Multimodal Navigation for Autonomous Service Robots

Rui Fernando Faustino Bettencourt

Thesis to obtain the Master of Science Degree in
Electrical and Computer Engineering

Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima
Member of the Committee: Prof. Alexandre José Malheiro Bernardino

November 2019
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.
Acknowledgments

First of all, I would like to thank my family for always supporting me in my academic and personal life. A special thanks to my parents and sister that were always there for me when I needed them.

I would also like to thank my supervisor, professor Pedro Lima, who introduced me to the robotics world and allowed me to work in the SocRob@Home group for my thesis and to gain experience in this area. I would also like to thank the team members who helped me either by discussing ideas or fixing problems in my code, in particular Enrico Piazza. I am also grateful for Rute Luz and Carlos Azevedo’s feedback on my work as well as João Pereira’s help in some experiments.

Finally, a special thanks to my friends who are not mentioned above for the never ending support and good times.
Abstract

Service robots for indoor environments is a very promising field, since it may solve many of the problems originated by the aging demographics in developed countries. These robots can assist people with mental or physical limitations or help anyone in their homes or workspaces, allowing them to use their time more productively. Thus, service robots need to be able to navigate in such environments, which might include narrow passages, and at the same time avoid, seamlessly, any static or moving obstacles. Moreover, the robot should be able to localize itself at all times to correctly perform its tasks. Since it is not possible to guarantee this, a safety measure to recover from the kidnapped robot problem should be implemented. In this thesis, a method is proposed to solve this challenge using state of the art algorithms with additional implemented components to create a fully operational navigation stack. To enable non-static goals, we introduced a dynamic goal component that follows a moving goal to its exact pose or to try and keep a certain distance from it. This dynamic goal can be an object, a person or any element that can be tracked over time and produce a pose estimation to be followed. The proposed method was implemented inside the framework SocRob@Home with the MBot robot. Using the robot in experiments at the ISRooBoNet@Home testbed and in the competition ERL Smart Cities, this thesis presents how the developed navigation stack achieves both the different navigation modes and the localization recovery.

Keywords

Navigation; Localization; Service Robots; Indoor spaces; Guiding; People Following; Waypoint Navigation; Dynamic Goal.
Resumo

Robôs de serviço para espaços interiores é atualmente um tema muito interessante. Estes robôs prestam assistência às pessoas com limitações físicas ou mentais e ajudam os utilizadores humanos em espaços domésticos ou profissionais, disponibilizando tempo para atividades mais importantes. Os robôs de serviço necessitam de se deslocar em espaços, por vezes estreitos, sem irem de encontro ao mobiliário ou a obstáculos em movimento. Além disso, os robôs têm de se manter permanentemente localizados para desempenharem corretamente as tarefas. Como é impossível garantir isto, é preciso que haja uma medida de segurança para que o robô recupere do problema de rapto do robô.

O método proposto recorre a algoritmos de vanguarda e pacotes com componentes implementados adicionalmente para criar uma stack de navegação completamente operacional. Para ativar os alvos dinâmicos é apresentada uma componente que segue um alvo em movimento até à sua pose exata ou que tenta manter-se a uma determinada distância. Este alvo dinâmico pode ser um objeto, uma pessoa ou algo cujo movimento é monitorizado pelo robô ao longo do tempo e gera uma estimativa de pose para ser seguida. O método proposto foi implementado no robô MBot na estrutura do SocRob@Home. Foram realizados testes no laboratório da universidade e na competição europeia ERL Smart Cities. A incerteza da localização, a robustez da navegação e o funcionamento do alvo dinâmico foram testes importantes para verificar o comportamento correto da stack de navegação, que realiza com sucesso os diferentes modos de navegação e a recuperação da localização.

Palavras Chave

Navegação; Localização; Robôs de Serviço; Espaços Interiores; Guiar Pessoas; Seguimento de Pessoas; Navegação por Pontos predefinidos; Destino dinâmico.
## Contents

1 Introduction .............................................. 2  
  1.1 Motivation ........................................ 3  
  1.2 Objectives ......................................... 4  
  1.3 Previous Work ..................................... 4  
  1.4 Contribution ....................................... 4  
  1.5 Outline of the Thesis ............................... 5  

2 State of the Art ....................................... 6  
  2.1 Localization ...................................... 7  
  2.2 Motion Planning ................................... 9  
  2.3 Guidance ......................................... 10  
  2.4 People Following ................................... 10  

3 Theoretical Background ............................... 12  
  3.1 Covariance and Variance ......................... 13  
  3.2 Bayes Filter ...................................... 14  
  3.3 Monte Carlo Localization ......................... 15  
  3.4 Augmented Monte Carlo Localization .......... 16  
  3.5 Adaptive Monte Carlo Localization ........... 19  
  3.6 Dynamic Window Approach ....................... 19  
  3.7 Dijkstra’s Algorithm ............................. 21  

4 Multimodal Navigation ............................... 23  
  4.1 ROS Components .................................. 24  
  4.2 Coordinate Frames ................................ 25  
  4.3 Localization ...................................... 25  
  4.4 Navigation ........................................ 28  
    4.4.1 Target Goal .................................. 29  
    4.4.2 Global Planner ............................... 32  
    4.4.3 Local Planner ............................... 33
List of Figures

1.1 Diagram of the overall behaviour of the navigation stack proposed. .......................... 5

2.1 Diagram of the navigation loop. ................................................................................... 7

3.1 Diagram of the overall behaviour of the several components of the proposed navigation. 13

4.1 Base Transform Library (tf) tree of the implemented navigation method. ................... 25
4.2 Robot’s reference axis ..................................................................................................... 27
4.3 Navigation’s architecture Diagram ................................................................................ 28
4.4 Selection of the optimal position ................................................................................... 31
4.5 Comparison between the two mentioned shortest path planners. .............................. 33
4.6 A person’s personal space ............................................................................................. 34
4.7 Guiding diagram ........................................................................................................... 35

5.1 Images from ERL Smart Cities episode 3. .................................................................. 38
5.2 Variances of $x$, $y$ and $yaw$ over time in a SciRoc run of episode 3 ......................... 39
5.3 Variances of $x$, $y$ and $yaw$ over time in ISR’s testbed for the kidnap recovery experiment with $\alpha_{\text{slow}} = 0.001$. ................................................................. 40
5.4 Variances of $x$, $y$ and $yaw$ over time in ISR’s testbed for the kidnap recovery experiment with $\alpha_{\text{slow}} = 0.01$. ................................................................. 41
5.5 Images of relocalization ................................................................................................. 42
5.6 Localization error from the robot’s estimated pose and the ground truth from Motion Capture System (MCS) ................................................................................. 43
5.7 Localizations over time of the ground truth (blue) and the robot’s estimated pose (red). 44
5.8 ERL Smart Cities’ episode 3 map used in the challenge .............................................. 44
5.9 Distance from the perceived robot location to the desired waypoint in meters at the ERL Smart Cities challenge ............................................................................ 45
5.10 University testing facilities for indoor robots .............................................................. 46
5.11 Distance from the perceived robot location to the desired waypoint in meters at the Institute for Systems and Robotics (ISR) testbed .......................................................... 47
5.12 Images of the path replanning for the obstacle avoidance experiment when the path created is impossible to follow and a new path needs to be generated ........................................ 48
5.13 Images of the path replanning for the obstacle avoidance experiment when there is room for the robot to go around the obstacle ................................................................. 49
5.14 Complete map used for the people following experiment ............................................. 50
5.15 Plot that depicts the distance to the followed person over time ................................... 50
5.16 Selection of goal for people following ............................................................................ 51
5.17 Plot that depicts the distance to the person to be guided and the distance from the robot to the goal at each time instant ................................................................. 53
5.18 Images that show the guiding mode execution ............................................................. 54
A.1 Behaviour of dynamic goal following a dynamic goal at 1,2 meters ............................... 62

List of Tables

4.1 Dynamic Goal message type ......................................................................................... 29
4.2 DWA Configurations ................................................................................................. 33
5.1 Average values of variance and standard deviation ...................................................... 39

List of Algorithms

3.1 Bayes filter Algorithm ................................................................................................. 15
Listings

4.1  Message to activate people following .................................................. 34
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>amcl</td>
<td>Adaptive Monte Carlo Localization</td>
</tr>
<tr>
<td>MCL</td>
<td>Monte Carlo Localization</td>
</tr>
<tr>
<td>KLD</td>
<td>Kullback–Leibler divergence</td>
</tr>
<tr>
<td>RGB-D</td>
<td>RGB and Depth</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman Filter</td>
</tr>
<tr>
<td>RDT</td>
<td>Rapidly Exploring Dense Trees</td>
</tr>
<tr>
<td>RRT</td>
<td>Rapidly Exploring Random Trees</td>
</tr>
<tr>
<td>DWA</td>
<td>Dynamic Window Approach</td>
</tr>
<tr>
<td>EBAND</td>
<td>Elastic Band method</td>
</tr>
<tr>
<td>TEB</td>
<td>Timed Elastic Band</td>
</tr>
<tr>
<td>hGPLVM</td>
<td>hierarchical Gaussian process latent variable model</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Derivative</td>
</tr>
<tr>
<td>tf</td>
<td>Transform Library</td>
</tr>
<tr>
<td>ERL</td>
<td>European Robotics League</td>
</tr>
<tr>
<td>ISR</td>
<td>Institute for Systems and Robotics</td>
</tr>
<tr>
<td>MCS</td>
<td>Motion Capture System</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Motivation ......................................................... 3
1.2 Objectives ......................................................... 4
1.3 Previous Work ..................................................... 4
1.4 Contribution ......................................................... 4
1.5 Outline of the Thesis .............................................. 5
1.1 Motivation

According to [1], a service robot is “a robot which operates semi- or fully-autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations”. Service robots have been studied and developed for over thirty years [1] and one of the most important components of every robot with mobility is its navigation, since “a robot accomplishes tasks by moving in the real world” [2]. Navigation can be implemented both in a known room with a predefined map or in a more challenging navigation mode where the robot needs to adapt to its surroundings. These challenging modes can include human interaction as is the case of guiding and people following or not as is the case of waypoint navigation. These modes and every component that allows a robot to navigate in an environment make up the navigation stack. The navigation can be referred to as multimodal if it has different operational modes, that differ from each other in the objective of the navigation. In the modes given as example, guiding mode differs from people following since the objective of it is to go to a static pose while making sure a person is tracked within a given range while the objective of people following is to be or trying to be at a certain distance from a person. And these modes differ from waypoint navigation since this mode just tries to get to a static pose.

All mobile robots require a good navigation stack to have a correct behaviour. The case study for this thesis is ISR’s MBot1 used by the team SocRob@Home [3] where a robust, efficient and intelligent navigation stack is a crucial component so that the robot is able to perform several indoor tasks with the goal of helping people in their daily lives. The results of this project are of great importance in competitions such as the European Robotics League (ERL) Consumer Service and Smart Cities Robot Challenges, or RoboCup@Home [4] where several students can implement their research in specific challenges. Most of the navigation modes described in this thesis were implemented in the robot during ERL Smart Cities competition, as it is shown in the results of Chapter 5.

The challenges that can be performed by a service robot can vary between domestic and professional environments where the robot can help in several areas. A service robot can, for example, assist in an industrial space or in other customer service roles where it can retrieve components and help reduce the time spent on these tasks by the clients and human staff [5]. The robot can also assist in the health sector by helping and entertaining sick children with specific needs by interacting with them in a safe and fun way [6].

All these examples are tasks that are evaluated in competitions nowadays [5] so that research can be made in order to have real contributions to the future in the mentioned areas. With each new competition, new knowledge arises which can make an impact in the future of service robots for the benefit of every user.

1from the FP7 MOnarCH robot
1.2 Objectives

Although most of the components used for navigation have been well established for some years now, there are still some improvements to be made and integrating all the different components for a good navigation is still a challenge. For a complete navigation stack, the following modes need to exist:

- **Waypoint navigation**, where the robot receives a static goal and has to plan the clearest and shortest path to this goal in a known map and navigate to it while avoiding obstacles.

- **Dynamic goal navigation**, where a goal that can change pose is sent to the navigation stack instead of a static one. This goal can be a simple pose in the map that can change its position and/or orientation in each time instant, e.g. the position of a tracked object or the position of a tracked person to follow.

- **Guiding navigation**, where there is a static or dynamic goal that the robot must reach whilst making sure that a certain tracked person is following the robot.

These different modes include not only components of navigation and localization, but also perception to detect and track an object or person. The final objective is to implement all these modes in a robot with existing state of the art components and to add necessary implementations to improve these components and accomplish all the different modes of navigation.

1.3 Previous Work

This project was developed on top of an already existing navigation stack that was implemented by previous members of the SocRob@Home team [3]. This navigation stack is integrated with the *Robot Operating System (ROS)* Middleware and takes advantage of some packages, such as the *move-base* package [7] and the *Adaptive Monte Carlo Localization (amcl)* package [8]. The packages were adapted by the members of SocRob@Home and configured to be used in the robots available. The people following mode implemented by the team uses a different navigation package developed which had some problems avoiding obstacles and walls while following the position estimation of a person. The people detector and tracker used in some of the different navigation modes that required perception were implemented by members of the SocRob@Home team.

1.4 Contribution

The main contribution of this thesis is a complete navigation stack with the modes mentioned in the objectives, with special focus on the *dynamic goal* package in order to follow a moving target pose.
These modes are represented in Fig. 1.1 by the Mode block that whether if it is waypoint navigation, guiding, people following or other dynamic goal configuration will choose the correct Goal format. This Goal in combination with the Human Awareness (if necessary for the selected mode), the Localization, the Map and the Sensor readings will control the Navigation of the robot.

The guiding mode was implemented so that there are guarantees that the tracked person was actually following the robot since the previous guiding robot was only a waypoint navigation. Besides these two main points, the Adaptive Monte Carlo Localization (MCL) parameters in the Localization needed in order to react to kidnapping situations were also studied and adjusted to the challenge in question so as to recover from wrong localizations.

![Figure 1.1: Diagram of the overall behaviour of the navigation stack proposed.](image)

### 1.5 Outline of the Thesis

This thesis outline is as follows:

- In Chapter 2, the state of the art in navigation is presented.
- In Chapter 3, some concepts necessary to the understanding of the final Chapters are explained such as covariance and variance, Bayes filter, MCL, Dynamic Window Approach (DWA) and Dijkstra's algorithm.
- In Chapter 4, the architecture of the solution implemented to achieve the objectives mentioned in Section 1.2 is described.
- In Chapter 5, the results of this implementation are presented through a series of tests performed either in the ERL Smart Cities competition or at ISRoboNet@Home tesbed for service robots.
- In Chapter 6, the results of this thesis are analysed, a general conclusion of all the work is presented and some future work is proposed.

2
State of the Art

Contents

2.1 Localization ................................................................. 7
2.2 Motion Planning ........................................................... 9
2.3 Guidance ................................................................. 10
2.4 People Following ......................................................... 10
In the following sections of this chapter, we present the state of the art in localization, motion planning, guidance and in people following since these are the four main components required for the objectives set in section 1.2. In Fig. 2.1, these components (with the exception of people following) and their connections in a navigation loop are depicted. An important problem in localization is the kidnapped robot problem, that occurs whenever the robot assumes a wrong localization as its localization in the given map. This problem will be studied in this chapter and in the next ones.

![Figure 2.1: Diagram of the navigation loop [reprinted from [9]].](image)

### 2.1 Localization

Having a properly localized robot is vital for a correct navigation. If the robot is not correctly localized, any plan made to reach the goal is wrong. Hence, this component has been thoroughly studied over the years and several localization methods have been proposed according to the type of vehicle and the intended function of the task it has to perform. In Fig. 2.1, this importance can be seen since the *Localization* block supplies the *Guidance* block the necessary pose estimate so that it can create a trajectory for the *Joint Controller*. *Localization* also supplies the operation point that *Joint Controller* works on.

Localization has two components, a global localization and a local position tracking. The global localization is performed when the robot does not know its initial pose and needs to determine it whereas local position tracking is the component that updates the localization as the robot moves in the map over time. Besides these two components, the kidnapped robot problem also exists as a version of the global localization with an additional difficulty of having to determine when the robot is or is not well localized and recover the correct localization if it is wrongly localized [10].

To study an uncertain environment, “the most general algorithm for calculating beliefs is given by the Bayes Filter” [10]. This algorithm is recursive which means that it depends on the beliefs of the previous time instants to determine the belief at its current time. With an accurate map available some probabilistic localization algorithms derived from Bayes filter can be used.
The most "straightforward application of Bayes filter to the localization problem is called Markov localization" [10]. This method is an algorithm that keeps a probabilistic distribution for the robot pose instead of a single hypothesis [11]. As written in [10], this method is able to address the global localization, the local position tracking and the kidnapped robot problem in static environments. A static environment is such that only the robot actions will affect the agent sensor readings, meaning that the method can fail if a dynamic object or a person enters the same environment. This method fails to be efficient as its computational resources grow with the size of the map [10].

A well studied case of a Bayes filter is Kalman Filter [12], useful for linear systems. It uses the mean $\mu_t$ and the covariance $\Sigma_t$ to represent the belief and is not applicable for discrete or hybrid states, working only for continuous states [10]. To extend the Kalman filter to be used in nonlinear problems, the Extended Kalman Filter (EKF) was introduced [10]. By linearizing with Taylor expansions, the model becomes an approximation instead of an exact value, maintaining a gaussian as the representation of the belief.

According to [13], EKF used for localization is a special case of the Markov localization where the measurement model has been linearized with Gaussian noise [10]. It can be used to increase the efficiency and successfully keep track of a robot's localization. Despite its accurate localization and efficiency, this method is mostly used in feature-based maps and is better suited for local localization with low uncertainty and in environments with distinct features, becoming "less applicable to global localization or in environments where most objects look alike" [10].

As it is presented in [14], another method and arguably the most popular, MCL is a sample-based representation and it is able to represent multi-modal distributions, it is able to be used in both global and local localization as opposed to EKF and it is more efficient than Markov localization since its computational cost does not depend on the size of the map, but on the number of particles. Although in its original version it could not recover from the kidnapped robot problem, a new version was introduced that can recover from this problem by adding random particles to the particle set when the localization accuracy is low. This new version was named Augmented MCL. To optimize the method, the sampling can be done in an adaptive manner [15] that changes the number of used particles depending on how spread out the particles are. To determine the number of particles, the method uses the Kullback–Leibler divergence (KLD) error estimate between particles.

The previously proposed probabilistic localization methods can use different kinds of sensors to determine a robot's localization. It can use odometry, sonar readings, laser readings, depth-camera data [16], WiFi energy [17] or any other sensor measurement that can be used to determine a robot's pose. These methods can also use the different sensor readings combined to improve their results.
2.2 Motion Planning

The motion planner is the part of the navigation that given an origin and target poses will give a path of a continuous sequence of poses that connects both taking into account the global costmap. In Fig. 2.1, the integration in the navigation loop of this component is shown, where it supplies a path to a certain goal to the Guidance block to follow. A costmap is an occupancy grid map divided into small squared cells of a certain size that gives each cell a probability value for that cell to be occupied. This value is a number between 0 and 100, where 0 means free and 100 occupied with highest confidence.

Several path planning approaches are used for motion planning such as:

- Roadmap, that obtains a graph from the space that will not make a robot hit an obstacle and through Voronoi diagrams or Visibility graphs creates a roadmap to be used by the motion planner [2].

- Cell decomposition, where the map is divided into simple regions called cells that then can produce a graph between each cell, where a shortest path planner can calculate for the graph the shortest path from two cells [2].

- Sampling-Based Planning, that incrementally builds a tree from a dense sequence of samples referred to as Rapidly Exploring Dense Trees (RDT). If the sequence of samples is random then the resulting tree is a Rapidly Exploring Random Trees (RRT) (biased to grow towards empty spaces), minimizing the overall path length. These trees grow until the goal is reached [18].

- Potential field, where the robot is treated like an electrically charged particle acting under the influence of a potential field. The robot has a force that attracts the robot to the goal and forces that push it away from obstacles [2]. This method has a problem of oscillations in narrow passages and it can get stuck in a local minima [9].

- Bug Algorithms, that assumes that the robot has a finite number of obstacles with specific boundaries. The robot also needs the Euclidean distance to the goal and the direction towards it as well as a sensor that provides the exact shape of the boundary of the obstacles within a small distance to the robot. With these premises, the robot can move towards the goal in a defined straight line $\gamma$ and if an obstacle is encountered, the robot goes around it until it reaches $\gamma$ again, resuming the navigation in this direction. The process stops when the robot reaches the goal [18]. Other option is to go again in the straight line $\gamma$ towards the goal, as the previous solution but instead of going around just until the intersection with $\gamma$, but until it navigated the whole perimeter of the obstacle. Then, it chooses the closest point in the perimeter to the goal, navigates towards it again around the obstacle and navigates towards the goal again repeating the process in every obstacle in the way [19].
Considering the use of cell decomposition into cells with the same size, the cells are considered as nodes in a graph and a shortest path planner can be applied to determine the shortest path from any point to another. The two main algorithms used for this problem are Dijkstra’s algorithm and A* algorithm [20]. Analysing both algorithms like it was done in [21], Dijkstra’s algorithm combined with gradient descent method allows the smoothest path. A* is the most efficient method but might not result in the optimal path in terms of smoothness, since this method does not look into every nearby map cell that is used in gradient descent method to get the optimal smoothest path.

2.3 Guidance

After the robot chooses a path towards its goal, it needs a guidance algorithm to take into account the robot surroundings and choose the best trajectory to follow the path as best as it can while avoiding obstacles. This is possible to observe in Fig. 2.1, where the Guidance block takes the path from the Motion Planner, the pose estimate from Localization and the detected obstacles and creates a trajectory (e.g., wheel velocities) for the Joint Controller to apply. Like it is mentioned in [21], some state of the art guidance algorithms are DWA, Elastic Band method (EBAND) and Timed Elastic Band (TEB). As it is shown in [22], DWA determines a velocity command from a determined set of velocities that can be reached by the robot according to its dynamic and kinematic constraints.

The guidance algorithm EBAND, that can be found in [23], supplies a trajectory by creating a virtual band which can be optimized with the use of artificial forces, contraction and stretching. Contraction is used to smooth and shorten it and stretching is useful when an obstacle is in the way. The algorithm can then use these properties for obstacle avoidance instead of the costly replan operation with the motion planner every time an obstacle is detected. A different elastic band method that was studied to be used as guidance algorithm is TEB. The article [24] explains that this algorithm is very similar to the previous EBAND, improved with the addition of temporal information. This allows the robot to take into account the robot’s dynamic constraints when applying the artificial forces to the path.

2.4 People Following

The topic of people detection and tracking has been studied for several years and there are different approaches to handle these cases. In [25], a people detector and tracker is proposed by using the data of a RGB camera. This method uses the position and articulation of the members to differentiate people by using a hierarchical Gaussian process latent variable model (hGPLVM) to model these individual limb dynamics. A Hidden Markov Model (HMM) is then used to extend the person detection to a people tracking over time. In [26] and [27], RGB and Depth (RGB-D) is used for the people tracking and
In [27], the authors focus on the tracking of a single operator and on the re-identification of this person. This is done so that people following can always track the same person, even in cluttered spaces and even after losing the person for a while. This is achieved by using a Bernoulli filter [28] and a Probability Hypothesis Density filter [29] combined. In order to make robot navigation more human aware, the article [30] proposes the merge of laser scans and RGB-D cameras in order to detect humans, fusing the several detectors with a Kalman filter.

In terms of people following, the integration of the tracker supplied by people tracking and navigation is a less studied topic. In [31], a people follower is presented but the navigation used by this proposed method is made specifically for this task. The method allows for following using distances defined by the notion of Proxemics [32], but fails to back away to maintain its distance if the person gets closer to the robot than it should. This method also uses voice interaction to interact with the person to be followed in order to provide feedback.

The method proposed in [33] can track and follow a person with a navigation stack built specifically using a Proportional Integral Derivative (PID) to calculate the angular velocity and one PID to calculate the linear velocity. The method is based on the comparison between two vectors, one from the robot center to the tracked person and one from the robot to the desired goal position from the robot to the person. The angular and linear velocity commands have the goal of equalizing the length of the vectors and bring the angle between them to zero.

Another possible way to follow a person is to track the person and aim to follow its tracked path, maintaining a distance to the person. In [34], a method uses this approach in one of its two different follower controllers that are presented. Firstly, a similar navigation objective to the method presented in [33] is used but a second controller is added to follow the tracked path, taking into account the distance to the person and the robot's kinematics.
3

Theoretical Background

Contents

3.1 Covariance and Variance ........................................... 13
3.2 Bayes Filter ......................................................... 14
3.3 Monte Carlo Localization ......................................... 15
3.4 Augmented Monte Carlo Localization ......................... 16
3.5 Adaptive Monte Carlo Localization ......................... 19
3.6 Dynamic Window Approach .................................... 19
3.7 Dijkstra’s Algorithm .............................................. 21
In this chapter, the theoretical background necessary to understand the approach suggested is presented. The Bayes filter, the Monte Carlo Localization (MCL) algorithm, the Augmented MCL algorithm, the Adaptive MCL algorithm, the Dynamic Window Approach (DWA) and the Dijkstra’s algorithm are presented in this chapter. A theoretical background on the navigation structure and on uncertainty measures is also supplied since they are required for the correct understanding of the Chapters 4 and 5.

Figure 3.1: Diagram of the overall behaviour of the several components of the proposed navigation.

In Fig. 3.1, the different components required for navigation are represented, combining the diagrams of Fig. 1.1 and Fig. 2.1. It needs a localization component, a map and sensor reading to navigate correctly. The navigation needs two main components, the motion planner and the guidance component to reach a certain navigation goal with the shortest distance and without hitting any obstacle. The mode of navigation is chosen from the four modes presented in the diagram that will automatically define which type of goal is required and if Human Awareness is needed. It is also possible to see that the resulting velocity command from the navigation block (specifically from Guidance) influences the localization since the vehicle moves with the command, that the sensors can modify the map and the map is needed for localization. Depending on whether the goal is dynamic or not, it may need the robot’s localization and the human awareness to function correctly, if the mode requires a dynamic goal.

3.1 Covariance and Variance

If a measurement unit for how much a single variable varies (or measurement uncertainty) is needed, variance can be used [10]. Another possible measurement unit is the standard deviation, that is the square root of the variance. A variance is defined as in Equation (3.1), where \( E[x] \) is the expectation of a discrete-space variable \( x \) expressed as in Equation (3.2), in discrete time. For every discrete-space variable \( x \), there is a specific event \( x_1 \) that \( x \) might result on, that has a probability \( p(x_1) \) of becoming true.
\[ \text{Var}(x) = \sigma(x, x) = E[(x - E[x])^2] = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2 \] (3.1)

\[ E[x] = \sum_{x_1} x_1 p(x_1) \] (3.2)

To measure how much two variables influence changes in one another, covariance is used. The covariance of two variables \( x \) and \( y \) is given by

\[ \sigma(x, y) = E[(x - E[x])(y - E[y])]. \] (3.3)

To represent all the variances and covariances, a covariance matrix is built. This matrix is symmetric and positive-semidefinite, with the variances of the variables in the diagonal and the covariances between the different variables in the other positions of the matrix. A covariance matrix for two variables \( x \) and \( y \) can be seen in Equation (3.4).

\[ \Sigma = \begin{bmatrix} \sigma(x, x) & \sigma(x, y) \\ \sigma(y, x) & \sigma(y, y) \end{bmatrix} \] (3.4)

### 3.2 Bayes Filter

A robot uses the sensors available to it to get information of its environment necessary for the robot to get knowledge of its surroundings. And since all sensors have an associated error in all measurements, what the robot can get is not the true state of its environment but an approximate calculation of its state from the data it has available.

To represent this approximate calculation, the concept of belief is introduced. Belief is used to have a probabilistic representation of the environment’s state taking into account the sensors measurements and the control commands up until the present time, since the algorithms cannot know directly the true state of the robot. A belief \( \text{bel}(x_t) \) over a state variable \( x_t \) at time \( t \) conditioned by all the past measurements \( z_{1:t} \) and all past control commands \( u_{1:t} \) is given by

\[ \text{bel}(x_t) = p(x_t \mid z_{1:t}, u_{1:t}). \] (3.5)

To calculate beliefs, a very general algorithm is given by the Bayes filter that calculates the belief distribution from the measurements and the control data. A single step of this algorithm is shown in Algorithm 3.1. This algorithm is applied recursively since to get the belief at time \( t \), the algorithm uses the belief of the previous time instant \( (t - 1) \). Besides this previous belief, the algorithm also needs the most recent control \( u_t \) and the most recent measurement \( z_t \).
Algorithm 3.1: Bayes filter Algorithm [10]

1 Algorithm Bayes filter \( \text{bel}(x_{t-1}), u_t, z_t) \):

2 for all \( x_t \) do

3 \( \text{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}) \text{bel}(x_{t-1}) \) \( dx \)

4 \( \text{bel}(x_t) = \eta \ p(z_t \mid x_t) \text{bel}(x_t) \)

5 endfor

6 return \( \text{bel}(x_t) \)

This algorithm has two main components: the prediction and the measurement update steps. The prediction step can be seen in line 3 of Algorithm 3.1, where the control \( u_t \) is processed. It calculates a belief \( \text{bel}(x_t) \) by the integral of the product between the belief in the previous time instant \( t-1 \) and the probability that \( u_t \) will induce a transition from \( x_{t-1} \) to \( x_t \). The measurement update is described in line 4 where the previous calculated belief from the prediction step \( \text{bel}(x_t) \) is multiplied by the probability of the measurement \( z_t \) to have been observed. This term is also multiplied by a constant \( \eta \) to normalize the result, that might not integrate to one. The algorithm repeats these steps for every possible posterior state \( x_t \) and returns the belief \( \text{bel}(x_t) \).

To begin the recursive application of the algorithm, an initial belief \( \text{bel}(x_0) \) is needed. If this belief is known, it should be initialized with a point mass distribution with the probability mass centered around the true value of \( x_0 \) and a probability 0 everywhere else. If the initial belief is not known, it should be initialized as a uniform distribution over the domain of \( x_0 \). If this initial belief is only partially known, it can be initialized with a non-uniform distribution.

It is also important to note that the algorithm makes a Markov assumption that the state is a complete summary of the past, implying that the belief is enough to represent the past history.

3.3 Monte Carlo Localization

Monte Carlo Localization, or MCL, is a localization algorithm that represents the belief \( \text{bel}(x_t) \) of a robot’s location by particles. As mentioned in 2.1, there is a differentiation between global and local localization, and MCL can be applied to both cases.

The basic MCL algorithm can be seen in Algorithm 3.2, that differs from the original Particle Filter algorithm only in lines 4 and 5. The input of this algorithm is the particle set until the previous time instant \( \chi_{t-1} \), the control action \( u_t \), the measurement \( z_t \) and the map \( m \).

In this algorithm, the belief \( \text{bel}(x_t) \) is represented by the set of \( N \) particles \( \chi_t = \{x_t^{[1]}, x_t^{[2]}, ..., x_t^{[N]}\} \) where each \( x_t^{[n]} \) represents the state of particle \( n \) in time instant \( t \).

The hypothetical state \( x_t^{[n]} \) is obtained by applying the sample_motion_model function, that can be either obtained from odometry or velocity, and takes into account the previous states \( x_{t-1}^{[n]} \) for particle \( n \) and the most recent control action \( u_t \).
Algorithm 3.2: Monte Carlo Localization Algorithm [10]

1 Algorithm MCL($\chi_{t-1}, u_t, z_t, m$):
2 $\bar{\chi}_t = \chi_t = \emptyset$
3 for $n = 1$ to $N$ do
4 $x_t^{[n]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[n]})$
5 $w_t^{[n]} = \text{measurement\_model}(z_t, x_t^{[n]}, m)$
6 $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
7 end for
8 for $n = 1$ to $N$ do
9 draw $x_t^{[n]}$ from $\bar{\chi}_t$ with probability $\propto w_t^{[n]}$
10 add $x_t^{[n]}$ to $\chi_t$
11 end for
12 return $\chi_t$

In line 5, the importance of the particle $n$, represented by $w_t^{[n]}$, is calculated from a measurement model that returns the likelihood of the measurement $z_t$, taking into account the map $m$ and all the states in $x_t^{[n]}$ of the particle $n$ until time $t$. The particle with state $x_t^{[n]}$ and weight $w_t^{[n]}$ is added to the temporary particle set $\bar{\chi}_t$ in line 6.

From lines 8 to 11, the algorithm resamples the particles from the temporary particle set $\bar{\chi}_t$ into the result of the algorithm, the set $\chi_t$. This resampling picks $N$ particles from $\bar{\chi}_t$, so that $\chi_t$ has the same number of particles as $\bar{\chi}_t$. However, this selection takes into account the weight of each particle in $\bar{\chi}_t$, so that the ones with greater weight are more represented. This means that the particles with higher weight in $\bar{\chi}_t$ will have several copies in $\chi_t$, and the ones with lighter weight will most likely disappear.

The size of the particle set $N$ has a direct influence in the accuracy of the approximation. The more particles the approximation has, the more accurate the algorithm is, with the setback of increasing the computational resources.

The end result of this algorithm is a sample set $\chi_t$ with many particles that can be the localization of the robot. How the final localization is chosen is by dividing these particles into clusters and choose the one with the biggest weight. From that cluster, the algorithm creates an average from all the particles of this cluster and creates a pose from it that it considers to be the robot localization.

3.4 Augmented Monte Carlo Localization

As mentioned before, MCL algorithm works efficiently both in the global and local localization but it cannot recover from the kidnapped robot problem or from wrong localizations. This is intuitive to understand since after localizing in a pose, all the particles are clustered around that pose. If this pose is not the correct localization and all the particles are around it, there will be no particles in any other pose of the map for the algorithm to converge to, making the correct localization impossible to reach.
To solve this limitation of the algorithm, another version of the algorithm was presented, Augmented MCL. This new version proposes a simple heuristic to solve the problem that can be summed up by the addition of random particles to the set when the robot perceives it is localized in a wrong location by checking the measurement's likelihood. This way, the particles will stop being so clustered and may converge into the correct localization. How quickly it can recover depends on the number of particles, on the heuristic to determine whether the robot is correctly localized, on the size of the map and in chance. This is because a new random particle can be immediately generated in the vicinity of the correct pose, or anywhere else in the map.

The added random particles could be, for example, a fixed number at each iteration, but a more interesting approach is to add a proportional number of random particles to an estimation of the location accuracy. This estimation of the robot's localization accuracy can be obtained from the probability of sensor measurements

\[ p(z_t|z_{t-1}, u_t, m) \] (3.6)

in relation to the average measurement probability. Since in particle filters the importance weight can be considered a stochastic estimate of Equation (3.6), the mean

\[ \frac{1}{N} \sum_{n=1}^{N} w_{t}^{[n]} \] (3.7)

can be used as the estimation of the measurement likelihood. Since this average can be always high due to, for example, noisy sensors and that there is no reference value to judge if this average is good or not, it is better to have a long term average estimation of the accuracy of the location and a short term one and compare both to determine the deterioration of localization accuracy.

This solution to the kidnapped robot problem implemented in the Augmented MCL algorithm can be seen in algorithm 3.3. The differences when comparing with MCL algorithm shown in 3.2 start with the declaration of 2 variables: \( w_{\text{slow}} \) and \( w_{\text{fast}} \). These variables are the average of the long-term and short-term averages, respectively. This means that the \( w_{\text{slow}} \) value takes into account past iterations with a bigger importance than \( w_{\text{fast}} \), that represents mostly the few last iterations, hence the name short-term average. The long-term average \( w_{\text{slow}} \) is given by

\[ w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}}) \] (3.8)

and the short-term average \( w_{\text{fast}} \) is calculated as

\[ w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}}). \] (3.9)
Algorithm 3.3: Augmented Monte Carlo Localization Algorithm \[10\]

Algorithm Augmented MCL($\chi_{t-1}$, $u_t$, $z_t$, $m$):

1. static $w_{\text{slow}}$, $w_{\text{fast}}$
2. $\bar{\chi}_t = \chi_t = \emptyset$
3. for $n = 1$ to $N$ do
   4. $x_t^n = \text{sample\_motion\_model}(u_t, x_{t-1}^n)$
   5. $w_t^n = \text{measurement\_model}(z_t, x_t^n, m)$
   6. $\bar{\chi}_t = \bar{\chi}_t + (x_t^n, w_t^n)$
   7. $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{N} w_t^n$
   8. end for
9. $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
10. $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
11. for $n = 1$ to $N$ do
    12. with probability $\max\left(0, 1.0 - \frac{w_{\text{fast}}}{w_{\text{slow}}}\right)$ do
    13. add random pose to $\chi_t$
    14. else
    15. draw $x_t^n$ from $\bar{\chi}_t$ with probability $\propto w_t^n$
    16. add $x_t^n$ to $\chi_t$
    17. end with
18. end for
19. return $\chi_t$

The importance that each of these averages give to each new iteration is defined by the two constants $\alpha_{\text{slow}}$ and $\alpha_{\text{fast}}$, that influence their respective average. For the algorithm to work as it is designed to, the following condition needs to be true: $0 \leq \alpha_{\text{slow}} \ll \alpha_{\text{fast}}$. The reason why this relation has to be true is to accomplish the long-term and short-term component. By being much greater than $\alpha_{\text{slow}}$, the constant $\alpha_{\text{fast}}$ impacts much more the value of $w_{\text{fast}}$ in each iteration than $\alpha_{\text{slow}}$, making it the intended short-term average. This operation is done in lines 10 and 11, where the value of $w_{\text{slow}}$ and $w_{\text{fast}}$ are updated with the difference from their current values to the current iteration estimation of accuracy. These differences are multiplied by $\alpha_{\text{slow}}$ and $\alpha_{\text{fast}}$, completing the update of the long-term average and the short-term one.

Other changes when compared to algorithm 3.2 are in line 8. This line results in the empirical measurement likelihood that was presented in Equation (3.7) that is later used in lines 10 and 11 to update the short and long-term averages as explained above.

The last difference from algorithm 3.2 is in the resampling loop, where a probabilistic condition is added to add random particles to $\chi_t$. The way it works is that for each $N$ iteration of the resampling loop either a completely random sample is added to $\chi_t$ or a particle from $\bar{\chi}_t$ is chosen just like in lines 9 and 10 of algorithm 3.2. This randomization or not of a particle is decided by a random condition with the probability

$$\max\left(1.0 - \frac{w_{\text{fast}}}{w_{\text{slow}}}, 0\right).$$

(3.10)
What this probability means is that if the short-term average is much greater or equal to the long-term average, there is no need to add random particles since the robot is likely well localized. If the short-term average is higher than the long-term one, then the relation between them is $0.0 < \frac{w_{\text{fast}}}{w_{\text{slow}}} < 1.0$, and random particles will be added proportionally to this relation.

This way, the global and local localization problem is handled efficiently, as well as the kidnapped robot problem.

### 3.5 Adaptive Monte Carlo Localization

After optimizing MCL with the addition of random particles to recover from the kidnapped robot problem, one more optimization can be made to increase the localization algorithm’s computational efficiency. This optimization is to vary the number of particles according to what is needed at each iteration of the algorithm.

What KLD sampling does is to bound the error introduced by the sample-based representation of the particle filter [15]. At each particle filter iteration, the KLD approach samples particles until their number is high enough to guarantee with probability $1 - \delta$ that the Kullback–Leibler distance between the maximum likelihood estimate and the belief estimate does not exceed a predefined limit $\epsilon$. The result of this approach is that there is a smaller number of particles if most samples are focused in a small area and more particles if the samples are more distributed along the state space.

The Adaptive MCL will then stop when the number of samples $n$ is no longer part of the interval defined by the expression

$$min_p \leq n < \frac{1}{2\epsilon} \lambda_{k-1,1-\delta}$$

where $k$ is the number of bins (sections of the map that are different from each other, similar to map cells), $min_p$ in the minimum number of particles and the values of $\delta$ and $\epsilon$ are fixed parameters.

### 3.6 Dynamic Window Approach

The Dynamic Window Approach or DWA is a guidance algorithm proposed in [22]. This method obtains its trajectory by computations made in the space of velocities.

A velocity in DWA refers to the velocity vector $(v_i, w_i)$ that can be visualized as a curvature (since the angular velocity can make it not linear), where $v_i$ is the linear velocity and $w_i$ the angular velocity. To reach a certain goal, the method needs to determine $n$ velocities for $n$ time instants from $t_0$ to $t_n$, increasing the search space exponentially with the size of $n$. To make it feasible, it is considered that the velocity is constant after $t_1$ (constant for $n-1$ time instants), making the search space two-dimensional.
This approximation is based on the premises that the search is repeated after each time interval and that the velocity is indeed constant if no new commands are provided.

To obtain the velocities in the state space for further analysis, the circular trajectories characterized by the velocity pair \((v, w)\) take into account the approximation mentioned before, making the state space a two-dimensional space. Then, all the velocity vectors that do not allow the robot to safely stop before reaching a detected obstacle are removed from the search space. These velocities can be defined as

\[
V_a = \left\{ (v, w) | v \leq \sqrt{2 \cdot \text{dist}(v, w) \cdot \dot{v}_b} \wedge w \leq \sqrt{2 \cdot \text{dist}(v, w) \cdot \dot{w}_b} \right\}
\]  

(3.12)

where \(V_a\) is the set of velocity vectors \((v, w)\) that do not hit any obstacle. The pair \((\dot{v}_b, \dot{w}_b)\) are the accelerations of braking in Equation (3.12) and \(\text{dist}(v, w)\) is the distance to the closest obstacle in the curvature defined by \((v, w)\). This distance is obtained by the product of the angle \(\gamma\) (angle between the robot and the obstacle) and the radius \(r\) of the curvature. Hence, the distance expression is

\[
\text{dist}(v, w) = \gamma \cdot r.
\]  

(3.13)

Another set of velocity vectors is determined by taking into account the dynamic constraints of the robot that remove all the velocity vectors that can’t be achieved by the limited acceleration of the robot, creating the set of velocities \(V_d\) that can be defined by

\[
V_d = \left\{ (v, w) | v \in [v_a - \dot{v} \cdot t, v_a + \dot{v} \cdot t] \wedge w \in [w_a - \dot{w} \cdot t, w_a + \dot{w} \cdot t] \right\}
\]  

(3.14)

where \(t\) is the time interval that the accelerations \((\dot{v}, \dot{w})\) will be applied in and \((v_a, w_a)\) is the actual velocity vector.

Equation (3.14) means that the velocities of the dynamic window are centered around the current velocities with boundaries in the maximum and minimum speeds it can achieve, calculated with the accelerations that can be applied. All velocities outside this range are left out of \(V_d\) since the robot cannot achieve them.

Considering the search space with all possible velocities \(V_s\), the search space used in DWA after the restrictions imposed above can be obtained by

\[
V_r = V_s \cap V_a \cap V_d.
\]  

(3.15)

After obtaining the final search space from Equation (3.15), the last step to take is to search for the optimal velocity. To do so, the objective function is defined as

\[
G(v, w) = \sigma \left( \alpha \cdot \text{heading}(v, w) + \beta \cdot \text{dist}(v, w) + \gamma \cdot \text{velocity}(v, w) \right)
\]  

(3.16)
and the pair \((v, w)\) that gets the maximum value of \(G(v, w)\) is chosen as the optimal. The objective function takes into account 3 main criteria: the heading of the robot towards the target goal, the clearance and the velocity, that can influence more or less the objective depending on the value of the constants \(\alpha\), \(\beta\) and \(\gamma\).

The heading component of the Equation (3.16) measures the angle \(\theta\) between the target goal and the robot heading at the point the robot reaches after exerting maximum deceleration in the time interval. This value is used to calculate

\[
heading(v, w) = 180 - \theta.
\]  

(3.17)

The second criterion to be used in the objective function is the clearance, that determines the distance to the closest obstacle detected as shown in Equation (3.13). If no obstacle is detected, this component of the function is set to a large constant value.

The other component to be checked for each velocity vector is the velocity that is just the projection of velocity \(v\), which means only the linear component of the robot's velocity.

The values of all three criteria are then normalized to \([0, 1]\) and the objective function is computed. It is important to note that the optimal resulting velocity vector depends on which values are used in the control constants. The recommended setting is that \(\alpha > \beta\) and \(\gamma\).

3.7 Dijkstra’s Algorithm

To solve the shortest-path problem on a weighted graph, Dijkstra’s algorithm can be used, as it is shown in [35]. A weighted graph means that the graph has several nodes connected to each other and the connections have an associated weight. This weight is the cost of the transition from a node to another. If it is used in a map divided by cells of the same size, each cell can be a node connected to the 8 nodes that correspond to the 8 connected map cells, and the weight of this connection is the same between all the nodes since the cells are all the same. In other cases, if the connection between nodes isn’t always the same, the weight needs to be different and Dijkstra’s algorithm works for both.

When the algorithm starts, it sets all nodes with the unvisited state. Every node can be visited or unvisited and visited states are never visited again. Besides this value each node also has a value that corresponds to a weight to reach that node. This weight will be denominated \(\Delta\) throughout the rest of this section. The \(\Delta\) is set as 0 in the starting node and infinite in all other nodes of the graph.

The current node is set as the start node in the beginning, it proceeds to check all unvisited neighbour nodes calculating the \(\Delta\) to the current node. This new \(\Delta\) is compared to the current \(\Delta\) of the node and the smallest value is kept. In the case that the new delta is the smallest value, the current node is saved as the parent node of the node in question. The \(\Delta\) is calculated as the \(\Delta\) of the current node plus the weight to the unvisited node. When all neighbour nodes are checked, mark the current node as visited.
Algorithm 3.4: Dijkstra’s Algorithm

1. Algorithm Dijkstra(graph, origin):

   1. create node set Q
   2. for each node i in graph do
      3. \( \Delta[i] = \infty \)
      4. parent_node[i] = None
      5. add node to Q
   3. end for
   4. \( \Delta[origin] = 0 \)
   5. while Q is not empty do
      6. current_node = node in Q with \( \min(\Delta[\text{current_node}] ) \)
      7. remove current_node from Q
      8. for each neighbour n of current_node do
         9. if n \( \in \) Q do
            10. new_\( \Delta \) = \( \Delta[\text{current_node}] + \text{length(\text{current_node}, n)} \)
            11. if new_\( \Delta \) < \( \Delta[n] \) do
               12. \( \Delta[n] = \) new_\( \Delta \)
               13. parent_node[n] = current_node
            14. end if
         15. end if
      16. end for
   17. end while
   18. return \( \Delta[] \), parent_node[]

and select the node that has the smallest \( \Delta \) and is unvisited. The algorithm ends when the goal node has been visited or when all remaining unvisited nodes have \( \Delta = \infty \), which means that the remaining unvisited nodes cannot be reached. This algorithm can be seen in Algorithm 3.4.

When the algorithm is finished, the path from the origin to the goal can be obtained tracing back from the goal node by checking the parent node of each node in the path until it reaches the origin. This algorithm’s result is an optimal shortest path solution for the selected graph, origin node and target node, and it will find a solution if the origin and target node belong to the same graph tree.
In this chapter, the ROS packages used as a base for the developed work are presented, building on top of it to create the proposed Multimodal Navigation. This method will be explained in this Chapter with these packages, with the content developed in Chapter 2 and in Chapter 3 as well as with new implementations to introduce features or improve the existing ones.

4.1 ROS Components

Since November 1 2009 [1], ROS has been available to help program and develop robots. ROS is a framework that facilitates the integration of robot software. Its libraries, tools and conventions simplify the task of working with robots.

Currently, the most used navigation package on ROS framework is navigation [36]. This package includes several others by default such as:

1. *amcl* [8], that uses from the different localization methods mentioned in Section 2.1 the Adaptive MCL, explained in Section 3.5. The configurations used will be further explained in Section 4.3.

2. *move_base*, that can be considered as the control unit of the navigation stack since it connects all the components towards the final goal of producing a velocity command for the motors [7].

3. *map_server*, that supplies the map data to the navigation stack while also being able to make changes to the current map as can be seen in [37].

4. *global_planner*, that defines a path planner from the location of the robot to the desired goal. It takes into account the current occupancy grid map and costmaps to avoid static obstacles. This path planner has several parameters that can be chosen, such as whether or not to use a gradient descent method to smooth the path chosen instead of just following the edges of the map cells. Another important parameter is the algorithm to be used to find the shortest path. How this package is used is described in detail in Section 4.4.2.

5. *costmap_2d*, that handles all the costmaps. These costmaps are occupancy grids produced from sensor data and from the map. It also generates the inflated cost area based on the parameters set beforehand in the configuration file [38].

6. *base_local_planner*, that creates a trajectory from the plan generated from the *global_planner* that allows the robot to safely navigate without hitting obstacles towards the goal. More information on how this package was used is described in Section 4.4.3.
4.2 Coordinate Frames

A robotic system has many components that move in relation to each other and it is important to know the position of each component in relation to the others, specially because most measurements are taken in different frames and need to be transformed to a same frame to allow a combination of the measurements. To solve this problem, the framework ROS has a library to handle this problem that is called Transform Library (tf), as can be seen in [39].

The idea of this library is to connect every frame of the system in a tree graph structure such that each frame is a node and the transform between two frames is represented as edges of the graph. When a node is connected to a parent node, it inherits all the connections that the parent node has, integrating the tf tree. This library allows to get for a certain component the transform from the coordinate frame of another component if those two components are in the same graph.

To perform the frame transform operation between two tf's, a reference frame and a target frame are required. The transformation returns the translation and rotation of the target frame in relation to the reference frame.

In the used implementation, the base tf tree used can be seen in Fig. 4.1.

![Figure 4.1: Base tf tree of the implemented navigation method.](image)

In Fig. 4.1, it is possible to see some of the most important frames and its connections, like the /map, /base_link, /odom and /person_position that are very important for the correct behaviour of the proposed method since without them the localization package amcl would not work correctly and the dynamic goal approach could not be used.

4.3 Localization

In Chapter 2, several robot localization methods were introduced, where the most important conditions that a robot needs to meet are that it can localize the robot properly and maintain the location when it moves. An indoor environment can be a domestic environment with well defined walls and furniture that
add some good reference points to the localization algorithm. It can also be a professional workspace as, for example, a coffee shop that can have walls in a square shape and barely any furniture other than some tables and chairs which can be moved and are hardly detected by the used sensors since their legs are very thin. In Chapter 5, examples of both types are used.

The localization algorithm must also be able to handle the kidnapped robot problem whether it is indeed moved by someone or even if it initially chooses a wrong location. In either case, it needs to recover to its correct localization to perform the tasks. Another factor is the resources this algorithm will use.

Service robots in indoor spaces can’t be too big or they will not be able to move around the environment and achieve their goals. This means that the amount of hardware they can have integrated in their shell is limited as are their computational resources. Therefore, if a localization algorithm works well but uses too many resources to allow the rest of the components to perform correctly, it won’t be a good choice.

For these reasons, the algorithm that is used for localization is Adaptive MCL, described in Chapter 3.5, for its global and local localization accuracy, its efficient amount of spent computational resources and the ability to recover from wrong localizations.

The algorithm \(^1\) uses the map generated by FASTSlam v2 [40](implemented by gmapping [41]), the laser scan (that can be the combination of several laser scans) and the transform tree to output a pose estimation. The transform tree, as explained in Section 4.2, is able to input the odometry information to this package in order to use it in the localization algorithm.

Some important settings of amcl used were:

- Particles from \([100, 20 000]\), which means that the number of particles can vary from a minimum of 100 particles to a maximum of 20 000 particles
- Minimum translational distance (distance in the position moved in any direction) moved before updating the filter: 0.05 m
- Minimum rotational movement (rotation of the orientation in any direction) before updating the filter: 0.01 rad/s
- Number of filter updates before resampling: 2
- Value of \(\alpha_{\text{slow}}\): 0.001
- Value of \(\alpha_{\text{fast}}\): 0.1

The maximum limit for the amount of particles as 20,000 was obtained experimentally as a good value to maintain the localization and not consume too many resources. The minimum translational distance

\(^1\)Documentation available in [8]
was set as 0.05 m (lower than the default 0.2 m) so it could update its location more frequently to keep a more accurate location. The same was done to the minimum rotational movement, decreasing the value from \( \pi/6 \) to approximately \( \pi/31.4 \) so as to update the location estimate more often, as mentioned above.

The number of filter updates before resampling was kept as two by default since it was recommended by the developers of the algorithm. This optimization reduces the amount of resamples made so it uses less computational resources.

The values of \( \alpha_{\text{slow}} \) and \( \alpha_{\text{fast}} \) are very important since they influence how fast the algorithm recovers from a wrong location but also how often it realizes that its location is wrong, adding random particles unnecessarily. This can cause the robot to always question its location and change it often, preventing a smooth and continuous movement. The chosen values were obtained and tested experimentally so that the robot was able to recover, even if taking more time, and simultaneously ensuring that the location was questioned less times so that the movement from navigation could be more fluid. These criteria were chosen because the robot can maintain its location correctly and a fluid movement is preferable when the robot interacts with people during its tasks.

The obtained localization is given by a pose with an associated covariance matrix with dimensions 6x6 shown in Equation (4.1). This is because the matrix needs to give information relating to the relationship between all the six degrees of freedom shown in Fig. 4.2, translation on x, y, z and rotation expressed as Euler angles roll, pitch, yaw.

Even though the robot has six degrees of freedom, since it is a terrestrial moving robot, it only has 3 degrees of freedom. Adding this to the fact that the position does not influence the variation of the rotations, the covariance of the position can be given by a 2x2 matrix. Since the covariance matrix is positive-semidefinite, \( \sigma(x, y) = \sigma(y, x) \) and represents the covariance of \( X \) and \( Y \), while \( \sigma(x, x) \) and \( \sigma(y, y) \) are the variances of these variables.
As for the rotations, roll and pitch are not used taking the assumption that the robot is always in the ground plane, so the robot can only rotate in the yaw term. This means that the yaw rotation variation does not depend on other rotations or in the position and can be reduced to the yaw variance value, obtained from the covariance matrix as the last eigenvalue.

\[
\Sigma_1 = \begin{bmatrix}
\sigma(x, x) & \sigma(x, y) & 0 & 0 & 0 & 0 \\
\sigma(y, x) & \sigma(y, y) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \sigma(\text{yaw}, \text{yaw})
\end{bmatrix}
\] (4.1)

This means that the matrix in Equation (4.1) can be simplified into a more simple two variable covariance matrix

\[
\Sigma_2 = \begin{bmatrix}
\sigma(x, x) & \sigma(x, y) \\
\sigma(y, x) & \sigma(y, y)
\end{bmatrix}
\] (4.2)

and the variance of the yaw represented by \( \sigma(\text{yaw}, \text{yaw}) \).

### 4.4 Navigation

The navigation stack are all the components necessary to choose the path towards the desired goal and decide the velocity commands to send the motors to follow this plan. The architecture scheme of the proposed navigation stack can be seen in Fig. 4.3.

**Figure 4.3:** Navigation’s architecture Diagram [reprinted from [7]]
In this diagram, it is possible to see how the different components interact with one another. The map server supplies the map and the sensor sources supply, for example, the laser scan data necessary to update the local and global costmap. The localization is given to the navigation by the amcl package, that uses the data from the sensors and the odometry of the robot.

### 4.4.1 Target Goal

The target goal is a very important component of the navigation since it is the main objective of the method and it triggers the several components to begin the process of path planning. In the available common navigation stack\(^2\), the received goal is a static pose. A static pose is a fixed position in a map with a certain orientation that does not change with time.

In order to achieve our objective, the navigation should be able to receive not only static goal poses but also dynamic ones. By using the tf library, it is possible to always have the pose of a certain tf even if it moves between two time instants. If a dynamic object or person that should be followed is represented in a frame, by calculating the transform to a reference frame like /map the pose of this dynamic target is known. If these two frames exist and are known, the transform between them results in the translation and rotation between frames.

If the reference frame is the /map and the target frame is, for example, a frame named /person1, the transformation between these frames will return the pose of the person in the map. This is true because all poses are points and orientations in relation to the origin of the map that is also the frame named /map in the tf tree shown in Fig. 4.1.

<table>
<thead>
<tr>
<th>type</th>
<th>denomination</th>
<th>description</th>
<th>optional/needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>activated</td>
<td>whether or not dynamic goal should be activated</td>
<td>needed</td>
</tr>
<tr>
<td>string</td>
<td>dyn_goal tf</td>
<td>reference frame to track</td>
<td>needed</td>
</tr>
<tr>
<td>string</td>
<td>origin tf</td>
<td>reference tf to be used</td>
<td>optional (default is /map)</td>
</tr>
<tr>
<td>float64</td>
<td>dist</td>
<td>distance r to be kept to the tracked goal</td>
<td>optional (default is 1.2 m)</td>
</tr>
</tbody>
</table>

By implementing the dynamic goal using coordinate frames, the default reference tf will be /map but the target frame has no default since it is not known which frame the algorithm should follow. This means that this parameter is required as an input for the algorithm to work.

The following new message type\(^3\) is introduced in table 4.1 and is used as an option to the static goal in move_base package. With this new message defined, the two following goals are accepted:

1. /move_base_simple/goal - topic that receives a static goal
2. /move_base_simple/dyn_goal - topic that receives a dynamic goal

\(^2\)Explained in [36]
\(^3\)Message type complete description in Appendix B
Each option has a specific message type, the static goal receives a pose and the dynamic one receives the message type described in table 4.1. These two options are mutually exclusive which means that they cannot be used at the same time.

As seen in Table 4.1, the dynamic goal has a distance parameter to define how far the robot should try to be from the goal. This value could be 0 if the robot should reach the target frame or a positive distance if the robot should keep a certain distance from the target.

If \( r > 0 \) (\( r \) is the radius of the circle and the distance to the dynamic goal) the robot does not have a defined goal pose but an infinite set of poses that can be set as the goal. To choose a pose from this infinite set an iterative process is introduced to calculate the best pose. This iterative process that is followed to determine this best pose is:

1. A circle of radius \( r \) is created around the target with a certain number of points that follow equation

   \[
   n_{pc} = r \cdot 50
   \]  

   where \( n_{pc} \) stands for number of points in the circle.

2. All the points are ordered by the distance to the robot.

3. One by one, the points are evaluated if they are a free cell in the costmap or not. If they are not, the point is discarded and the next point in the ordered list is evaluated.

4. If the point can be set as a goal, the algorithm checks if there is any obstacle between the robot and the target. If there is, the point is discarded and the next point is evaluated.

5. The closest point to the robot that satisfies these conditions is set as the best pose.

6. The orientation the robot should have is set so that base_link is oriented towards the target pose.

7. The pose with the chosen position and orientation is chosen as the navigation goal for this dyn_goal at the current iteration.

This process can be seen in Fig. 4.4 and is performed every time the robot perceives that the target frame has moved at least a certain amount defined in a threshold or until the dynamic goal mode is deactivated. This threshold can be, for example, 10 cm.

In the proposed iterative process to choose the goal pose of the robot, in step 4, the way the method checks if there is a clear path towards the target is by making a straight line from the point being analysed and the desired target. This line is made of a finite sample set of points composed of a number of points that vary according to the chosen \( r \). This number of points is given by Equation (4.5), where the points are equally distanced from each other by the value obtained by Equation (4.4), as can be seen in Fig. 4.4(c).
Figure 4.4: Selection of the optimal position
Each point of the created finite sample set of points has a coordinate \((x, y)\), and they have an interval \(\delta\) between each \(x\) of

\[ \delta = \frac{x_{\text{target}} - x_{\text{origin}}}{n_{\text{pl}}} \]

(4.4)

where the variable \(x_{\text{target}}\) is the \(x\) of the followed target, \(x_{\text{origin}}\) is the \(x\) of the best pose (point that is analysed to be set as goal) and the number of points in the line \(n_{\text{pl}}\) is represented by

\[ n_{\text{pl}} = r \cdot 20. \]

(4.5)

This interval is then applied starting from \(x_{\text{origin}}\), incrementing the value of \(x\) at each iteration and calculating the correspondent value of \(y\) for it with a linear equation such as Equation (4.6), where \(m\) and \(b\) are calculated from \(x_{\text{origin}}\) and from \(x_{\text{target}}\). The iteration stops before it reaches \(x_{\text{target}}\).

\[ y = m \cdot x + b \]

(4.6)

In Fig. 4.4, the position for the dynamic goal is chosen taking into account there is a wall between the robot and the target. In Fig. 4.4(a), the circumference around the target has been determined. In Fig. 4.4(b), the robot has ordered the candidate points by proximity and the closest point is highlighted. Next, in Fig. 4.4(c), the first point A has been discarded since its cost in the map is high because it is located in a wall. The second closest point is selected and its location is checked in the map. Since it is in a free position, it moves to the next step, checking if there is a free path from point B and the target. The algorithm will create points between point B and the target and check the occupancy of each one, performing step 4. Since there is a wall, the costmap cost of one or more of these generated points will be high, which means that point B is discarded. Next, in Fig. 4.4(d), point C is evaluated. Like point B, it is in a free location but, when performing step 4, this time the algorithm returns that the path from C to the target has no high costmap cells and this point is chosen as the location to send the robot to, as seen in Fig. 4.4(e). In the next Fig. 4.4(f), the orientation is chosen so that the robot will be looking at the target and the process is concluded.

### 4.4.2 Global Planner

The chosen global planner from the motion planners mentioned in Chapter 2 is a cell decomposition algorithm with Dijkstra’s algorithm (explained in Chapter 3.4) as the path planning algorithm. The reason why Dijkstra’s algorithm was chosen and not A* was to have the smoothest and optimal path over the best efficiency, as can be seen in Fig. 4.5. This was important so that the robot, that interacts frequently with humans, would not come as aggressive with abrupt movements. Gradient descent method was also used to create a smoother path and the planner was set so as not to plan in unknown spaces of the
map since the map used had all the possible locations where the robot could go to mapped. In addition, the tolerance was set to 0 so that the planner very accurately aimed towards the desired goal and the quadratic approximation of the potential was used instead of a simpler method.

(a) Using A* algorithm [42]  
(b) Using Dijkstra’s algorithm [42]

Figure 4.5: Comparison between the two mentioned shortest path planners.

4.4.3 Local Planner

The local planner chosen from the proposed guidance algorithms mentioned in Chapter 2 for the navigation stack was DWA, explained in Chapter 3.6, since it can correctly choose a velocity command to send to the base controller, following a global path and is able to avoid dynamic obstacles.

The main configurations used for this component are related to the dynamical constraints of the robot, setting the limits of the robot’s speed and acceleration for translation and rotation. Instead of setting these values only once, it might be necessary to change the top speed of the robot depending on which action the robot should perform. For this, several configurations were made to easily change the value of these Adaptive MCL parameters in runtime. These configurations can be seen in table 4.2.

<table>
<thead>
<tr>
<th></th>
<th>only_front</th>
<th>only_back</th>
<th>front_and_back</th>
<th>front_and_back_slow</th>
</tr>
</thead>
<tbody>
<tr>
<td>max velocity in X axis</td>
<td>0.6 m·s⁻¹</td>
<td>0.6 m·s⁻¹</td>
<td>0.6 m·s⁻¹</td>
<td>0.4 m·s⁻¹</td>
</tr>
<tr>
<td>min velocity in X axis</td>
<td>0.0 m·s⁻¹</td>
<td>0.0 m·s⁻¹</td>
<td>0.0 m·s⁻¹</td>
<td>-0.4 m·s⁻¹</td>
</tr>
<tr>
<td>max velocity in Y axis</td>
<td>0.6 m·s⁻¹</td>
<td>0.6 m·s⁻¹</td>
<td>0.6 m·s⁻¹</td>
<td>0.4 m·s⁻¹</td>
</tr>
<tr>
<td>min velocity in Y axis</td>
<td>-0.6 m·s⁻¹</td>
<td>-0.6 m·s⁻¹</td>
<td>-0.6 m·s⁻¹</td>
<td>-0.4 m·s⁻¹</td>
</tr>
<tr>
<td>max rotational velocity</td>
<td>0.6 rad·s⁻¹</td>
<td>0.6 rad·s⁻¹</td>
<td>0.6 rad·s⁻¹</td>
<td>0.4 rad·s⁻¹</td>
</tr>
<tr>
<td>acceleration limit X axis</td>
<td>0.6 m·s⁻²</td>
<td>0.6 m·s⁻²</td>
<td>0.6 m·s⁻²</td>
<td>0.3 m·s⁻²</td>
</tr>
<tr>
<td>acceleration limit Y axis</td>
<td>0.6 m·s⁻²</td>
<td>0.6 m·s⁻²</td>
<td>0.6 m·s⁻²</td>
<td>0.3 m·s⁻²</td>
</tr>
</tbody>
</table>

In table 4.2, the velocity in a certain axis uses the axis configuration as shown in Fig. 4.2, so to make only_front the minimum speed on axis X is set to 0 m·s⁻¹. For back_only the maximum velocity in axis X was set to 0, which forces the robot to only move backwards and sideways. To make a
slower configuration so that the robot moves slower, all velocities and accelerations are reduced. These configurations are implemented in order for them to be imported when a certain specific navigation is required.

These configurations are important to produce different types of movements for the navigation modes in order to improve the robot behaviour according to the selected mode.

### 4.5 People Following

People following is an example of a dynamic goal, since to follow a person the navigation stack needs to constantly track this person, determine its position and navigate towards it or back away from it, according to the defined parameters. The detecting and tracking component used is originated from previous members of the team SocRob@Home and is available at [43]. From this component, the tf frame `tracked_person` indicates the pose of the tracked person to be followed. Getting the transformation from this frame to the `/map` frame, the pose of the person can be obtained.

Since the pose of the person can be obtained from two tf frames in the same tree, the dynamic goal approach introduced in Chapter 4.4.1 can be used, with the target frame as `/tracked_person` and the reference frame as `/map`, setting the desired distance to the person. Taking into consideration the Proxemics theory [32] analysed in [44], the distance of 1.2 meters was used in the tests and experiments, since it is the limit between the personal and the social space, as shown in Fig. 4.6.

![Figure 4.6: A person's personal space [44]](image)

This is everything that needs to be done to run a people following mode if the dynamic goal component is integrated in the system. The message sent to the topic `/move_base_simple/dyn_goal` can then be, for example, as simple as in Listing 4.1 and to deactivate it, a message with `activated=False` needs to be sent. Another way to stop the dynamic goal movement is, as mentioned in 4.4.1, to send a static goal.

```python
dyn_goal_msg(
    activated = True,
```
4.6 Guiding

Guiding mode, like the people following mode, requires the component of human perception. To perform this mode, a guiding mode was designed so that the robot first verifies if there is someone being tracked and awaits a person. When the person starts being tracked, the robot will aim to keep tracking them while moving toward a predefined goal. This goal is considered to be static, although if a dynamic goal is required, as long as it does not need human perception as well, it can work.

To start the movement, the robot sends a goal to `move_base` with the configuration `only_back`, presented in table 4.2, so that the robot moves towards the goal while maintaining the target person in front of its camera, necessary for the tracking, as seen in Fig. 4.7(a). The robot then calculates, at each time instant, the distance to the tracked person and if it lost them. In these cases, it will stop the movement and say something to attempt to bring back the person in front of it or simply wait for the person to catch up, as seen in Fig. 4.7(b). This mode keeps running until the goal is reached.
Results

Contents

5.1 Localization ......................................................... 37
5.2 Waypoint Navigation ............................................. 43
5.3 People Follower ..................................................... 49
5.4 Guiding ............................................................... 52
The results obtained and discussed in this Chapter demonstrate the navigation stack capabilities and robustness as it undergoes multiple tests. The experiments are divided into four main components:

- The localization, where this component is tested to see if it can maintain the location and if it can recover in case of a kidnapped robot situation.

- The waypoint navigation, that should correctly plan the shortest path to a goal and travel to it while avoiding obstacles and maintaining its localization.

- The people following mode, that should aim to keep a certain distance to the tracked person.

- The guiding mode that should successfully navigate to a waypoint while making sure that the tracked person is following it.

These results obtained were recorded from experiments performed either in the ISRoboNet@Home testbed or in the ERL Smart Cities competition challenge, where the navigation stack was used.

The robot used was MBot robot from MONarCH project and produced by IDMind, with 1.02 x 0.57 x 0.67 meters of dimension (H x W x L). The mobile base has 4 Mecanum omnidirectional wheels allowing it to move in any direction. The robot weights between 40 to 50 kilograms, it has a built-in touch screen and a neck that can rotate nearly 180 degrees. Other than this, it has an Orbbec Astra S depth camera attached to its forehead in a support controlled by a motor that allows the camera to change its orientation to the ground, giving it 2 degrees of freedom.

For human interaction, the robot has built-in speakers on its head, built-in leds on its face to express emotions and an attachable microphone to record speech. In terms of computing power, the robot has a i7-3770T processor with a 16GB RAM and a NVIDIA GeForce GTX 1060 6 Gb as gpu.

As for sensors, besides the camera and the microphone, the robot has two Hokuyo URG-04LX-UG01 lasers, one at the front and one at the back, that allow it to have almost 360 degrees of laser sweep coverage, if not for some blind spots next to the sides of the robot.

5.1 Localization

In this section, three experiments are presented to test the localization. Firstly, the local localization is tested by navigating during a long time and testing whether or not the localization can be kept within a given accuracy tolerance. The data analysed is given by part of a run of episode 3 of ERL Smart Cities’ competition that totaled 5 minutes and 23 seconds or 323 seconds.

This episode was named “Deliver coffee shop orders”. At this challenge, the robot assists people in an arena resembling a coffee shop by checking the state of the tables, taking orders, bringing them to
the customers and guiding a new client to a free table. Images from this challenge can be seen in Fig. 5.1 and videos recorded are available in YouTube\(^1\).

(a) MBot inspecting a table.  
(b) MBot serving 2 customers at their table.

Figure 5.1: Images from ERL Smart Cities episode 3.

Secondly, the kidnapped robot problem and the global localization are tested by making the robot localize after an estimate of its position is given, navigating for a while and then producing a situation of kidnapping by turning off the odometry and moving the robot (the same as if the robot would be picked up and carried some distance). The robot then needs to realize it is wrongly localized and relocalize itself.

For the second experiment, two different values for the recovery behaviour constants are used to allow comparison. The values used are \(\alpha_{\text{slow}} = 0.001\) and \(\alpha_{\text{slow}} = 0.01\) while \(\alpha_{\text{fast}}\) was kept as 0.1.

In the third experiment, the error on the localization is calculated by comparing the robot's estimated localization to a ground truth localization given by the Motion Capture System (MCS) that uses 12 OptiTrack PRIME13 cameras (1.3MP, 240FPS). This system is located in Institute for Systems and Robotics (ISR)'s testbed that can track at a high rate a rigid body with some reflecting markers.

5.1.1 Maintaining Localization

The results presented from SciRoc are part of a successful run where the robot correctly performed all tasks. Although there is no numeric data for the ground truth, by synchronizing the video with the recorded data timestamp and comparing them, it is possible to verify that the robot seems to be localized throughout the experiment.

To see how accurate the measurement is, the uncertainty is presented through the variances of the coordinates \(x, y\) and \(\text{yaw}\), as explained in Chapters 3.1 and 4.3. The variance of these measurements, presented in \(m^2\) for \(\sigma(x, x)\) and \(\sigma(y, y)\) and in \(\text{rad}^2\) for \(\sigma(\text{yaw}, \text{yaw})\), can be seen in the graph of Fig. 5.2.

\(^1\)https://youtu.be/xZwMvkZsw4Q
In the graph of Fig. 5.2, the robot is already localized and performing the task of scouting the coffee tables. It is possible to see that the position variances are quite low, always below 0.018 m$^2$ of uncertainty which is approximately equal to a standard deviation of 0.134 meters for $x$ and for $y$. For $\text{yaw}$ in this experiment, the variance never went above 0.007 rad$^2$, translated to a standard deviation of approximately 0.0837 rad or 4.796 degrees.

These are however the top values that were never achieved by the localization process. The average variances in these 323 seconds can be seen in table 5.1.

Table 5.1: Average values of variance and standard deviation

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$\text{yaw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average variance</strong></td>
<td>$0.004$ m$^2$</td>
<td>$0.005$ m$^2$</td>
<td>$0.002$ rad$^2$</td>
</tr>
<tr>
<td><strong>Average standard deviation</strong></td>
<td>$0.066$ m</td>
<td>$0.070$ m</td>
<td>$0.043$ rad</td>
</tr>
</tbody>
</table>

It is also possible to determine the average value of $\sigma(x, y)$, the covariance of the two translational variables that resulted in $-4.31 \cdot 10^{-5}$ m$^2$, denoting a low influence of the variables on one another.

In table 5.1, it is possible to see that the estimated average uncertainty of the localization in a long run after being localized is around 0.07 meters for the position coordinates and 0.043 radians/2.46 degrees, which allows for a smooth navigation in the scenario environment that is resistant to the kidnapped robot problem.
5.1.2 Recovery from Kidnapping situations

In this experiment, the robot would start with an estimate of the position, navigate remotely operated for a while, then the odometry is disabled and the robot is moved for some meters. After kidnapping the robot, the odometry is turned back on and the robot is driven remotely again until it recovers its localization. The instants where the robot is kidnapped and when it recovers its position are tagged in the graph by a label.

For $\alpha_{\text{slow}} = 0.001$, the variances of the variables are plotted in the graph of Fig. 5.3.

![Graph showing variances of x, y, and yaw over time for kidney recovery experiment with $\alpha_{\text{slow}} = 0.001$.]

The graph of Fig. 5.3 shows a system that is mostly confident of its localization even after getting kidnapped. There is only a moment where the variance gets very high when the robot is wrongly localized.

The spike in uncertainty at around time instant 80 seconds produced an attempt to relocalize the robot that failed, although bringing the robot closer to its actual localization. The uncertainty that generated this relocalization is small, but high enough to produce a correct localization.

It is possible to see from the graph of Fig. 5.4 that a higher $\alpha_{\text{slow}}$ when the $\alpha_{\text{fast}}$ is constant originates a bigger uncertainty when the robot is wrongly localized, attempting to relocalize more often. The fact that it took amcl longer to relocalize the robot for $\alpha_{\text{slow}} = 0.001$ than for $\alpha_{\text{slow}} = 0.01$ in this experiment does not mean that it is always like that since this time depends on many factors, including chance.

What can be concluded is that for a lower value of $\alpha_{\text{slow}}$, the robot will remain more confident of its localization.
Figure 5.4: Variances of $x$, $y$ and $yaw$ over time in ISR’s testbed for the kidnap recovery experiment with $\alpha_{slow} = 0.01$.

Localization even when it is not correct, though both can recover from the kidnapped robot problem. This means that the robot will trust more on its localization and have a smoother navigation.

In Fig. 5.5, it is possible to see the robot relocalizing itself for $\alpha_{slow} = 0.001$. In frame 5.5(a), the robot is correctly localized just before the kidnapping is done by turning off the odometry and moving the robot backwards around 2 meters. The odometry is turned back on and the robot is wrongly localized as seen in Fig. 5.5(b).

In Fig. 5.5(c), the robot is confident of its wrong localization but in frame 5.5(d), when the robot enters the corner, the particles start to spread into two clusters that eventually lead to the robot slowly moving into the right localization as shown in Fig. 5.5(e) and Fig. 5.5(f).

5.1.3 Ground Truth Comparison

This experiment will determine the error in localization when using the MCS as ground truth. The ground truth of this system is not perfect since the map of the robot needs to be exactly with the same rotation as the calibration of the cameras, and that the reflecting markers need to be set up in the robot’s head to be seen throughout the testbed. Since the head has some movement in relation to the base of the wheels, this adds an additional error. With these limitations of the ground truth system, the experiment designed was a waypoint navigation to three waypoints around the testbed while recording both the
(a) Robot localized before kidnapping.  
(b) Robot after kidnapping.  
(c) Robot wrongly localized before relocalization.  
(d) Particles start dividing into two clusters.  
(e) The algorithm recovers the localization by having most particles around the true localization.  
(f) Robot correctly relocalized.

Figure 5.5: Images of relocalization
robot's estimation of its location and the ground truth from the MCS.

![Localization Error](image)

**Figure 5.6**: Localization error from the robot's estimated pose and the ground truth from MCS

In Fig. 5.6, the error between the robot's estimated pose and the MCS's ground truth is shown. It is possible to see that the error is always below 0.15 meter and maintained an average value of 0.09 meter error throughout the 90 seconds experiment. Considering the map has a resolution of 0.025 meter and the limitations of the MCS, the values represent a good result. In Fig. 5.7, the poses of both the robot's localization algorithm and the MCS's ground truth over time. It is possible to see that throughout the performed navigation, the error was reduced except when the robot was close to the bedroom.

### 5.2 Waypoint Navigation

One of the most used navigation modes in any mobile robot is waypoint navigation. Waypoints are known poses in the map that are used as static goals in waypoint navigation.

This section will present the result of two experiments. The first is part of a run of ERL Smart Cities’ episode 3, where the robot visits all tables to check them, visiting two waypoints for each table. Then the robot goes to a free table, asks what the clients would like to order and moves to the counter to ask for the clients’ orders. Later, the robot returns to the table where the request was made to deliver the order. This part of the challenge has twelve waypoints in a coffee shop with clients in random tables.

The second one is a small experiment in the ISR domestic robot testbed that shows how the robot
can replan its path after sensing some obstacle that makes the original plan unachievable. The robot will then have to avoid an obstacle on its course as well. This experiment is a common test performed in ERL’s challenges [45] test obstacle avoidance.

Figure 5.8: ERL Smart Cities’ episode 3 map used in the challenge
5.2.1 ERL Smart Cities Challenge

The map of the coffee shop that was used by the participating robots can be seen in Fig. 5.8. In this figure, besides the used map, it is also possible to see the waypoints created. The numbers in the yellow circles were added to the map to reference each waypoint more easily.

![Figure 5.9: Distance from the perceived robot location to the desired waypoint in meters at the ERL Smart Cities challenge](image)

The waypoints used are:

1. Table 1, closer waypoint and delivery waypoint (t₁_c and t₁_d)
2. Table 2, most distant waypoint for this table (t₂_f)
3. Table 2, closer waypoint and delivery waypoint (t₂_c and t₂_d)
4. Table 3, most distant waypoint for this table (t₃_f)
5. Table 3, closer waypoint and delivery waypoint (t₃_c and t₃_d)
6. Table 4, most distant waypoint for this table (t₄_f)
7. Table 4, closer waypoint and delivery waypoint (t₄_c and t₄_d)
8. Table 5, most distant waypoint for this table (t₅_f)
9. Table 5, closer waypoint and delivery waypoint (t₅_c and t₅_d)
10. Waypoint close to the counter to check and get the order (c,c)

All of the waypoints mentioned above are used at least once in the experiment with waypoint 1 being used three times: to check, to ask the order and to deliver the order.

In Fig. 5.9, the navigation to these waypoints is shown in a graph that represents the distance to the desired target at each time instant as perceived by the robot. Between each color change there is a transition of desired goal and a label that indicates which waypoint has been selected.

This experiment shows that the navigation stack can travel several waypoints consecutively in a cluttered space reaching the goal with the selected precision of 0.1 meters. In all waypoints, the robot only stopped when it was less than 0.1 meters away from the goal.

5.2.2 Obstacle Avoidance

The second waypoint navigation experiment was performed in the lab with the map shown in Fig. 5.10(a) with the specifications shown in Fig. 5.10(b).

In this experiment, two navigation waypoints are sent as goal to the navigation stack. Firstly, a waypoint close to the bottom right corner of the map shown in Fig. 5.10(a) is sent. Since this waypoint is close to a plant decoration it will be henceforth denominated as Plant. On the way to this goal, a person blocks the path of the robot forcing it to choose another path to the goal. After reaching this goal, the robot receives another goal in the Entrance waypoint and on the way to it a person will stand in front of its path so it has to avoid him/her.
In Fig. 5.11, the graph of the distance to the goal perceived at each time instant is presented. This graph has the two main parts filled with different colours, the first part with a light colour when it is navigating towards the Plant waypoint and the second with a darker colour when the chosen waypoint is the Entrance. In the graph, the instants where the goals are sent have a tag with the goal name and the instants for replanning and obstacle avoidance are also tagged. Tag A represents the moment of replanning and tag B corresponds to the obstacle avoidance moment.

There are two more instants on the way to the Entrance where the robot stops. The robot stops as a safety measure when the robot starts shaking too much. It is also possible to see that the robot overshoots the goal of the plant, represented by the spike of distance at time instant 30 s. The robot recovers from the overshoot three seconds later, reaching the goal with the intended pose.

To analyse the path planning, some images of the path in the used map are presented in Fig. 5.12 and Fig. 5.13.

In Fig. 5.12(a), the position of the robot can be seen as the red particle set and by its footprint, the path originated from the global planner can be seen as the purple line and the local path as the green line.

The robot follows the path until it detects through laser scans a person blocking its path just after the frame shown in Fig. 5.12(b). After detecting the person, the robot replans a way to the goal through another route. This new plan can be seen in Fig. 5.12(c).

As for the obstacle avoidance in Fig. 5.13, the robot is at Plant waypoint and plans the path to Entrance as illustrated in Fig. 5.13(a). Some time into the path following, in Fig. 5.13(b), the robot turns a corner and finds an obstacle in its path which makes it replan its global path around this found obstacle.
Figure 5.12: Images of the path replanning for the obstacle avoidance experiment when the path created is impossible to follow and a new path needs to be generated.

obstacle, as shown in Fig. 5.13(c). The robot successfully avoids this obstacle by following the new route as evidenced by Fig. 5.13(d).

This experiment effectively shows that the navigation stack can properly replan in case an obstacle is perceived by the robot if another path is available. In the experiment, the sensor used to detect obstacles were two laser scanners that work very well for obstacles at 0.14 meters from the floor. For smaller obstacles or obstacles that have a small base, the robot would need to use other sensors to detect these obstacles like obstacle detecting from its depth camera.
5.3 People Follower

To test the people following mode, the robot was activated with that mode and a person walked at a normal pace in front of the robot. It is important to note that the people tracking and detector is not being evaluated and so there is no other person testing the resilience of the tracker. What is important to test is that given the correct position of a person, the dynamic goal component works correctly in choosing the correct goal, choosing a viable path and avoiding obstacles in the navigation towards the selected goal.

The experiment was performed in the testbed [46] of ISR and in the rest of the Mobile Robotics Laboratory in the 8th floor of the north tower of IST. The whole map used can be seen in Fig. 5.14.
During the experiment, the robot should try to keep the distance of 1.2 meters to the person as indicated in Chapter 4.5. This result can be seen in the plot of Fig. 5.15.

In this plot, the distance to followed person and a reference line at 1.2 meters are presented. As can be seen, the distance always tends to approximate to 1.2 meters even though some spikes occur when the person increases their speed or the robot needs to slow down in its navigation.

In the images presented in Fig. 5.16, some goal choosing results are presented. The robot position is presented as its footprint and/or the cluster of red pose particles, the position person to be followed
(a) Goal choice process operating in a corridor.  
(b) Goal choice and obstacle avoidance path.

(c) Goal choice process operating in an open space.  
(d) Goal choice process when the person is closer than the target distance.

(e) Goal choice process inside the testbed [46].  
(f) Goal choice process when there are obstacles between the closest points and the person.

Figure 5.16: Selection of goal for people following.
is shown by a yellow sphere and a tf frame and the poses at 1.2 meters to the considered target are printed as a small blue sphere. Additionally, the blue and purple areas represent the inflation areas of the costmap, the pink line depicts the chosen global path and the green line the local one.

In Fig. 5.16(a), the robot tracks the person through a corridor with a corner which walls are 1 meter high. This allows the robot to see the person on the end of the corridor and calculate the best possible goal to get closer to the person. The robot is also able to follow the path on this narrow corridor without hitting any walls.

Observing Fig. 5.16(b), the robot has an obstacle in the way but is able to create a path while avoiding the obstacle towards the chosen goal.

In Fig. 5.16(c), the robot and the person are in an open space with a distance of over 3 meters between them and correctly chooses the goal to move to and navigates towards it. In frame 5.16(d), the person gets close to the robot which makes it plan a path backwards towards the desired distance of 1.2 meters.

The goal choosing process also worked in the more limited space of the house environment of the testbed, as seen in Fig. 5.16(e). In addition, the feature to only choose a goal that has a clear path towards the target is working, as observed in frame 5.16(f).

5.4 Guiding

In this section, an experiment was conducted to test the guiding mode explained in Chapter 4.6. The experiment is the same task performed at the end of episode 3 of SciRoc challenge where the robot needs to guide a new customer inside the coffee to a free table.

This task was adapted to the environment shown in Fig. 5.10(a). The initial position is inside the testbed behind the sofa. Then the robot goes to the Entrance waypoint to guide a new person inside the testbed towards the dining table. When it reaches the Entrance, it starts guiding the tracked person towards the given goal but trying to maintain a distance of 2 meters between itself and the tracked person. If the distance becomes greater than this threshold, the robot will stop and wait for the person to get closer, using speech to warn him/her that it is waiting.

In the graph of Fig. 5.17, the distance of the robot to the current goal and to the tracked person are plotted. The plot has two shaded areas, the left one represents the navigation towards the waypoint Entrance and the right side the guiding navigation towards the Dining table.

The robot sees the person and starts tracking him/her before reaching the first goal, so it started guiding immediately. The robot started moving towards the table but the person stayed in the same place so the distance from the person to the robot increased to over 2 meters. The robot stopped and waited for the person to get closer before it started moving again. The person stopped once more before
Figure 5.17: Plot that depicts the distance to the person to be guided and the distance from the robot to the goal at each time instant.

getting to the table that made the robot stop again to wait. In the end, the robot reached the final goal with the person closer than 2 meters from it.

To visualize some situations of this mode, Fig. 5.18 was added. In the frame 5.18(a), the robot has reached the Entrance and is ready to start guiding the customer. Next, the path to the goal is chosen, as can be seen in Fig. 5.18(b). After the navigation, the robot reaches the goal with the target person close by in Fig. 5.18(c). Lastly, a picture taken in the last task of a run in SciRoc’s episode 3 is presented, showing the robot guiding a person to its table.
(a) Robot at the Entrance waypoint prepared to begin guiding mode.

(b) Robot plans the path to the selected guiding goal.

(c) Robot reaches the guiding goal with the tracked person.

(d) Robot guiding a person in the SciRoc competition.

Figure 5.18: Images that show the guiding mode execution.
6

Conclusions

Contents

6.1 Conclusions .............................................. 56
6.2 System Limitations and Future Work ...................... 57
6.1 Conclusions

The objective of the thesis was to design, implement and test a navigation stack that could operate in several modes such as waypoint navigation, guiding and people following. In order to achieve the objective, state of the art components were chosen for localization, local planner and global planner with the move_base package from the framework ROS proving to be important to bring all these components together. The parameters required for the correct behaviour of the navigation stack were tuned to get the best of the components and achieve a smooth collision free navigation. In addition, some components were implemented or changed to improve some functionalities of the stack as well as enable some modes.

The most important component implemented was the dynamic goal that can follow a goal pose moving in time, turned into a ROS package described in Appendix A. The goal can be set directly towards this dynamic pose or try to keep a certain distance from it, depending on the parameters supplied. An example application for this component is the people following mode when the person to follow is considered a moving pose in the time domain. Using the results of an already implemented people detector and follower from the SocRob@Home team [3], a tf frame is received with the pose of the person to track. With this, the dynamic goal component can be activated with the distance to be kept from the pose and it will attempt to always keep this distance from the selected tf frame until the component is deactivated.

All these components were tested to verify their correct behaviour. The results of the localization revealed that it can maintain an accurate estimation for a long time period and recover from the kidnapped robot problem if it arises, even if it takes a long time. It is possible to change the value of $\alpha_{slow}$ and $\alpha_{fast}$ in order to trust more in its localization and take more time to recover from wrong localizations or try to be more accurate by sacrificing the smoothness of the navigation. This will depend on each system’s priorities. It was also tested if the robot can navigate in different environments with different levels of obstacles from a waypoint to another, succeeding in both reaching the goals and avoiding obstacles.

The people following mode using the dynamic goal component was tested afterwards. In these tests, the robot performed how it was supposed to which was to keep trying to be at a certain distance from a moving person while avoiding obstacles and navigating both in open and in narrow spaces. As for the guiding mode, it performed as expected, moving only when the tracked person was within a certain distance. The most important navigation contributions to this mode are navigating backwards to keep the sensors towards the person and the ability to start and stop the navigation.

Taking these tests into account, the navigation stack can perform all the different proposed modes. Some videos of the results obtained can be seen in a YouTube playlist\(^1\).

\(^1\)https://www.youtube.com/playlist?list=PLx27vcpSOb_fdHu0DuN-0QPCwCOhx2GLH
6.2 System Limitations and Future Work

The biggest limitation of the system is that in dynamic goal the target choice has some problematic cases. These cases can be, for example, if the robot has two candidate points with the same distance to the robot and both have a clear path to the robot, the robot will choose the one that is at the top of the list that could be any of the two, even though one of them might take quite longer to reach than the other. This means that an heuristic to take into account the path from the robot to the candidate points could be added. A solution could be to only stop the method after getting 3-5 candidate points (instead of stopping when the closest point to the robot is considered a good goal) and for all of them check the distance of the path and use that information to choose which candidate point should become the navigation goal.

Another good alternative would be to call a path planner from the robot to the target pose and from that path, trace back the path until it finds the first point that is at the desired distance from the target. This approach has some limitations, for example, when the robot is closer to the target than the desired distance, this method would not produce a path to make the robot back some distance. A combination of the methods can perhaps produce a better solution.

Another optimization to be made is related to the following component. When the person moves, the dynamic goal will generate what it considers to be the best goal with the smaller distance. This makes sense if the only objective of the robot is to be at a certain distance of a moving goal. However, if the objective is to follow the path of a moving goal, the navigation should take into account the path of the target when defining its path. This can happen more often if the goal moves much faster than the robot and appears close to it after going around an obstacle. With the current method, the robot will not follow the path of the goal, but instead move directly towards the goal since it is the shortest path. To add this feature, a Bayesian filter could be added to save the path information and an heuristic to balance the goal’s path history and the shortness of the path.
Bibliography


A

**dyn_goal**

dyn_goal is a node that allows the ROS navigation package to follow a dynamic goal exactly or to a certain distance. The node will continue following the goal until it is deactivated. If the distance to be kept to the dynamic goal is higher than 0, the algorithm will choose a free cell in the map with a path available to the dynamic goal as the navigation target.

![Figure A.1: Behaviour of dynamic goal following a dynamic goal at 1,2 meters.](image-url)
A.1 Subscribed Topics

- dyn_goal(dyn_goal_msg) - Configurations of the node
- goal(geometry_msgs/PoseStamped) - Static goal received by the navigation stack
- costmap(nav_msgs/OccupancyGrid) - Costmap to check the availability of cells

A.2 Published Topics

- goal(geometry_msgs/PoseStamped) - Static goal sent to navigation stack
- cmd_head (std_msgs/UInt8MultiArray) - Command to move the head towards the dynamic goal
- visualization_marker_array (visualization_msgs/MarkerArray) - Visualization markers to see the possible goals

A.3 Parameters

- visualization (bool, default: false) - Variable to choose whether or not the visualization of the goal options should be activated
- tracking (bool, default: true) - variable to choose whether or not the robot should track the dynamic goal target

---

1 Message type described in Appendix B
B.1 File:

dyn_goal_msgs.msg

B.2 Raw Message Definition

# Control message for the dyn_goal package
bool activated # whether or not the dyn_goal package should be active
string dyn_goal_tf # the name of the frame that should be followed
string origin_tf  # the name of the reference frame
float64 dist    # distance that should be kept from the dynamic goal
B.3 Compact Message Definition

bool activated
string dyn_goal_tf
string origin_tf
float64 dist