-28: Domestic scene. Generated trajectories from each method and the relative distances kept for each simulation.

The standard ROS and the social navigation method, drive the robot along the shortest path to the target location and inevitably, the robot closely interacts with the human since there is not enough space. In the case of standard ROS navigation the robot enters the intimate

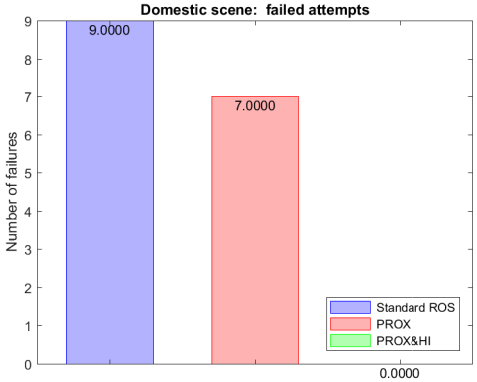
zone in most of the cases. In the case of the social navigation, the robot tries to fit and pass between the proximity costmap and the inflation limits of the obstacle. In some cases, depending on the human velocity, it penetrates the intimate zone, but in fewer cases than in the standard ROS navigation. On the contrary, the proposed framework made the robot to respect and not enter the proximity zone in almost all of the trials. As it can be seen in Figure 3-28f, there is only one trial in which the robot entered the proximity zone, but not the intimate, thus avoiding collision. This trial is associated to the outlier path of Figure 3-28e, which passes at the same side of the obstacle with the human.



(a) Minimum relative distance.



(b) Integral of relative distance.



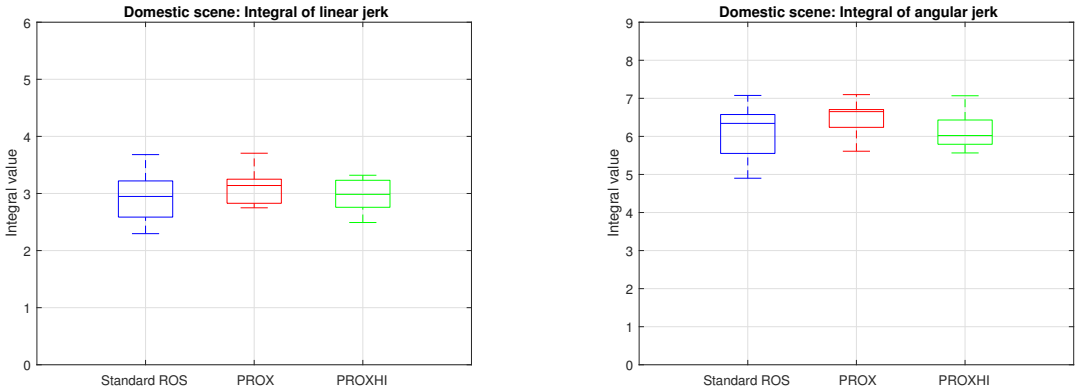
(c) Average failed attempts.

Figure 3-29: Domestic scene. Comparison of the metrics that evaluate human comfort. Figure 3-29a shows the minimum relative distance kept by each method. Figure 3-29b shows the integral of the relative distance. Figure 3-29c shows the average number of failed attempts.

Figure 3-29 reveals that the proposed framework maintains more than five times larger relative distance and five times larger integral of relative distance comparing to the other two methods. This high value is also justified by Figure 3-28f, where the relative distance curves stay, in general, way above the proximity zone. Apart from the distance-related metrics, it is remarkable that the proposed framework did not create any dangerous situations, with the number of failed runs to be zero. Both the standard ROS and the social navigation demon-

strated numerous failures, since they preferred to squeeze the robot between the human and the obstacle, rather than selecting a different path.

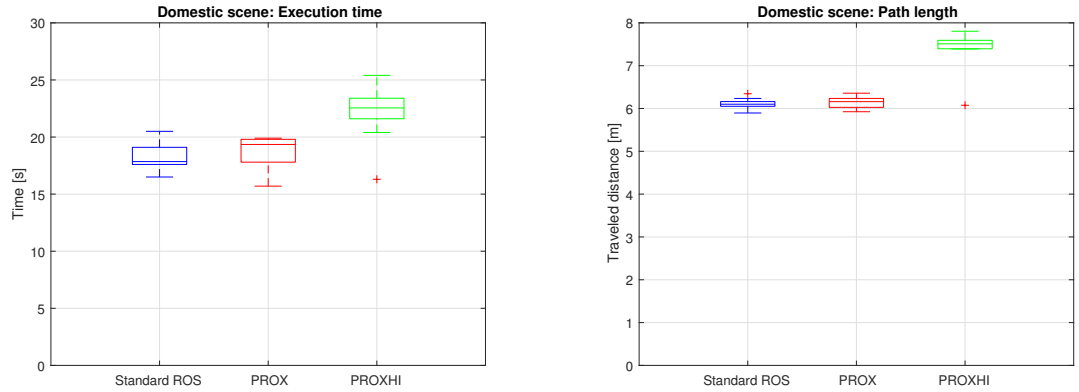
Figure 3-30 shows the results for the integral of the linear and angular jerk. All three methods exhibit comparable values, yielding similar smoothness in motion.



(a) Integral of linear jerk. **(b)** Integral of angular jerk.

Figure 3-30: Domestic scene. Comparison of the integrals of the linear and angular jerk. Higher values of the integrals yield more erratic motion.

The last figures concern the comparison in navigation performance for the domestic scene scenario. Figure 3-31 illustrates the execution time and the path length for all three methods. As expected, the standard ROS and the social navigation demonstrated very similar median time and almost equal median path length, since in these cases the robot followed similar motion behavior. On the contrary, the proposed framework made the robot to perform a larger circumvention and this is depicted in a higher median time ($\approx 4s$) to reach the target location, and almost 1.5 m more traveled distance.



(a) Execution time. **(b)** Path length.

Figure 3-31: Domestic scene. Comparison of the execution time and path length for each navigation method.

3-3 Discussion

Hallway passing

In the hallway passing scenario, the social navigation and the proposed framework outperformed the standard ROS navigation in comfort, since they exhibited higher values of relative distance and zero failed attempts. This claim is evident in Figure 3-10. An interesting remark is that the integral of the relative distance is comparable for all the navigation methods. This result yields that although the social navigation and the proposed framework maintained a larger minimum distance, yet the robot remained within the critical area (approximately a radius of 2.5 m around the human) for some time. Therefore, as also explained by the example of Figure 3-6, the total integral has similar values. The relatively long stay of the robot in the critical zone is due to the confined hallway. If the hallway was wider, the robot should avoid even entering the proximity zone of the human. That would yield higher value of the integral of the relative distance. Despite the similar values for the integral of distance, the number of failed attempts in the case of standard ROS navigation is huge in comparison with the other two methods. In particular, in all the trials the robot entered the intimate zone, whereas in the other two cases all the runs were successful. Hence, it can be concluded that the social navigation and the proposed framework are comparable in terms of human comfort and both outperformed the standard ROS navigation.

Regarding smoothness, plots in Figure 3-11 reveal that the standard ROS navigation produced the smoothest robot motion. However, this result is fictitious. Since there are no costs around the human, but only the ones coming from the inflation zone around him, the robot moves straight and only reacts when the human is really close. Then it makes a slight maneuver to avoid collision, which under realistic circumstances would be insufficient and would certainly lead to collision. In this case, a real person would stop the moment it realized that the robot will bump into it. This action would give time to the robot to stop and turn on spot in order to bypass the human. Still, a motion like that would certainly result in high values for the jerk, comparable or even higher than for the other two methods. Consequently, the low value of the jerk is disregarded as it does not reflect realistic results. On the contrary, the social navigation and the proposed framework exhibit comparable jerk both on the translational and the rotational motion. The slightly lower value produced by the proposed framework is also pinpointed in Figure 3-9e, where the colored robot trajectory is marginally smoother than the corresponding one in Figure 3-9c. This reflects the ability of the proposed method to infer the human destination and assign the intention costmap to this direction, making the robot to perform an earlier avoidance maneuver and move to the right side of the corridor more smoothly. Therefore it is argued that the methods performed similarly in terms of smoothness. The increased smoothness demonstrated by the standard ROS navigation is disregarded due to the unrealistic assumption of a 'dummy' virtual human in our simulated experiments, which does not cause physical collisions with the robot.

As far as navigation performance is concerned, the similar results on execution times and path lengths for all the methods, yield that the incorporation of the proximity and the human intention layer did not affect the efficiency of the planner. It can be argued that a relatively straightforward motion followed by an abrupt maneuver, as in the case of the standard ROS navigation, can be alleviated by a slightly diverging but smooth motion as in the case of the other two methods.

Intersection

In contrast to the hallway passing, an intersection scenario does not necessarily presage collision if both the robot and the human do not deviate from their paths. Thus, as Figure 3-16b illustrates, the number of trials in which the robot penetrates the intimate zone is significantly lower than in the hallway scenario. However, there are still enough dangerous situations. In general, all methods (on average) managed to maintain larger distances from the human. The minimum distance is shown in Figure 3-17a and the integral of the distance in Figure 3-17b. The proposed framework performs similarly to the social navigation and this result is associated to the adjustment of the human intention costmap with respect to the relative distance and the velocity of the human, as explained in Appendix B-4. By modifying the size of the costmap, as the robot and the human come closer, a more polite approach is facilitated since the relative distance is decreased with a lower rate over time. Besides, the superiority of the proposed framework is evident in Figure 3-17c where the number of failed attempts is zero, whereas the standard ROS and the social navigation failed to produce successful motion at some human velocities. It is highlighted that, the standard ROS navigation produced the highest number of failed attempts.

With reference to smoothness in motion, the differences in the jerk values of Figure 3-18 between the social navigation and the proposed framework are negligible. In addition, since the proposed framework did not create any dangerous situations, it is argued that it produced the friendliest motion in this particular scenario. Moreover, the friendliness of the proposed framework is apparent in Figure 3-16e. In most of the runs (there is one outlier), the robot makes a relatively smooth maneuver to the left communicating its intention to move to the back of the human in order to reach the target location. The respective maneuver in Figure 3-16c, is a bit sharper, making the robot motion less effusive. The directness of the paths generated by the standard ROS navigation does not clarify the intention of the robot and could create worries to the human.

However, there can be interaction scenarios where the proposed framework does not manage to completely eliminate dangerous situations. As an example, a perpendicular crossing scenario is investigated. As illustrated in Figure 3-22c, in that case the proposed framework produced one failed attempt. In an attempt to keep the robot even further away from the human, the intention costmap was kept constant and extended by a fixed number of prediction steps. This modification led to higher relative distance (as shown in Figures 3-22a and 3-22b) and to zero failed attempts (Figure 3-22c). Though, this improvement came at the expense of a more erratic motion. In Figure 3-21a the robot, for most of the simulations, first attempts to move to the right (probably aiming to pass in front of the human) and then, as the path deviates a lot (and the distance cost increases), the robot changes its mind and moves back to the left, in order to pass the human from behind. This hesitation in motion is pinpointed by the increased jerk values for the constant intention costmap, as can be seen in Figure 3-23. The social navigation and the proposed framework with adjustable costmap instead, have lower and comparable jerk values.

Regarding navigation performance, the standard ROS navigation produced more direct paths and hence it exhibits slightly smaller execution time and path length, than the other two contestants and the modified version with the constant intention costmap. Yet, these differences are not significant and as a result, also in this scenario it is argued that the navigation performance is not downgraded by the incorporation of the human intention layer.

Domestic scene

This scenario demonstrates the inefficiency of the social navigation method in confined environments. The inability of the social navigation to deal with confined and cluttered environments is overcome by the proposed navigation framework as shown in Figure 3-27. The framework infers the most probable destination (purple sphere) and the human path planning module finds the anticipated path towards this destination. The costmap is assigned along this path. The generated motion, at the mean human velocity, evidently increases human comfort, since the robot prefers the longer, yet more human-friendly path, at the other side of the obstacle and does not even interact with the human.

The key weakness of both standard ROS and social navigation methods is that, even if there is some space available to find a collision-free path, they will dictate the robot to follow it, since they do not account for the future positions of the human. However, the ability of the proposed framework to anticipate the future human positions and capture them within the intention costmap, led the robot to find an alternative path around the obstacle, as illustrated in Figure 3-28e. In almost all the trials (there is one outlier), the robot preferred a slightly larger path, at the expense of increasing human comfort. The overwhelming prevalence of the proposed framework in terms of human comfort, is depicted in Figure 3-29, since the proposed method did not create any dangerous situations, while the other two methods failed for the majority of trials.

In terms of smoothness, the standard ROS and the social navigation exhibit some erroneousness in motion, when the robot confronts the human and performs abrupt maneuvers, as shown in Figures 3-28a-3-28c. On the other hand, the proposed framework produces some erratic motion, when the robot tries to maneuver among the central and other obstacles in the down right part of Figure 3-28e. This erroneousness may arise due to the effort of the planner to find the least cost path, since at that point of motion, the length weight and the distance from obstacles weight compete each other intensively. Eventually, the total values for the integral of the linear and angular jerk are comparable for all the methods.

Up to this point, it can be argued that the proposed framework, obviously outperformed the other two methods in terms of comfort and demonstrated comparable performance in terms of smoothness. In terms of navigation performance, the proposed framework exhibits increased execution time and path length comparing to the other two methods. This is a straightforward consequence, since the robot traveled along a larger path, taking more time to reach the target location. Nevertheless, it is argued that this slightly poorer navigation performance is more than acceptable, since the robot managed to dramatically increase human comfort.

Chapter 4

Conclusions

In this thesis it was investigated how robot motion can become more friendly in domestic environments, by taking explicitly the presence of humans into account, in order for service robots to be accepted as daily interaction partners.

A human-aware navigation framework was proposed, that builds on top of the layered costmap architecture used in robot navigation, by adding a layer with the context of human intention. The new layer models the intention of the human to visit a known destination in the environment and commands the robot to avoid future locations which the human is anticipated to occupy, on his way towards his destination. Furthermore, a simplistic path planner is implemented in order to infer the most probable unobstructed path that a human would follow, and hence the future positions of the human, when moving to reach his target position. The intention costmap is assigned along this path.

The layered costmap provides a generic, modular and efficient way to incorporate new contexts into the navigation process. Therefore, it was selected as the architecture for the implementation of our framework, answering to the first research question of the thesis. Hence, the first conclusion of this thesis is:

Conclusion 1: The layered costmap architecture is an efficient way to incorporate social constraints and human intention in the navigation process.

In order to evaluate the proposed framework, three interaction scenarios that are common in domestic environments were defined. The considered scenarios include a hallway passing, two cases of intersection and a domestic scene with static obstacles. The proposed framework was tested and compared, in these scenarios, against two state-of-the-art navigation methods; the standard navigation used in the ubiquitous open-source Robot Operating System (ROS) and the social navigation proposed by Lu et al. [21].

The simulated experiments showed that incorporating the human intention layer, led to a more human-aware and friendlier robot motion. Knowing the intention of the human makes the robot respect more the human path and tries not to intersect with it. This finding is in accordance with the second research question of this thesis.

In most of the simulated experiments the proposed method, denoted as PROX&HI (Proximity & Human Intention), led the robot to keep larger distances from the human, during their interaction, and to perform an avoiding maneuver sooner. The distance kept from the human is considered as an important metric for quantifying human comfort, in the context of this thesis. Therefore it can be concluded that:

Conclusion 2: The proposed framework performed similarly or better than the other two methods, in terms of human comfort.

Apart from human comfort, also smoothness was evaluated. Smoothness is predominant in order for a motion to be socially accepted, since an erratic motion can be proved to be confusing for a human. In our design, the method followed to improve the exhibited smoothness of the proposed framework, was to adjust the intention costmap based on the human velocity and the relative distance. This method was able to improve smoothness in most of the trials, but there where cases where it led the robot to enter the intimate zone of the human and create dangerous situations. Overall, regarding smoothness it can be concluded that:

Conclusion 3: The proposed framework did not exhibit significant improvement in smoothness.

PROX&HI, is evidently superior in cases where obstacles are present in the environment and people maneuver amidst these obstacles to reach their destination, as in the domestic scenario. In such cases the assignment of the intention costmap along the circumvented human path, can prevent interaction with the robot at all. To this end, the implementation and inclusion to the planning process of the human planner, as a means of human motion prediction method, proved to be essential. Hence it can be concluded that:

Conclusion 4: The incorporation of the human planner into the proposed framework, improved friendliness in situations where obstacles are present and people do not walk straight towards their destination.

Finally, in terms of navigation performance, PROX&HI exhibited similar performance with both the standard ROS navigation and the social navigation method, with negligibly larger execution times and path lengths. It is argued that such slight differences are welcomed in trade-off for increased friendliness. The last conclusion of this thesis is:

Conclusion 5: Navigation performance was not deteriorated by the incorporation of the human intention layer.

To summarize, the proposed method performed better than the other two navigation methods in terms of human comfort, by maintaining a larger distance from the human and being slightly smoother in many cases. Smoothness can be improved by controlling the velocity of the robot on top of the intention costmap. In terms of navigation performance, all three methods exhibited similar results, an outcome showing that the incorporation of the intention layer did not downgrade the efficiency of the navigation planners.

4-1 Future work

Promising results have been found towards increasing friendliness in motion, by incorporating human intention in the path planning process. However, this does not exhaust the topic. Recommendations in order to further validate the proposed framework, as well as guidelines for implementing it on a real robot, are given in this Section.

Testing the framework with other planners. Although the proposed framework demonstrated similar or higher smoothness than its competitors in most of the simulated experiments, in some cases it led the robot to intrude the intimate zone of the human. As already explained, the intention costmap was adjusted based on the relative distance between the robot and the human and scaled by the magnitude of human velocity. This method essentially, shrinks linearly the size of the costmap around the human, enabling the robot to approach gradually and without high erroneousness in its motion. Nevertheless, a more sophisticated approach would be to dictate a velocity to the robot, which renders the robot to exhibit a more polite behavior, namely slowing down in order not to enter the intention costmap and wait till the human has passed. This approach could be more efficient and generic in intersection scenarios, ensuring collision avoidance in more crossing directions. For this purpose, the navigation planner should be modified and incorporate a search of the admissible velocities that will not result in collision. To this end, velocity obstacle methods (e.g. [54], [55]) could be utilized and tested with the proposed framework.

One of the most significant features of the proposed framework is its modularity. This means that the framework is independent of the planner used. Consequently, it is encouraged to test it with other planners as well. This recommendation is in accordance with the previous one. It is expected that the incorporation of human intention would be an asset for any costmap-based path planning approach and a tool to explore the efficiency of the planner itself, in different contexts of navigation.

Testing in more complex scenarios. Up to this point of implementation, almost 80% of the framework has been designed for multiple humans in the environment. Work is still need to be done in order to finalize and polish the code and for the method to be functional when multiple humans are present.

Another powerful characteristic of the proposed framework is the ability of the human path planner to find the shortest, collision-free path in real time. This means that, it can find the human path also in the presence of moving obstacles which can cause the person to maneuver in order to avoid them. However, in the simulated environment we were not able to produce moving obstacles (apart from the robot itself) and that was a limitation in conducting more experiments. In addition, there is the realistic difficulty in capturing the full dimensionality of a moving object by the robot sensors. Let us assume that a mobile vacuum cleaner is moving inside a room. The laser sensors capture only the side where the IR beams reflect, but it cannot provide any more information about the actual shape and size of the cleaner. Only when the vacuum cleaner approaches the robot enough, it can be fully captured by the RGB-D sensor and provide this information to the human path planner. Otherwise, a non-feasible path may be found which passes through the (unidentified) area of the moving

vacuum cleaner. Furthermore this problem of partial captivity of objects can also occur for static objects.

Improvements for real-life application. After designing, implementing and testing the proposed framework in a simulated environment, an attempt to implement it on the real Marco robot was made. Albeit the framework is functional enough, there are significant problems regarding human detection in the environment. These problems arise from two sources. The first one is the noisy sensory data. Noisy measurements, mainly as far as the computation of the human velocity is concerned, lead to memory crashes since the costmap is scaled by this velocity. Therefore if there are huge differences between consecutive readings, the abrupt changes in the size of the costmap lead the navigation system to crash. Hence, for real life application of the framework, some filtering in the measurements is required.

The second source of problems is the big number of false positives of the human detection module. Often enough, pairs of chair and table legs are falsely considered as human ones. Hence, detection in domestic environments may require a sophisticated training method for the human detection module, similar to the one described in Section 2-2-4.

Last but not least, as described in Section 2-2-9, the human planner makes use of the local costmap of the robot, in order to find the optimal path for the human. Hence, if it is desired to infer the human path in a larger area of the environment, the local costmap needs to be extended. However, this comes at a cost of computational complexity. In a simulated environment, this complexity is manageable, however in real-life it increases dramatically and may impose limitations in the extension limits of the local costmap.

Conducting a human-factor experiment. Since human comfort is a highly personal and subjective feeling, it is really difficult to quantify it. In the context of this thesis, metrics were defined (minimum relative distance and integral of relative distance within a critical area) in an attempt to quantify and measure comfort. Another interesting approach would be the conduction of a human-factor experiment, in which participants would be exposed to the considered interaction scenarios and then would be asked to evaluate their interaction with the robot and grade the behavior of the robot when each navigation method is applied. For this purpose common official acceptance tests, such as the Van der Laan acceptance test [1], can be employed.

[1] http://www.hfes-europe.org/accept/accept.htm

Appendix A

A brief introduction to the Robot Operating System (ROS)

Robot Operating System (ROS) [1] is an open-source *robotics middleware*, namely a collection of software frameworks intended for development of robot software. Albeit ROS is not an operating system, it provides the services that usually are available in an operating system, such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes and package management. The most significant attribute of ROS is the fact that it provides a variety of tools and libraries for obtaining, developing and compiling code across multiple computers and platforms. The main goal or ROS is to support code *sharing* and *reuse* as well as *collaboration* among the world community of roboticists and developers. Hence, ROS can be described as a distributed framework of processes that are individually designed but jointly used at the same time. To this end, ROS also supports a federated system of code *repositories*, where code is stored and is publicly available for use and improvement. Robotics developers are free and encouraged to exploit the available code and contribute to the robotics society by adding their unique innovations. Certainly, ROS is not the only framework that provides these advantages. However, what makes ROS unique is its acceptance and fame within the robotics society. As the author of [56] claims: "... the level of widespread support for ROS across the robotics community. This 'critical mass' of support makes it reasonable to predict that ROS will continue to evolve, expand, and improve in future".

ROS was started back in 2007, by Stanford University, building on existing robotic middlewares such as *switchyard* [2]. From the next year, the main responsibility for ROS development was under a robotics company called Willow Garage [3]. Later, in 2013, ROS custody passed to Open Source Robotic Foundation (OSRF) [4]. ROS is designed such that the code can be used across many computers and also be compatible with other robot software frameworks. So far, ROS has been integrated with OpenRAVE, Orocos and Player. Developers aim to design

[1] http://wiki.ros.org
[2] https://en.wikipedia.org/wiki/Robot_Operating_System

ROS in many programming languages. Until now, ROS is designed in C++, Python and Lisp and some libraries have also been written in Lua and Java. Although ROS is not a real time operating system, nevertheless real time code can be run through ROS. ROS libraries are structured in a Unix-like manner, mainly due to their dependence on open-source software. The main Operating System (OS) supporter of ROS is Ubuntu Linux. Many releases or, as they are known in ROS parlance, *distributions*, have made their appearance with the first one to be Box Turtle in 2010. The ROS version used in this thesis is the 8th one, named ROS Indigo Igloo, introduced in July 2014. The OS used for installing and running ROS is Ubuntu Trusty 14.04 LTS.

A-1 ROS concepts

ROS is structured in three levels of concepts: the ROS Filesystem Level, the ROS Computation Graph Level and the ROS Community Level. These levels and the main concepts included in each of these, will be briefly discussed in the following subsections. An extensive overview of these levels can be found in `http://wiki.ros.org/ROS/Concepts`.

A-1-1 ROS Filesystem Level

The Filesystem Level is the hierarchy in which the different files and utilities are structured within ROS.

Packages A ROS package is a coherent collection of files, including executables, libraries, tools, configuration and supporting files, intended to serve a purpose. All these files, give to the package a certain functionality to fulfill its purpose. A package constitutes the lowest level of ROS hierarchy.

Manifests A manifest (`package.xml`) is a supporting file that contains the description of a package. The description consists of a name, version, license information etc. The manifest is important because it also defines the dependencies of the package, namely the functionalities in which the package depends on in order to work properly. These functionalities are usually included in other packages. Hence, a key feature of ROS is the several dependencies between packages.

CMakeLists A `CMakeLists.txt` is a script for an industrial build system called `CMake`. This script contains a collection of commands that specify which files are the executables, where shoud they be executed, which files and libraries should be included and installed for the executables etc. Every package has a `CMakeLists.txt` and a manifest `package.xml`.

[3] `https://en.wikipedia.org/wiki/Willow_Garage`
[4] `https://www.osrfoundation.org/`

Metapackages A metapackage is a **package** that represents a group of related packages. It is used for solving some compatibility issues with former versions of ROS. In earlier ROS versions (before groovy), there existed a collection of packages, called a **stack**. This concept was gradually ruled out by the metapackage. There is a key difference between metapackages and stacks. Metapackages are packages with a manifest file and -no other packages are stored inside their directory-, whereas stacks are directories with a stack manifest file, including packages.

Workspaces A workspace is a folder where we can modify install and build packages. Different packages that depend on each other (or not) exist together in the same workspace. In the workspace there is a folder called `src` in which the packages are included. In the workspace there are two more folders called `build` and `devel` respectively. These folders are automatically generated when we build the workspace and contain build-related files, object code and the executables themselves [56]. Inside the `devel` folder, among others, there is a script called `setup.bash` which locates the packages and the executables inside the workspace. By sourcing this script, the workspace and all of its packages and utilities become available for use. For building a workspace, ROS Indigo uses a build system called `catkin` [1].

Launch files A launch file is a script in XML format. A launch file, as its name addresses, launches one or more executable scripts. This file can also sets arguments, modify parameters and pass values inside the executable scripts . More information about launch files can be found in http://wiki.ros.org/roslaunch/XML.

Parameter files A Ain't Markup Language (YAML) parameter (or configuration) file are files written in YAML language which is a human-readable data serialization language. These files are used to map parameter names to values. They simply assign values to parameters, which are defined inside the executables.

Repositories A repository is a collection of files that is under *version control*. Repositories can contain workspaces, one or more packages and can be stored and accessed by a Version Control System (VCS). Most of the publicly available ROS packages are stored in Github [2].

A-1-2 ROS Computation Graph Level

The Computation Graph is the *peer-to-peer* network of ROS processes that are processing data together.

Nodes The core Computation Graph units are the small individual programs called *nodes* that run simultaneously. They are simple processes that perform computations. In a robot system there are multiple nodes. For example, a node controls the camera of the robot, one node the laser scanner, another one performs localization etc. ROS does not require a node

[1] http://wiki.ros.org/catkin
[2] https://github.com/

to do a particular job however it requires each of the node to strictly have a unique name. A node can be written in C++ or Python.

ROS Master For any task to perform in ROS, the multiple nodes have to communicate together. This communication is facilitated by the ROS Master . The ROS Master is a process that sets up communication between the nodes, provides naming and services and helps the nodes to locate its other and start to communicate.

Parameter Server The Parameter Server is a centralized mechanism that keeps track of parameter values. Parameters are simple strings that are assigned a value. The value can be a boolean, integer, floating point number or other. Nodes must be able to access these parameters, which are stored in the parameter server. The parameter server is currently part of the ROS Master.

ROS Messages Messages are data structures and constitute the core mechanism that ROS uses to exchange information between nodes. It is simple the way that nodes communicate with each other. A message has a *type* and comprises of some *fields*. The message type concerns the **data type** of the message and determines the content of the message, namely what kind of information the message carries. The fields show details of the message type. For example a velocity message comprises of an angular and a linear vector, a time stamp etc. Each of these fields are assigned values that can be of several kinds, in this case the values usually are floating point numbers.

Topics In order for the messages to be sent and received by the nodes, they are organized into named topics . The topic is basically a *name* that is used to identify the content of the message. The idea of node communication is based on a **publishing/subscribing** mechanism. A node that wants to share information, **publishes** this information in messages on a topic with a name (preferably) associated to the content of the information. A node that is interested in this information, **subscribes** to the corresponding topic. The ROS Master ensures that the *publisher* and the *subscriber* node can find each other and exchange the information. A single node might publish or subscribe in multiple topics. For example a node that performs localization, subscribes to the Inertial Measurement Unit (IMU) sensor messages and to the laser scanner data. The same node can publish the pose of the robot, but also the velocity of the robot base. In addition, the same topic can be published by two or more nodes. For example, a node that performs motion planning has to know the pose of the robot. The node subscribes to a `robot_pose` topic which is published by the IMU node, but when no IMU data are received, the topic is published by the visual odometry node.

Services A service is an alternative way for communication between nodes and aims to overcome some of the message's limitations. A service has two main characteristics:

- A service is a *bi-directional* form of communication. A node sends a call to another node and waits for a response. On the contrary, for messages this guarantee of receiving the information does not exist. Often messages are published with no nodes subscribing to them.

- Services are a form of *one-to-one* communication. This means that a request is made by one node and the response is received by the same node. On the other hand, messages implement a *many-to-many* communication, since multiple nodes can publish/subscribe to the same messages.

The node that makes the request is called a server node and the node that receives the request (and publishes a response) is called the client node.

Bags One of the most significant attributes of ROS is the ability to *record* and *replay* messages . This provides a powerful way to test robot software even if the actual sensors and hence, the real messages, are not available or difficult to collect from the sensors. The recorded data are stored in *bag files*. By running a bag file it is possible to replay the messages and make them available as if they were really published.

Practical example

Let us assume that we have a mobile robot with an RGB-D sensor. The task is to perform mapping using filtered data from this sensor. For this purpose a node `camera` obtains the measurements from the RGB-D sensor and publish them. The measurements are published as Image/msgs messages on a `\camera_images` topic. Another node `filter_images` subscribes to `\camera_images`, filters them and publishes a `\camera\filtered_images` topic. However the `filter_images` node, does not care where the Image/msgs messages are coming from. This means that if for some reason the operation of the `camera` node terminates, we will not see any error, but simply that no messages are received. If we start another node called `camera_2` which obtains measurements from another sensor, the `filter_images` node will start again publishing filtered images. This example illustrates the sense of *loose coupling* between the nodes. In Figure A-1 the basic structure of the ROS Computation Graph Level is shown.

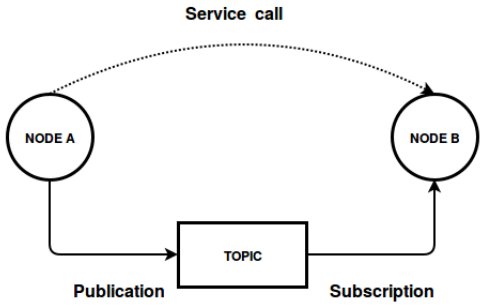


Figure A-1: The basic structure of the ROS Computation Graph. (Adopted from ROS Wiki).

A-1-3 ROS Community Level

As already mentioned, ROS is strongly supported by its community. There are multiple sources that someone can search for help and advice. The main forum for documenting

information about ROS, is the ROS Wiki (wiki.ros.org/Documentation). There a variety of tutorials is available as well as information on software, hardware, education material and community events. The main Q&A forum is the ROS Answers (http://answers.ros.org/questions/), where someone could post his question or problem and let the community of developers help him to find the answer. It is also possible to subscribe to the ROS-users mailing list to receive updates and announcements. Moreover, the ROS discourse forum (https://discourse.ros.org/) contains several threads with comments and discussions around multiple robotics related topics. Last but not least, ROS community strongly supports individual contribution. To this context, there are available tools and tutorials to facilitate the generation of Wiki pages, repositories and of course new packages. There is also a ticket system for finding and reporting issues and bugs on existing software (http://wiki.ros.org/Tickets). Finally, ROS is connected with VCS, with the officially preferred to be Github. Tutorials and guidelines on how to connect and use Github are available on the ROS Wiki.

A-2 Tools and simulators

In this Section, some additional concepts and tools, necessary for the implementation of any robot navigation system are briefly described. In addition visualization tools and simulators, used in this thesis are introduced.

Coordinate Frames/Transforms A typical robotic system consists of multiple 3D coordinate frames that change over time . These frames include the world (or global) frame, the base frame of the robot, wheel frames, camera frame, arm frames etc. In order to be able to determine the pose and the configuration of the robot at each time instant, we need knowledge of the position and orientation of these frames and the spatial relations between them. These computations are performed by the `tf` package which provides all all the information about the coordinate frames of a robot. Through this package we are able not only to get a transformation between two frames, but also to insert one, by creating new frames or modifying the existing ones. A robot system composed of mutliple frames is shown in Figure A-2 [1].

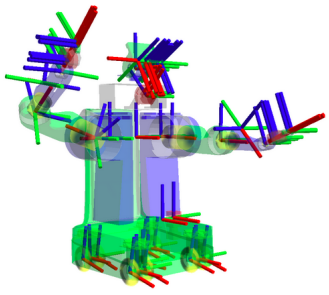


Figure A-2: A robot composed of many different 3D coordinate frames. Each frame has three colors, one for each axis. The transformations and all relevant information about the frames are provided by the `tf` package.

[1] http://wiki.ros.org/tf

Client Libraries The ROS client libraries are simply collections of code that facilitates programming over ROS. These libraries implement in code most of the ROS concepts and enable the user to write nodes, topics and services. Such a client library can be written in any language but, for the time being, ROS support and guarantees are provided only for C++ (`roscpp`) and Python (`rospy`).

Pluginlib and Plugins `pluginlib` is a is a C++ library for loading and unloading `plugins` from within a ROS package. Plugins are dynamically loadable classes that are loaded from a runtime library. In simple words, by using plugins, the user is able to insert or append his part of code in an existing application without the need to access the source code of the application. An example is the way that the user can create a planner for path planning for the robot. The class that includes all the necessary objects and members to write code for path planning in ROS, is called `base_global_planner`. By writing his own code for a new planner and insert it as a plugin to the `base_global_planner` class, the user is able to button the new code to the existing ROS path planning class, without the need to access and modify the source code of the path planning process. This means that the robot will simply use the new planner to design a path, as long as the new planner is inserted as a plugin.

GUIs ROS enables visualization supervision and interaction with the robot during runtime via dedicated Graphical User Interface (GUI)s. ROS uses a Qt-based [2] framework for GUI development, called `rqt` . `rqt` implements the several GUI tools in the form of plugins as dockable windows on our system. `rqt` GUIs include the standardized procedures of a GUI, allow multiple dockable windows without the need to run multiple GUIs, ease editing of the user's customized interface and strong support by the ROS community. There are numerous tools provided by the `rqt` GUI such as the `rqt_graph` and `rqt_robot_monitor` that can be used to inspect relations between nodes and topics and detect erroneous connections. Several additional tools are implemented in the form of plugins (http://wiki.ros.org/rqt). An example of a `rqt` GUI is shown in Figure A-3.

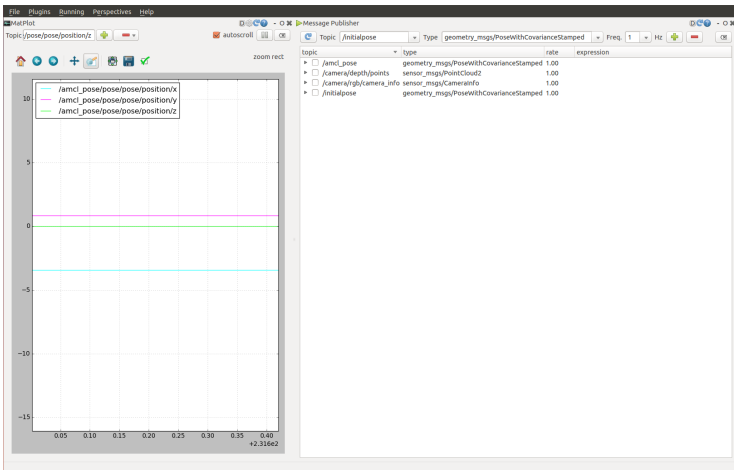


Figure A-3: A `rqt` GUI for visualizing, plotting and inspecting the relations between nodes and topics in a robotic system running in ROS. On the left dockable window the robot pose topic is plotted and on the right some topics have been added to the inspection queue.

Rviz `rviz` is a visualization tool for ROS. It uses a specific type of messages, called *markers* to visualize basic shapes and models on the screen. Models can be for example human figures or some common objects. The markers messages have a `visualization_msgs/Marker` type, which includes fields with information about the position and orientation of the marker, the colors, some geometrical features like volume, etc. Apart from markers, `rviz` utilizes also other display types such as images odometry, grids and others. A complete catalog can be found here: http://wiki.ros.org/rviz/DisplayTypes. The most important feature of `rviz`, is that it can visualize not only the robot itself, but also its actions. For example the path that the robot follows, the costmap that the robot takes into account during navigation, the laser scan etc. `rviz` also offers additional options for setting navigational goals, perform localization, executing motions and others. An illustration of the `rviz` tool is shown in Figure A-4.

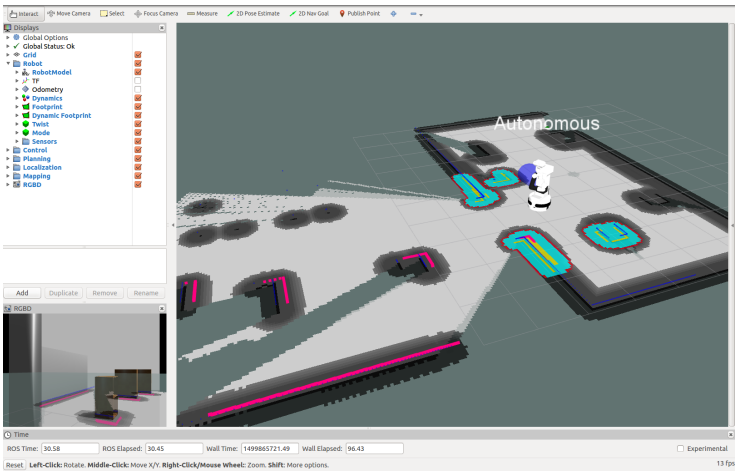


Figure A-4: A snapshot of an `rviz` window during a robot performing 3D mapping. On the left there is a list with the visualized topics and objects. There is also a view of what the robots sees through its camera (bottom left). On the right the virtual environment is visualized. On the top there is a bar with options such as setting a navigation goal or performing localization.

[2] https://www.qt.io/

Gazebo Gazebo [3] is a robot simulator that is under the stewardship of OSRF. It can simulate robots in indoor (mainly) and outdoor environments and is available for ROS in the form of a package that provides several plugins.

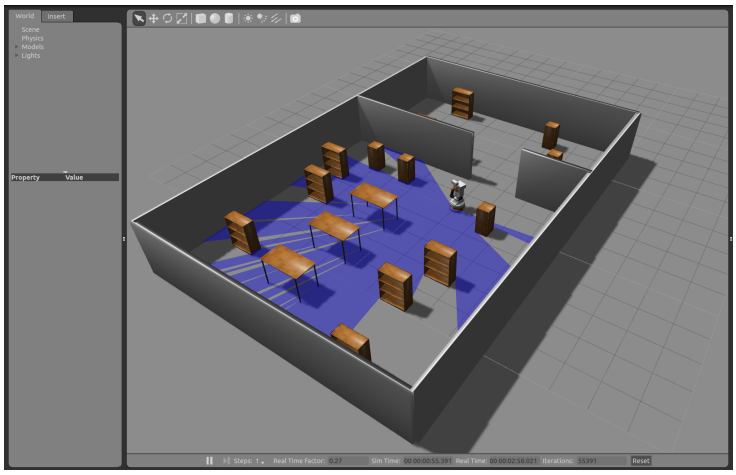


Figure A-5: An indoor environment and the TIAGo robot simulated in Gazebo.

Gazebo was integrated with ROS and the PR2 [4] robot in 2009, by John Hsu of Willow Garage. Since then, it enjoys great development and massive support from the ROS community. Inside a project in ROS, Gazebo usually is started as a ROS node called `gazebo` and supports subscribed and published topics as well as services (http://wiki.ros.org/gazebo). An illustration of a Gazebo world is shown in Figure A-5. In this thesis, Gazebo version 2.0 is used as it is guaranteed to be compatible with the TIAGo robot.

[3] http://gazebosim.org/
[4] http://wiki.ros.org/Robots/PR2

Appendix B

The proposed framework implementation over ROS

In this Appendix a brief discussion of the core Robot Operating System (ROS) packages used for autonomous navigation is made and the ROS implementation of the proposed human-aware navigation framework is described. The framework is implemented according to the ROS navigation stack architecture. The navigation stack is a set of algorithms and libraries that make use of sensor information and odometry in order to control the robot motion using messages. The navigation stack is ubiquitous and easily used with almost any robot. Sometimes it may require some modification in the form of extra nodes and configuration files, but the general principles are the same. In the following Section, we will focus on the key elements of the navigation stack. The rest of the Sections concern the implementation of the individual components used in our framework.

B-1 ROS Navigation Stack

B-1-1 Overview

The navigation stack obtains information from sensors and odometry and generates velocity commands that are sent to the mobile base of the robot. These commands determine the motion of the robot, namely the path that it follows and are usually a product of a motion planning algorithm. Prerequisite to use the navigation stack, is the robot to run ROS, to have a transformation (`tf`) tree available and to be able to work with the publishing/subscribing message mechanism. The dynamics of the robot posses some challenges to the out-of-the-box usage of the navigation stack, hence some configuration may be necessary to adhere to these dynamics.

As far as hardware is concerned, the navigation stack is destined only for differential drive and holonomic wheeled robots. The velocity commands must be consistent with a (x, y, θ) velocity form (x, y linear and θ angular velocity). The existence of at least one sensor (such

as a laser scanner) is indispensable. The readings from the sensors are used for mapping and localization. Finally, in its initial form the ROS navigation stack was designed to work with square-shaped robots, so some difficulties with large rectangular or circular robots may arise. A general scheme of the ROS navigation stack is shown in Figure B-1 [1]. In the following Sections the main consisting components will be discussed.

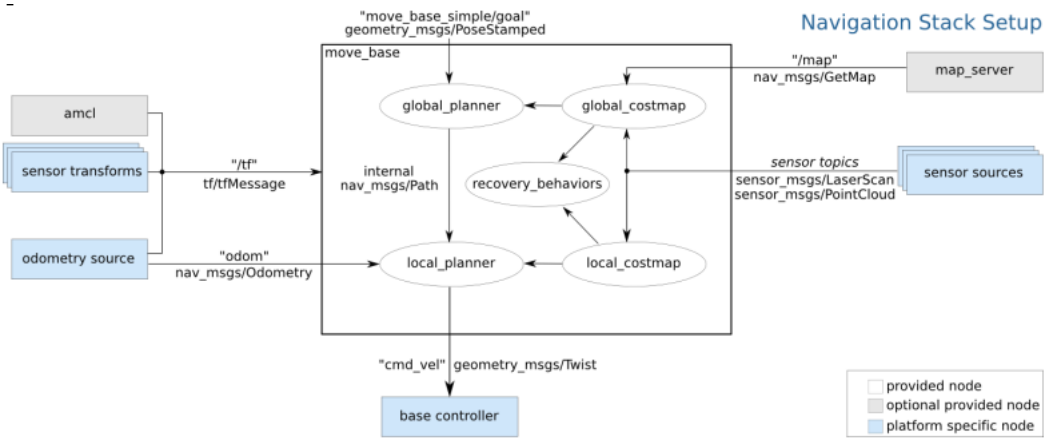


Figure B-1: A scheme of the ROS navigation stack, with the nodes and structures necessary for moving the robot successfully in the environment.

B-1-2 `map_server` package

The `map_server` package provides the map data (information about the structure of the environment) and several tools for manipulating and storing this data. The map is provided as a ROS service. The map is stored in information contained in two types of files. An image file that containes encoded data of the occupancy grid of the map and a Ain't Markup Language (YAML) configuration file that contains the map meta-data and names of the image file.

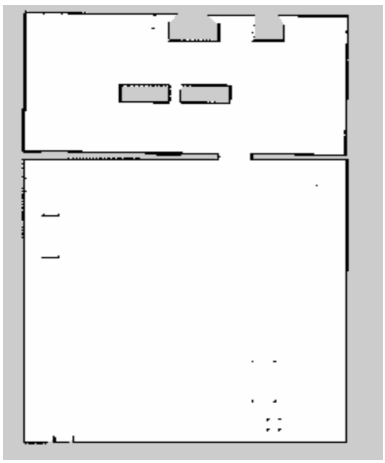
The **image file** contains the occupancy information of every cell of the world grid, in the color information of the corresponding pixel. This simply means that dark pixels denote occupied cells, white pixels denote empty cells and intermediate color pixels denote unknown cells. The state of a cell (occupied, empty or unknown) is determined by some threshold values stored in the YAML file. The occupancy value (probability) of a cell is calculated as:

$$\texttt{occ\_value} = (255 - \texttt{color\_avg})/255.0$$

where `occ_value` is the occupancy value of a cell and `color_avg` is the 8-bit value as a result from averaging over all color channels. For example, in a 24-bit color image, a pixel with the color 0x0a0a0a has an occupancy value of 0.96 yielding that the corresponding cell is occupied. The pixel with a color of 0xeeeeee has an occupancy value of 0.07 yielding an empty cell. Translated in ROS message values, the occupancy value is an integer between 0 and 100, with 0 meaning completely empty and 100 totally occupied. The special value -1 denotes

[1] http://wiki.ros.org/move_base?distro=lunar

unknown value. An example of an image file of a map, with the occupancy information in color values is shown in Figure B-2a.



(a) The content of an image file of a map.

```
map.yaml
1 image: submap_0.pgm
2 resolution: 0.050000
3 origin: [-50.000000, -50.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
```

(b) The YAML file.

Figure B-2: Examples of an image file of a map that contains the occupancy information of the environment encoded in a color image and the corresponding YAML file. These two files contain the information of the map, needed for navigating the robot with the ROS navigation stack.

The **YAML file** contains several meta data about the the map and the image file. In particular it contains the following fields:

- **image**: directory to the image file that contains the occupancy information, or the name of the file if it is in the same directory with the YAML file.
- **resolution**: the resolution of the map in meters/pixel.
- **origin**: The 2D position and orientation of the lower-left pixel in the map, as (x, y, yaw), with yaw in counterclockwise rotation (yaw=0 means no rotation).
- **negate**: If enabled, reverses the empty-occupied / white-black semantics without affecting the threshold values though.
- **`occupied_thresh`**: Threshold value for being occupied. Pixels with occupancy value grater than this threshold are considered totally occupied.
- **`free_thresh`**: Threshold value for being free. Pixels with occupancy value below this threshold are considered completely empty.

An example of a YAML file of a map, is shown in Figure B-2b. Apart from providing the map, the `map_server` publishes topics related to the map data and values, services and tools for saving and retrieving a map. Detailed information can be found at http://wiki.ros.org/map_server.

B-1-3 `amcl` package

The `amcl` package implements the Adaptive Monte Carlo Localization (AMCL) probabilistic localization system, presented in Section 2-2-3. It contains many algorithms of those presented in the book Probabilistic Robotics, by Thrun, Burgard, and Fox. The `amcl` package

takes as input a map of the environment obtained through laser, laser measurements, transform messages and gives out the estimate for the pose of the robot (at each time instant). The package initializes a particle filter according to the parameters provided. It performs localization of the robot when the service `global_localization` is called. The package requires a known map of the environment which can be obtained either a priori or by a Simultaneous Localization and Mapping (SLAM) approach.

`amcl` computes the transformation between the incoming laser reading s and the odometry frame (which usually is the base frame of the robot), so in the `tf` tree, a path must exist between the frame in which the readings are obtained and the odometry frame. It is important to mention that by construction the package looks up the transform between the laser's frame and the robot base frame at initialization, and latches this transformation. This means that the `amcl` cannot work with a laser that moves with respect to the base of the robot. `amcl` is superior in comparison to a pure dead reckoning method since it accounts for the odometry drift. This is done because the `amcl` estimates the transformation between the map (world) frame and the robot base frame and publishes the estimates in the odometry frame. This transformation inherently accounts for the odometry drift, whereas dead reckoning considers the transformation only between the robot base frame and the odometry frame, so with no reference to the world frame. This reasoning is illustrated in Figure B-3 [2].

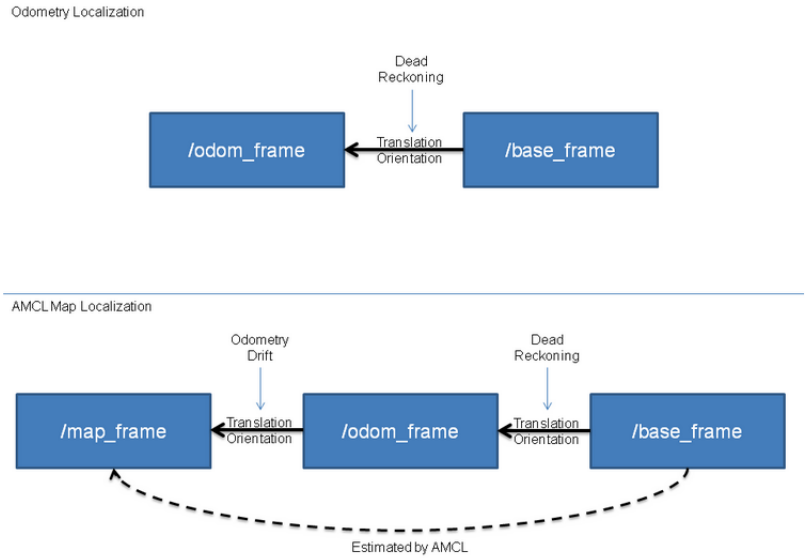


Figure B-3: Difference in localization between dead reckoning and `amcl`. `amcl` estimates the robot pose with respect to the world frame accounting for the odometry drift.

B-1-4 `move_base` package

The `move_base` package provides the velocity commands for the robot mobile base, in order to reach a specified goal in the map. The path that the robot has to follow towards the goal is provided by a navigation planner. In fact `move_base` incorporates two types of planners, the *global planner* and the *local planner* to accomplish the desired navigation task. The

[2] http://wiki.ros.org/amcl

global planner uses the global costmap of the environment (the entire static world) to plan a long-term path. The local planner uses the local costmap which is an area around the robot (typically extended up to the robot's sensor range) updated constantly as new sensory data are obtained. If changes occur in the local costmap, such as moving objects or humans, the local planner replans a path to avoid these moving obstacles with a tendency to stick to the already found global path. The level of how close the local path should stay to the global plan is usually a user preference that can be introduced as an optimization constraint to the planner.

The robot achieves the desired goal by approaching it with some user-specified tolerance. The key node that orders the robot to move is the `move_base` node. This node is capable to allow to the robot to perform the so-called *recovery behaviors* if it gets (or think it is) stuck. The recovery behaviors are ordered and are performed with a certain succession as shown in Figure B-4 [3].

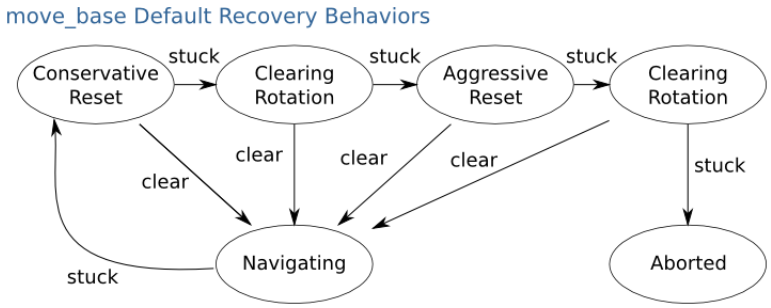


Figure B-4: Recovery behaviors that the robot will perform in succession in order to get unstuck. If everything fails the robot will abandon the effort to reach the goal and will mark the goal as infeasible.

The goal of `recovery_behaviors` is to clear the space so the robot can move again. The first action is to clear obstacles outside a user-defined area (conservative reset). Secondly, the robot will perform an on-spot rotation to clear out space (clearing rotation). If also the second action fails the robot will aggressively clear obstacles outside the area it can rotate (aggressive reset). Afterwards, it will attempt another clearing rotation and if this fails too it will abort the effort and mark the goal as infeasible to reach.

As already mentioned the `move_base` node utilizes two types of planners, the global and the local planner. The global planner is used as a plugin (See Appendix A-2) which must adhere to the `nav_core::BaseGlobalPlanner` class, specified in the `nav_core` package. Respectively, the local planner is used as a plugin which adheres to the `nav_core::BaseLocalPlanner`. This plugin architecture allows for different planners to be inserted in tte `move_base` node and tested within the ROS navigation stack. In the following the `nav_core` and the global and local planners packages will be reviewed.

`nav_core`

The `nav_core` package provides useful interfaces for the navigation stack. More precisely it provides the `BaseGlobalPlanner`, `BaseLocalPlanner` and `RecoveryBehaviors` interfaces.

[3] http://wiki.ros.org/move_base

Whereas the latter interface was explained in the previous Section, in this Section the planner interfaces are explained. What is important about the `nav_core` package is that it provides a plugin architecture that enables changing planner and adding new features. In consistence with Figure B-1, where the rough structure of the navigation stack is depicted, the particular components within the `move_base` that belong to the `nav_core` package are the `global_planner` , `local_planner` and `recovery_behaviors` packages. These, in conjunction with the global and local costmaps, produce the desired collision-free path and the velocity commands for the robot base given a goal. The goal is given as a stamped pose message, that is a 3D pose with a timestamp, to the `global_planner`. The `global_planner` by using information from the global costmap produces a long-term collision free path. This path is inserted to the `local_planner` which by using now the constantly updated local costmap, designs an online short-term collision free path, trying to stick close to the global one. The `nav_core` interface, via the `local_planner` interface, outputs a twist (linear and angular velocity) for the robot base in order to follow the local path.

Base Global Planner

The global planner is written as a plugin for the `move_base` node. It is merely a class which is provided by the `nav_core::BaseGlobalPlanner` interface. As a result, any global planner must adhere to this interface, namely must be written in C++ code as a child class of `nav_core::BaseGlobalPlanner`. The default global planners in the `nav_core::BaseGlobalPlanner` interface are the `global_planner` and the `navfn`.

`navfn` This planner uses Dijkstra's algorithm to find a global path between a starting and a goal position within a grid. The global path is the minimum cost path.

`global_planner` The `global_planner` is a more flexible version used instead of `navfn` and incorporates two well-known algorithms for finding a path, Dijkstra or A*. In addition, it enables a different approximation of the optimal path (least cost path) and provides the option of a grid path, namely a path that follows grid boundaries. A comparison between the resulting path of `navfn` and `global_planner` can be found at http://wiki.ros.org/global_planner.

Base Local Planner

The `base_local_planner` package provides two different implementations for local planners which adhere to the `nav_core::BaseLocalPlanner` class. The two implementations available by default in the `move_base` package, are the Dynamic Window Approach (DWA) and Trajectory Rollout planners.

The main job of the `base_local_planner` package is to provide a controller to drive the base of the robot along the desired path. However, to account for moving or unexpected obstacles (that are not present in the global map) it performs sampling in the control space of the robot to find a local path that does not lead in collision with the detected obstacles. In practice, the local planner searches in the control space of the robot to find these dx, dy and $d\theta$ velocities

that if applied they will not lead to collision. For this purpose the planner selects a local area around the robot and represents it as a cost grid map with a value function. Each cell in the grid map is assigned a cost according to the context of the value function. The value function can incorporate several desired contexts such as proximity to obstacles, proximity to goal, resemblance to the global path, speed, change of orientation etc. The search area specified by the local planner is the local costmap and is updated in each cycle while also a new search is performed. The local costmap (as well as the global one) follow the layered costmap architecture, hence new layers correspond to new context to the value function.

According to the above reasoning, the principal idea of both the DWA and Trajectory Rollout planners is the following (Figure B-5):

1. Sample the control space of the robot ($dx, dy, d\theta$ velocities).
2. For every sampled velocity, simulate a trajectory using the current state of the robot to investigate how the robot would move if the sampled velocity was applied for a relatively short period of time.
3. Based on the value function and the desired contexts that the robot should exhibit, evaluate the score (the total cost) of each simulated trajectory.
4. Eliminate infeasible trajectories (those led to a collision)
5. Choose the best performing trajectory (least cost or some other metric) and apply the corresponding sampled velocity to the base of the robot.
6. Clear out the scores and the search space an repeat the process.

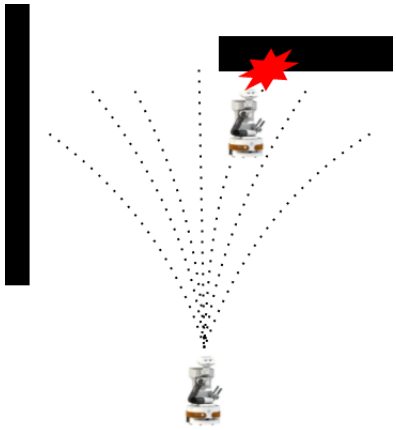


Figure B-5: Visualization of the DWA approach. The algorithm samples the control space of the robot and for every sampled velocity it simulates a trajectory using the current state of the robot to investigate how the robot would move if the sampled velocity was applied for a relatively short period of time. Then the trajectories that lead to collision are eliminated and the optimal trajectory is evaluated.

Dynamic Window Approach DWA was proposed by Fox et al. in their seminal paper [57]. According to the method, the purpose of DWA is to find a $(dx, dy, d\theta)$ velocity tuple which will give an optimal trajectory with respect to the robot's current pose by searching the set of sampled velocities for just one simulation step ahead, given the acceleration limits of the robot. On the other hand the Trajectory Rollout [58], searches the set of sampled velocities over the entire forward simulation period given the acceleration limits of the robot.

Furthermore, DWA searches for velocities in the control space, which are restricted to be admissible. This means that the robot must be able to stop before reaching the closest obstacle on the trajectory dictated by these admissible velocities. Also, since DWA searches for one simulation step ahead, it will only consider velocities within a dynamic window, which is defined to be the tuple of velocities that are reachable within the next time interval, given the current translational and rotational velocities and acceleration. This makes DWA a more efficient algorithm, since it uses a reduced search space.

The score evaluation of a trajectory is an optimization problem which depends on the different contexts as stated in step 3 above. DWA aims to find the optimal (or best scoring) trajectory by minimizing an objective function of the form:

cost = `pdist_scale` ∗ (distance(m) to path from the endpoint of the trajectory)
+ `gdist_scale` ∗ (distance(m) to local goal from the endpoint of the trajectory)
+ `occdist_scale` ∗ (maximum obstacle cost along the trajectory in obstacle cost (0-254))

`pdist_scale` is the weight for how much the local planner should stay close to the global path. `gdist_scale` is the weight for how much the controller should attempt to reach its local goal, regardless of path, and also controls speed. Finally, `occdist_scale` is the weight for how much the robot should attempt to avoid obstacles. This factor is where the cost value of each cell in the local costmap is taken into account. The cost value of each cell is determined by the layered architecture and the individual cost that each layer has assigned to the cell in the `updateCosts` phase. In this thesis the costs for **penetrating the proximity of humans** and **blocking human intention** have been added to this factor of the objective function. Therefore, they are considered by the local planner accumulated with the costs of the obstacles and inflation layers. An analytical description on the additional parameters and different objective functions that can be used with the DWA can be found at `http://wiki.ros.org/base_local_planner`.

B-1-5 Vector Field Histogram (VFH)

The Vector Field Histogram (VFH) [49] is a real-time obstacle avoidance method which uses a *2D Cartesian histogram* grid as a representation of the world. The VFH computes the desired steering commands for the robot, using a two-step process. In the first step the histogram grid is reduced to an *one-dimensional polar histogram*, constructed around the location of the robot. Every sector (bar) of the polar histogram represents the *obstacle density* in the corresponding direction. In the second step, the method determines a direction, namely a sector of the polar histogram, which has low obstacle density and is pointing as close as possible to the target of the robot. Then, the suitable steering command is given to the robot in order to head towards this direction.

Initially the environment is split into sectors, which form a histogram, around of the location of the robot. Then the histogram is reduced to a polar histogram where each sector has an obstacle polar density value, determined by the sensor readings. In the case of the VFH version used in this thesis, these densities are determined by the values of the costamp, namely the values of the cost of the cells that are included in each sector. The density value is mapped

on the histogram by a distributing function (which in our case is unknown). The function of the original VFH approach is given by:

$$h_s(q) = \int_{\Omega_s} P(p)^n \left(1 - \frac{d(q,p)}{d_{max}}\right)^m dp$$

where $\Omega_s = \{p \in W \setminus p \in s \wedge d(q,p) < d_0\}$. The above equation denotes the density h of the histogram sector s and is proportional to the probability P that a point (or cost of a cell in our case) is occupied by an obstacle and also proportional to a distance factor that increases as the distance to the considered point decreases.

The resulting polar histogram consists of *peaks* (sectors/directions with high polar density) and *valleys* (directions with low density). The robot basically tries to find a valley that is close to the sector, namely the direction, where the target location is. The set of candidate directions is the set of consecutive sectors with a polar density below a threshold and which are closest to the sector that contains the target position. This set is called the *selected valley*. The idea is illustrated in Figure B-6.

Now that the set of candidate directions (selected valley) has been determined, the next task is to chose a direction from this set. This is done by following a heuristic strategy that depends on the sector that contains the target or on the size of the selected valley. There are three cases that form the heuristic strategy:

- Case 1: the target sector is actually inside the selected valley. Then the solution is $s_{sol} = s_{target}$, where s_{target} is the sector that contains the target location.
- Case 2: the target sector is not in the selected valley and the number of sectors of the valley is greater than a number of sectors m. Then the solution is $s_{sol} = s_i \pm m/2$, where s_i is the sector of the valley that is closer to the target sector s_{target}.
- Case 3: the target sector is not in the selected valley and the number of sectors of the valley is less or equal than m. Then the solution is $s_{sol} = (s_i + s_j)/2$ where s_i and s_j are the extremities of the valley.

The resulting solution is the sector s_{sol}, whose bisector θ_{sol} is the direction solution of the acVFH method. The velocity v_{sol} is inversely proportional to the distance from the closest obstacle. The command sent to the robot is the control $u = (v_{sol}, \theta_{sol})$ [6].

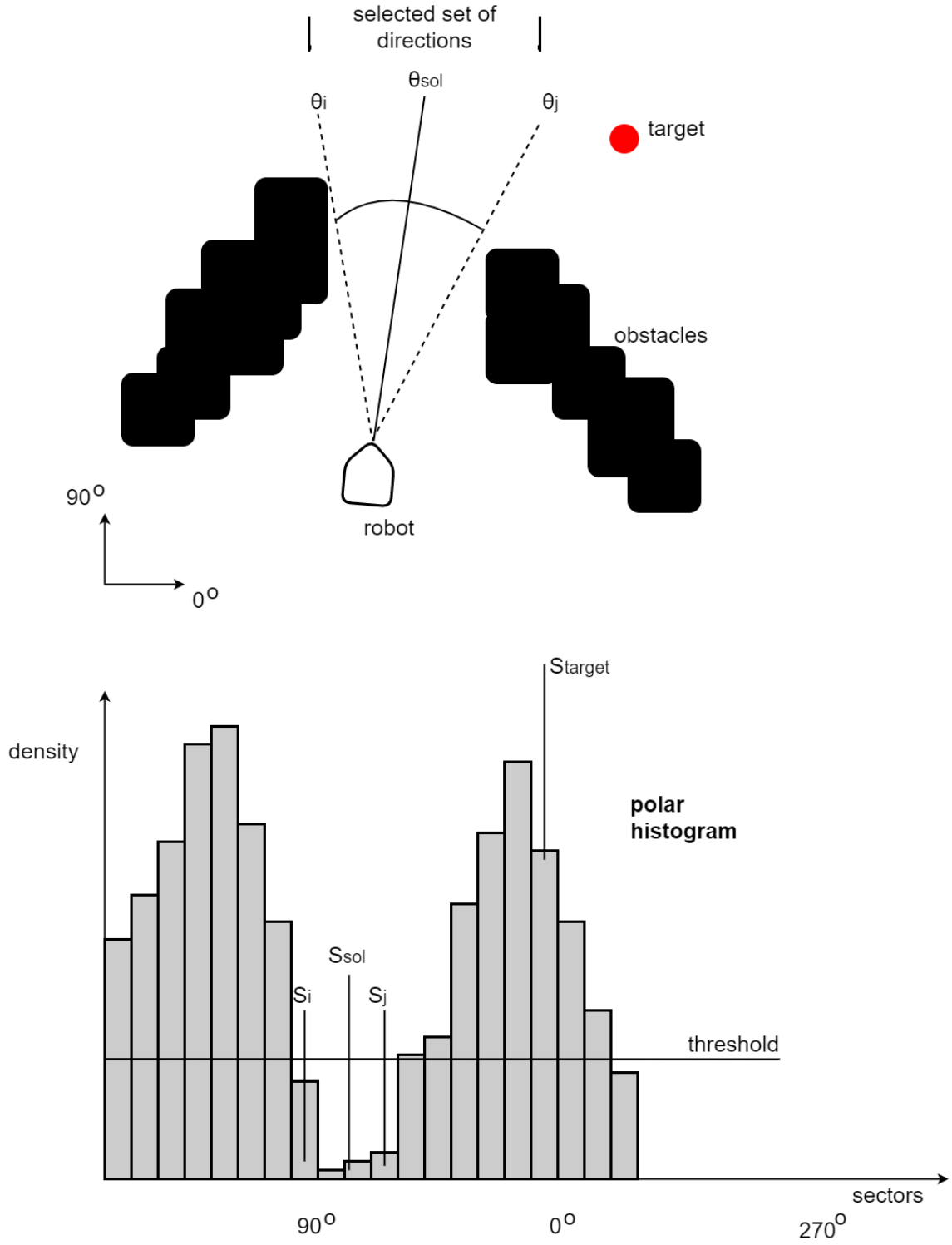


Figure B-6: Visualization of the VFH method. The candidate valley is the set of sectors with density values less than the threshold. The heuristic rule followed is case 3, since the sector target s_{target} is not in the valley and the number of sectors is smaller than m (for example $m = 10$). Hence, he direction solution θ_{sol} is given by the bisector of the sector $s_{sol} = (s_i + s_j)/2$. θ_i and θ_j are the bisectors of sectors s_i and s_j respectively (Adapted from [6]).

B-1-6 `costmap_2d` package

It has already been discussed how the layered costmap architecture works and serves as a descendant of the occupancy grid for safe and efficient navigation. In this Section, the inner mechanisms of the `costmap_2d` interface, which offers the layered architecture and the implementation details of how new layers are added to the map are investigated.

The `costmap_2d` provides all the necessary functionalities for building a layered costmap. Particularly, it takes in sensor data regarding the environment, builds the occupancy grid, inflates the obstacles and adds the cost values to each cell according to the specified contexts in each layer. The package initializes the `map_server` for the creation of the costmap and offers an option for a so-called *rolling-window* costmap. A rolling-window costmap, places the robot at the center of a square and updates only the area delimited by this square. Basically, the local costmap is such a rolling-window costmap. The package offers also configuration options for the sensors (e.g. changing the range of the laser sensor to extend the local costmap).

The main interface of the costmap is the class `costmap_2d::Costmap2DROS` which contains the majority of the costmap functionalities. The `costmap_2d::Costmap2DROS` object is a wrapper for a `costmap_2d::Costmap2D` object that exposes its functionality as a C++ ROS Wrapper. The `costmap_2d::Costmap2D` class implements the basic data structure for storing and accessing the two dimensional costmap. The interface `costmap_2d::LayeredCostmap` keeps track of the layers. Each layer is incarnated as a plugin and appended to the `LayeredCostmap` class. A more detailed explanation about how the layers are written and added in the layered costmap is given in Section B-1-8.

The package uses the sensors of the robot to *mark* a cell as occupied or to *clear* a cell. Marking is performed by adding an index in a 2D array which represents the grid of the costmap. Clearing is slightly more complicated since the cell, in order to be cleared, is traced from the origin of the sensor and onwards and then the index of the cell is removed.

Each cell in the grid can have one of 255 cost values. However, the costmap structure can only represent three (as discussed in Chapter 2), namely occupied, empty or unknown. An occupied cell is assigned a `costmap_2d::LETHAL_OBSTACLE` cost, an empty cell a `costmap_2d::FREE_SPACE` cost and an unknown cell a `costmap_2d::NO_INFORMATION` cost.

The update process of the layered costmap is discussed in Chapter 2. The update process is performed at a specific rate determined by the `update_frequency` parameter. At each cycle, sensory data are obtained and marking and clearing operations are executed. Then the new cost values are assigned (by the `updateCosts` method) and each cell has one of the aforementioned values. Afterwards, the cells that are occupied are inflated by user-defined radius. Other operations defined by the contexts of each layer are also carried out. To perform correctly all the necessary actions to the cells, the `costmap_2d` package makes extensive use of transformations between the sensors frames, the robot frame and the world frame. These transformations are provided by the `tf` package (Section A-2) and must be up-to-date at each cycle. There can also be specified some tolerance regarding the time waiting for a transformation to become available. If no transformation is obtained for a specific amount of time. Then the navigation stack immobilizes the robot. In the following a closer look in the mechanisms of the layers and the way they are incorporated in the costmap is taken.

B-1-7 The Layers

Static Layer The static layer obtains the static map as a encoded occupancy grid (Figure B-2a) from the `map_server` via a service call. As the static world does not really change, the static layer incorporates stable data and during the update process is does not modify the master costmap significantly. The static layer is a `costmap_2d::StaticLayer` object of the class `costmap_2d::Layer`.

Obstacles Layer The obstacle layer uses sensor topics to mark or clear cells in the costmap. Usually the layer performs these operations in 2D. An extension has been created that uses point cloud data to detect 3D obstacles [39]. The obstacles layer is a `costmap_2d::ObstacleLayer` object of the class `costmap_2d::Layer`. In the implementation of TIAGo robot, there are several obstacle layers, each one for every sensor used. Hence, there is a `obstacle_laser_layer` that updates cells according to the laser readings, a `obstacle_rgbd_layer` that considers readings from the RGB-D sensor and `obstacle_sonar_layer` that updates the costmap based on sonar measurements, but the latter is not used in the Marco robot prototype.

Inflation Layer The inflation layer inflates the occupied cells over a user-specified radius by assigning decaying cost values in the neighboring cells. It is basically a buffer zone around the lethal obstacles in which high, but non-lethal costs are assigned in order to prevent the robot to approach the lethal obstacles too much for safety. The inflation zone specifies how much the footprint of the robot is allowed to approach the obstacle as explained in Figure 2-2. In particular, the center of the footprint should never enter the inflation zone. The costs within the inflation zone follow a decay exponential function as shown in Figure B-7 [1]:

Within the inflation zone, five different semantic values are defined, describing the relation between the cell and the robot. The five values are:

- `Lethal` cost means the cell is occupied so if the robot inserts this particular cell will definitely collide with the obstacle.
- `Inscribed` cost means that a cell is inside the robot's inscribed radius (shown in Figure B-7). This means that if the robot center is at a cell that its distance from an obstacle is less than the robot's inscribed radius, then the robot is in collision with some obstacle.
- `Possibly circumscribed` cost is similar to `Inscribed`, but using the robot's circumscribed radius as cutoff distance. In this case if the robot is at a cell which is at or above the cutoff distance, then it depends on the robot's orientation if there will be a collision or not. The word "possibly" is used because in general, the cutoff value is a user defined cost and can be assigned to a particular cell for exhibiting a desired behavior. For instance, if the user intends the robot to follow larger circumventions in corners in order to enter the blind area more safely, then he can assign different cutoff values close to the corner to keep the robot further away from it.
- `Freespace` cost is assumed to be zero and this means that the cell is empty and the robot can be there without danger.
- `Unknown` cost means that there is no information for the particular cell and is up to the user to command the robot how to deal with this area.

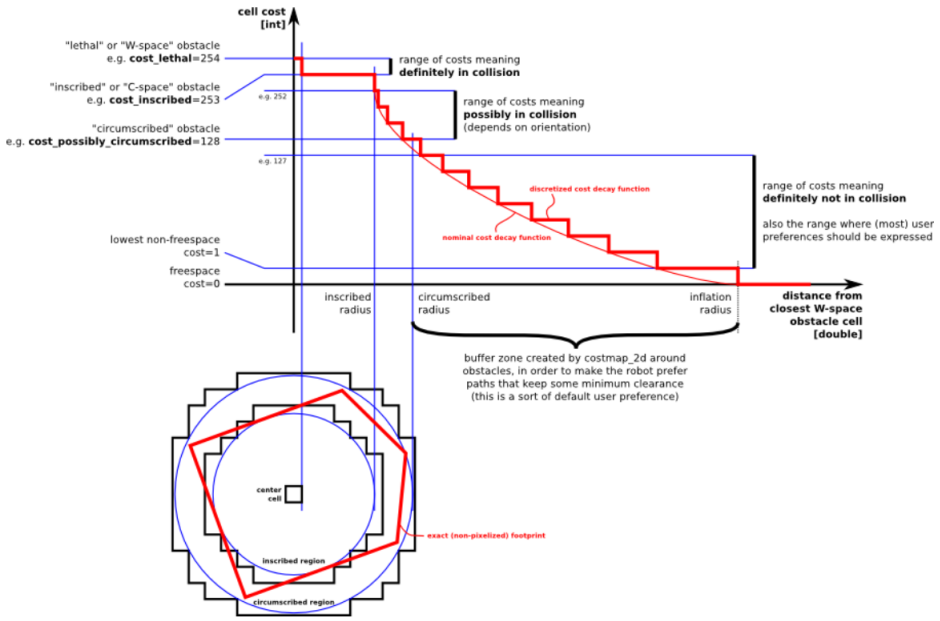


Figure B-7: The decay function that the costs around a lethal cell follow in order to prevent the robot from approaching the obstacle too much. The costs decrease with distance and have several cost values in the range 0-255, depending on how close the robot is to the obstacle

The rest of the costs have a value between `Possibly circumscribed` and `Freespace`. This value depends on the distance of the cell from the lethal cell and the decay function used. The two important parameters that must be selected from the user is the `inflation_radius` which is the radius in meters in which the obstacles are inflated and the `cost_scaling_factor` which is a factor for applying the costs in the inflation zone. The decay function that the costs in the inflation zone follow is defined as:

$$exp(-1.0 * \texttt{cost\_scaling\_factor} * (\text{distance from obstacle} - \texttt{inscribed\_radius}) *$$

$$* (\texttt{costmap\_2d::INSCRIBED\_INFLATED\_OBSTACLE} - 1)$$

where `costmap_2d::INSCRIBED_INFLATED_OBSTACLE` has the value 254. Since the `cost_scaling_factor` is multiplied by a negative in the above function, increasing the factor will decrease the cost values.

B-1-8 Incorporating layers in the master costmap

The proximity layer and the human intention layer were accumulated with the standard navigation layers in the layered costmap. To add an extra layer in the costmap it must be written as a plugin of the `costmap_2d::Costmap2D` interface. More precisely, a layer is an object class `costmap_2d::Layer` of the `costmap_2d::layered_costmap` class. To this end the proximity layer is defined as a `ProxemicLayer` child of the `costmap_2d::Layer` class and the human intention layer as a `HumanIntentionLayer` child of the `costmap_2d::Layer`

[1] http://wiki.ros.org/costmap_2d

class. The layers are inserted in the costmap as plugins since they both adhere to the layered costmap class. They can be complied separately and they are updated separately from the rest of the layers according to the process described in Chapter 2.

After having written the plugin that enables the layers to be inserted in the costmap, the layers must also be declared in the costmap YAML configurations files. There is one such file for the global and one for the local costmap. Then the layers are taken into account by the `nav_core` interface which incorporates the information from the global and local costmaps and dictate the planners to find the minimum cost path.

B-2 Human Detection

A mentioned in Section 2-2-4, for detecting humans in the environment the approach of Arras et al. [41] was used. This approach is implemented by Pantofaru in the `leg_detector` package (`http://wiki.ros.org/leg_detector`). The package is a component of the `people` package [2] which implements also a velocity tracker. The velocity tracker publishes both information about the pose and the velocity of each human. It subscribes to the laser readings topic (`sensor_msgs/LaserScan`) and publishes the x, y position and $v_x.v_y$ velocity of a person (`people_msgs/Person`). The person messages are pushed in an array message that contains information about all the persons in a single people message (`people_msgs/People`). The scheme of the human detection module is illustrated in Figure B-8.

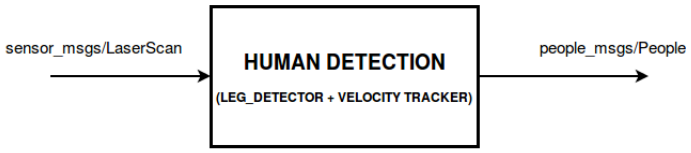
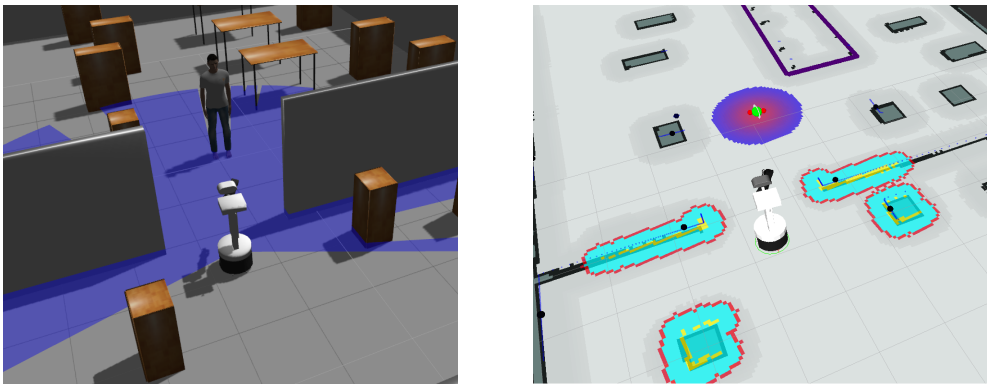


Figure B-8: General scheme of the human detection module. The module takes in laser scan readings and outputs an array with the position and velocity of every detected human at each reading cycle.

The package publishes also visualization markers for the laser readings and the detected humans. An example of a detection (in simulation) with the associated markers is shown in Figure B-9.



(a) The robot detects a simulated human. **(b)** The visualization of detection.

Figure B-9: Human detection. The black dots represent the laser readings, the green sphere the detected human. Around the human the proximity costmap is formed with a circular shape since the velocity is zero (the human is standing). The light blue area denotes the local costmap of the robot.

B-3 Intention Prediction

The task of the `intention_prediction` node is to compute the probabilities of visiting each destination according to the equations of Section 2-2-8. The node takes in a set of hard-coded destinations (x and y positions) and the human position included in the `people_msgs/People` messages. The node solves the Hidden Markov Models (HMM) and outputs the `most_probable_destination` as a stamped point (`geometry_msgs/PointStamped`). In addition, it provides visualization markers in the form of arrows that point to the probable destinations as shown in Figure B-10.

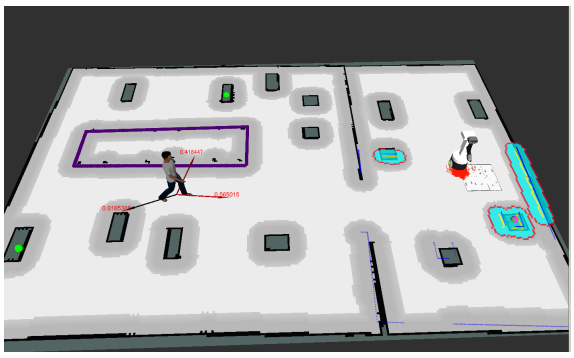


Figure B-10: Visualization of the human intentions calculated by the `intention_prediction` node. The most probable destination is given by the purple sphere. The other destinations are denoted by green spheres. The most probable intent is denoted by the light red arrow.

[2] https://github.com/wg-perception/people

B-4 Human Planner

The human planner does the job of estimating the path that the human is expected to follow in order to reach the most probable destination. The planner consists of two nodes:

`plan_human_path` node The `plan_human_path` node computes the shortest collision free path for the human, given the position of the human at each time, the end destination and the costmap of the environment. As explained in Section 2-2-9, the given costmap is the local costmap of the robot as the path planning for a person is performed based on the robot's perception of the environment. Given the position of a person, the end destination and the costmap, the optimal path is found by means of the Dijkstra algorithm. The basic implementation idea is to use the utilities of the `nav_core` package and apply them on the human instead of the robot. Consequently, the `navfn` planner, which carries the Dijkstra algorithm is employed to determine the human path. The resulting path for a simulated human is shown in Figure B-11. The `plan_human_path` node takes in people messages (`people_msgs/People`) in which the start point for the Dijkstra algorithm is contained (human position) and the `most_probable_destination` stamped point (`geometry_msgs/PointStamped`) constituting the end point. The node outputs the expected human path. The path is optimal in terms of distance only. It is an educated guess that the person simply tends to take the shortest unobstructed path to pursue a desired destination. The path is included in a `nav_msgs/Path` message.

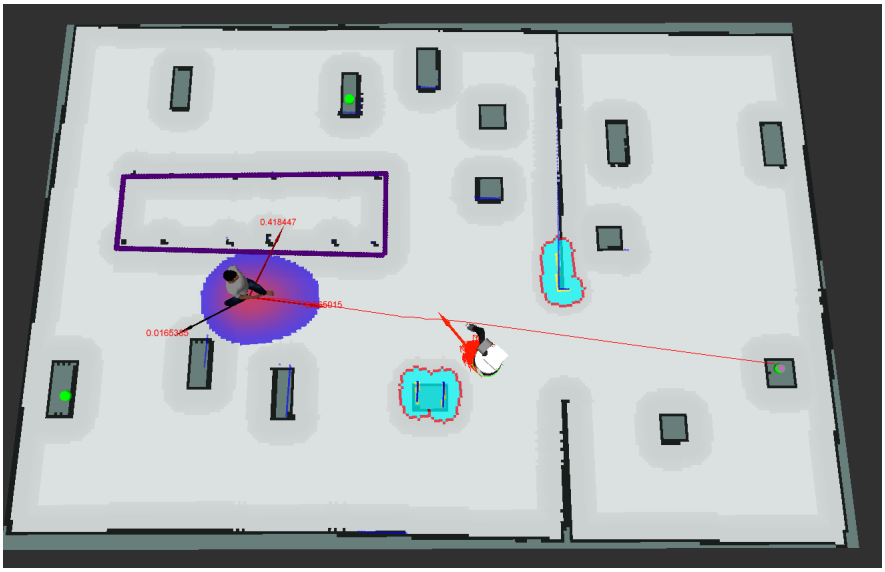
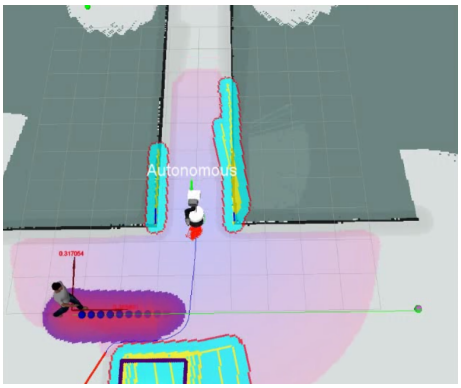


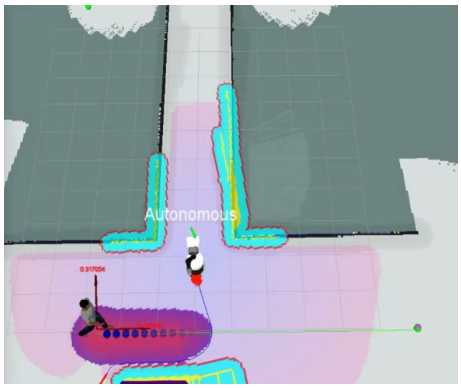
Figure B-11: The human path provided by the `plan_human_path` node. The node takes in the human position, the end destination and the local costmap of the robot. The path has been found using the Dijkstra algorithm. The path is optimal in terms of distance and is assumed that the person simply tends to take the shortest unobstructed path to pursue a desired destination (given by the purple sphere). In the Figure only the proximity layer is added to the costmap.

`human_path_estimation` node The second node of the human planner module is responsible for determining a number of predictions steps along the expected human path, for extending

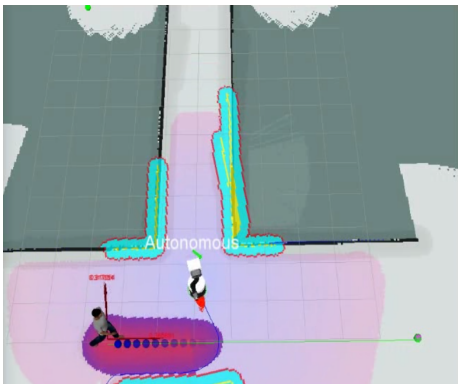
the human intention Gaussian cost model. In the initial implementation the number of prediction steps was a user-preference and was primarily chosen based on the clutter of the environment, since a severe extension of the model could result in blocking feasible paths for the robot and getting itself stuck. However, another limitation of this implementation was the fact that often the robot tended to make confusing maneuvers because it thought that it would be able to pass in front of the human while the cost model had not yet captured the humans' future positions. This erratic behavior of the robot is illustrated in Figure B-12. The robot initially thought that it will manage to pass by the front side of the human, but as the costmap evolved and captured further the front space, the robot regretted its initial choice and performed awkward rotation in place to take the path in the back of the person. This behavior is not regarded as human-friendly since the person might think that the robot will collide with it in the immediate future, watching it to try to pass in front.



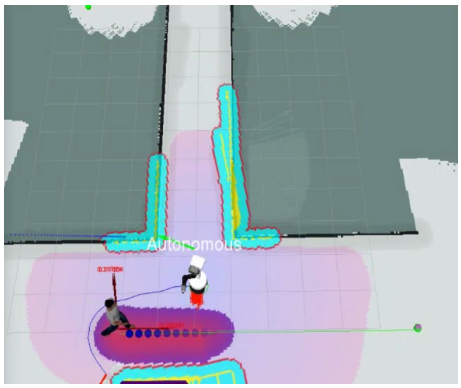
(a) The robot moves to its goal.



(b) It thinks it will be able to pass in front of the person.



(c) It still tries to pass in front.



(d) The costmap captured the available space forcing the robot to change its mind.

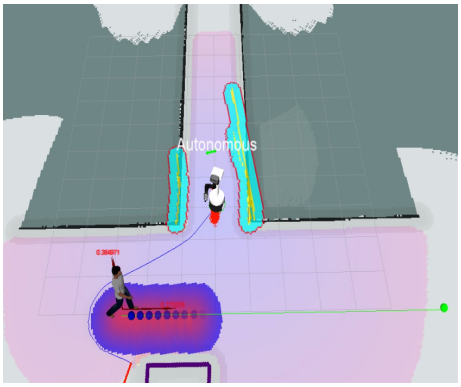
Figure B-12: Erratic behavior of the robot. The robot initially believed that it will manage to pass by the front side of the human, but as the costmap evolved and captured further the front space, the robot regretted its initial choice and performed awkward rotation in place to take the path in the back of the person.

In order to improve this behavior of the robot, the `human_path_estimation` node was modified as to scale the extension of the Gaussian cost model respectively to the relative distance

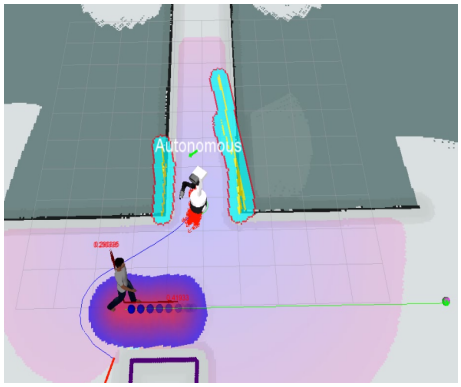
between the robot and the human as well as the velocity of the human. For this purpose the number of prediction steps (blue spheres in Figure B-12), is determined by the formula:

$$\text{number\_of\_steps} = \frac{\text{relative\_distance}}{\text{magnitude} * \text{scale\_factor}}$$

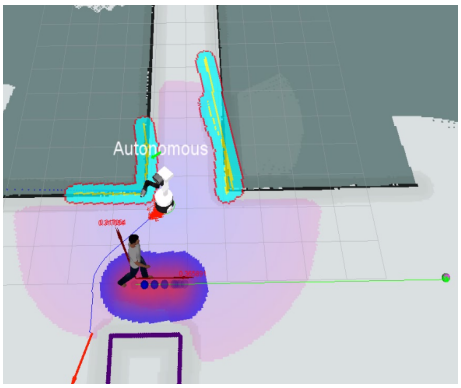
where number_of_steps is the number of prediction steps for the intention model, relative_distance is the relative distance between the human and the robot, magnitude is the magnitude of velocity of the person and scale_factor is a scaling factor which is defined by the user. By using this adaptive formula for selecting the number of prediction steps, we managed to improve the robot behavior as can be seen in Figure B-13. The larger the relative distance is, the more prediction steps are applied to the human intention model. When the robot is close to the human there is no need to extend the costmap a lot in the front side. The formula is effective in many scenarios. Nevertheless, this approach is rather coarse and a more sophisticated selection of the steps is advised.



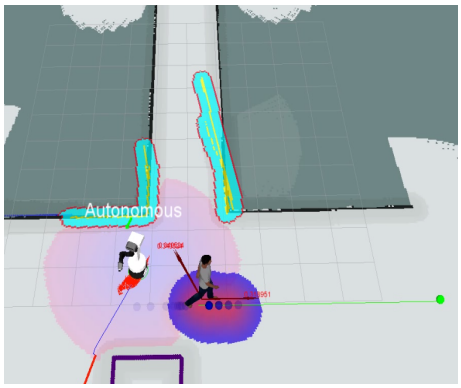
(a) The robot takes an initial decision to pass in the back.



(b) The robot sticks to its decision.



(c) It continues moving in a legible way.



(d) The robot did not interact at all with the human.

Figure B-13: Improvement of the erratic behavior of the robot by using an adaptive formula which scales the number of prediction steps with respect to the relative distance between the robot and the human and the velocity of the human..

In order to account for the relative distance between the human and the robot, apart from the position of the human, necessary information is also the position of the robot. The latter as already stated (Section 2-2-3) is provided by the AMCL localization method which is implemented by the `amcl` package. The output of this package is the estimated pose (and velocity) of the robot which is a message of type `nav_msgs/Odometry`. The position field of this message is inserted in the `human_path_estimation` node and is used in the calculation of the adaptive formula.

B-5 Graph of the proposed framework

In this Section a summary of our implementation method for the accumulation of the proximity and the human intention layers into the layered costmap is made. In order to construct the proximity layer, the only information needed is the position of the human. Then costs are assigned following a combination of two Gaussian distributions which are centered at the human as explained in Section 2-2-7. The position of the human is obtained by the human detection module in the form of `people_msgs/People` messages. The graph of this implementation is shown in Figure B-14.

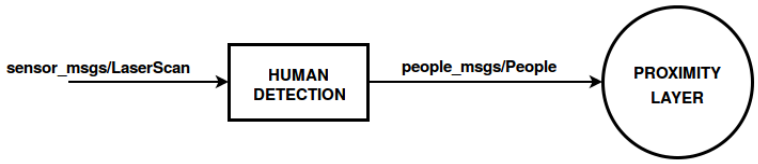


Figure B-14: Graph of the implementation of the proximity layer.

It is important to mention that the Proximity Layer is constructed by a `proxemic_layer` C++ executable file, which apart from implementing the Gaussian model, also performs the `updateBounds` and `updateCosts` methods to update the master (layered) costmap.

The implementation of the human intention layer is rather more complex. First the human intentions need to be computed. The problem is settled as a Bayesian classifier [47] and the probability of a human to pursue a destination is conditioned on his current orientation. The desired probability can be solved as a HMM problem. For this reason the `intention_prediction` node, takes in the position of the human and a set of manually selected destinations and based on the HMM method of [30] outputs the most probable destination that the human is heading to, given the current orientation. The destination has a `geometry_msgs/PointStamped` type.

After having computed the most probable destination, the next task is to find a path for the human to go there. The `plan_human_path` node computes the path considering the human position (`people_msgs/People`) as starting point and the destination (`geometry_msgs/PointStamped`) as end point. The node embraces also the local costmap of the robot and by employing the Dijkstra algorithm, outputs the shortest collision human path as `nav_msgs/Path` message. Consequently, the `human_path_estimation` node assigns the intention Gaussian cost model along the aforementioned path. The number of steps, by which the model is expanded along the path is determined based on the relative distance between the

human and the robot, and the velocity of the human. For this reason the node subscribes to the `people_msgs/People` message for the human and to the `nav_msgs/Odometry` message for the robot. The node publishes the prediction steps in the form of future human locations contained in a `people_msgs/Predictions` message. Finally, the position of the human, the most probable destination and the number of prediction steps are incorporated by a `human_intention_layer` C++ executable, which constructs the cost model and executes the `updateBounds` and `updateCosts` methods. The graph of the implementation of the human intention layer is shown in Figure B-15.

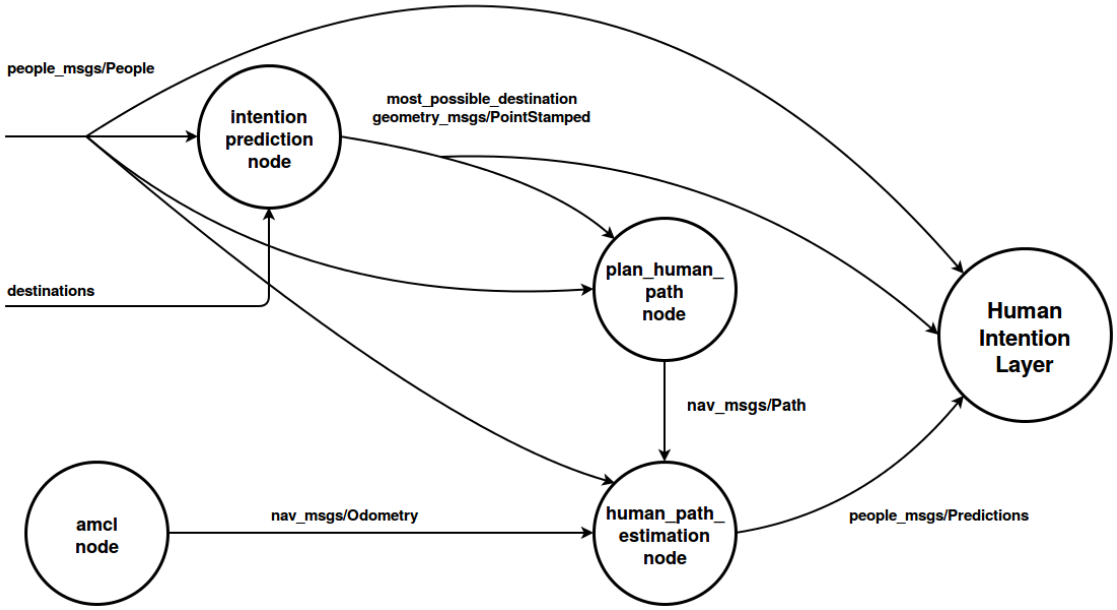


Figure B-15: Graph of the implementation of the human intention layer.

Bibliography

[1] A. de Jong and C. van Duin, "PBL: Regional population and household projections, 2011 - 2040: Marked regional differences," pp. 1–19, 2011.

[2] P. Raja and S. Pugazhenthi, "Optimal path planning of mobile robots: A review," *International Journal of the Physical Sciences*, vol. 7, no. 9, pp. 1314–1320, 2012.

[3] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 03, pp. 463–497, 2015.

[4] T. Thoa Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, 2016.

[5] C. Dondrup and M. Hanheide, "Qualitative Constraints for Human-aware Robot Navigation using Velocity Costmaps*," pp. 586–592, 2016.

[6] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[7] S. M.Lavalle, "Planning Algorithms," p. 842, 2006.

[8] S. G. Tzafestas, *11 - Mobile Robot Path, Motion, and Task Planning*. 2014.

[9] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013.

[10] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From Proxemics Theory to Socially-Aware Navigation: A Survey," *International Journal of Social Robotics*, vol. 7, no. 2, pp. 137–153, 2015.

[11] C. Lichtenthäler and A. Kirsch, "Towards Legible Robot Navigation - How to Increase the Intend Expressiveness of Robot Navigation Behavior," *International Conference on Social Robotics - Workshop Embodied Communication of Goals and Intentions*, no. October 2013, 2013.

[12] C. Lichtenthäler and A. Kirsch, "Legibility of Robot Behavior : A Literature Review," 2016.

[13] E. A. Sisbot, K. F. Marin-Urias, R. Alami, and T. Siméon, "A human aware mobile robot motion planner," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 874–883, 2007.

[14] D. Feil-Seifer and M. Mataric, "Human Robot Interaction," *Encyclopedia of Complexity and Systems Science, Springer*, pp. 4643–4659, 2009.

[15] E. T. Hall, *The Hidden Dimension.* 1966.

[16] M. L. Walters, K. Dautenhahn, K. L. Koay, C. Kaouri, R. T. Boekhorst, C. Nehaniv, I. Werry, and D. Lee, "Close encounters: Spatial distances between people and a robot of mechanistic appearance," *Proceedings of 2005 5th IEEE-RAS International Conference on Humanoid Robots*, vol. 2005, pp. 450–455, 2005.

[17] H. Huettenrauch, K. S. Eklundh, A. Green, and E. A. Topp, "Investigating spatial relationships in human-robot interaction," *IEEE International Conference on Intelligent Robots and Systems*, pp. 5052–5059, 2006.

[18] R. Kirby, R. Simmons, and J. Forlizzi, "COMPANION: A Constraint-Optimizing Method for Person-Acceptable Navigation," *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, pp. 607–612, 2009.

[19] T. Kruse, A. Kirsch, E. A. Sisbot, and R. Alami, "Exploiting human cooperation in human-centered robot navigation," *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, pp. 192–197, 2010.

[20] T. Kruse, P. Basili, S. Glasauer, and A. Kirsch, "Legible robot navigation in the proximity of moving humans," *Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO*, pp. 83–88, 2012.

[21] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered Costmaps for Context-Sensitive Navigation", 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014).

[22] M. Kollmitz, K. Hsiao, J. Gaa, and W. Burgard, "Time dependent planning on a layered social cost map for human-aware robot navigation," *2015 European Conference on Mobile Robots, ECMR 2015 - Proceedings*, 2015.

[23] F. Zanlungo, T. Ikeda, and T. Kanda, "Social force model with explicit collision prediction," *EPL (Europhysics Letters)*, vol. 93, no. 6, p. 68005, 2011.

[24] G. Ferrer, A. Garrell, and A. Sanfeliu, "Social-aware robot navigation in urban environments," *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*, pp. 331–336, 2013.

[25] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning Motion Patterns of People for Compliant Robot Motion," *The International Journal of Robotics Research*, vol. 24, no. 1, pp. 31–48, 2005.

[26] B. Okal and K. O. Arras, "Learning Socially Normative Robot Navigation Behaviors with Bayesian Inverse Reinforcement Learning," pp. 2889–2895, 2016.

[27] D. Carton, A. Turnwald, D. Wollherr, and M. Buss, "Proactively Approaching Pedestrians with an Autonomous Mobile Robot in Urban Environments," *Experimental Robotics, Springer STAR 88*, vol. 88, no. 287513, pp. 199–214, 2013.

[28] J. Rios-Martinez, A. Renzaglia, A. Spalanzani, A. Martinelli, and C. Laugier, "Navigating between people: A stochastic optimization approach," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 231855, pp. 2880–2885, 2012.

[29] C. G. Keller and D. M. Gavrila, "Will the pedestrian cross? A study on pedestrian path prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 494–506, 2014.

[30] A. Nagariya, B. Gopalakrishnan, A. K. Singh, K. Gupta, and K. M. Krishna, "Mobile robot navigation amidst humans with intents and uncertainties: A time scaled collision cone approach," *Proceedings of the IEEE Conference on Decision and Control*, vol. 2016-Febru, pp. 2773–2779, 2016.

[31] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi, "Motion Planning through Symbols and Lattices," pp. 4–9, 2004.

[32] M. Labbe and F. Michaud, "Memory management for real-time appearance based loop closure detection," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1271–1276, 2011.

[33] M. Labbe and F. Michaud, "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation,"

[34] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1271–1278, 2016.

[35] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, "Feature-Based Prediction of Trajectories for Socially Compliant Navigation," *Proceedings of Robotics: Science and Systems*, 2012.

[36] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[37] S. Thrun, "Learning Occupancy Grid Maps With Forward Sensor Models," no. 1998, pp. 1–28, 2003.

[38] D. V. Lu, "Contextualized Robot Navigation," pp. 1–139, 2014.

[39] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 300–307, 2010.

[40] D. Fox, "KLD-Sampling: Adaptive Particle Filters," *Advances in Neural Information Processing Systems 14*, 2001.

[41] K. O. Arras, Ó. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2D range data," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3402–3407, 2007.

[42] M. Munaro, F. Basso, and E. Menegatti, "Tracking people within groups with RGB-D data," pp. 2101–2107, 2012.

[43] M. Munaro and E. Menegatti, "Fast RGB-D people tracking for service robots," *Autonomous Robots*, vol. 37, no. 3, pp. 227–242, 2014.

[44] T. Linder, S. Breuers, B. Leibe, and K. O. Arras, "On Multi-Modal People Tracking from Mobile Platforms in Very Crowded and Dynamic Environments," pp. 5512–5519, 2016.

[45] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 53. 2013.

[46] D. V. Lu, D. B. Allan, and W. D. Smart, "Tuning Cost Functions for Social Navigation," pp. 1–10.

[47] G. Ferrer and A. Sanfeliu, "Bayesian Human Motion Intentionality Prediction in urban environments," *Pattern Recognition Letters*, vol. 44, no. March 2015, pp. 134–140, 2014.

[48] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," 1989.

[49] J. Borenstein and Y. Koren, "The Vector Field Histogram: Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[50] E. Pacchierotti, H. I. Christensen, and P. Jensfelt, "Embodied social interaction for service robots in hallway environments," *Springer Tracts in Advanced Robotics*, vol. 25, pp. 293–304, 2006.

[51] D. V. Lu and W. D. Smart, "Towards more efficient navigation for robots and humans," *IEEE International Conference on Intelligent Robots and Systems*, no. c, pp. 1707–1713, 2013.

[52] Y. Tamura, T. Fukuzawa, and H. Asama, "Smooth collision avoidance in human-robot coexisting environment," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 3887–3892, 2010.

[53] C. Lichtenthäler, T. Lorenz, M. Karg, and A. Kirsch, "Increasing perceived value between human and robots - Measuring legibility in human aware navigation," *Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO*, pp. 89–94, 2012.

[54] P. Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using Velocity Obstacles Abstract," no. 1986, pp. 760–772, 1995.

[55] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," *Springer Tracts in Advanced Robotics*, vol. 70, no. STAR, pp. 3–19, 2011.

[56] J. M. O'Kane, *A gentle introduction to ROS*. 2013.

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

[58] B. P. Gerkey and K. Konolige, "Planning and Control in Unstructured Terrain," *ICRA Workshop on Path Planning on Costmaps*, 2008.

Glossary

List of Acronyms

ADLs	Activities of Daily Living
HiT	Heemskerk Innovative Technology
SACRO	Semi Autonomous Care Robot
HAN	Human-Aware Navigation
HRI	Human-Robot Interaction
HMM	Hidden Markov Models
SLAM	Simultaneous Localization and Mapping
DOF	Degrees of Freedom
AMCL	Adaptive Monte Carlo Localization
VFH	Vector Field Histogram
ROS	Robot Operating System
OSRF	Open Source Robotic Foundation
OS	Operating System
VCS	Version Control System
YAML	Ain't Markup Language
IMU	Inertial Measurement Unit
GUI	Graphical User Interface
HRI	Human-Robot Interaction
BHMIP	Bayesian Human Motion Intentionality Prediction

PDF Posterior Density Function

DWA Dynamic Window Approach

List of Symbols

$\alpha_{ij}(t)$	Transition probability
$\beta_t(i)$	The backward parameter of a HMM
η	Normalization factor
$\gamma_t(i)$	Desired HMM probability
λ	Set of HMM parameters
$\mathbf{X}_n$	The n observed trajectory of a human
$\mathbf{x}_n(t)$	The detection at time t of the human along the trajectory n
$\mathcal{N}(u; 0, \sigma_u)$	Gaussian distribution of zero mean and covariance σ
$\phi_{ij}(t)$	Angle between two destinations d_i and d_j
σ	Design parameter for covariance
σ_x	Covariance of Gaussian function in the x direction
σ_y	Covariance of Gaussian function in the y direction
magnitude	Magnitude of the velocity of a human
number_of_steps	Number of prediction steps of the human intention cost model
relative_distance	Relative distance between the human and the robot
scale_factor	Scaling factor for the velocity
`color_avg`	8-bit value
`num_of_steps`	Number of prediction steps
`occ_value`	Occupancy value of a cell
θ	Orientation of the Gaussian
$\theta_{1:T}$	Set of observations up to time T
θ_{sol}	Bisector of the solution sector s_{sol}, direction solution of the VFH method
A	Amplitude of Gaussian function
$a_t(i)$	The forward parameter of a HMM
D	Set of probable destinations
d_i	A destination in the environment
$d_{rel}(t)$	The relative distance over time
f	Factor for scaling the intention costmap
f_{Gauss_i}	Gaussian function at step i
f_{Gauss}	Gaussian function
$factor$	Design parameter for velocity scaling
fc	Scaling factor
h	A human in the environment
h	Obstacle polar density
j_{ang}	Angular jerk
j_{lin}	Linear jerk
L	Total path length
m	Number of destinations

P_{Int_i}	Human intention to pursue destination d_i
s	Sector of polar histogram
s_{sol}	Solution sector of the VFH method
T	Set of human detections
t_0	Starting time of the navigation run
t_f	Time that the robot reaches at the target position
t_{task}	Total execution time of a navigation task
u_{ni}	Angle between the current orientation of the human and destination d_i
v_{sol}	Velocity solution of the VFH method
x	The x coordinate
x_0	The x coordinate of center of Gaussian
x_r	The x position of the robot
y	The y coordinate
y_0	The y coordinate of center of Gaussian
y_r	The y position of the robot
Z_{int}	Intimate zone of the human