

POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering



Master's Degree Thesis

Person-aware autonomous navigation for an indoor sanitizing robot in ROS2

Supervisor

Marcello CHIABERGE

Candidate

Vittorio MAYELLARO

December 2022

*To my mother, for showing
me what it means to have
unwavering strength of will.
Have courage and keep smiling.*

Summary

In recent decades, intelligent systems have increasingly become part of our everyday lives to the point that robots able to perceive their surrounding environment and interact with it are not a dream anymore. With the advent of Industry 3.0, IT and computer technology were used to automate processes, but this is already the past because, nowadays, in the era of Industry 4.0, manufacturers are integrating new technologies, including IoT (Internet of Things), cloud computing and analytics, AI and machine learning into their production facilities and throughout their operations. For centuries people have been interested in building machines that mimic living beings. In living beings, perception and action are tightly coupled, which is also fundamental for the development of autonomous mobile robots. With the widespread use of robotic arms in industrial manufacturing, robotics has seen its greatest success to date. Despite their success, unfortunately, these commercial robots suffers from a fundamental disadvantage: lack of mobility. As intelligent agents capable of engaging actively with the industrial environment, mobile autonomous robots are being created in order to better achieve the level of flexibility envisioned by the Industry 4.0 revolution.

In many fields, including manufacturing, assembly, packing, transportation, search and rescue, healthcare and surgery, robots are used extensively. The usage of robots in social contexts, on the other hand, is still in an earlier stage. [1]. An extensive amount of research has been carried out in the area of localization, mapping, and exploration for autonomous mobile robots and almost all of the work has been aimed at environments where the robot is able to build 2D map based on its sensor's output. In particular, in the last few years, an increasingly number of research about autonomous navigation in social contexts has been carried out, which is significant in terms of how much this topic is currently being considered.

Person-aware indoor navigation focuses on the ability of the robot to automatically detect a person, its position and velocity in real time. Indeed, this is crucial for people-aware indoor mapping, obstacle avoidance and path planning.

This thesis project aims to improve the navigation strategy of a sanitizing robot by introducing a person-aware module. Through the usage of ROS2 the local and global costmap of Nav2 are modified to address the presence of people and maintain an acceptable social distance. The goal is to create an approach that makes it possible for autonomous robots to navigate indoors without relying on GPS or other external signals, which often fail in indoor environments.

A research about the autonomous navigation problem in social contexts has been carried out and the solution we propose uses computer vision to detect people and distinguish them from static obstacles. The developed algorithms are subsequently used to determine the optimal path through the environment by combining multiple probabilities of success based on each sensor's output.

This project is linked to a collaboration between the Interdepartmental Centre for Service Robotics of Politecnico di Torino (PIC4SeR) and the Innovation Centre of Intesa San Paolo. The proposed solution intends to build the groundwork for more extensive and focused approach to tackle the issue of autonomous navigation in social contexts.

Acknowledgements

First of all I would like to thank my supervisor Prof. Marcello Chiaberge, who supported me and helped me achieve this goal. With these few words, I would like to express my appreciation for good advice and helpfulness during both the thesis work and team activities.

None of this would have been possible without the great contribution of Andrea, Chiara and Mauro who have accompanied me along this journey. A genuine thank you for your teachings that have been and will always be an inspiration for my work. Thank you for showing me that it is always possible to improve.

There are no words to express my gratitude to the entire PIC4SeR, an extremely inspiring and stimulating environment that has become my second home for more than a few months. Sincere thanks to all PhD students and researchers of PIC4SeR for always being helpful and teaching me to look at the world from a new perspective.

My heartfelt thanks go to Team RoboTO, thank you for changing my life. In this environment I had the opportunity to experience what it means to passionately pursue and truly live my dreams. In particular, I would like to thank Francesco P., Luca, Francesco G., Corrado, Marco and Jiang; you have been more than university colleagues, thank you for being trusted companions. I hope we will be able to build a solid future together, this is just the beginning.

I would also like to sincerely thank Giuseppe, Alessandro, Daniele, Vito, Martina, Sabrina, Francesca, Mirko and Ilaria for being my second family here in Turin. Since we arrived in this new city, we have experienced many new adventures together

and we still have many more to come. Thank you for always being present in my life, never leaving me alone.

The most important thanks goes to my family. Entire pages would not be enough to express my gratitude to you. Thank you for making me the man I am today and thank you for all the sacrifices you have made for me, I have reached this milestone mainly thanks to you.

Lastly, I would like to thank all the people who are persistently pursuing their ambitions: do not give up, the world belongs to those who never stop dreaming.

Table of Contents

List of Tables	XI
List of Figures	XII
Acronyms	XV
1 Introduction	1
1.1 Overview	1
1.2 Robotics for sanification and disinfection	2
1.3 Robots in social contexts	3
1.4 Indoor navigation technologies	5
1.5 Related works	6
1.5.1 Visual Perception	8
2 ROS2 Framework	9
2.1 Overview	9
2.2 The ROS 2 Graph	11
2.3 RViz	13
2.4 Gazebo	14
2.5 ROS2 Navigation Stack	15
2.5.1 Costmap2D	17
3 Hardware architecture	21
3.1 Overview	21
3.2 Ventilation and purification system	22
3.3 KUKA youBot Platform	22

3.4	RPLIDAR A2	23
3.5	Time-of-Flight sensor	24
3.6	OAK-D stereo camera	25
3.7	Intel RealSense T265	26
3.8	CO_2 sensor	27
4	Implementation	28
4.1	Overview	28
4.2	Nodes, custom messages and topics	29
4.2.1	Messages	29
4.2.2	Nodes and topics	30
4.3	People detection and tracking	32
4.3.1	MobilenetSSD	32
4.3.2	Object tracking	33
4.4	Social Costmap Plugin	35
4.4.1	Costs assignment	39
5	Simulations and tests	42
5.1	Overview	42
5.2	Environment configuration on Gazebo	43
5.3	Simulation scenarios	45
5.4	Social layer parameters	48
5.5	Set of metrics	49
5.6	Simulation results	50
5.6.1	Front passing test	51
5.6.2	Diagonal passing test	52
5.6.3	Orthogonal passing test	54
5.6.4	Wall test	55
5.6.5	Cut test	57
5.6.6	X test	58
5.6.7	Tab test	61
6	Conclusions and future developments	64

List of Tables

1.1	Social zones in proxemics according to [6]	4
2.1	Ported and new packages in Nav2	16
4.1	Virtual methods for the implementation of new costmap layers [30]	37
5.1	Parameters of the Gaussian function	48
5.2	Set of metrics for front passing test	52
5.3	Set of metrics for diagonal passing test	53
5.4	Set of metrics for orthogonal passing test	55
5.5	Set of metrics for wall test	56
5.6	Set of metrics for cut test	58
5.7	Set of metrics for X test	60
5.8	Set of metrics for Tab test	63

List of Figures

1.1	Some mobile robotics application examples	2
1.2	Mobile robotics in the Healthcare	3
1.3	EMIEW2 by HITACHI	4
1.4	Indoor navigation in an office-like environment	6
1.5	Example of objection detection results using MobileNet SSD [12] . .	8
2.1	ROS2 distributions and their EOL	10
2.2	ROS 2 Foxy Fitzroy layered architecture [15]	10
2.3	PoseStamped message	11
2.4	ROS2 topic	11
2.5	ROS2 service	12
2.6	ROS2 action	12
2.7	rqt_graph visualization for a basic turtlesim example	13
2.8	Example of RViz visualization	14
2.9	Example of Gazebo visualization	14
2.10	Nav2 architecture [20]	15
2.11	A stack of costmap layers, showing the numerous contextual behav- iors achievable with the layered costmap approach [8]	18
2.12	Assignment of Inflation Layer costs [21]	20
3.1	Mobile robot deployed for air sanitation	21
3.2	VORT ARIASALUS 200 ventilation and purification system	22
3.3	KUKA youBot omni-directional mobile platform	23
3.4	RPLIDAR A2 360° Laser Scanner	24
3.5	RViz visualization of lidar measurements in simulation	24

3.6	VL53L5CX multizone ranging sensor [23]	25
3.7	OAK-D stereo camera [24]	26
3.8	Intel RealSense T265 Tracking Camera [25]	26
3.9	Infineon Technologies XENSIV™ PAS CO2 [26]	27
4.1	General pipeline to understand the system elements' interactions	29
4.2	Various applications of MobilNet models [12]	32
4.3	MobilenetSSD layered architecture [28]	33
4.4	Object tracker inputs and outputs [29]	34
4.5	Visualization of the detections in our solution	35
4.6	Social Layer virtual methods main tasks step by step	38
4.7	Local reference frame for costs assignment according to an asymmetric 2D Gaussian function	40
4.8	Visualization of a person moving towards the robot in a simulated world	41
5.1	Cafe world visualization in Gazebo (left) and its map (right)	43
5.2	Visualization of the spawned actor in Gazebo	44
5.3	Visualization of the spawned Nexus 4WD Mecanum robot in Gazebo	44
5.4	Front passing qualitative schematization	45
5.5	Front passing qualitative schematization	45
5.6	Orthogonal passing qualitative schematization	46
5.7	Wall test qualitative schematization	46
5.8	Cut test qualitative schematization	47
5.9	X test qualitative schematization	47
5.10	Tab test qualitative schematization	48
5.11	Front passing test robot and person path representation	51
5.12	Front passing test person-robot distance over time graph	51
5.13	Diagonal passing test robot and person path representation	52
5.14	Diagonal passing test person-robot distance over time graph	53
5.15	Orthogonal passing test robot and person path representation	54
5.16	Orthogonal passing test person-robot distance over time graph	54
5.17	Wall test robot and person path representation	55
5.18	Wall test person-robot distance over time graph	56

5.19	Cut test robot and person path representation	57
5.20	Cut test person-robot distance over time graph	57
5.21	X test robot and person path representation	58
5.22	X test person1-robot distance over time graph	59
5.23	X test person2-robot distance over time graph	59
5.24	Tab test robot and person path representation	61
5.25	Tab test person1-robot distance over time graph	61
5.26	Tab test person2-robot distance over time graph	62

Acronyms

AI

artificial intelligence

DNN

deep neural networks

ROS

robot operating system

BT

behaviour tree

IMU

inertial measurement unit

TOF

time-of-flight

Chapter 1

Introduction

1.1 Overview

To date, mobile robotics is one of the fastest growing areas of scientific research. Due to their capabilities, mobile robots can nowadays replace humans in several tasks. A robot is autonomous when it has the ability to determine the actions to be taken to perform a task, using a perception system that helps it. It also needs a cognition unit or a control system to coordinate all the subsystems that comprise the robot. The domains of locomotion, perception, cognition, and navigation make up the foundation of mobile robotics [2]. One of the most important tasks of an autonomous system of any kind is to acquire knowledge about its environment. This is accomplished by taking measurements with a variety of sensors, then deriving meaningful data from those results [3]. Signal analysis and specialized fields like computer vision and sensor technology are involved in perception. To accomplish the goals of the mobile robot, cognition is in charge of evaluating the input data from sensors and taking the appropriate actions. Thus, It is responsible for the control system architecture. Navigation requires knowledge of planning algorithms, information theory, and artificial intelligence [2]. Navigation ability is thus the result of the combination of perceptual data, localization, cognition, and motion control.



(a) BACCHUS by Robotnik
<https://robotnik.eu/projects/bacchus-en/>



(b) MiR250 Hook by Mobile Industrial Robots
<https://www.mobile-industrial-robots.com/solutions/mir-applications/mir-hook-250/>



(c) KMR QUANTEC by KUKA
<https://www.kuka.com/en-de/products/mobility/mobile-robots/kmr-quantec>



(d) Spot by Boston Dynamics
<https://www.bostondynamics.com/products/spot>

Figure 1.1: Some mobile robotics application examples

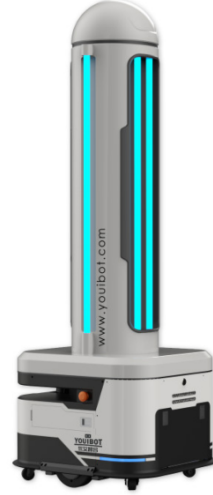
1.2 Robotics for sanification and disinfection

From a technical standpoint, the use of robots is a part of the proactive strategy to stop the spread of infectious diseases and shield healthy individuals from infection. In an effort to combat the challenges and hardships brought on by the COVID-19 pandemic, the development of service robots in the healthcare industry has been sparked. Over the past year, a variety of robots have been used to carry out a

number of crucial duties and functions using a variety of techniques [4].



(a) Mobile Robot for Air Disinfection by
NOLL Sondermaschinenbau
<https://www.noll-sondermaschinenbau.de/>



(b) ARIS-K2 by Shenzhen Youibot
Robotics <https://en.youibot.com/>

Figure 1.2: Mobile robotics in the Healthcare

Currently, human personnel carried out the majority of indoor disinfection in contaminated environments, including hospitals. Despite wearing protective gear, frontline healthcare workers are still exposed to the infection when they have direct patient contact. This unquestionably raises the possibility that a healthy individual, either working in the health sector or not, will be exposed to contagious virus. In contrast, disinfection robots might result in a more safe, efficient and effective disinfection process [5].

1.3 Robots in social contexts

Nowadays, every industry has undergone a fundamental change as a result of the use of robots in industrial settings. In many fields, including manufacturing, assembly, packing, transportation, search and rescue, healthcare, surgery, and laboratory research, robots are used extensively. On the other hand, the use of robots in social contexts is still in its early stages [1]. Robots are gradually entering into public

areas to help people with their duties and while coming into contact with them it is crucial that their general motion behavior inspires confidence and causes people the least amount of distress possible. The robot should move in a way that is perceived as safe and should make people feel at ease. [6]



Figure 1.3: EMIEW2 by HITACHI

There are social norms that govern how people should move past one another in public contexts. A robot should, at the very least, exhibit motion behavior that adheres to identical rules and it takes meticulous planning to make the robot behave in a way that is convincingly human-like and encourages confidence. According to Pacchierotti et al. [6], who studied the rules from proxemics used to design a passing strategy, the space around a person is divided into four distance zones:

Social Zone	Distance
Intimate distance	up to 45 cm
Personal distance	from 0.45m to 1.2m
Social distance	from 1.2m to 3.5m
Public distance	beyond 3.5m

Table 1.1: Social zones in proxemics according to [6]

Pacchierotti et al. also suggest three parameters as the most significant for the robot passing behaviour:

- Robot speed: the average forward speed of the robot during the passing maneuver.
- Signaling Distance: the person-robot distance along the robot direction of motion.
- Lateral Distance: the distance that the robot maintains from the person at the passing point along the direction perpendicular to that of the corridor

Estimating the human's location and pose in the frame is the initial step in several pipelines for understanding human behavior. Furthermore, the capability of robots to predict future actions is crucial for understanding human behavior. Early research on trajectory forecasting for social robot navigation among humans achieved impressive results. From Kalman filters to solutions that use deep learning and reinforcement learning, a lot of innovation is being carried on in this context.

1.4 Indoor navigation technologies

Autonomous indoor navigation is a subfield of the study of autonomous robotics and it is one of the greatest challenges of the coming decades in robotics. It is becoming increasingly necessary for mobile robots to be able to navigate autonomously in a variety of different environments and without a GPS, such as in dense urban areas, homes, tunnels, office buildings etc.

Autonomous systems must be able to navigate by responding and adapting to their changing environments; they must be robust to failure of sensors and/or control methods and they must adapt as the environment changes. LiDAR could be used to identify common obstacles, but what if we wanted to distinguish between how a robot behave when it encounters specific impediments, such as people? The most obvious solution for this would be to use video data from a camera mounted on the robot. Computer vision is a technique of processing information from a video sequence or image sensor to recognize objects, determine their 3D shape and motion, and track their positions. Computer vision provides a potentially powerful

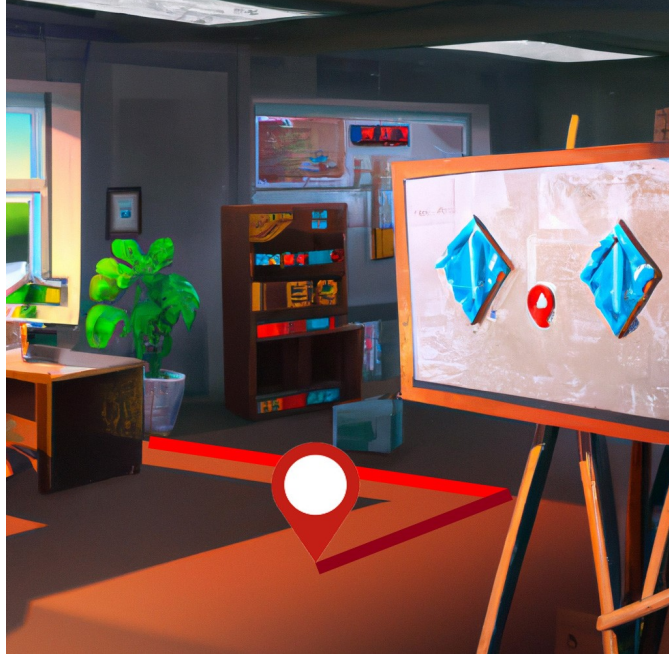


Figure 1.4: Indoor navigation in an office-like environment

tool for developing mobile robots that are able to navigate indoors on their own. However, occlusion and shadows of obstacles, huge computational demands by models for optimal control and obstacle avoidance together with high sampling rate requirements for vision sensors are still huge challenges to be solved.

1.5 Related works

For a long time, the navigation of mobile robots, both remotely controlled and autonomous, has been of considerable academic interest. In recent years, the robotics research community is extensively working on autonomous navigation of robots, more specifically on person-aware solutions. Although numerous problem scenarios are either devoid of people or merely consider them to be static impediments, they are nevertheless useful for studying general navigation skills, which are essential for social robots. Additionally, many traditional navigational techniques have been modified to incorporate a social component while still utilizing the same underlying algorithms, as also proposed in our solution. The long-term objective is

to develop robots that can comprehend human behavior and social conventions, using these inputs to communicate in a way that is intuitively understood by humans [1]. In addition to the standard obstacle avoidance challenges, navigating around people presents unique challenges. For instance, a chair perception of the robot’s path around it is not of interest; nevertheless, a person who believes the robot is approaching too closely may react negatively.

Robots can now navigate around people using a variety of techniques. Kirby [7] addressed social navigation issues by adjusting the robot’s path in consideration of people’s personal space and other social norms. His method adds values to the costmap in the vicinity of the human’s identified location in accordance with a 2d Gaussian, presumably accounting for the person’s direction of travel. In general, this strategy works effectively since it directs the robot to follow smooth paths far from people. Unfortunately, the laser-based tracking technology for person detection and tracking employed by Kirby didn’t work very well in practice. Indeed, a better solution would have been employing a multi-sensor strategy to locate people more accurately. In other words, even if a laser can offer rather precise range readings, it might be challenging to identify which signals are related to people. The tracker might reach more accuracy by combining the laser ranges with, for instance, a vision system that recognizes people and their positions.

David V. Lu et al. created and implemented a new method in the ROS Navigation Stack called layered costmaps [8]. It records different types of constraints or obstacles by processing costmap data in semantically distinct layers, and then it alters a master costmap that is used for path planning. A significant reason for including more complex costs in the costmap is modelling limitations brought on by human-robot interaction. They devised a Proxemic Layer that employs Kirby’s mixture of Gaussians model to prevent robots from approaching people too closely using their location and velocity (as determined by laser scans of the person’s legs) [9]. In contrast, we implemented the proxemic layer in the ROS2 Navigation Stack employing the Gaussian functions as proposed by Kirby and using the computer vision to detect and track people in the area of interest.

1.5.1 Visual Perception

To date, robotics relies heavily on visual perception, which enables a full and detailed knowledge of the surrounding environment. The development of Deep Neural Networks (DNN) and Artificial Intelligence (AI) is still ongoing on a global scale. The employment of visual perception on robots is, together with the ever-present need to entrust robots with tasks that would normally be performed by human operators, what is bringing autonomous navigation in social contexts so much in vogue. Indeed, the increasingly evident improvement of computer vision technique for real-time people detection is fundamental for the accuracy of the developed solution in social contexts. For instance, G. Doisy et al. [10] developed an adaptive person-following algorithm based on depth images and mapping. In the proposed solution, they used depth images for real-time human pose recognition, using the position of the head joint to estimate the ground coordinates of the detected person.

Standard deep learning-based object detection algorithms such as SSD [11] networks inform the robot about the presence of objects, in our context, people, in its field of view. Our pipeline combines both RGB and depth images, employing mobilnet-ssd and a built-in tracker of our depth camera to detect and track humans in the field of view of the robot in order to treat people differently from static obstacles.



Figure 1.5: Example of objection detection results using MobileNet SSD [12]

Chapter 2

ROS2 Framework

2.1 Overview

The Robot Operating System (ROS) is an open-source middleware that has undergone rapid development and has been widely used for robotics applications. It was built almost entirely from scratch and has been maintained by Willow Garage and the Open Source Robotics Foundation since 2007. It provides numerous libraries and tools to help software developers create robotics applications. ROS, on the other hand, must meet the increasing demands of real-time distributed embedded systems and only runs on a few operating systems. To satisfy these requirements, it has undergone a significant upgrade to ROS2. [13]. There are numerous ROS2 versions, and the most of them receive ongoing updates and support until their EOL date (End of life). In this thesis project ROS2 Foxy has been used. ROS2 was developed in order to leverage what is great about ROS1 and improve what is not. The most intriguing aspect of this updating process is that it is always possible to utilize a bridge to connect the most recent ROS2 version in use with ROS1 such that neither one loses any functionality. Greater control over the condition of the ROS 2 system is possible with Lifecycle (or Managed) nodes [14], which are one of the most important novelties of ROS 2. A managed node can be thought of as a "black box" because it offers a known interface and operates in accordance with a known life cycle state machine. This gives node developers flexibility in how they implement managed life cycle features and guarantees that any tools developed for managing




Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023

Figure 2.1: ROS2 distributions and their EOL

nodes can be used with any compliant node. Currently, ROS 2 is employed in a variety of applications, including autonomous cars, industrial robots, and humanoid robots mainly for the purposes of navigation, control, motion planning, vision, and simulation. One of the most useful characteristics of ROS 2 is that it provides an high level of abstraction of the hardware from the software which means that users create robotics applications without needing physical hardware on hand. ROS 2 Foxy Fitzroy and its basic layered architecture is shown in Figure 2.2.

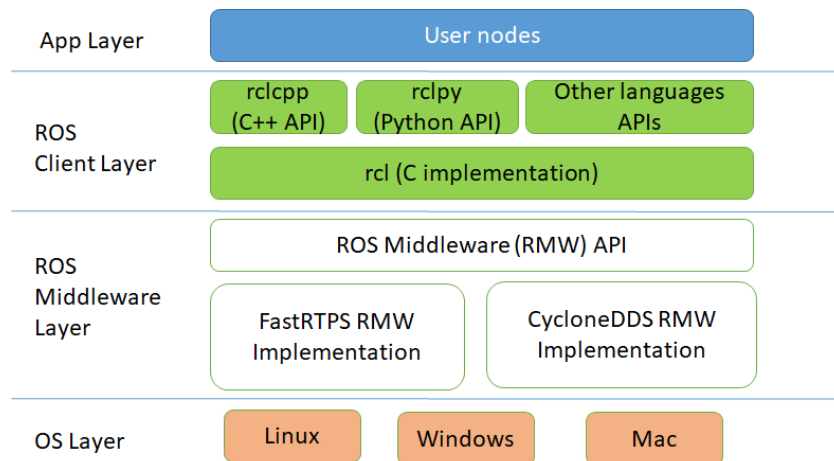


Figure 2.2: ROS 2 Foxy Fitzroy layered architecture [15]

2.2 The ROS 2 Graph

Developers get access to a diverse number of communication patterns using RCLCPP. These are intended to encapsulate typical node-to-node communication patterns. The ROS 2 graph [16] is a network of ROS 2 elements that process data concurrently. If all of the executables were mapped, it would include all the links between them. This graph is composed by the following elementary components:

- **Nodes:** A ROS node is basically a process running inside the application, usually responsible for a single purpose (e.g. acquire camera data), that can send and receive data to other nodes via topics, services, actions or parameters.
- **Messages:** A ROS message is a simple data structure used to store data useful for nodes communication. They can be standard-type (e.g. String, Int, Bool) or specifics for the application of interest as shown in figure 2.3.

```
# A Pose with reference coordinate frame and timestamp

std_msgs/Header header
Pose pose
```

Figure 2.3: PoseStamped message

- **Topics:** ROS topics are the means through which nodes can communicate by exchanging messages. Since topics use anonymous publish/subscribe logic, the creation and acquisition of information are separated and each node is not aware of the "identity" of the nodes it is communicating with. In general, a node may simultaneously subscribe to any number of topics while publishing to others topics. Each topic can only receive messages of the same type.

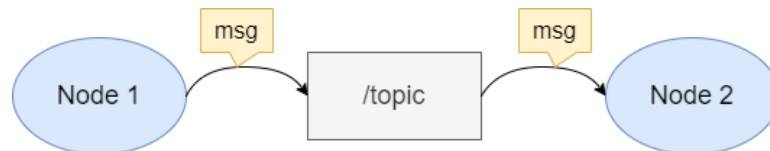


Figure 2.4: ROS2 topic

- **Services:** Unlike ROS topics, where the publisher-subscriber model is used, ROS services are built on a call-and-response synchronous model. Indeed, services only deliver data when they are specifically requested by a client. Services are defined using srv files.

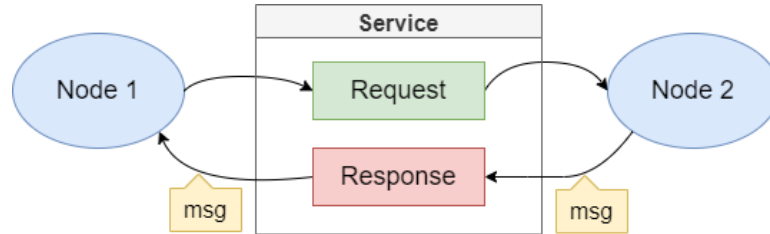


Figure 2.5: ROS2 service

- **Actions:** ROS actions are another way to perform node communications particularly suited for long running tasks. Action clients communicate with an action server in a manner similar to services in order to accomplish a task and receive a response. Furthermore, feedback is useful to implement a mechanism to communicate the small steps a goal has taken whereas the result is sent when the goal has been completed. Unlike services, an action server notifies the client of its progress as the action is being carried out.

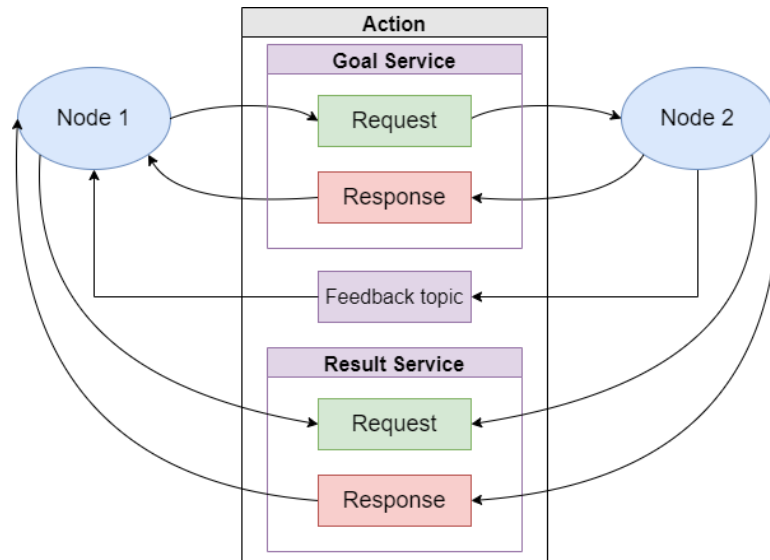


Figure 2.6: ROS2 action

- **Parameters:** A ROS parameter is a node's configuration value. In general, parameters can be seen as node settings. Indeed, a parameter server is best used for static data.

These basic building blocks allow us to create extremely complicated systems, and using the command `rqt_graph` to represent all the system's nodes and topics is extremely useful to understand its overall behaviour.

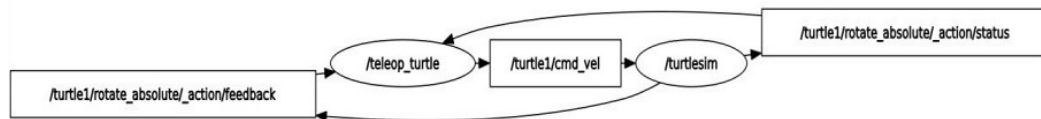


Figure 2.7: `rqt_graph` visualization for a basic turtlesim example

2.3 RViz

RViz [17] is a ROS graphical interface that allows to visualize numerous of information published on topics. To create a custom RVIZ configuration, such as in figure 2.8, a lot of plugins can be employed:

- **Global Options:** allows to choose general settings like the background colour and fixed frame.
- **Grid:** allows to visualize a grid laying on the xy plane.
- **RobotModel:** allows to visualize the Robot Model according to its description from the URDF model.
- **TF:** shows the position and the orientation of all the frames that compose the TF Hierarchy.
- **Laser Scan:** shows data from a laser scan, with various options for rendering modes, accumulation, etc.
- **Grid Cells:** shows cells from a grid, usually obstacles from a costmap from the navigation stack.

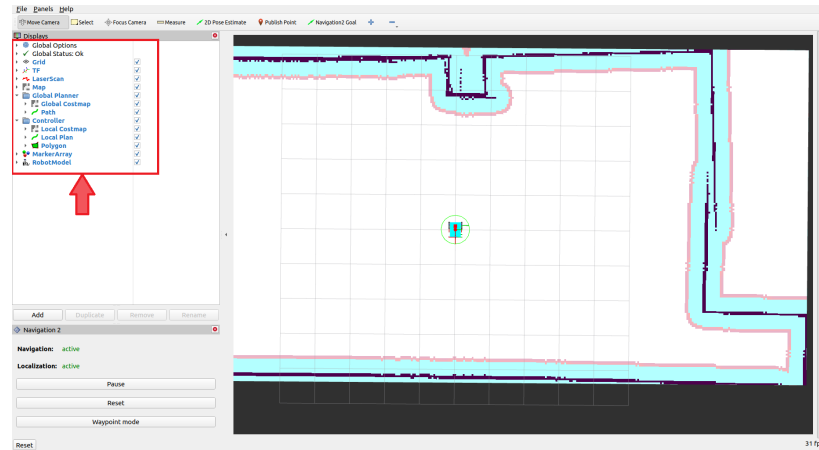


Figure 2.8: Example of RViz visualization

2.4 Gazebo

Gazebo is an open-source 3D robot simulator widely used in both industry and academics. It does physics calculations, provides sensor data, and allows actuator control. The simulation environment offered by Gazebo includes a full set of development libraries and cloud services. It is also particularly well suited for testing control strategies safely. Indeed, the primary uniqueness of this software is that it enables robot system development without the need for physical hardware and, consequently, without endangering the robot and the surrounding environment.

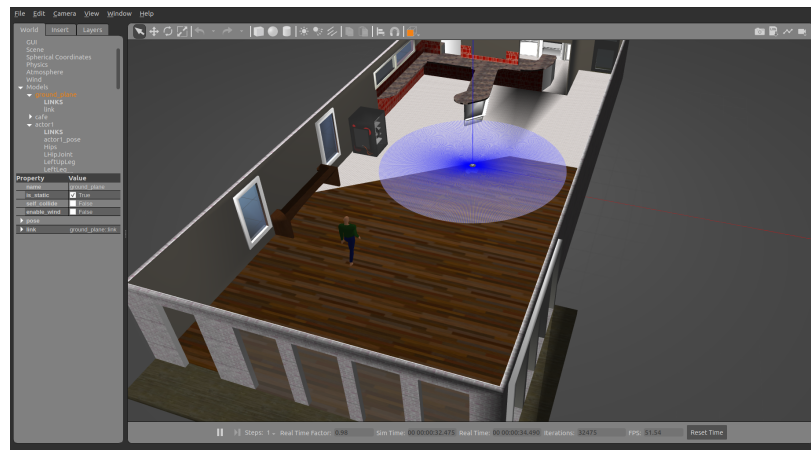


Figure 2.9: Example of Gazebo visualization

2.5 ROS2 Navigation Stack

In general, odometry and sensor data are gathered by a navigation stack, which then produces velocity signals to be sent to a robot. Furthermore, for a robot's shape and dynamics to operate at a high level, the navigation stack must be set up properly. In this thesis project, the ROS 2 navigation stack (also known as Nav2) has been employed. The core middleware employed for Nav2 [18] is ROS 2. Actions are widely used in the ROS 2 navigation stack to manage ongoing processes. In this stack, action servers are utilized to send and receive NavigateToPose action messages from the top-level BT navigator, which is the true novelty of Nav2. In order to plan paths, control efforts, and recoveries, the BT navigator also uses them to connect to the ensuing smaller action servers. Behavior trees (BT) are used more frequently in sophisticated robotic tasks since they are better suited for these tasks compared to the single process state machine that was employed for the ROS 1 navigation stack. They are organized as a tree of tasks to be performed. It creates a framework that is more scalable and easy for people to understand for defining multi-step or many-state applications. The planner, behavior, smoother, and controller servers are four of Nav2's action servers. [19] To sum up, comparison

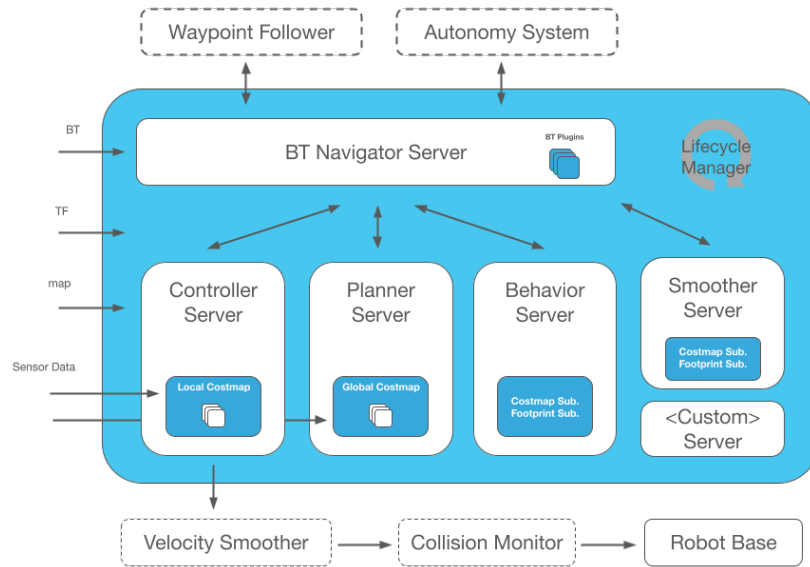


Figure 2.10: Nav2 architecture [20]

between the new and ported packages from ROS1 navigation stack to Nav2 are

listed in table 2.1 [20].

Nav2 ported and new packages
<code>amcl</code> \rightarrow <code>nav2_amcl</code>
<code>map_server</code> \rightarrow <code>nav2_map_server</code>
<code>global_planner</code> \rightarrow <code>nav2_planner</code>
<code>Navfn:</code> \rightarrow <code>nav2_navfn_planner</code>
<code>DWA:</code> \rightarrow <code>DWB</code>
<code>nav_core</code> \rightarrow <code>nav2_core</code>
<code>costmap_2d</code> \rightarrow <code>nav2_costmap_2d</code>
NEW: <code>nav2_bt_navigator</code> : replaces <code>move_base</code> state machine
NEW: <code>nav2_lifecycle_manager</code> : Handles the server program lifecycles
NEW: <code>nav2_waypoint_follower</code> : Execute navigation through waypoints
NEW: <code>nav2_system_tests</code> : Set of integration tests and basic tutorials
NEW: <code>nav2_rviz_plugins</code> : RViz plugin to interact with Nav2
NEW: <code>nav2_experimental</code>
NEW: <code>navigation2_behavior_trees</code>

Table 2.1: Ported and new packages in Nav2

2.5.1 Costmap2D

By means of the ROS 2 navigation stack, the robot uses a costmap representation of the environment as a foundation on which planner and controller servers compute a preferred path through obstacles while minimizing a cost function. This is possible in Nav2 thanks to the specific package `nav2_costmap_2d`, which subscribes to the sensor data topics and builds a 2D or 3D space representation as an occupancy grid: each cell can assume an integer value value between 0 and 255 depending on the sensor data. In a classic costmap, which we refer to as a monolithic costmap, all the data is kept in a single grid of values. Due to its simplicity (there is only one area to read from and write values to), in the past the monolithic costmap has become the dominant technique. As a result, the costmap's values lost a lot of their semantic context. David V. Lu et al. [8] developed and fully implemented the layered costmaps approach, which divides the processing of costmap data into levels that are separated by semantics in order to boost the capability to deal with numerous and different contexts. The layered costmap is introduced in order to overcome the restrictions the monolithic costmap imposes, which can be summarized as follows:

- Most of the information in the costmap is stored in one location, a conflict between the sensor data and the values in the global costmap (i.e. provided by the static map) cannot be easily solved since the costmap update strategy depends on the origin of the data and other semantic information.
- It is challenging to determine how long any specific cost value has been present in the monolithic costmap due to the absence of semantic information. Regardless of how much of that space was really updated, in practice this means to conservatively estimate a region of the map that encompasses the whole area that may have been updated and update it roughly in a square around the robot.
- It is infeasible as the number of data sources increases.
- It limits the varieties of information that can be used, allowing for only one interpretation of the costmap's values.

Indeed, the layered costmap approach guarantees the following benefits:

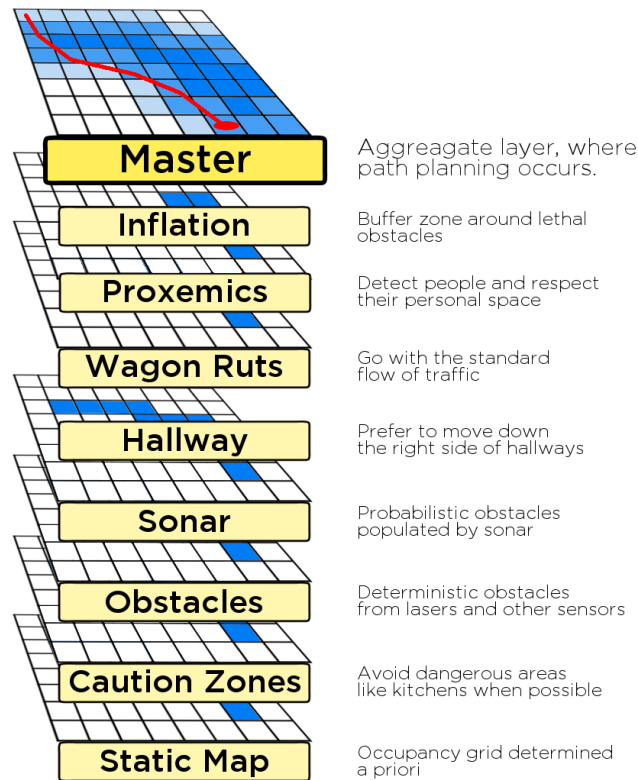


Figure 2.11: A stack of costmap layers, showing the numerous contextual behaviors achievable with the layered costmap approach [8]

- **Clearer Update Step:** the layered costmap approach adds various types of costmap information to distinct levels, separating the update phase more clearly and removing discrepancies between the competing costmap information sources.
- **Dynamic Update Areas:** Only areas of the map that each layer deems required is updated by the layered costmap. As only values inside a predetermined bounding box are updated, the costmap is more stable and efficient.
- **Ordered Update Process:** The explicit ordering of the layered costmap allows each costmap to be designed to combine the value from the previous layer with the value from the subsequent layer as a maximum, minimum, or other mathematical function of the two.

The 2D costmap implementation provided by `nav2_costmap_2d` gathers sensor

data from the outside environment, creates a 2D or 3D occupancy grid from the data, and then inflates costs depending on the occupancy grid and a user-specified inflation radius. The basic costmap layers are:

- Static Layer: Gets static map provided at launch and loads the corresponding occupancy information into the costmap
- Obstacle Layer: Updates continuously a 2D costmap using raycasting from 2D laser scans to empty spaces detected as free
- Inflation Layer: Inflates lethal obstacles in costmap out from occupied cells with exponential decay. In order to accomplish this, 5 distinct symbols for costmap values are defined [21]:
 - A "lethal" cost indicates that a cell contains a genuine impediment. The robot would clearly be in collision if its center were in that cell.
 - A cell has a "inscribed" cost if it is closer to an actual obstacle than the robot's inscribed radius. Therefore, if the robot center is in a cell that is at or over the written cost, the robot is undoubtedly colliding with some obstacle.
 - Similar to "inscribed," "possibly circumscribed" cost uses the robot's circumscribed radius as the cutoff distance. It thus depends on the orientation of the robot whether it collides with an obstruction or not if the robot center is located in a cell at or above this value.
 - "Freespace" cost is zero, meaning that the robot shouldn't be prevented from travelling in that zone.
 - A cell's "unknown" cost indicates that there is no information available.
 - Depending on how far they are from a "Lethal" cell and the decay function the user provides, all other costs are given a value between "Freespace" and "Possibly restricted."

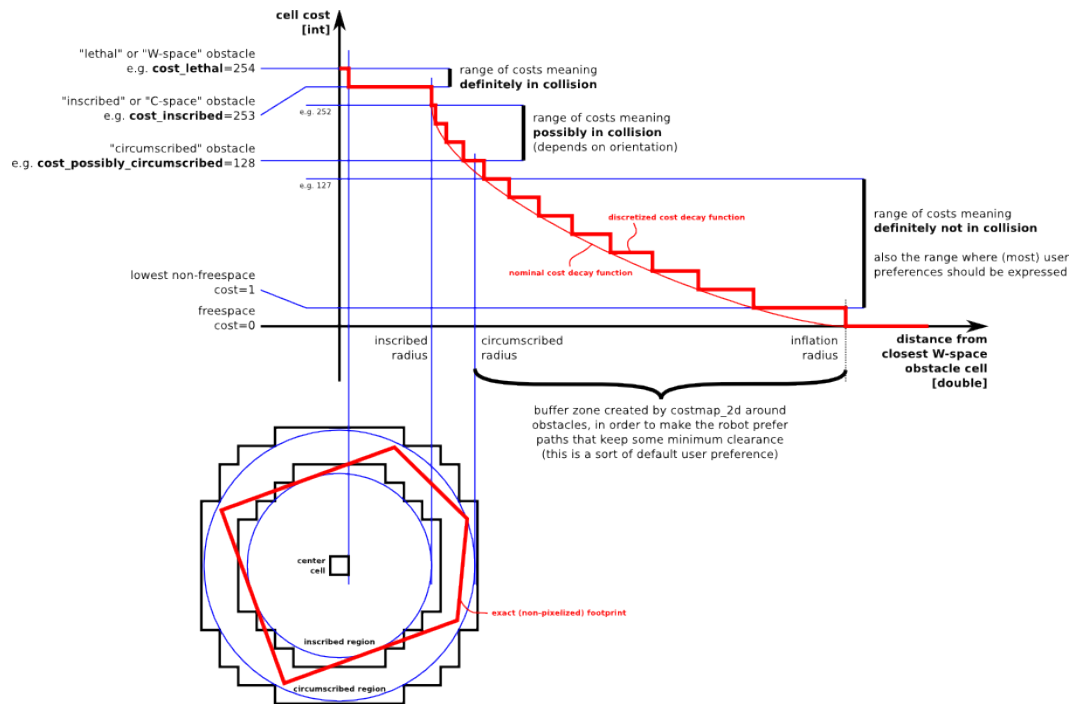


Figure 2.12: Assignment of Inflation Layer costs [21]

Chapter 3

Hardware architecture

3.1 Overview

This chapter focuses on providing a general overview of all the hardware required for the thesis project. A wide range of sensors are needed to accurately sense the surroundings so that the robot can operate in the indoor environment. The mobile robot to be deployed for air sanitation is shown in figure 3.1.



Figure 3.1: Mobile robot deployed for air sanitation

3.2 Ventilation and purification system

The mobile robot's ventilation and purification system [22] draws in air and purifies it by passing it through a filter system that retains the impurities in it. In addition, the sanitizing device (photocatalysis) with which it is equipped exerts an antibacterial and antiviral action, ensuring high levels of room healthiness. When in ventilation mode, it restores the optimal oxygen concentration while reducing relative humidity and carbon dioxide levels to provide the required air exchange. Furthermore, the purification unit is also effective against the SARS-CoV2 responsible for the COVID-19 pandemic.



Figure 3.2: VORT ARIASALUS 200 ventilation and purification system

3.3 KUKA youBot Platform

The KUKA youBot platform guarantees a 360-Degree freedom of movement (x , y , θ) with its omnidirectional wheel system. Despite the limitations on the surface they can be deployed on, omnidirectional wheels are particularly beneficial to use in intelligent and small autonomous robots, especially in indoor environments.

The most significant advantage of omniwheels is the holonomic movement (i.e. the controllable degree of freedom is equal to total degrees of freedom).

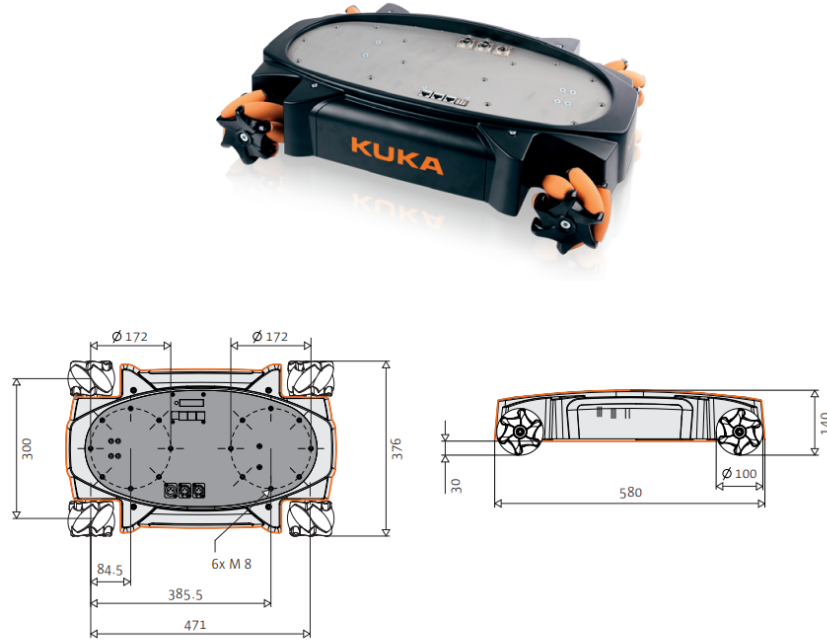


Figure 3.3: KUKA youBot omni-directional mobile platform

3.4 RPLIDAR A2

The RPLIDAR A2 360° Laser Scanner uses a low-cost laser triangulation measurement technique created by SLAMTEC, performing very well in a variety of indoor and outdoor conditions. LiDAR (Light Detection and Ranging) systems produce point clouds (i.e. a set of 3D points) and measure distances using laser light waves. The distance is calculated by timing the flight of a light pulse, and gyros measure the laser's direction as it is being transmitted. An object can be located by comparing the measured point cloud with one that is recorded in a database. In figure 3.5 it is possible to observe the implementation of a LiDAR on RViz.



Figure 3.4: RPLIDAR A2 360° Laser Scanner

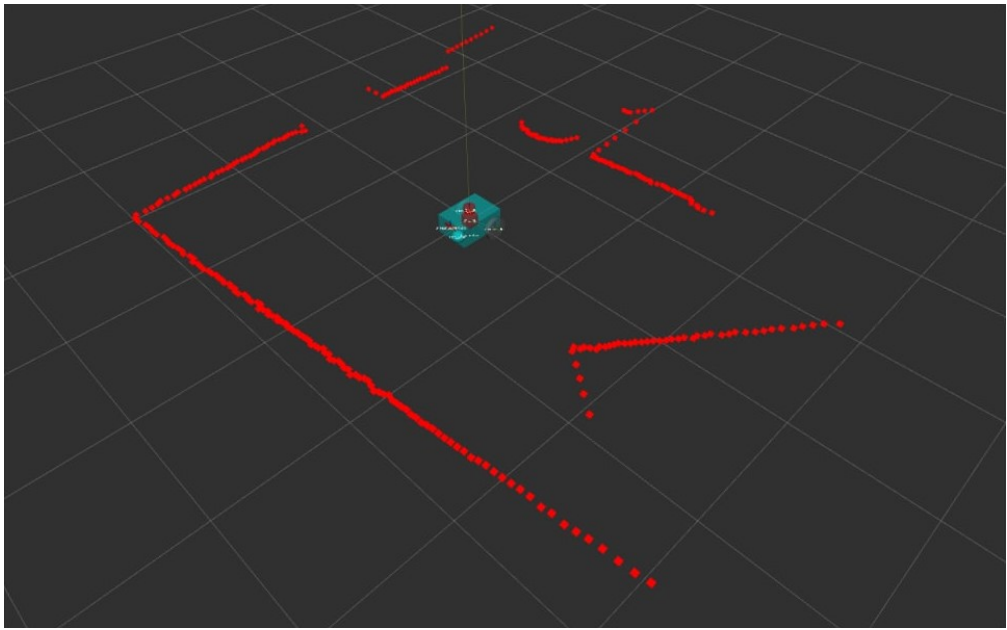


Figure 3.5: RViz visualization of lidar measurements in simulation

3.5 Time-of-Flight sensor

When emitting light, Time-of-Flight (ToF) is particularly effective for range-finding and distance sensing. It offers much wider range, quicker readings, and more accuracy as compared to ultrasound. In our robot, a VL53L5CX [23] time-of-Flight 8x8 multizone ranging sensor with wide field of view is employed to achieve the best ranging performance. Its most significant features are:

- easy integration thanks to the flexible power supply options

- small dimensions and low power consumption
- fast and accurate distance measurements



Figure 3.6: VL53L5CX multizone ranging sensor [23]

3.6 OAK-D stereo camera

In this thesis project we have employed the Luxonis OAK-D camera [24] to acquire image data, estimate depth, and execute the people detection and tracking model. OAK—D is a spatial AI powerhouse that can run complex neural networks simultaneously while obtaining depth from two stereo cameras and color information from a single 4K camera in the middle. It is worth to highlight that the algorithm used to compute the depth is often run on the host platform. A stereo camera operates on the same foundation as a human eye: binocular vision. The depth of an item is determined by stereo disparity, which is the process of estimating the distance to an object comparing its locations as perceived by two separate cameras. Furthermore, the cameras resolution is directly related to the depth measurements accuracy. Our OAK-D stereo camera consists in:

- Two monocalera with a 1280x800p resolution for depth estimation
- A color camera with a 4032x3040p resolution

- An Intel Movidius Myriad X Vision Processing Unit that can run AI models on the camera
- An inertial measurement unit (IMU) that can be used to determine the orientation of the camera in the 3D space.



Figure 3.7: OAK-D stereo camera [24]

3.7 Intel RealSense T265

Intel RealSense T265 Tracking Camera [25] is employed for self-localization of the mobile robot. It has two fisheye lenses for feature detection but it is not a depth camera. Furthermore, it has low power consumption and a small form factor which allows to put it in the bottom front of the robot to better exploit its functionalities.



Figure 3.8: Intel RealSense T265 Tracking Camera [25]

3.8 CO_2 sensor

Infineon Technologies XENSIV™ PAS CO2 [26] is a compact carbon dioxide (CO_2) sensor, designed on the basis of the photoacoustic spectroscopy (PAS) concept, that saves more than 75 percent space of current genuine CO_2 commercial sensors. Its high accuracy and robust performance together with its small form factor allow for an easy integration into both low and high-volume applications.

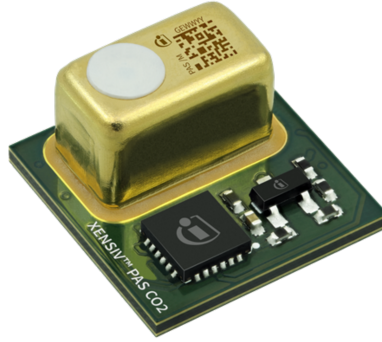


Figure 3.9: Infineon Technologies XENSIV™ PAS CO2 [26]

Chapter 4

Implementation

4.1 Overview

This chapter is intended to indicate the general approach we pursued for the implementation of our solution. The work presented in this chapter refers partially to [27] which offered me a great way to start the implementation of a "proxemic layer" in ROS2. The OAK-D camera will be used for people detection and tracking and the people position and velocity will be transformed from the `camera` reference frame into `odom` reference frame. These information will be published on the `ppl_odom` topic to which the global and local costmap are subscribed. The social layer will use those information to assign costs around each detected person according to a 2D Gaussian shape with variances proportional to the people velocity and oriented in the direction of their movement. In Figure 4.1, it is possible to observe a general pipeline schematization useful to give a high-level overview of the proposed solution. It is worth to highlight that a first implementation has been carried out in simulation in order to get a qualitative idea of the general performance of the social layer regardless of the people detection technique, and this will be discussed in Chapter 5. Then, an implementation of our solution on the real robot has been carried out and led to promising but not sufficient results, which will be the main scope for future improvements.

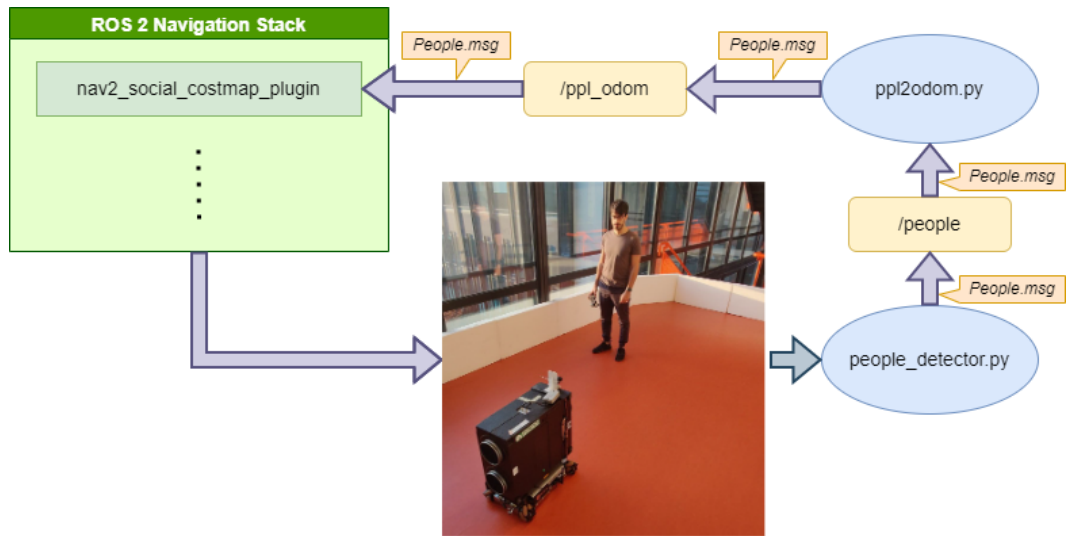


Figure 4.1: General pipeline to understand the system elements' interactions

4.2 Nodes, custom messages and topics

4.2.1 Messages

The communication of the detected people information occurs through two message types:

- Person.msg

```
int16 id
geometry_msgs/Point position
geometry_msgs/Point velocity
bool islost
```

- People.msg

```
std_msgs/Header header
Person[] people
```

The `id` of a person is an identifier directly related with the id of the bounding box got from the tracklets data. A person `position` and `velocity` are expressed

as three-dimensional vectors (i.e. x,y,z) with components in the reference frame of interest. Due to the limited field of view of the camera, during the movement of the robot, people will not be detected anymore even if they are near the robot (e.g. when passing on the side of a person). A possible solution would be creating custom heuristics for the planner or employing more than one camera on the robot to get an overall wider field of view. However, a very elementary solution, that is employed in the real case solution taking into account simplicity and hardware limitations, is to suppose that the person has virtually remained in the last position it was detected in, for a certain amount of time. That's the main reason for the introduction of the flag `islost` in the `Person.msg`. When a person is not detected anymore its id will be set to a new value and the transformation `camera` \rightarrow `odom` will not be performed for the remaining time (otherwise, since the virtual person is spawned by `people_detector` before publishing the detected people, its position would be fixed in the robot local coordinates system, moving along with it).

4.2.2 Nodes and topics

According to the general pipeline depicted in figure 4.1, two are the fundamental packages to ensure that our solution is properly implemented: `people_pub` and `nav2_social_costmap_plugin`. The package `people_pub` has two fundamentals node that are able to provide the people position and velocity in the `odom` reference frame to the social layer:

- `people_detector`: by means of the camera, we can acquire positions and velocities of detected people referenced to the `camera` reference frame. Inside this node, before publishing to the topic `/people` we are transforming people positions and velocities into the robot reference frame. As previously explained in section 1.2.1, when a person get out of the camera field of view, it will be "marked" as lost and its information will keep to be transmitted for a certain amount of time.
- `ppl2odom`: after getting the detected people information from the topic `/people` this node is meant to transform people positions and velocities from the robot reference frame to `odom`.

The composition of the package `people_pub` is the following:

```
~/dev_ws/src/people_pub
├── models
│   └── mobilenet-ssd_openvino_2021.4_5shave.blob
├── msg
│   ├── People.msg
│   └── Person.msg
├── src
│   ├── __init__.py
│   ├── ppl_tracker.py
│   └── ppl2odom.py
├── CMakeLists.txt
└── package.xml
```

The composition of the package `nav2_social_costmap_plugin` is the following:

```
~/dev_ws/src/nav2_social_costmap_plugin
├── include
│   └── nav2_social_costmap_plugin
│       └── social_layer.hpp
├── src
│   └── social_layer.cpp
├── CMakeLists.txt
├── package.xml
└── social_layer.xml
```

4.3 People detection and tracking

By means of the OAK-D stereo camera we are able to acquire image data, estimate depth, and execute the people detection and tracking model.

4.3.1 MobilenetSSD

Object detection is a computer technology that, thanks to computer vision and image processing allows to detect objects of a certain class (e.g. humans) in images and videos. MobilenetSSD is an object detection model able to determine the bounding box and category of an object given an image. This Single Shot Detector (SSD) object detection model can accomplish quick object identification, leveraging Mobilenet as its structural support. A single convolutional network constitutes the SSD architecture, which learns to predict bounding box coordinates and classify these regions in only one "shot". The SSD [11] method relies on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the occurrence of object class instances in those boxes. The final detections are then obtained by a non-maximum suppression step. Offset values (cx, cy, w, h) from the default box are contained in boxes. For each of the object categories, scores include a level of confidence. Each prediction has a bounding box and 21 scores for all of the classes; the class with the maximum score is determined to be the bounded object's class. MobileNet [12] is a class of efficient models using

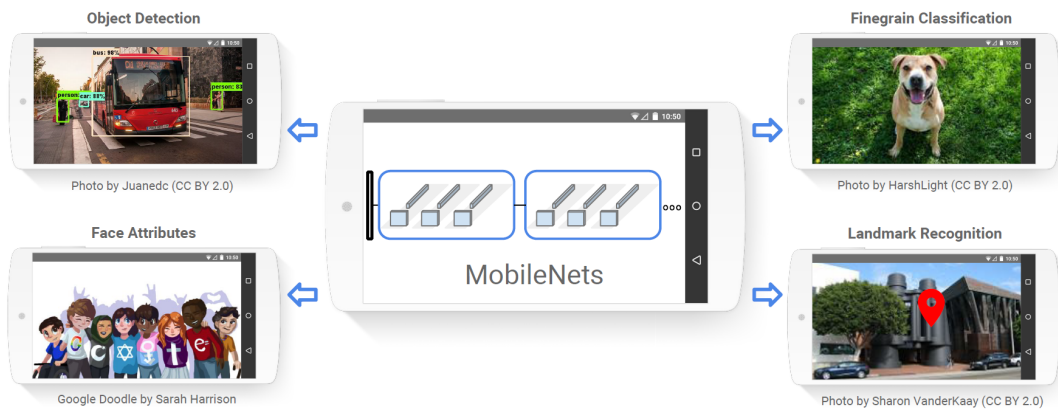


Figure 4.2: Various applications of MobilNet models [12]

depth-wise separable convolutions that, compared to a network with conventional convolutions of the same depth, dramatically reduces the number of parameters. Basically, depthwise separable convolution is a depthwise convolution followed by a pointwise convolution. The SSD network is composed of the Mobilenet base architecture and some convolution layers following it (Figure 4.3). Thanks to the combination of MobileNet and the SSD method, MobilenetSSD takes in input a $300 \times 300 \times 3$ image and outputs $1 \times 3000 \times 4$ boxes and $1 \times 3000 \times 21$ scores. In our

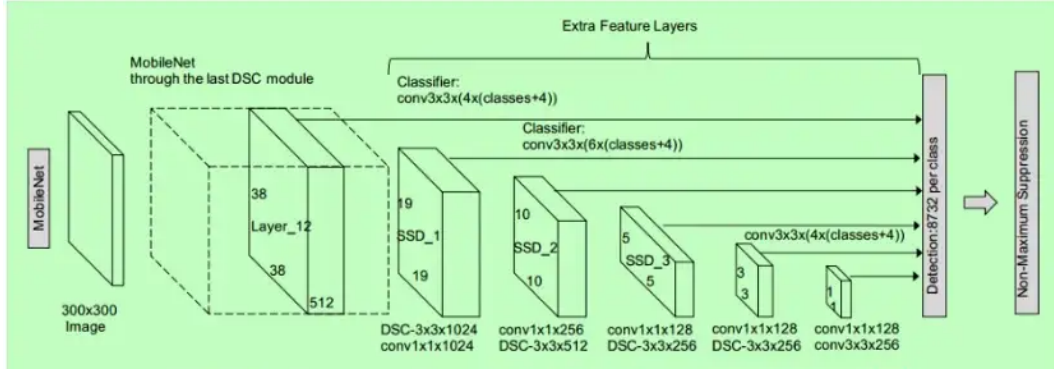


Figure 4.3: MobilenetSSD layered architecture [28]

solution, only people are detected and tracked since other objects are not of our interest.

4.3.2 Object tracking

DepthAI OAK-D camera incorporates a robust object tracker that is able to track detected objects from the image detections using Kalman filter and hungarian algorithm. Its inputs and outputs are depicted in Figure 4.4. We can distinguish two categories of tracking [29] supported in the above-mentioned DepthAI object tracker, which are:

- **Zero term tracking:** it carries out object association, therefore it doesn't conduct prediction or tracking based on earlier tracking data. Object association would entail mapping tracked objects that have been detected and are being tracked from earlier frames with detected objects from an external detector.

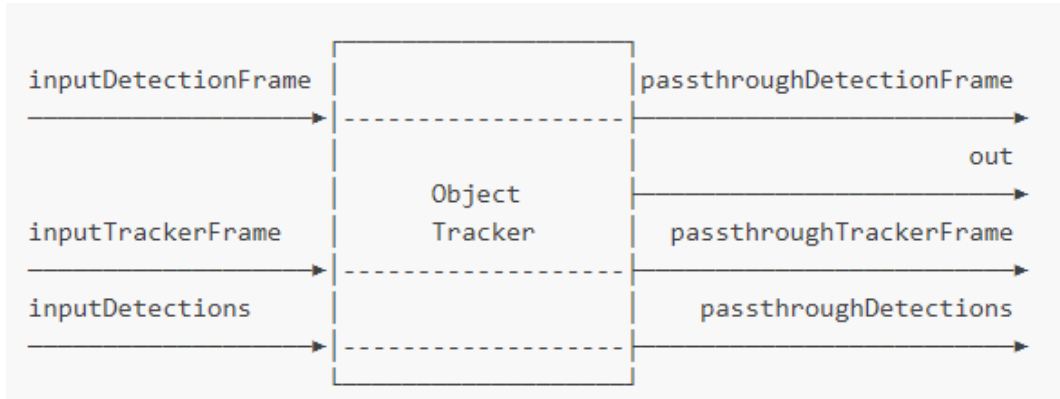


Figure 4.4: Object tracker inputs and outputs [29]

- **Short term tracking:** By tracking objects between frames, short-term tracking eliminates the requirement to perform object recognition on each frame. This works well with neural network models that can't run at 30 frames per second since tracker can supply tracklets when there was no inference, allowing the entire system to operate at 30FPS.

The DepthAI object tracker allows to use four different types of trackers [29], two per each category:

- **SHORT_TERM_KCF:** Kernelized Correlation Filter tracking employs circulant matrix to enhance the processing speed.
- **SHORT_TERM_IMAGELESS:** Short-term tracking enables to track objects on frames where object detection was not performed, by extrapolating object trajectory from previous detections.
- **ZERO_TERM_COLOR_HISTOGRAM:** performs object tracking using the position, shape, and input image data, such as the RGB histogram.
- **ZERO_TERM_IMAGELESS:** It employs only the rectangular shape of the identified object and its position to track it. It doesn't use the color information of the objects it is tracking. Compared to ZERO TERM COLOR HISTOGRAM, it has a better throughput but lower accuracy.

In our solution we use, as object tracker, the ZERO_TERM_COLOR_HISTOGRAM

that guarantees good performances. Furthermore, in Figure 4.5 it is possible to observe a visualization of the detections from our camera.



Figure 4.5: Visualization of the detections in our solution

4.4 Social Costmap Plugin

In the first place, the plugin class `nav2_social_costmap_plugin::SocialLayer` is inherited from the basic class `nav2_costmap_2d::Layer`. The class `Layer` provides a set of virtual methods API for the implementation of new costmap layers [30]. These methods are stated in table 4.1.

Virtual Method	Method description
onInitialize()	This method is called at the end of plugin initialization and it is employed to execute any required initialization.
matchSize()	This method is called when the map size is changed.
onFootprintChanged()	This method is called when the footprint is changed.
reset()	It can contain any code to be executed during costmap reset.
updateBounds()	This method is called to get information from the plugin about which portion of the costmap layer requires updating. To ask the plugin which portion of the costmap layer requires updating, a method is called. Robot position and orientation are the method's two input parameters. Its four output parameters are pointers to the boundaries of the window. These boundaries are utilized for performance purposes in order to update the portion of the window when new information is available rather than updating the entire costmap every time an iteration occurs.

updateCosts()	Every time a costmap recalculation is necessary, the method is called. It only updates the costmap layer within of the bounds of its window. The method takes 4 input parameters, including the computation window boundaries, and outputs 1 reference to a costmap master grid as a result. A costmap_intrinsic to the Layer class is made available to the plugin for modifications. One of the update methods listed below—updateWithAddition(), updateWithMax(), updateWithOverwrite(), or updateWithTrueOverwrite—should be used to update the master grid with values that fall within the window bounds.
---------------	---

Table 4.1: Virtual methods for the implementation of new costmap layers [30]

The most important virtual methods, that are obviously employed in our solutions are `onInitialize()`, `updateBounds()`, `updateCosts()` since they are essential for a proper functioning of our solution. In particular, these methods, in our solution, perform the following tasks:

- `onInitialize()`: contains the declaration of some useful ROS parameters with their default values and execute the subscription to the topic `/ppl_odom` containing the position and velocity information of all the detected people in the `odom` reference frame.
- `updateBounds()`: it is devoted to update the computation window boundaries according to the people data received from the topic it subscribed to in the initialization phase.
- `updateCosts()`: in this method the gaussian costs are computed and their values are written directly to the resulting costmap.

The pipeline depicted in figure 4.6 describes the fundamentals steps that our layer follows:

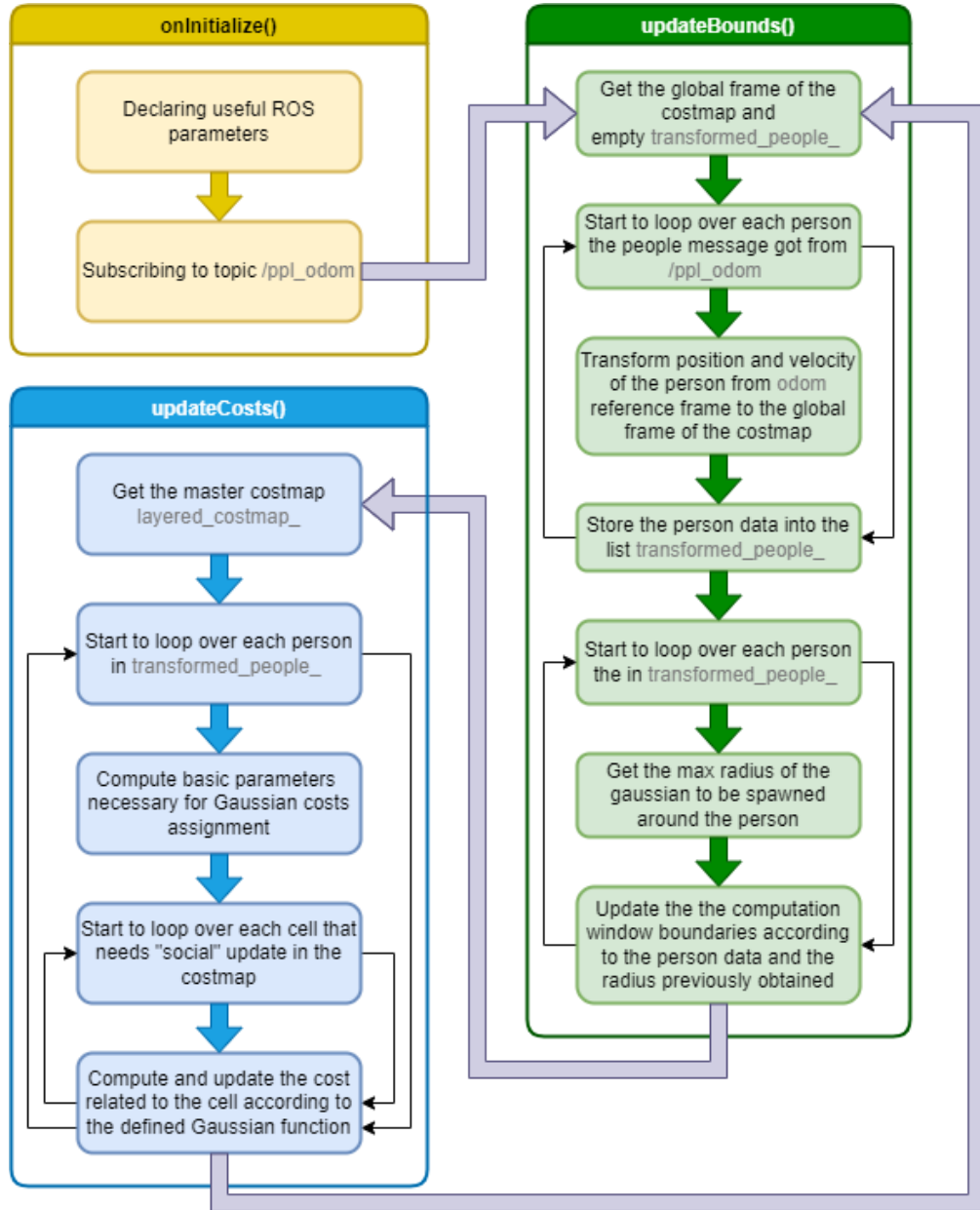


Figure 4.6: Social Layer virtual methods main tasks step by step

4.4.1 Costs assignment

The final and most important stage for the implementation of our "social layer" has been to meaningfully integrate into the costmap the information about people that have been detected. In particular, the two-dimensional Gaussian distribution, which is widely used, is the subject of our investigation. In general, the mean and variance of a typical one-dimensional Gaussian function are required for its definition in (4.1).

$$f(x) = A \cdot e^{-\frac{(x - \mu)^2}{2\sigma^2}} \quad (4.1)$$

Instead, for a symmetric two-dimensional Gaussian, the mean is the center of the function (x_0, y_0) :

$$f(x) = A \cdot e^{-\left[\frac{(x - x_0)^2 + (y - y_0)^2}{2\sigma^2}\right]}$$

In our solution, we want the cells to be inflated more in the direction that people are moving to. As a consequence, as proposed in [7] and [9], we are implementing an asymmetric two-dimensional Gaussian through the combination of two such functions with differing variances σ_x and σ_y :

$$f(x) = A \cdot e^{-\left[\frac{(x - x_0)^2}{2\sigma_x^2} + \frac{(y - y_0)^2}{2\sigma_y^2}\right]} \quad (4.2)$$

This function simply ensures symmetry along one axis (instead of two), which in general doesn't need to be parallel to the x or y axes. In particular, given a person P centered in $C(x_c, y_c)$ referring to the map reference frame, we define a local coordinate frame with origin in C, X'-axis oriented along the direction of the velocity vector, Z'-axis outgoing the costmap plane and Y'-axis defined in accordance with the right-hand rule as depicted in figure 4.7. Therefore, according to (4.2), we can rewrite the asymmetric two-dimensional Gaussian function as stated in (4.3).

$$f(x) = A \cdot e^{-\left[\frac{(r \cdot \cos(\theta - \theta_c))^2}{2\sigma_x^2} + \frac{(r \cdot \sin(\theta - \theta_c))^2}{2\sigma_y^2}\right]} \quad (4.3)$$

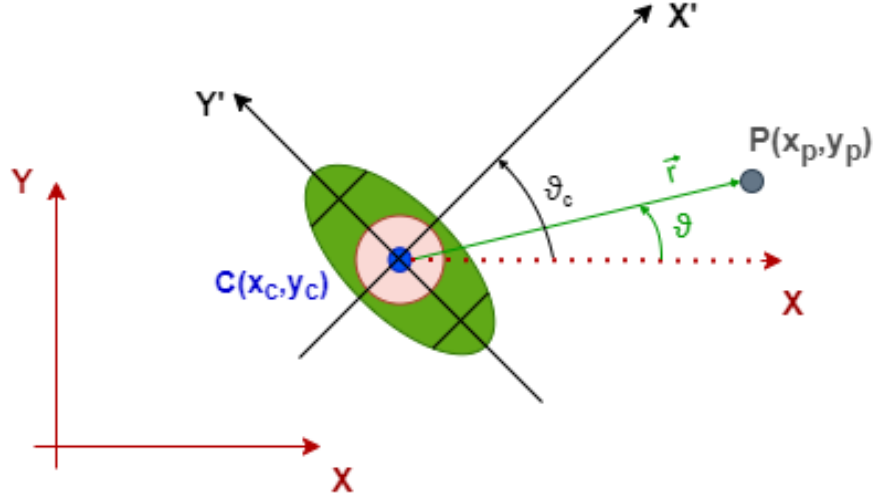


Figure 4.7: Local reference frame for costs assignment according to an asymmetric 2D Gaussian function

At first, since we want the cells to be inflated more in the X' -axis direction, we compute a "speed factor" that modulates how "stretched" the Gaussian shape is, in the motion direction. This factor is the sum of a constant component and a component that is directly related to the velocity vector magnitude: $factor = a + mag \cdot b$. In our solution we choose as reasonable values $a = 1.0$, $b = 6.0$. Subsequently, we compute the shortest angular distance between θ and θ_c in order to understand if the cell of interest is in the front region (to be inflated more) or in the rear region. Indeed, if the shortest angular distance is less than $\pi/2$ the considered cell is in the front region (in our solution this implies $\sigma_x = 0.25 \cdot factor$, $\sigma_y = 0.25$) whereas in the other cases the considered cell is in the rear region (in our solution this implies $\sigma_x = 0.25$, $\sigma_y = 0.25$). The final results obtained from the implementation of the social layer is shown in Figure 4.8 in the simulated world (on the left) a person is moving towards the robot and on RViz (on the right) it is possible to notice the costmap layer addition.

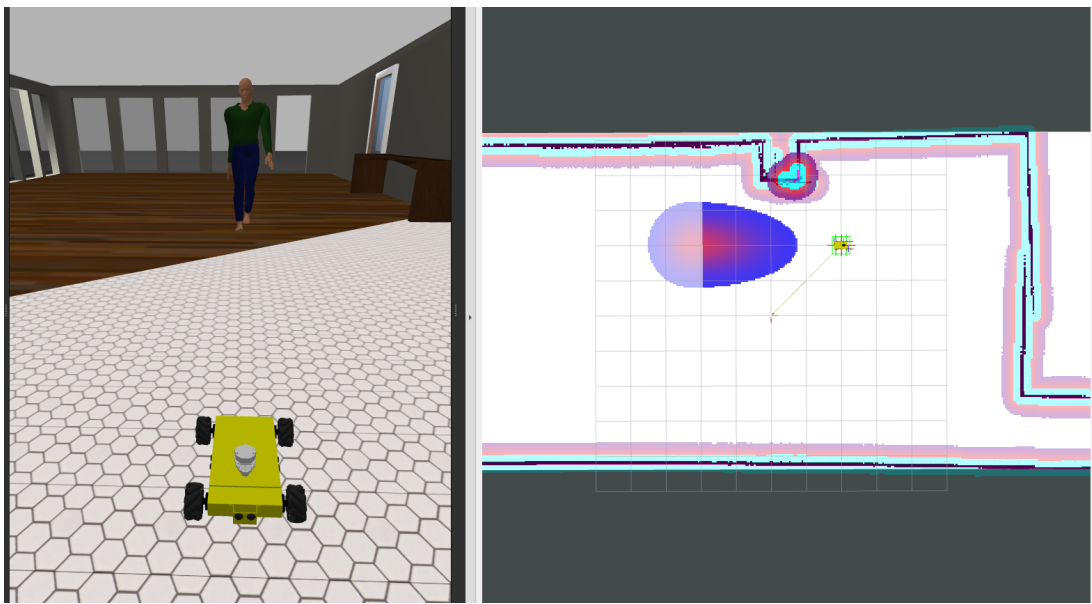


Figure 4.8: Visualization of a person moving towards the robot in a simulated world

Chapter 5

Simulations and tests

5.1 Overview

Before deploying our solution on the robot we set up a simulation to understand how good was the behaviour of the robot when dealing with a gaussian shape inflated around people moving. This chapter explains how the simulation was set up and the results obtained using the assumptions from the previous chapters and configurations for ROS 2, Gazebo, and RViz. The software used for the simulation are:

- **Gazebo:** used to generate of the simulation environment with one or more people, the robot and the simulation world.
- **RViz:** used as visualization software tool, especially to monitor the local and global costmap.

It is worth to denote that, both in the simulation and in the real tests, `map` and `odom` reference frame were assumed to be coincident. Indeed, the command `ros2 run tf2_ros static_transform_publisher 0 0 0 0 0 0 map odom` has been executed.

5.2 Environment configuration on Gazebo

To begin with, a launch file has been set up to ensure that all entities within the Gazebo description file and the robot model were spawned together, and all the nodes of interest for the autonomous navigation of the robot could be launched just by executing one command on the terminal. Taking this into account, to prevent all sort of anomalies, the environment needed to be properly configured. The world used for the simulation is the `cafe.world` (Figure 5.1) proposed in the Gazebo tutorial about making an animated model (actor). Here, we spawn an actor (Figure

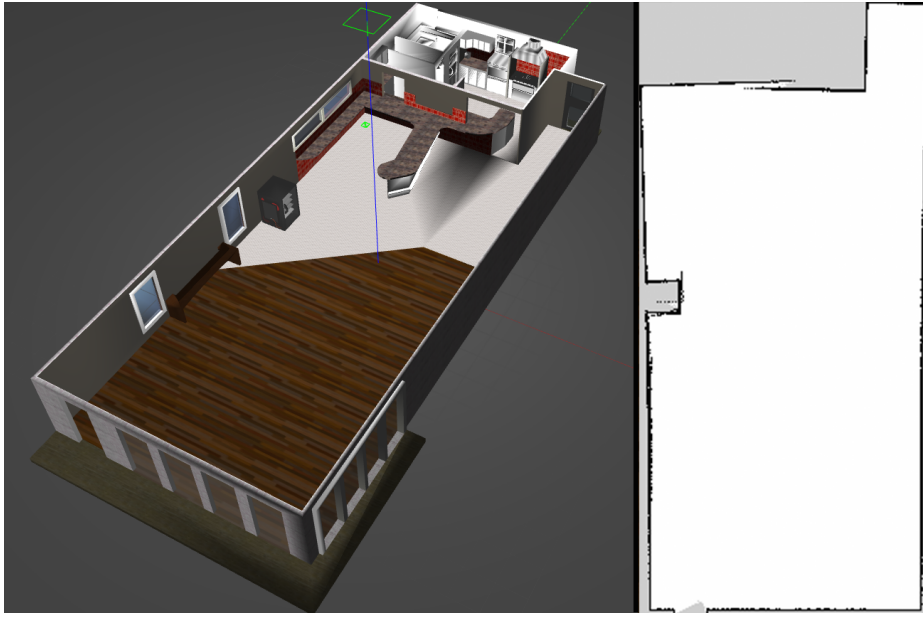


Figure 5.1: Cafe world visualization in Gazebo (left) and its map (right)

5.2) that is intended to follow a predetermined trajectory and, calling the service `get_entity_state`, we are able to get its position and velocity over time.

In the simulation, the pipeline `people_detector.py` \rightarrow `/people` \rightarrow `ppl2odom.py` is replaced by `ppl_publisher.py` which is the node designated for the publication of people data, acquired by getting the state of the "actor" entity, in the topic `/ppl_odom`.

The mobile robot employed for the simulation (Figure 5.3) is a Nexus 4WD Mecanum robot [31], which is different from the mobile platform we employed in our sanitation robot but it is equivalent from a qualitative point of view for the



Figure 5.2: Visualization of the spawned actor in Gazebo

evaluation of the performance of the implemented "social layer" since both have a mecanum wheel based platform.

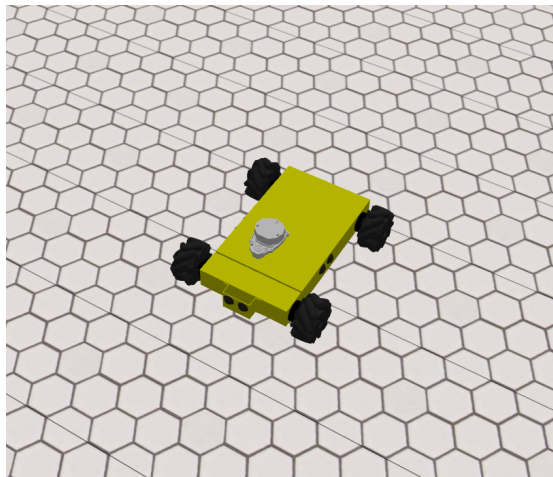


Figure 5.3: Visualization of the spawned Nexus 4WD Mecanum robot in Gazebo

5.3 Simulation scenarios

For performance evaluation purposes, some prototypical "social" encounters have been considered, and in particular the following tests have been carried out:

- Front passing test: the robot and the person start their "navigation" in two different points of the environment having intersecting trajectories since both are heading "against" each other.

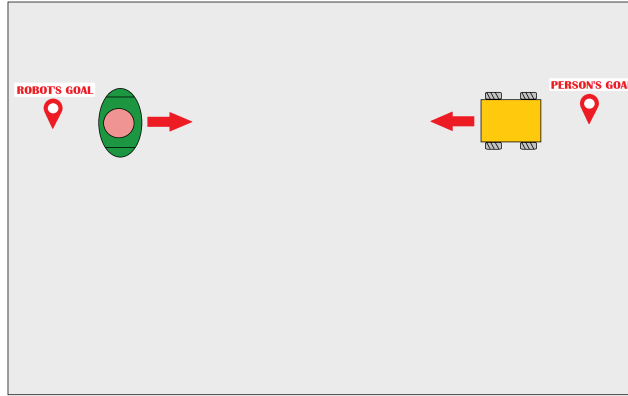


Figure 5.4: Front passing qualitative schematization

- Diagonal passing test: similar to the front passing test but the robot is intercepting the person trajectory diagonally.

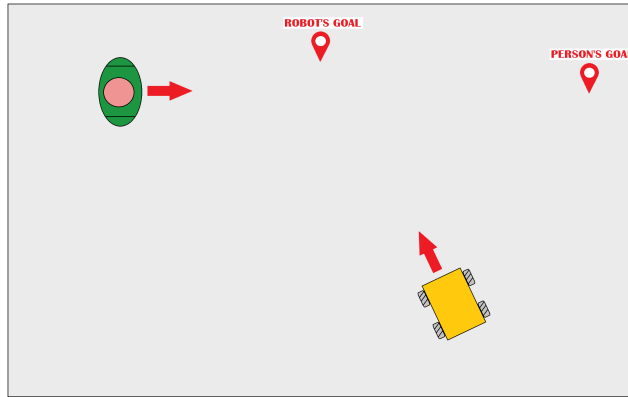


Figure 5.5: Diagonal passing qualitative schematization

- Orthogonal passing test: similar to the front passing test but the robot is intercepting the person trajectory orthogonally.

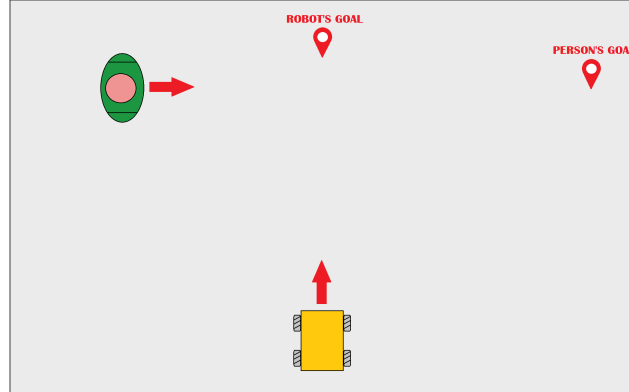


Figure 5.6: Orthogonal passing qualitative schematization

- Wall test: similar to the front passing test but with a wall on one side.

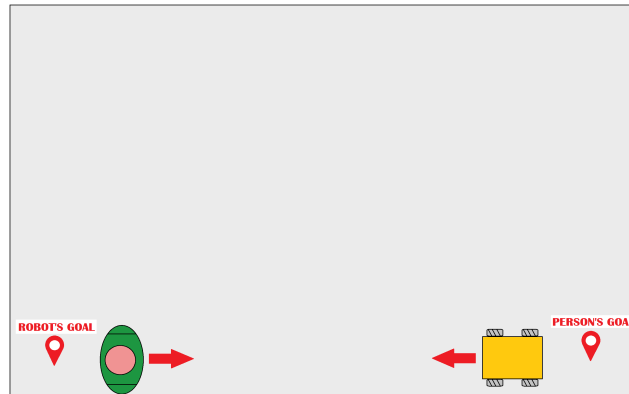


Figure 5.7: Wall test qualitative schematization

- Cut test: passing with a person walking next to the robot in its same direction and suddenly cutting in front of it.

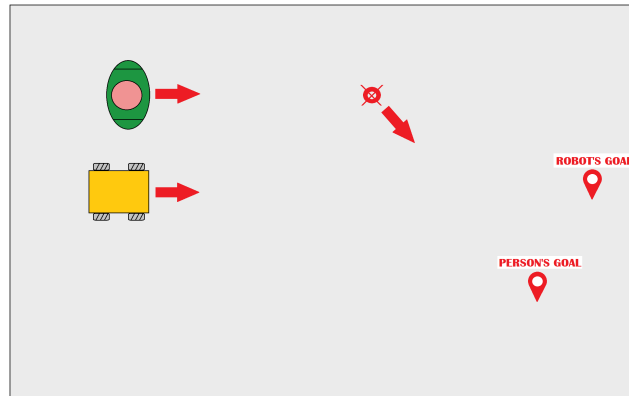


Figure 5.8: Cut test qualitative schematization

- X test: two people starting their navigation from adjacent corners of the room crossing it diagonally to get to the opposite corners; also the robot is crossing diagonally the room.

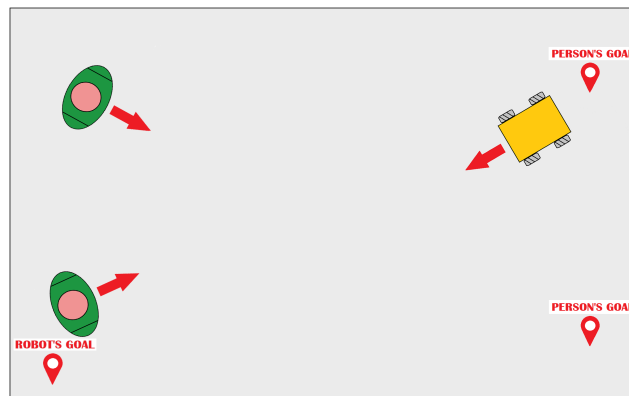


Figure 5.9: X test qualitative schematization

- Tab test: similar to the front passing test with a further person walking next to the robot.

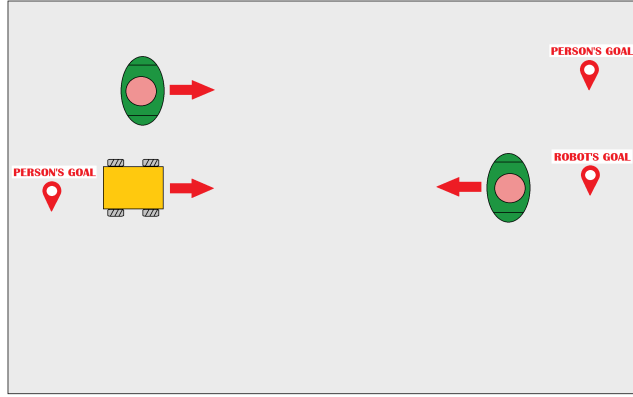


Figure 5.10: Tab test qualitative schematization

5.4 Social layer parameters

A substantial performance assessment of our approach would require a lot of time and this is out of the scope of this work, which instead intends to build the groundwork for more extensive and focused future approaches at PIC4SeR. For this reason, we are assuming some reasonable parameters in table 5.1 for the Gaussian function. It is worth to highlight that increasing values of σ lead to wider Gaussians

Name	Description	Value
Cutoff	Min cost value to publish on costmap	10.0
Amplitude (A)	Max cost, at the peak of the Gaussian	200.0
Covariance (σ)	Covariance for the definition of σ_x and σ_y	0.25
Speed factor	Gaussian stretching in the motion direction	$1.0 + 6.0 \cdot mag$

Table 5.1: Parameters of the Gaussian function

whereas bigger values of the speed factor lead to a more "stretched" Gaussian along the motion direction. We have chosen $\sigma = 0.25$ since we noticed that this value is

enough to guarantee more or less the robot-person distance to be around 1 metre during "social" encounters.

5.5 Set of metrics

A first study of the literature has been carried out in order to gather some of the metrics relevant for the evaluation of the performance of the human-aware navigation. Indeed, the proposed set of metrics includes some of the metrics found in the literature [32] [33] (some metrics are identical while some are modified) plus some more. To date, the metrics implemented are the following:

- Time to reach the goal: time in seconds from when the robot started its navigation till it successfully reached the destination.

$$T_p = T_{goal} - T_{init}$$

- Path length: the distance in metres traveled by the robot from its starting location to its destination.

$$L_p = \sum_{i=1}^{N-1} \|x_r^i - x_r^{i+1}\|_2$$

- Path length no people: the straight line distance in metres among the initial and final position of the robot (it is the distance the robot would have traveled if no person was encountered during the navigation).

$$L_{p0} = \|x_r^{goal} - x_r^{init}\|_2$$

- Human-aware path efficiency: the ratio of the path length without people (L_{p0}) and the path length (L_p).

$$\eta = \frac{L_{p0}}{L_p}$$

- Minimum, maximum and average distance to people: the minimum, maximum and average distance in metres between the robot and each person.

$$RPD_{min} = \min ||x_r^i - x_p^i||_2 \quad \forall i \in N$$

$$RPD_{max} = \max ||x_r^i - x_p^i||_2 \quad \forall i \in N$$

$$RPD_{avg} = \text{avg} ||x_r^i - x_p^i||_2 \quad \forall i \in N$$

- People space intrusions. The Proxemics theory, which establishes personal spaces around people for interaction, serves as the foundation for this metric. These areas are defined as:

- Intimate: distance shorter than 0.45m.
- Personal: distance between 0.45 and 1.2m.
- Social: distance between 1.2 and 3.6m.
- Public: distance longer than 3.6m.

Thus, these metrics classify the distance between the robot and each person at each time step in one of the Proxemics spaces in order to obtain a percentage of the time spent in each space for the overall trajectory:

$$RPD_{prox}^k = \frac{1}{N} \sum_{i=1}^N F(||x_r^i - x_p^i||_2 < \delta^k) \cdot 100$$

where N is the total number of time steps during the robot navigation, δ^k defines the distance range for classification of the space, and $F(\cdot)$ is the indicator function (a function that maps elements of the subset of a set to one, and all other elements to zero).

5.6 Simulation results

This section is intended to showcase the results obtained in the different simulation scenarios depicted in section 5.3. Each subsection is dedicated to charts and metrics for a single simulation scenario.

5.6.1 Front passing test

This section is intended to show the results obtained through the simulation of a front passing test that is qualitatively depicted in figure 5.4.

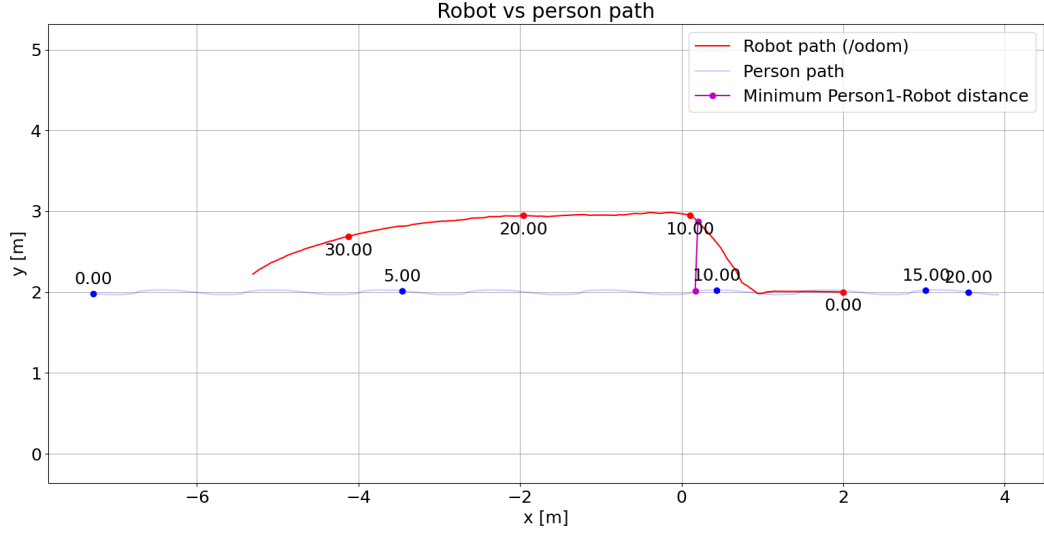


Figure 5.11: Front passing test robot and person path representation

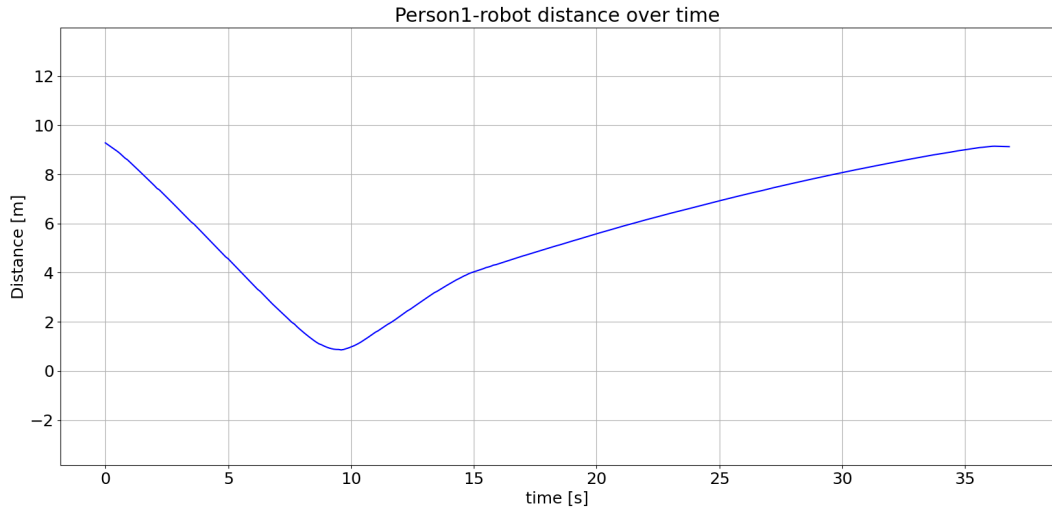


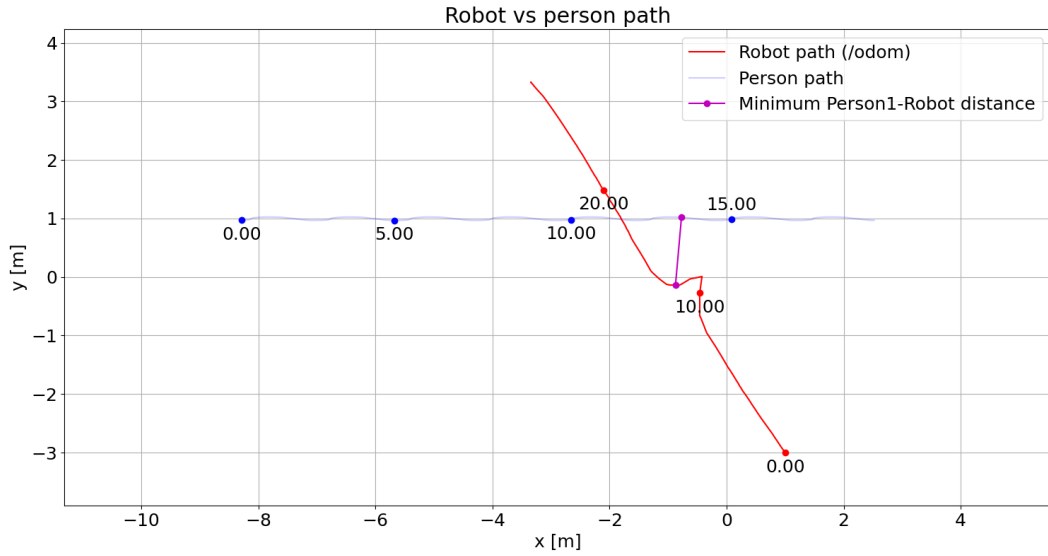
Figure 5.12: Front passing test person-robot distance over time graph

Metric	Value	Metric	Value
T_p [s]	36.6	RPD_{avg} [m]	5.756
L_p [m]	7.908	$RPD_{prox}^{intimate}$	0.0%
L_{p0} [m]	7.313	$RPD_{prox}^{personal}$	5.15%
η	0.9247	RPD_{prox}^{social}	17.07%
RPD_{min} [m]	0.858 at $t = 9.55$ s	RPD_{prox}^{public}	77.78%
RPD_{max} [m]	9.281 at $t = 0.0$ s		

Table 5.2: Set of metrics for front passing test

5.6.2 Diagonal passing test

This section is intended to show the results obtained through the simulation of a diagonal passing test that is qualitatively depicted in figure 5.5.


Figure 5.13: Diagonal passing test robot and person path representation

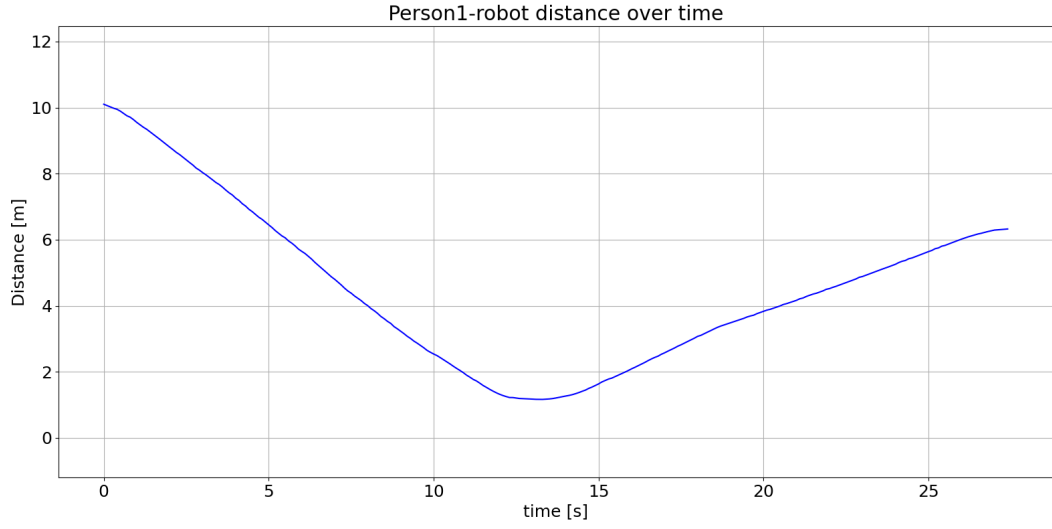


Figure 5.14: Diagonal passing test person-robot distance over time graph

Metric	Value	Metric	Value
T_p [s]	27.35	RPD_{avg} [m]	4.532
L_p [m]	8.268	$RPD_{prox}^{intimate}$	0.0%
L_{p0} [m]	7.679	$RPD_{prox}^{personal}$	4.0%
η	0.9288	RPD_{prox}^{social}	35.27%
RPD_{min} [m]	1.164 at t = 13.25 s	RPD_{prox}^{public}	60.73%
RPD_{max} [m]	10.1 at t = 0.0 s		

Table 5.3: Set of metrics for diagonal passing test

5.6.3 Orthogonal passing test

This section is intended to show the results obtained through the simulation of a orthogonal passing test that is qualitatively depicted in figure 5.6.

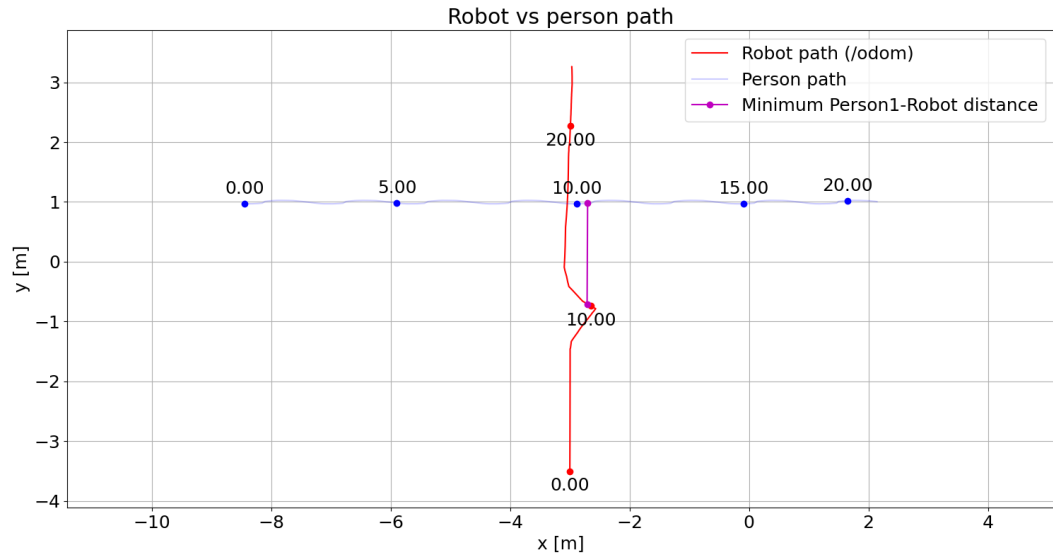


Figure 5.15: Orthogonal passing test robot and person path representation

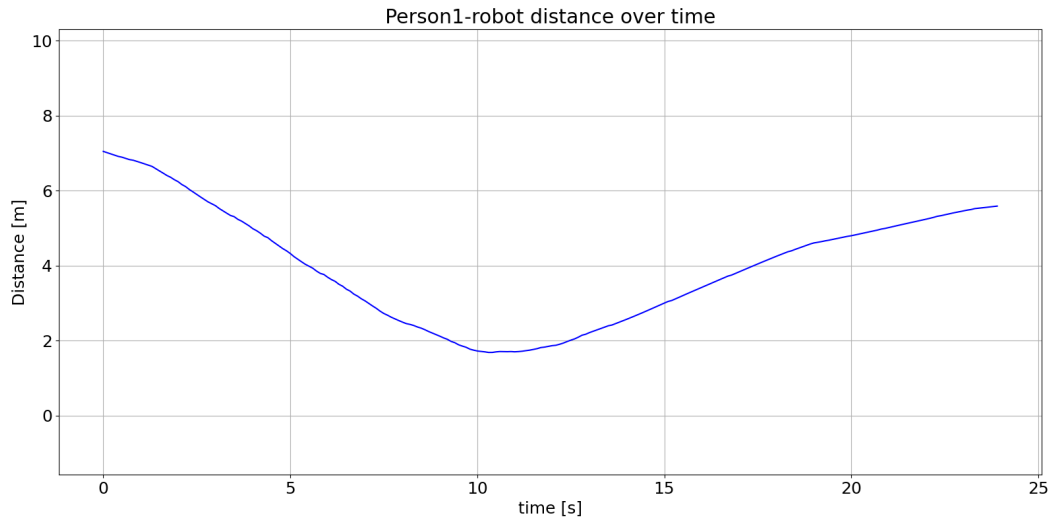


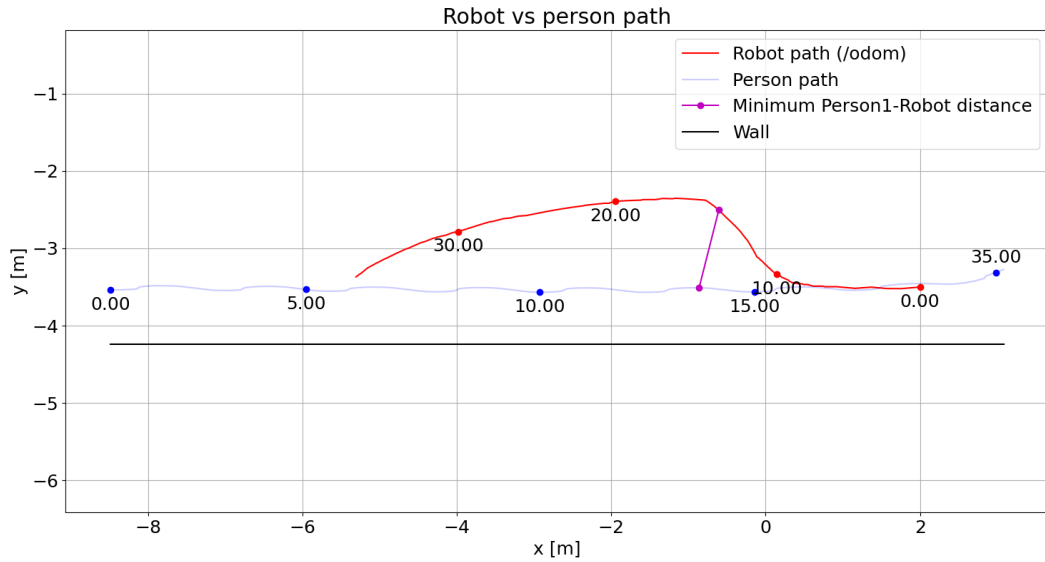
Figure 5.16: Orthogonal passing test person-robot distance over time graph

Metric	Value	Metric	Value
T_p [s]	23.8	RPD_{avg} [m]	3.969
L_p [m]	7.138	$RPD_{prox}^{intimate}$	0.0%
L_{p0} [m]	6.763	$RPD_{prox}^{personal}$	0.0%
η	0.9474	RPD_{prox}^{social}	42.92%
RPD_{min} [m]	1.683 at t = 10.3 s	RPD_{prox}^{public}	57.08%
RPD_{max} [m]	7.046 at t = 0.0 s		

Table 5.4: Set of metrics for orthogonal passing test

5.6.4 Wall test

This section is intended to show the results obtained through the simulation of a wall test that is qualitatively depicted in figure 5.7.


Figure 5.17: Wall test robot and person path representation

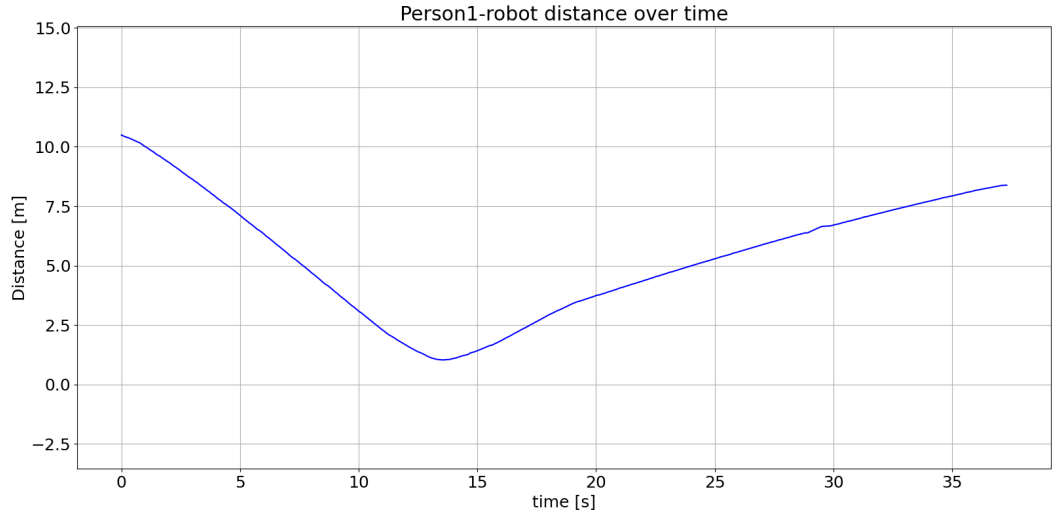


Figure 5.18: Wall test person-robot distance over time graph

Metric	Value	Metric	Value
T_p [s]	37.1	RPD_{avg} [m]	5.356
L_p [m]	7.958	$RPD_{prox}^{intimate}$	0.0%
L_{p0} [m]	7.308	$RPD_{prox}^{personal}$	4.07%
η	0.9183	RPD_{prox}^{social}	23.58%
RPD_{min} [m]	1.037 at t = 13.55 s	RPD_{prox}^{public}	72.36%
RPD_{max} [m]	10.491 at t = 0.0 s		

Table 5.5: Set of metrics for wall test

5.6.5 Cut test

This section is intended to show the results obtained through the simulation of a cut test that is qualitatively depicted in figure 5.8.

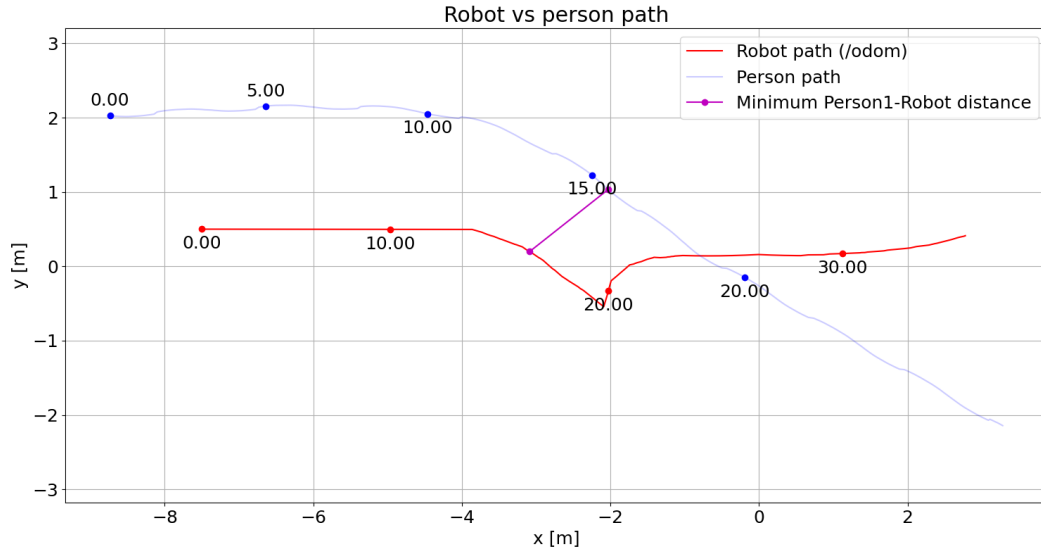


Figure 5.19: Cut test robot and person path representation

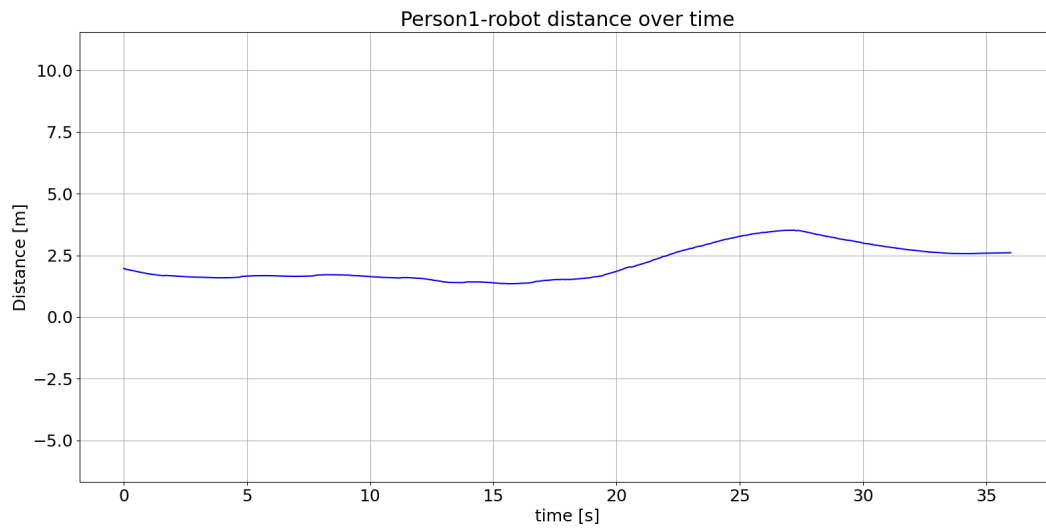


Figure 5.20: Cut test person-robot distance over time graph

Metric	Value	Metric	Value
T_p [s]	35.75	RPD_{avg} [m]	2.157
L_p [m]	10.97	$RPD_{prox}^{intimate}$	0.0%
L_{p0} [m]	10.278	$RPD_{prox}^{personal}$	0.0%
η	0.9369	RPD_{prox}^{social}	100.0%
RPD_{min} [m]	1.348 at t = 15.6 s	RPD_{prox}^{public}	0.0%
RPD_{max} [m]	3.52 at t = 27.15 s		

Table 5.6: Set of metrics for cut test

5.6.6 X test

This section is intended to show the results obtained through the simulation of an X test that is qualitatively depicted in figure 5.9.

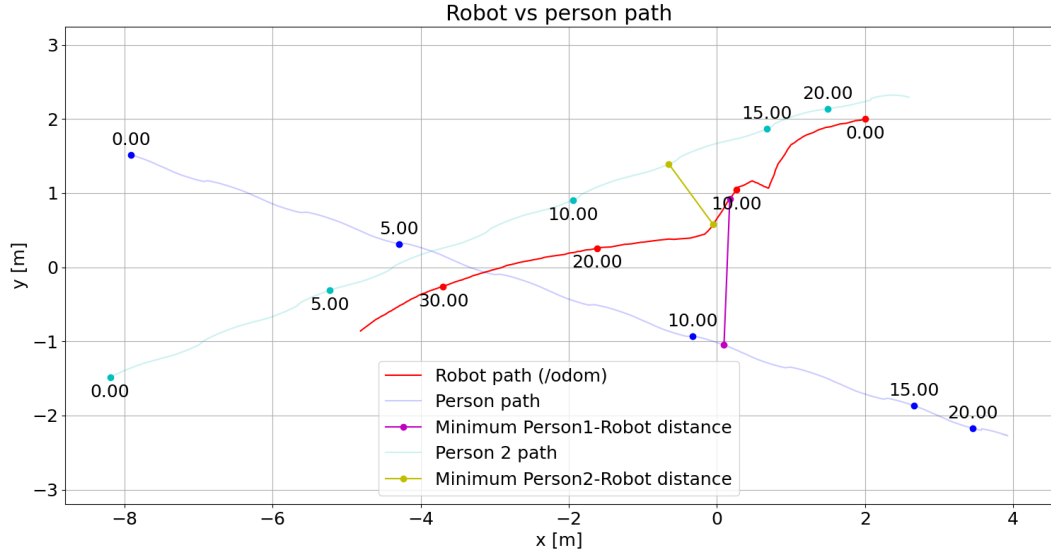


Figure 5.21: X test robot and person path representation

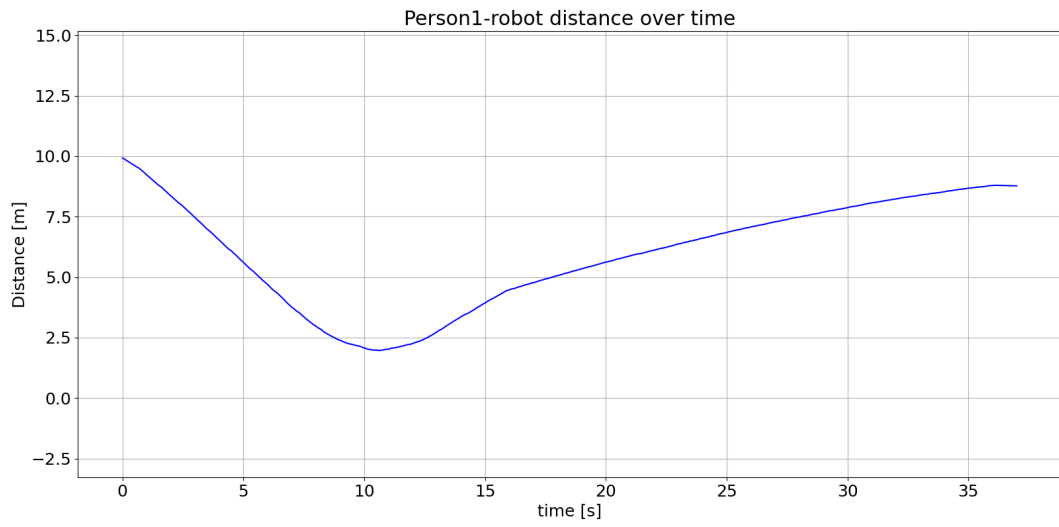


Figure 5.22: X test person1-robot distance over time graph

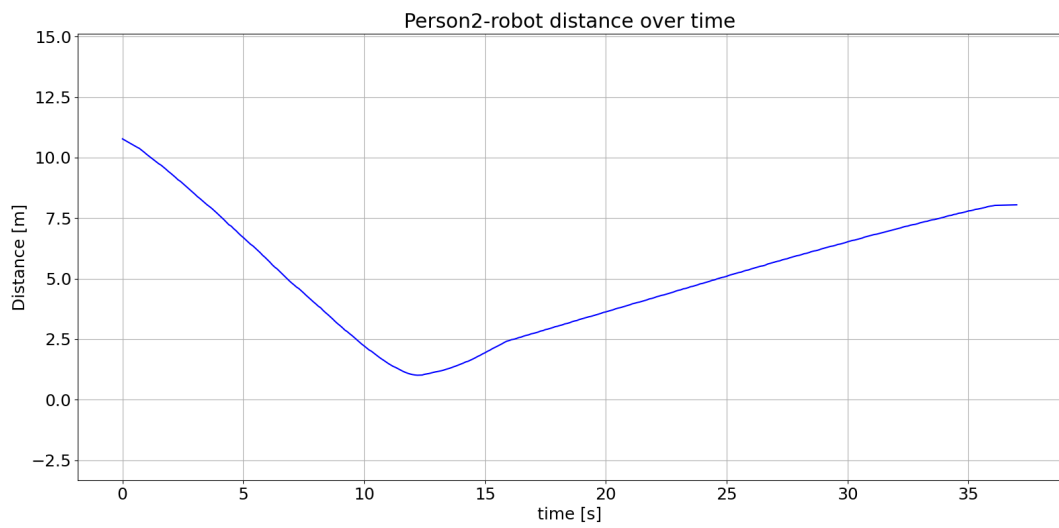


Figure 5.23: X test person2-robot distance over time graph

Metric	Value
T_p [s]	36.6
L_p [m]	7.873
L_{p0} [m]	7.39
η	0.9387
PERSON 1	
RPD_{min} [m]	1.97 at t = 10.6 s
RPD_{max} [m]	9.927 at t = 0.0 s
RPD_{avg} [m]	6.006
$RPD_{prox}^{intimate}$	0.0%
$RPD_{prox}^{personal}$	0.0%
RPD_{prox}^{social}	19.41%
RPD_{prox}^{public}	80.59%
PERSON 2	
RPD_{min} [m]	1.011 at t = 12.15 s
RPD_{max} [m]	10.773 at t = 0.0 s
RPD_{avg} [m]	5.163
$RPD_{prox}^{intimate}$	0.0%
$RPD_{prox}^{personal}$	4.58%
RPD_{prox}^{social}	26.42%
RPD_{prox}^{public}	69.0%

Table 5.7: Set of metrics for X test

5.6.7 Tab test

This section is intended to show the results obtained through the simulation of a Tab test that is qualitatively depicted in figure 5.10.

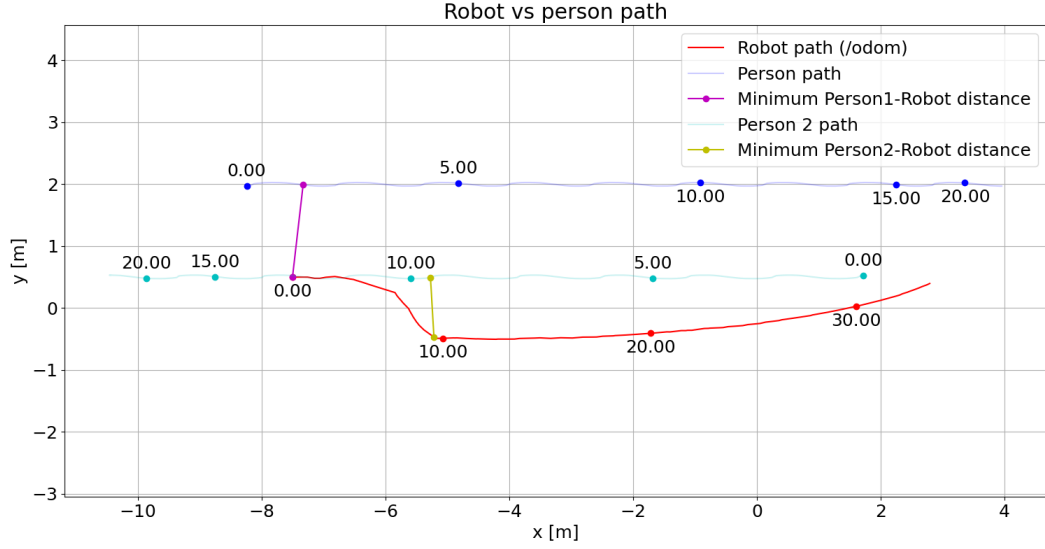


Figure 5.24: Tab test robot and person path representation

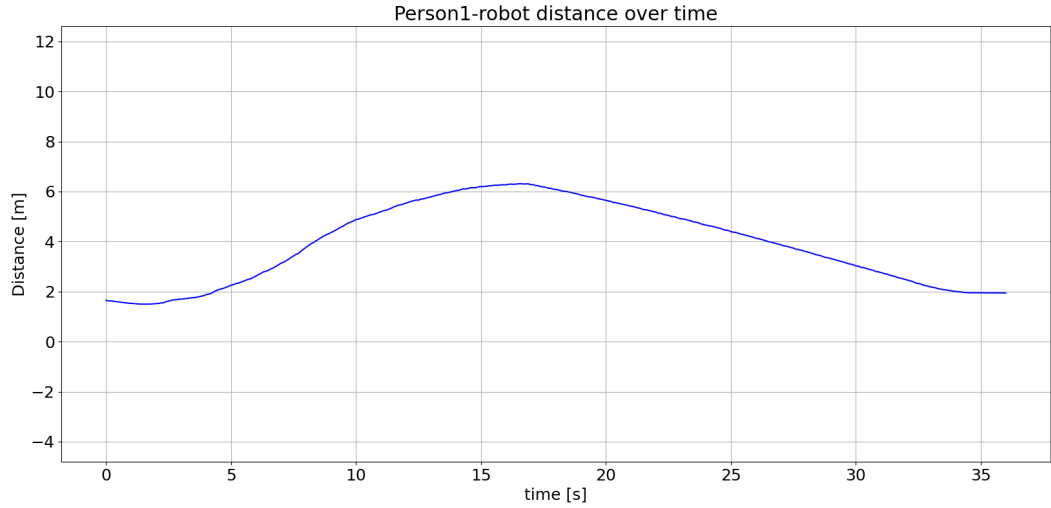


Figure 5.25: Tab test person1-robot distance over time graph

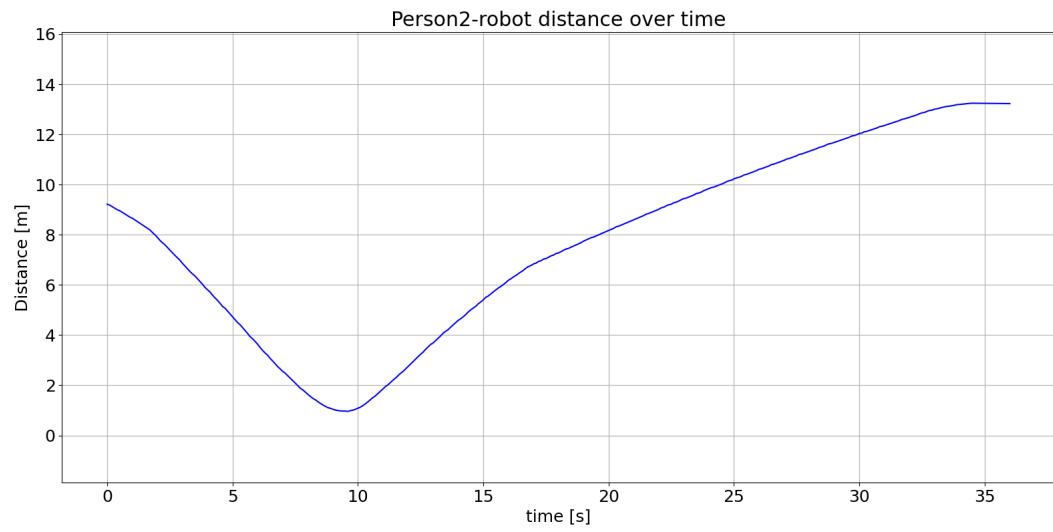


Figure 5.26: Tab test person2-robot distance over time graph

Metric	Value
T_p [s]	34.85
L_p [m]	10.773
L_{p0} [m]	10.292
η	0.9553
PERSON 1	
RPD_{min} [m]	1.499 at t = 1.5 s
RPD_{max} [m]	6.315 at t = 16.55 s
RPD_{avg} [m]	3.946
$RPD_{prox}^{intimate}$	0.0%
$RPD_{prox}^{personal}$	0.0%
RPD_{prox}^{social}	43.77%
RPD_{prox}^{public}	56.23%
PERSON 2	
RPD_{min} [m]	0.959 at t = 9.55 s
RPD_{max} [m]	13.244 at t = 34.45 s
RPD_{avg} [m]	7.725
$RPD_{prox}^{intimate}$	0.0%
$RPD_{prox}^{personal}$	4.43%
RPD_{prox}^{social}	14.4%
RPD_{prox}^{public}	81.16%

Table 5.8: Set of metrics for Tab test

Chapter 6

Conclusions and future developments

The topic of autonomous robot navigation is very extensive. The initial state-of-the-art study, which helped identify the issue in the context of indoor navigation, has been very useful in order to identify which were theoretically the best approaches to tackle this problem. We were able to successfully integrate a social layer in the local and global costmap that could render "human-aware" the navigation of robots. The proposed social layer approach presents a method for considering people differently from classical objects, that is simple to integrate with the existing ROS2 Navigation Stack. A first implementation of the social layer has been carried out in simulation in order to get a qualitative idea of its overall performance regardless of the people detection technique as discussed in Chapter 5. Then, an implementation of our solution on the real robot has been carried out and led to promising but not sufficient results, which will be the main scope for future improvements. Indeed, we were able to successfully implement the detection and tracking of people in the robot's environment using a stereo camera. The biggest problem encountered during the implementation on the robot was the limited field of view of the camera. Indeed, during the movement of the robot, people could not be detected anymore even if they were near the robot (e.g. when passing on the side of a person). There are numerous solution useful to tackle this issue (i.e. creating custom heuristics for the planner, employing more than one camera on the robot to get an overall

wider field of view or using sensor fusion with lidar data to have more reliable information). The proposed solution, was to suppose that the person had virtually remained in the last position it was detected in, for a certain amount of time. This was theoretically a good solution, however due to computer vision technological limitations also "false" detections remained virtually on the costmap for some time, which is undesirable. In summary, the proposed solution is a solid groundwork for more extensive and focused approach to tackle the issue of autonomous navigation in social contexts using ROS2. Indeed, the modularity of this solution makes the integration of a "social module" easy for service robots that are intended to coexist with humans and to fulfill various tasks, from manufacturing to human assistance.

Bibliography

- [1] Ronja Möller, Antonino Furnari, Sebastiano Battiato, Aki Härmä, and Giovanni Maria Farinella. «A survey on human-aware robot navigation». In: *Robotics and Autonomous Systems* 145 (2021). ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2021.103837>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889021001226> (cit. on pp. iii, 3, 7).
- [2] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. «A review of mobile robots: Concepts, methods, theoretical framework, and applications». In: *International Journal of Advanced Robotic Systems* 16.2 (2019). DOI: 10.1177/1729881419839596 (cit. on p. 1).
- [3] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2nd. The MIT Press, 2011. ISBN: 0262015358 (cit. on p. 1).
- [4] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O'Dwyer, Fionn Merz, and Mark McConnell. «Service Robots in the Healthcare Sector». In: *Robotics* 10.1 (2021). ISSN: 2218-6581. URL: <https://www.mdpi.com/2218-6581/10/1/47> (cit. on p. 3).
- [5] Guang-Zhong Yang et al. «Combating COVID-19-The role of robotics in managing public health and infectious diseases». In: *Science Robotics* 5.40 (2020). DOI: 10.1126/scirobotics.abb5589. eprint: <https://www.science.org/doi/pdf/10.1126/scirobotics.abb5589>. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abb5589> (cit. on p. 3).

- [6] Elena Pacchierotti, Henrik I. Christensen, and Patric Jensfelt. «Evaluation of Passing Distance for Social Robots». In: *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*. 2006, pp. 315–320. DOI: 10.1109/ROMAN.2006.314436 (cit. on p. 4).
- [7] Rachel Kirby. *Social robot navigation*. Carnegie Mellon University, 2010 (cit. on pp. 7, 39).
- [8] David V Lu, Dave Hershberger, and William D Smart. «Layered costmaps for context-sensitive navigation». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 709–715 (cit. on pp. 7, 17, 18).
- [9] David V Lu and William D Smart. «Towards more efficient navigation for robots and humans». In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1707–1713 (cit. on pp. 7, 39).
- [10] Guillaume Doisy, Aleksandar Jevtic, Eric Lucet, and Yael Edan. «Adaptive person-following algorithm based on depth images and mapping». In: *Proc. of the IROS Workshop on Robot Motion Planning*. Vol. 20. 12. 2012 (cit. on p. 8).
- [11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. «Ssd: Single shot multibox detector». In: *European conference on computer vision*. Springer. 2016, pp. 21–37 (cit. on pp. 8, 32).
- [12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on pp. 8, 32).
- [13] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. «Exploring the Performance of ROS2». In: *Proceedings of the 13th International Conference on Embedded Software*. EMSOFT '16. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2016. ISBN: 9781450344852. DOI: 10.1145/2968478.2968502. URL: <https://doi.org/10.1145/2968478.2968502> (cit. on p. 9).

- [14] *Managed nodes*. URL: https://design.ros2.org/articles/node_lifecycle.html (visited on 11/10/2022) (cit. on p. 9).
- [15] *ROS 2 Foxy Fitzroy: Setting a new standard for production robot development*. URL: <https://aws.amazon.com/blogs/robotics/ros-2-foxy-fitzroy-robot-development/> (visited on 11/10/2022) (cit. on p. 10).
- [16] *Understanding nodes*. URL: <https://docs.ros.org/en/foxy/Tutorials/Beginner-%20CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html> (visited on 11/05/2022) (cit. on p. 11).
- [17] *RViz User Guide*. URL: <http://wiki.ros.org/rviz/UserGuide> (visited on 11/05/2022) (cit. on p. 13).
- [18] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. «The Marathon 2: A Navigation System». In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020. URL: <https://github.com/ros-planning/navigation2> (cit. on p. 15).
- [19] *Navigation concepts*. URL: <https://navigation.ros.org/concepts/index.html#ros-2> (visited on 11/07/2022) (cit. on p. 15).
- [20] *ROS to ROS 2 Navigation*. URL: <https://navigation.ros.org/> (visited on 11/07/2022) (cit. on pp. 15, 16).
- [21] *Costmap2D Package Summary*. URL: http://wiki.ros.org/costmap_2d (visited on 11/09/2022) (cit. on pp. 19, 20).
- [22] *VORT ARIASALUS 200*. URL: <https://www.vortice.it/it/trattamento-aria/depuratori/centralizzati/25044> (visited on 11/10/2022) (cit. on p. 22).
- [23] *VL53L5CX Time-of-Flight sensor*. URL: <https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html> (visited on 11/12/2022) (cit. on pp. 24, 25).
- [24] *OpenCV AI Kit: OAK—D*. URL: <https://store.opencv.ai/products/oak-d> (visited on 11/10/2022) (cit. on pp. 25, 26).
- [25] *T265*. URL: <https://www.intelrealsense.com/tracking-camera-t265/> (visited on 11/10/2022) (cit. on p. 26).

- [26] *XENSIV™ PAS CO2 sensor*. URL: <https://www.infineon.com/cms/en/product/sensor/co2-sensors/pasco2v01/> (visited on 11/12/2022) (cit. on p. 27).
- [27] *social_navigation_layers*. URL: http://wiki.ros.org/social_navigation_layers (visited on 11/13/2022) (cit. on p. 28).
- [28] *Object Detection using SSD Mobilenet and Tensorflow Object Detection API*. URL: <https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd0691> (visited on 11/23/2022) (cit. on p. 33).
- [29] *ObjectTracker*. URL: https://docs.luxonis.com/projects/api/en/latest/components/nodes/object_tracker/ (visited on 11/23/2022) (cit. on pp. 33, 34).
- [30] *Writing a New Costmap2D Plugin*. URL: https://navigation.ros.org/plugin_tutorials/docs/writing_new_costmap2d_plugin.html (visited on 11/13/2022) (cit. on pp. 35, 37).
- [31] *Nexus 4WD Mecanum Wheel Mobile Robot*. URL: <https://www.nexusrobot.com/product/4wd-mecanum-wheel-mobile-arduino-robotics-car-10011.html> (visited on 11/20/2022) (cit. on p. 43).
- [32] Noé Pérez-Higueras, Roberto Otero, Fernando Caballero, and Luis Merino. «HuNavSim: A ROS2 Human Navigation Simulator for Benchmarking Human-Aware Robot Navigation». In: () (cit. on p. 49).
- [33] Abhijat Biswas, Allan Wang, Gustavo Silvera, Aaron Steinfeld, and Henny Admoni. «SocNavBench: A Grounded Simulation Testing Framework for Evaluating Social Navigation». In: *J. Hum.-Robot Interact.* 11.3 (July 2022). DOI: 10.1145/3476413. URL: <https://doi.org/10.1145/3476413> (cit. on p. 49).