# POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI



Master of Science in Mechatronic Engineering

Master of Science Thesis

# Modelling and Validation of Socially-Aware Navigation Algorithms for Mobile Robots in Populated Environments


**Supervisors**
Prof. Alessandro Rizzo
Dr. Giada Galati

**Candidate**
Giacomo Vignolo
Student ID 302159


December 2023

# Acknowledgements

All'inizio di questo elaborato, che rappresenta l'atto conclusivo del mio lungo e impegnativo percorso universitario, desidero dedicare alcune righe a tutti coloro che sono stati al mio fianco, rendendo più leggero e piacevole questo viaggio.

Un sentito ringraziamento va al mio relatore, Alessandro Rizzo, e alla Dott.ssa Giada Galati, per la loro infinita disponibilità, pazienza e tempestività di risposta ad ogni mio dubbio e perplessità. Un grazie speciale va ad Andrea, collaboratore essenziale per il progetto di tesi, e a tutti i dottorandi e tesisti che ho conosciuto in questi mesi al DET e nel laboratorio di robotica. Ogni giorno mi sono sentito parte del gruppo, condividendo momenti utili per la mia crescita professionale, ma anche divertenti, come il giorno in cui sono arrivato in ritardo per preparare le frittelle di cipolla per tutti.

Senza il sostegno dei miei genitori e della mia famiglia, non sarei mai potuto arrivare fin qui. Grazie per essere stati sempre al mio fianco, nonostante molte volte l'ansia e la pressione mi abbiano reso antipatico soprattutto nei vostri confronti. Grazie papà, per avermi dato l'opportunità di raggiungere questo traguardo. Grazie mamma, per le borsate di cibo e perché è principalmente grazie a te se in molti momenti non mi sono lasciato prendere dallo sconforto. Grazie Laura, che per prima mi ha insegnato ad usare il pc e mi ha regalato il primo Arduino, e grazie Matteo, sempre pronto a darmi consigli "politecnici". Grazie nonna Anna per tutti i sughi che mi hai preparato in questi anni e per avermi fatto sentire a casa ogni domenica mattina, quando ero felice di smettere di studiare per venire a prenderti in macchina per pranzare tutti insieme. Grazie anche a tutti gli altri.

Ringrazio anche tutti i miei colleghi, gli amici vecchi e quelli nuovi che in questi anni hanno sopportato le mie lamentele e assecondato le mie pazzie, come le gite in montagna dell'ultimo minuto o le cene improvvisate con quello che rimane nel frigo per non sprecare nulla.

Grazie Alex, con te ho condiviso tutti i corsi della magistrale e siamo stati un ottimo team. Il rapporto che si è creato va al di fuori delle mura universitarie, e come sai, non sono molto bravo a esprimermi in questi casi. Ti ringrazio per avermi spronato a dare il meglio ogni giorno e spero che anche per te sia stato così. Sto

ancora aspettando l'arbanella di pesto di pistacchio!

Grazie Dave, Luca, Silvia, Eli, Mauri e Asti. Via Pigafetta è stata per me un luogo felice quanto la Liguria. Nonostante fossi entrato in casa da quasi sconosciuto, mi avete subito fatto sentire bene e avete sempre avuto tempo di scambiare due parole quando serviva. Auguro a chiunque di avere nella loro vita dei coinquilini come voi. E poi, siete stati le mie cavie preferite per i miei esperimenti culinari. Che si fa alla cena di Natale? Iniziate a pensarci che manca molto poco.

Grazie Gabriele, Giulio, Mariagrazia, Stelio, Mattia, Federico e a tutti i miei colleghi e compagni di studio di questi anni. Dagli aperitivi con gnocchi alla romana prima delle sessioni di fisica due, fino alle giornate passate ad insegnarci il dialetto vicendevolmente, ho passato dei bei momenti con voi che sicuramente non dimenticherò.

Grazie a Fra e Patty. Ci siete sempre state, anche se ci vediamo pochi giorni all'anno. Grazie a Rachi, Bru, Ilyas, Bubi, Virgi, Bertu, Perro, Dodo, Fil, Toni, Zuna, Stella, Dami... E a tutti gli amici che hanno condiviso con me dei bei momenti negli ultimi anni e che qui non riesco a citare. Sono quello che sono e sono riuscito ad arrivare fino a qui anche grazie ai momenti di svago passati con voi e alle discussioni che abbiamo avuto. Sono cambiato molto negli ultimi anni ed ognuno di voi mi ha trasmesso qualcosa. Spero di aver fatto altrettanto.

Infine, vorrei ringraziare tutti i miei amici "storici" di Pallare. A causa della lontananza abbiamo perso un po' i contatti, ma quando ci ritroviamo insieme alle feste di paese o alle castagnate, mi fate sentire sempre parte di qualcosa che non ha mai smesso di esistere. Ora che ho finito di studiare non ho più scuse e riusciremo ad organizzare la tanto agognata pizzata!

Grazie a tutti, senza di voi non ce l'avrei mai fatta.

Ora inizia il difficile. Ora inizia il bello.

**Abstract**

In our current era, robots have transcended industrial confines and are being deployed across multiple sectors, necessitating a heightened degree of autonomy. Anticipating the imminent future, it is plausible that human-robot interaction will be part of our daily activities, and as a consequence, robotic entities will inevitably manage their navigation within environments shared with humans. In these contexts, a robot must engage in special behaviors designed to ensure safety in its interactions with humans. This consideration extends beyond physical safety to include psychological aspects associated with sharing an environment with an automaton. To ensure that both aspects of the robot's behavior are effectively integrated, the robot should move with the aim of causing as little disturbance as possible to moving pedestrians. This involves a commitment to being socially accepted by respecting social and cultural norms imposed by society. Although existing algorithms for socially-aware robot navigation ensure safety, many of them generate unnatural trajectories and have limitations in predicting pedestrians' future movements. Thus, the primary objective of this thesis is to develop a navigation algorithm for mobile robots capable of predicting human future movements, with the aim of ensuring the generation of natural trajectories and guaranteeing comfort for interacting humans. To achieve our goal, the modeling of human motion in a shared environment, and consequently, the decision-making process during navigation, is conducted by introducing Game theory. Game theory serves as a mathematical tool in which players make decisions that may influence each other. Unlike the majority of the solutions in the literature, game theory models the decision-making process of humans, treating them and the robot as rational agents that interact with each other. Therefore not only the interaction between the robot and the pedestrians individually is modeled, but the prediction takes into account also the mutual influence of the pedestrians. In our approach, named Game Theory Planner (GTP), each agent aims to find an optimal sequence of actions to generate an optimal trajectory to follow. The solution of the game is considered to be the attainment of the well-known Nash equilibrium. Simulations have shown good performance, but the time needed for computations makes the overall algorithm impractical for real-time implementation on a real robot. To overcome this issue, the algorithm is used to create a training dataset for a fully-connected neural network, which is used in practice to replace the computation of the action decided by the GTP. The proposed approach has been validated through a Monte Carlo numerical simulation performed in Gazebo, reproducing a virtual representation of operating conditions. Specifically, the GTP is compared with two other state-of-the-art algorithms: the Social Force Model (SFM) and Optimal Reciprocal Collision Avoidance (ORCA).

The comparison has been performed using state-of-the-art metrics, focusing on the naturalness of the movements and the perceived comfort. Each algorithm is inserted as a local planner into the navigation stack of a simulated robot (Locobot wx250s of Trossen Robotics), and several point-goal trials are performed in an environment shared with simulated pedestrians. The evaluation procedure indicates that GTP reaches a similar performance to the other approaches, opening the possibility of reaching remarkable results with few predicted future improvements.

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**AI**
artificial intelligence

**AS**
average speed

**CNN**
convolutional neural network

**CPD**
closest pedestrian distance

**DOF**
degree of freedom

**DRL**
deep reinforcement learning

**FCNN**
fully connected neural network

**FL**
fuzzy logic

**GA**
genetic algorithm

**GTP**
game theory planner

**HRI**
: human robot interaction

**LP**
: linear program

**ML**
: machine learning

**MPC**
: model predictive control

**NN**
: neural network

**ORCA**
: optimal reciprocal collision avoidance

**PLR**
: path length ratio

**PR**
: path regularity

**QP**
: quadratic program

**ROS**
: robot operating system

**RVO**
: reciprocal velocity obstacles

**SBRA**
: sequential best response approach

**SFM**
: social force model

**VO**
: velocity obstacle

# Chapter 1

# Introduction

The following sections will elucidate the research context, the goals, and the methodology employed in this thesis. Subsequently, a detailed outline of the organization of this work will be provided.

## 1.1 Research context

In our current era, robots have transcended industrial confines and are being deployed across multiple sectors, necessitating a heightened degree of autonomy. Looking toward the imminent future, it is plausible that human-robot interaction will become a routine part of our daily activities. As a consequence, robotic entities will inevitably navigate through environments shared with humans. In these contexts, a robot must engage in special behaviors designed to ensure safety in its interactions with humans. This consideration extends beyond physical safety to include psychological aspects associated with sharing an environment with an automaton. To ensure that both aspects of the robot's behavior are effectively integrated, the robot should move with the aim of causing as little disturbance as possible to moving pedestrians. This involves a commitment to being socially accepted by respecting social and cultural norms imposed by society.

This objective can be achieved by invoking the concept of Socially-aware navigation, which is defined in [1] as: "the strategy exhibited by a social robot which identifies and follows social conventions in order to preserve a comfortable interaction with humans. The resulting behaviour is predictable, adaptable and easily understood by humans".

Navigation, in this context, becomes an extremely intricate aspect of service robot design, evolving into a complex interdisciplinary task rather than just a technical one. To enhance the acceptability of the robot, Kruse et al. [2] suggest three key features:

- *Comfort*, or the absence of annoyance and stress for humans in interaction with robots. It can be ensured by defining measurable constraints, and one of the most utilized concepts is the *personal space*, a region of space around humans that must be respected by the robot to ensure the comfort of individuals.

- *Naturalness*, or the similarity between robots and humans in low-level behaviour patterns. This concept represents the ability of the robot to mimic a human-like trajectory, ensuring that the overall path will be more predictable. Therefore, a human will interpret it as more friendly, resulting in higher social acceptability. This concept is closely related to the idea of Anthropomorphism, defined as humans' tendency to attribute human-like characteristics to non-human objects or entities [3]. Moreover, it can be measured by considering the smoothness of the generated path.

- *Sociability*, or the adherence to explicit high-level cultural conventions. It is closely related to the target culture and the tasks that the robot has to pursue. For example, avoiding a collision by passing on the right side of a hallway or waiting in a queue are aspects of this concept. Its evaluation is more challenging and is often done through real-world experimentations. The assessment of sociability, along with comfort and naturalness, is commonly carried out using questionnaires in the literature.

In our research project, the focus is on the first two topics of this list as central paradigms upon the design of a model able to predict future human movements. This model could then be integrated into the navigation framework of the robot by developing an appropriate algorithm, allowing it to be fully integrated and socially-accepted into a crowded space.

## 1.2 Thesis aim and our approach

The primary objective of this thesis is to develop a navigation algorithm for mobile robots capable of predicting human future movements, whose aims are to ensure the generation of natural trajectories and to guarantee comfort for interacting humans.

To achieve our goal, the modeling of human motion in a shared environment, and consequently, the decision-making process during navigation, is conducted by introducing Game Theory, a mathematical tool in which players make decisions that may influence those of the others [4]. This approach is considered innovative, as the majority of proposed models in literature often focus on individual predictions for each moving entity. In contrast, game theory takes into account the influence that each agent would have on the others. In our context, both the influence that

the robot has on pedestrians and the influence that each pedestrian would have on each other are considered. This concept is known as *interaction-awarness* [5].

Following the considerations explained above, the developed algorithm is named *Game Theory Planner* (GTP). The proposed method has been validated through a Monte Carlo numerical simulation performed in Gazebo [6] [7], reproducing a virtual representation of operating conditions (as shown in Figure 1.2). Specifically, the GTP is compared with two other state-of-the-art algorithms: the Social Force Model (SFM) [8] and Optimal Reciprocal Collision Avoidance (ORCA) [9]. The comparison has been performed using state-of-the-art metrics, in order to have a quantitative evaluation of its performance in terms of naturalness of motion and perceived user comfort during interactions. Therefore, each algorithm is inserted as a local planner into the real-time simulation of the navigation stack of a real robot (Locobot wx250s of Trossen Robotics [10], represented in Figure 1.1).

**Figure 1.1:** Locobot WX250s from Trossen Robotics[10]

## 1.3 Thesis organization

In the following, the thesis will be organized as explained below:

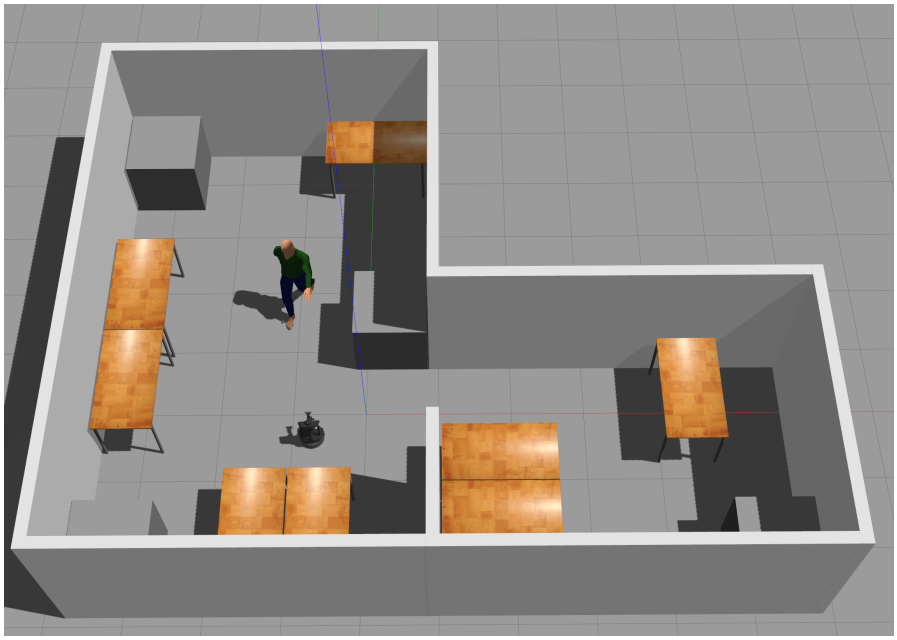

**Figure 1.2:** Sample representation of a simulation in Gazebo virtual environment

- **Chapter 2** will provide a comprehensive study of the fundamentals of autonomous robot navigation.
- **Chapter 3** will review the primary methods employed in recent years for socially-aware robot navigation. It will include a detailed analysis of Human-Robot Interaction modeling, with a focus on the concepts of Comfort, Naturalness, and Sociability.
- **Chapter 4** will offer detailed descriptions of the algorithms selected for implementation on the simulated robot. It will provide an in-depth overview of the proposed *Game Theory Planner*.
- **Chapter 5** will outline the software tools used during the design and simulation procedures. It will also describe the hardware components that compose the simulated robot.
- **Chapter 6** will focus on the preliminary simulations conducted in the Matlab environment. It will highlight important parameters and discuss potential future improvements.
- **Chapter 7** will provide an overview of the conducted evaluation tests and introduce the obtained results.

- **Chapter 8** is the concluding chapter, where an analysis of the results will be presented along with conclusions drawn from them. Additionally, considerations for potential future developments and upgrades to the proposed approach will be discussed.

# Chapter 2

# Robot navigation

In the following sections, a comprehensive overview of the main components of the navigation framework for an autonomous robot is presented. The focus is primarily on *path planning* procedures proposed in the literature. This discussion aims to provide a clearer overall view of the available solutions and outline the considerations that led to the design of an approach based on Game Theory.

## 2.1 Basics

Robot navigation, in summary, is based on four central concepts [11]:

- Perception: the process in which the sensory system captures the environment.
- Localization: the identification of the position of the robot within a map created using the data acquired in the perception phase.
- Path-planning: the computation of a valid sequence of configurations that the robot must follow in order to reach its goal without collisions.
- Motion control: the actuators of the agent are fed by proper commands in order to follow the desired path.

The previously explained steps are organized following the scheme in Figure 2.1. The whole process starts with a goal pose given to the robot. It starts the perception process by scanning the environment and the data coming from different types of sensors are combined together into a virtual representation of the environment. The map created is used to localize the robot within the space in a process called SLAM (Simultaneous Localization and Mapping) [12]. The actual pose just found is fed to the path-planning algorithm along with the goal one and the map, obtaining a path to be followed in order to avoid collisions and reach the desired location. Finally,

the planned trajectory is given to the motion controller, which computes the needed commands that generates the wanted movements of the actuators mounted on the robot. After a certain amount of time, called time step of the motion process, the process is repeated, with the dynamic update of the current position of the agent into the environment.

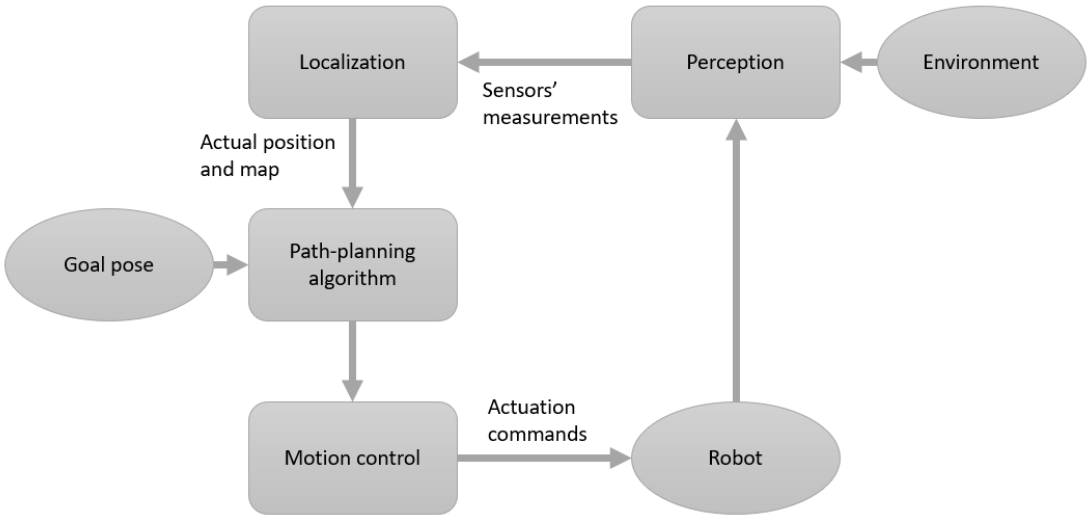


**Figure 2.1:** Robot navigation scheme

### 2.1.1 Path planning

The aim of this project is to develop a human-aware algorithm that is able to plan a feasible trajectory in a shared environment while ensuring safety, naturalness and social acceptability. In order to reach that goal, we have to focus on the path generation process performed by the path planning step. Normally, the planning paradigm can be divided into two strongly connected steps [13]:

- Global planning: algorithm aimed to find a feasible course path from the current pose to the given final goal pose.
- Local planning: algorithm aimed to generate the motion commands that the robot has to perform in order to reach a local goal in the immediate vicinity, that is given as input to the global planner.

The Global Planner uses a global representation of the world, usually generated taking into account past data acquired during preliminary operations like mapping the environment, in order to generate a feasible path that the robot is driven to follow. The global path is normally divided in sections, whose limits are used as

local goal points directly fed into the Local Planner. The latter uses a more detailed representation of the environment, a local map, in order to generate the commands needed to reach the given local goal. The process is subsequently repeated with the next local goal computed until the global pose is reached. A detailed scheme reporting the subdivision can be seen in Figure 2.2. On the other hand, a simple representation of the different explained parts into a map is reported in Figure 2.3.

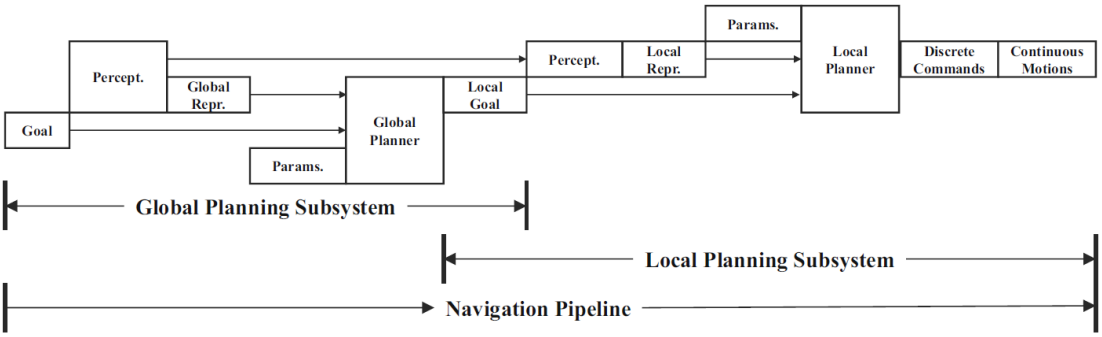


**Figure 2.2:** Navigation pipeline[13]

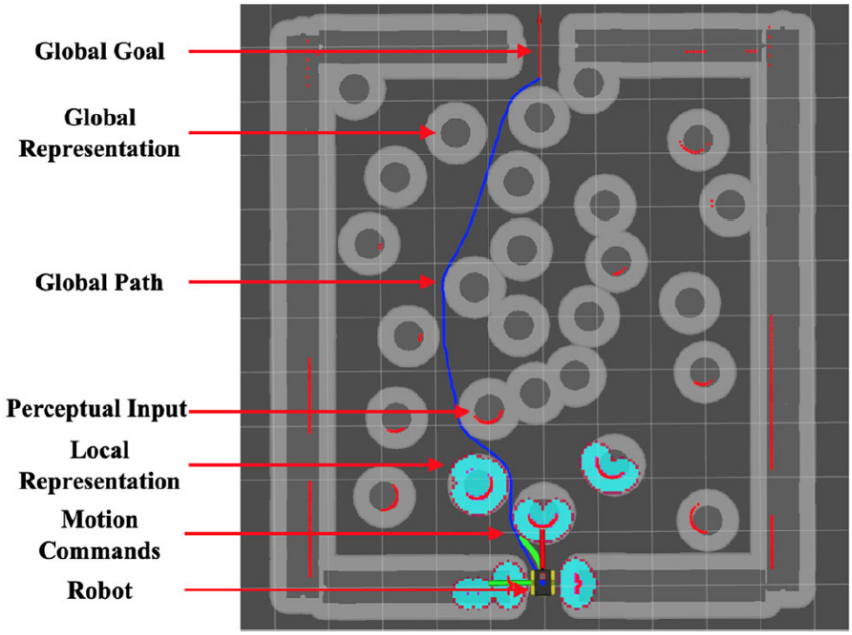


**Figure 2.3:** Navigation elements represented into a sample map[13]

In a practical implementation, all the main ingredients of a navigation pipeline are coded in blocks linked together by a middle-ware able to provide assistance

in data transfer, data acquisition and command actuation. In our case, we will use ROS (Robot Operating System) and its framework to perform such tasks. A deeper analysis of the ROS navigation stack will be available in Chapter 5.

## 2.2 Popular paradigms

During its movements, a mobile robot will primarily encounter two types of obstacles: static obstacles, such as walls, stationary objects, or furniture, and dynamic obstacles, including moving objects, opening doors, vehicles, or pedestrians. Over the years, numerous algorithms have been proposed and evaluated as navigation planners for mobile robots. The primary objective of these algorithms is to determine a sequence of configurations that the robot should adhere to in order to reach the desired goal, while also steering clear of collisions with obstacles. Generally, it is less challenging to address static obstacles as opposed to dynamic ones. In this section, we will concentrate on path-planning strategies that demonstrate the capability to guide the robot through a dynamic environment.

In the literature, a distinction is made between *classical strategies*, which rely on functional modeling of the environment to determine a path with the minimum cost necessary to follow, and *heuristic approaches*, which are integrated with rules derived from past experience or expert knowledge [11]. In general, classical approaches tend to struggle when dealing with dynamic environments, with some exceptions. Heuristic approaches, on the other hand, have gained significant importance in the context of robot navigation within dynamic and complex environments. Below, there is a group of examples that are well-recognized in literature.

Classical algorithms, in the context of robot navigation, may include methods such as the Roadmap approach, Cell Decomposition, Mathematical Programming, and the Artificial Potential Field (APF) technique [11]. The APF method is the sole classical approach capable of handling dynamic and uncertain environments, and it experienced a sudden surge in popularity among researchers. In 1985, O. Khatib introduced the APF method as one of the pioneering concepts for real-time collision avoidance in the field of robotics [14]. Initially, the approach was developed for manipulators and was later adapted and extended for application in the realm of mobile robots. In this method, the robot is essentially represented as a point mass moving within a force field generated by both the goal (an attractive pole) and the obstacles (repulsive poles). This field exerts forces on the robot, directing it towards the desired goal while avoiding obstacles. An example of implementation can be found in [15]. The primary limitation of this algorithm is its constrained applicability due to the tendency to become stuck in deadlock situations when encountering a local minimum within the force field. This issue, coupled with the algorithm's inability to anticipate the future movements of dynamic entities and

its absence of interaction modeling with dynamic obstacles, makes the approach unsuitable for social navigation, aligning with the majority of methods falling into the classical classification.

Transitioning to the other classification group, heuristic approaches encompass methods such as Genetic Algorithms (GA), Fuzzy Logic (FL), Neural Networks (NNs), Particle Swarm Optimization (PSO), and a multitude of other techniques [11]. Among the examples provided, Fuzzy Logic (FL) is often regarded as the most popular due to its readability and straightforward implementation process. This paradigm operates on the assumption that some processes or decisions are more accurately modeled using imprecise state descriptors like "warm", "cold", "big" or "small", rather than binary states (True/False) [16]. The input data, which is derived from sensors, is transformed into a state described in relation to its membership within a specific "Fuzzy Set". In the context of navigation, these sets could include descriptions like "dynamic obstacle approaching from the left" or "obstacle remaining stationary". Subsequently, a series of logical operations are executed to compute the fuzzy output, which, in this instance, might determine the velocity required for the robot to safely navigate the environment. An example of implementation can be found in [17].

Heuristic methods, such as the one previously described, are better suited for mobile robot navigation in dynamic and complex environments compared to classical approaches. Their unique characteristics also make them a suitable baseline for the development of social navigation, with necessary modifications to incorporate aspects of social acceptability and modeling of human-robot interaction. The following chapter will delve into a comprehensive examination of state-of-the-art approaches that have been utilized over the past decades within the realm of socially-aware robot navigation.

# Chapter 3

# Socially-aware robot navigation

Nowadays robots and automata have achieved remarkable levels of autonomy and precision in their movements. Notably, among the most exemplary projects currently under development and enhancement, we can point to the work of Boston Dynamic [18]. From industrial manipulators to humanoid robots, their creations exhibit remarkable reliability, enabling them to "address the most challenging automation tasks of today and tomorrow". However, it is important to note that their utilization in everyday environments remains somewhat restricted. In recent years, there has been a notable increase in the availability of service robots designed to assist with human tasks and enhance everyday life in domestic settings. These solutions are becoming more readily attainable and accessible at competitive prices, making it increasingly plausible to anticipate a rising presence of them in our daily lives. Indeed, there are numerous examples of mobile robots being utilized in environments where people share the same space. For instance, there has been significant progress in the field of service robotics in healthcare, particularly in response to challenges posed by COVID-19 [19] [20]. Additionally, robots have found applications in sectors such as hotels [21], restaurants [22] [23] and elderly assistance [24].

In these contexts, navigation has become an exceptionally challenging task due to the dynamic nature of the environment. What is more, success in accomplishing a given task is not solely reliant on the safety of the robot's movements but is also intricately linked to its social acceptability. Specifically, socially-aware navigation is defined as the strategy exhibited by a social robot which identifies and follows social conventions in order to preserve a comfortable interaction with humans [1]. The resulting movements of a robot using this concept as a starting point for its navigation framework will be predictable, adaptable and easily understood by

humans.

## 3.1 Related work

As previously mentioned in Chapter 1, once an embodied agent starts moving within a space shared with humans and engages in interactions, it must adhere not only to safety regulations but also a multitude of social norms. These norms can differ from one location to another and may be influenced by cultural factors or by the users' subjective perception of a robot [16].

To enhance the social acceptability of the robot, Kruse et al. [2] suggest Comfort, Naturalness and Sociability as key features to be ensured (see Chapter 1). Modeling human motion is a fundamental aspect in achieving these goals.

### 3.1.1 Human-robot interaction modeling

Considering the main goals of human-aware navigation expressed in the introduction of this section, a deep look on the consideration behind human motion modeling must be done. This step is crucial for highlighting the design and tuning methodologies elucidated in subsequent sections of the work.

#### Proxemics

When a robot moves in a shared space, it can be the cause of discomfort in encountered pedestrians. For instance, this intimate reaction can be caused by moving too close or too fast. A fundamental aspect in this context is the so called *proxemics*, defined as "the study of spatial distances that individuals maintain in various social and interpersonal situations" [1]. This subject assume the existence of certain unwritten rules that guide the definition of a *personal space*, defined as "the region around humans that they actively maintain into which others cannot intrude without causing discomfort" [1]. An example of the typical arrangement of humans into a shared environment is reported in Figure 3.1.

This space can be translated into a distance, that is considered as the minimum gap that a robot should maintain to its nearest person in the space in order to ensure users' comfort. In literature, we can found a model for human management of space that divides it into concentric zones described by distances from the human body [1] [2]. The most important zones are represented in Figure 3.2, whose limits are listed below:

- public zone ($>$ 3.6 m)
- social zone (1.2-3.6 m)


**Figure 3.1:** Typical arrangement of humans observing reciprocally their personal space, highlighted as blue circles [1].

- personal zone (0.45-1.2 m)
- intimate zone (<= 0.45 m)

The robot must move in the space avoiding collisions, remaining at a distance that ensures the comfort of the interacting pedestrians. In our work, we consider acceptable a trajectory that avoids the personal zone of the pedestrian, using the distances previously listed as a reference for the algorithms' parameters tuning.

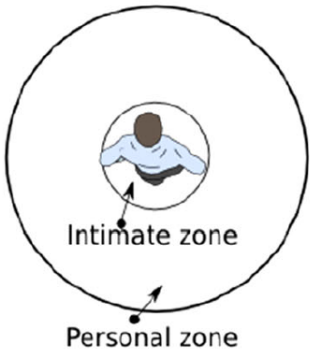


**Figure 3.2:** Representation of the intimate and personal zone [1].

Another interesting consideration about the human management of space is related to the concept of *information process space* (IPS). It is defined as "the space within which all objects are considered as potential obstacles when a pedestrian is planning future trajectories" [25]. The paper points out that the shape of the IPS is conic, and pedestrians do not pay attention in elements that draw an angle

more than 45 degrees from the walking direction [1]. A simple representation can be found in Figure 3.3.

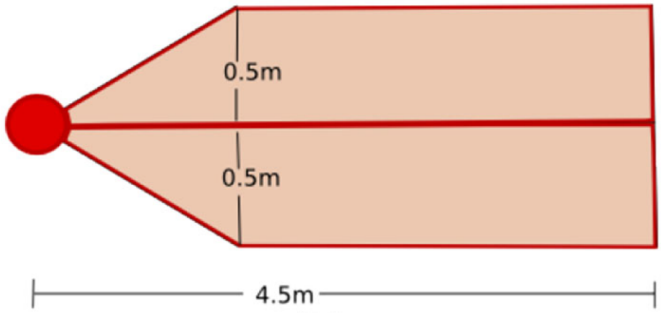


**Figure 3.3:** Representation of the information process space [1].

Following these considerations, our GTP algorithm is designed considering a limited field of view of the robot during its computations, in order to reach an higher human-likeness, aiming to an increase of social acceptability. More details can be found in Chapter 4.

### Natural motion

A fundamental assumption in the context of human robot interaction is that if a robot mimics human (or animal) behavior, interactions become more straightforward and intuitive for the individuals encountered. Therefore the concept of naturalness, or human-likeness, should be considered in the design of methods employed in robot navigation. This concept is strictly related to the interpretation that the human being produces looking at the robot movements: a trajectory is considered natural if it is predictable, understandable, readable or legible [2].

What is more, an important feature to be taken into account is the smoothness of the trajectory generated by the considered path planning algorithm. It is related to the geometry of the path, but also to the velocity profile [2]. A possible quantitative measure of this concept is presented into Chapter 7, where a comprehensive summary of the main evaluation metrics is done in order to define the one used in the final experimental evaluation.

These concepts can be linked to the so called "anthropomorphism", defined as the humans' attitude to attribute human-like characteristics to non-human objects or entities [3].

Similarity to human motion and acceptability are considered as proportional to each other, but when the motion is too human-like in some manner the person involved enters into the well known uncanny valley problem [26]. Basically, if the robot is too similar to a human being in his behaviour or facets, the human will

enter into a discomfort condition that is difficult to eliminate. The fact that the robot used in this work is not human-shaped obviously avoid totally this problem, which however is worth mentioning.

### 3.1.2 Solutions proposed in literature

Considering the basics of HMI presented in the previous section, we can summarize the requirements that a socially-aware path planning algorithm should have [2]:

- Respect personal zones, as studied in proxemics theory.
- Respect affordance spaces, identifying the regions of space where movement is feasible and navigating by adhering to the cues and recommendations provided by the environment.
- Avoid culturally disapproved behaviours.
- Avoid erratic motions or noises that cause distraction.
- Reduce velocity when approaching a person.
- Modulate gaze direction, that in a mobile robot context can be associated to the pointing direction of the movement, trying to maximize the smoothness of the path.

Following these main objectives, the scientific community relating to the study of autonomous navigation of social robots in the past decades has attempted to create numerous path planning algorithms. In the following sections we propose a simple double independent classification of the main examples found in literature, whose discriminant criteria are the period of time in which the procedure has been proposed and the overall shape of the generated trajectories, underling a specific behaviour in interaction. The learning-based approaches are here treated as separate entities even if they can be inserted in both the classifications easily, due to their particular declinable nature and for the sake of order.

#### Temporal classification

From a temporal point of view, the social navigation algorithms of autonomous robots can be divided into two distinct temporal strands:

- *Decoupled modeling*
- *Interaction modeling*

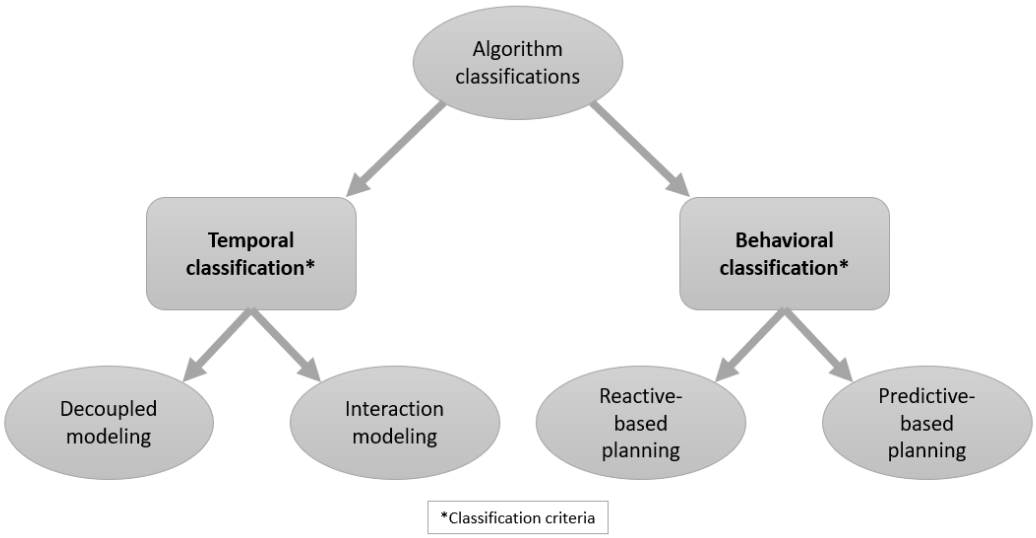


**Figure 3.4:** Algorithm classifications proposed.

The first algorithms to be developed focused on decoupled models [27], where individual agents in the environment are treated as self-contained and independent entities. The pedestrians are treated as dynamic obstacles, without taking into consideration any kind of prediction or interaction. One of the most famous methods is the dynamic window approach [28], where the dynamic of the robot is studied in order to define a set of reachable velocity in a short time interval, known as dynamic window, among with the moving velocity is chosen by means of an objective function minimization. Other examples that are worth mentioning are the randomized kinodynamic planning approach [29] and the velocity obstacle method[30], that is deeply discussed in Chapter 4.

The main problem of using decoupled models is the tendency to enter into the serious problem of freezing robots[31], defined as the situation in which the planner decides that all forward paths are unsafe, and the robot freezes in place (or performs unnecessary maneuvers). This problem, along with the uncertainty related to the future movements of pedestrians, ends up with oscillating trajectories that are potentially unsafe and unnatural. To address these challenges, researchers have concentrated on modeling interactions between humans and robots, as well as the interactions among humans during navigation within shared spaces.

The most widespread algorithm that falls into this second group is the social force model (SFM)[8], which tries to model the iterations by considering each agent in the shared space as a material point subject to an attractive force from the goal and repulsive forces generated by other individuals and obstacles. This is one of the algorithm chosen for performance comparison with our proposed GTP, therefore

there is a deeper discussion in Chapter 4.

Another widely used algorithm is the reciprocal velocity obstacle (RVO) [32, 33], which is an evolution of the previously cited VO concept taking into account the reciprocal interaction between the agents. The method is then upgraded into the optimal reciprocal collision avoidance (ORCA)[9] algorithm. Also the ORCA algorithm is chosen as a reference for performance comparison in the testing procedure of this project. Therefore, also in this case a deeper analysis is done in Chapter 4. Other examples that are worth mentioning are the learning-based approaches [13], which are treated as a separate group in the last section of this chapter.

### Behavioral classification

From a behavioural point of view, the path planning algorithms of social robots can be divided into two principal groups [34]:

- *Reactive-based planning*
- *Predictive-based planning*

A simple representation of the general concept behind this division is shown in Figure 3.5

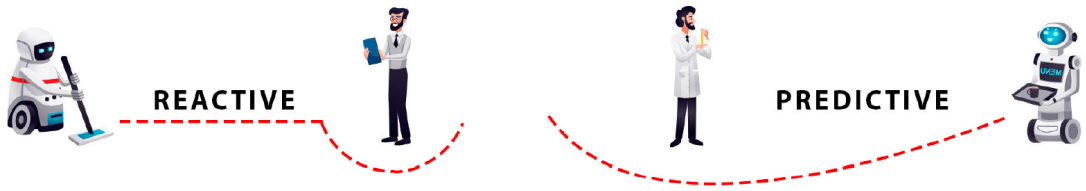


**Figure 3.5:** Navigation methods for robot social behavior[34].

Starting from the first group, a reactive-based planning algorithm generates a trajectory in which the robot changes its moving direction when a human appears to be colliding with it [34]. Generally in this type of approaches, there is not an explicit prediction of human movement, but only a rough check about a possible collision or not. Recalling proxemics theory, the robot moves in the space until the distance between the closest pedestrian enters into a position near the personal zone, inducing the computation of an avoidance manoeuvre. The resulting trajectory is sickle-shaped, as it can be seen on the left side of Figure 3.5. Both SFM and ORCA fall into this category. The pros of the reactive-based methods are the more affordable computational complexity and the more reliable usage in unstructured environments, rather than prediction-based ones. The main drawback is the

less friendly and human-like movement generated, making these approaches less predictable from a human point of view. What is more, the absence of prediction can cause freezing situations in which the robots enter too much into the personal zone of the pedestrian and the robot must be stopped in order to ensure safety. This situation has been experienced during the real testing of the algorithm on the locobot, ending up with the consideration that a predictive-based algorithm should be more suitable in indoor applications, where the space is limited and the map is structured.

Focusing now on the second group, a predictive-based planning algorithm predicts the future states of the people around the agent and then computes its trajectory by taking into account that information to avoid collisions in advance [34]. Therefore adopting this solution for navigation, the trajectory generated should be more natural and the user should perceive high comfort. The main issue related to these algorithms is the quite high computational complexity, that make necessary the usage of dedicated hardware components, for instance a GPU, where performing fast computations. The results analyzed in literature for these approaches are generally better than reactive ones [34]. The GTP algorithm proposed in this project falls into this category, as well as the reference works [4] [5], that point out the idea of using game theory as a modeling base for interaction with humans and prediction of their movements (more information can be found in Chapter 4).

Other prediction approaches use, for instance, probabilistic frameworks or learning-based path planners. An example of probabilistic approach can be found in [35], where the solution proposed, named RR-GP, uses Gaussian processes (GP) to predict the movement of the opponents and the rapidly exploring random tree (RRT) to identify probabilistically feasible paths. Taking into consideration the learning-based predictors, one of the most cited ones is SocioSense [36]. This approach uses Bayesian learning and Personality Trait theory in order to train a CNN from real trajectories' states of human motion recorded into surveillance videos, ending up with a planner that improves significantly the performance of long-term predictions by comparison with other approaches.

Overall, many authors in literature assert that prediction methods in future applications should take precedence over the reactive ones [34], due to their smoother and more natural trajectory generation.

### Learning-based algorithms

In recent years, thanks to the advances in computational capabilities of mobile-board computers mounted in service robots, algorithms that use machine learning as a base for their decisions became very popular among researchers. Due to their extremely various application field, learning algorithms can be treated as a separate group, that may intersect the other classifications previously addressed.

The section will start with general considerations about ML processes and then a quick evaluation of the most used approaches is presented, along with some practical examples.

The usage of learning-based algorithms presents several advantages when used in socially-aware robot navigation, that can be summarized as:

- Time consumed and computational complexity reduction: a proper trained ML system can efficiently replace a very complex and time-consuming algorithm used, for instance, as predictor or decision-making process.
- Reduction of human engineering effort.
- Overall increase in performance compared to classical navigation algorithm based on direct programming of each functionality.

Along with the benefits, this solution comes with a group of disadvantages, summarized as:

- Difficult interpretability: ML systems can be difficult to interpret, making it challenging to understand the causes of a certain behaviour. As a consequence debugging and making diagnosis of problems it's more complicated than traditional approaches.
- Data requirements: the learning process requires a huge amount of data, which can be expensive or time to collect or generate. Additionally the quality of the training sets influences the final results.
- Safety: connected to limited interpretability, in some cases it is possible to experience unwanted behaviours, which can be dangerous in environment shared with humans. It's important to careful design recovery or emergency behaviours in order to ensure that the robot's movement remains safe and socially acceptable in all situations.
- Lack of practice experience: many learning-based approaches have been studied mostly in simulation and have occasionally been applied to simple real-world environments.
- Bias: Machine learning methods can amplify existing biases in the data, leading to unwanted behaviours. It is important to carefully curate the training data to ensure that the robot is not inadvertently perpetuating biases. Not being able to see similar goal configurations during training makes it hard to generalize to arbitrarily specified goals.
- Difficult comparability: there is a lack of common metrics that makes it difficult to compare classical and machine learning methods. A deeper discussion about this topic can be found in Chapter 7.

Following the consideration expressed in [13], there are in general two starting approaches when a ML system is designed in the field of mobile robot navigation. The first one consists on the complete replacement of the navigation stack, the latter one focuses on the replacement of only a single component of the framework.

*Learning the entire navigation stack* is equivalent to consider the navigation task as a black box system with raw unprocessed perceptions signals as input and motion commands as outputs. The process is relatively straightforward and the behaviour depends primarily on the data used in the training process, which can be taken from real records of human trajectories, like in the previously cited Sociosense [36], or autonomously generated by means of numerical simulations. The most used and replicated learning frameworks in this context are Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), where the algorithms try to mimic the way of learning of humans or animals. One of the first example of application of DRL in navigation can be found in [37], that reach the goal to prevent collisions with dynamic obstacles, but does not take into account social conventions. An upgraded version of the algorithm is presented in [38]. Another worth to mentioned example is [39], that primarily concentrates on modeling the interaction between robots and crowds, utilizing information concerning the positioning of individuals in relation to the agent, and therefore it also explores the interaction between individuals as they engage with each other.

Passing to the alternative approach, that is *learning navigation subsystems*, the main difference is that the overall framework remains fixed, maintaining its fundamental features described in Chapter 2, while only the target functionality is replaced by a ML system. The researchers have focused their attention primarily in the local planner learning, with very few attempts on the global planner side [13]. In our work we use a fully connected neural network to replace the decision-making process into the local planner algorithm in order to implement the real-time version of the GTP, falling in a certain way in this presented learning-based algorithm category (see Chapter 4 for more details). Another example that uses this methodology is proposed in [40], where a classical global planner is used and a local trajectory planner and velocity controller adjust the behavior of the robot using simple deep attention mechanisms. In other examples found in literature, also other subsystems can be replaced by a learned version. For instance, in [41] there is a learning system aiming to produce the cost-maps used in the navigation pipeline.

# Chapter 4

# Algorithms description

In order to have more significant results and comparisons, three algorithms are developed and tested through the project. The first two, SFM and ORCA, are state of the art algorithms that have been widely used as comparative baselines. They can be defined as the two main representative algorithms for the last two decades[27].

The third one, which is named Game Theory Planner (GTP), is a modified version of a game theory based algorithm that takes inspiration from [4]. The employed predictive method have been tested in [42] by our department research group, using a qualitative questionnaire based evaluation procedure. It is expression of an innovative strategy of socially-aware motion planning, since not so many works are dedicated to game theory algorithms.

## 4.1 Social Force Model

The first algorithm presented, social force model (SFM), has been developed by a collaborator of the project team. This section will explain the basics of the algorithm functionalities and its main characteristics, leaving some technical details and comments on the effective realization, since this is out of the scope of this thesis redaction.

### 4.1.1 Basics

The SFM is a mathematical model describing the behaviour of a human moving into a crowd. It has been firstly introduced in 1995 by Helbling et al.[8]. Moreover, due to the low computational complexity, immediately it became one of the most popular models. Today this model is considered one of the most representative state-of-the-art models of social navigation.

The fundamental functioning principle operates on the presupposition that an individual exposed to a sensory stimulus will respond by adopting a behavior influenced by their personal objectives. Normally a pedestrian is used to react in an automatic way, following some rules built up personal experience. SFM tries to put these rules into an equation of motion, assuming that human's acceleration, described as the variation of the preferred velocity, is related to a vectorial quantity $\mathbf{F}_\alpha(t)$ that can be interpreted as a *social force* [8].

Therefore, pedestrians are regarded as point-like entities within a field of fictitious forces. These forces determines in them a tendency to move in a specific direction and at a certain speed, influenced by their interactions with the environment and other agents.

### 4.1.2 Social forces definition

Let us assume the presence of $n \in \mathbb{N}$ pedestrians moving into a bi-dimensional environment, whose main aim is to reach the personal goal position $\mathbf{p}_i^{goal}$ while avoiding obstacles and other agents. Following the explanations of the original paper in which the model has been introduced in the research field [8], the main social forces that determine the model motion are:

- **Goal attraction force:** the agent in position $\mathbf{p}_i(t)$ is attracted by the goal position $\mathbf{p}_i(t)$ with a force equal to

$$\mathbf{F}_i^{goal}(t) = \frac{\mathbf{v}_i^{desired} \frac{\mathbf{p}_i^{goal} - \mathbf{p}_i(t)}{\left\|\mathbf{p}_i^{goal} - \mathbf{p}_i(t)\right\|} - \mathbf{v}_i(t)}{\alpha_i} \tag{4.1}$$

  where $\alpha_i \in \mathbb{R}^+$ is called "relaxation time", tuning the attraction, and $\mathbf{v}_i^{desired}$ is the velocity that the agent want to reach during his motion.

- **Other pedestrians repulsive forces:** premising that an individual in motion heightens the feeling of discomfort when another person comes in close proximity, a repulsive force is generated proportionally to the distance among the actual agent and the others in the shared environment. The formal definition [43] is

$$\mathbf{F}_{i,j}^{rep}(t) = A_i \exp\left\{\frac{r_i + r_j - \left\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\right\|}{B_i}\right\} F_{i,j}^{fov}(t)\mathbf{n}_{i,j}(t) \tag{4.2}$$

  where $j$ represents a generic opponent in the environment, $A_i$ and $B_i \in \mathbb{R}^+$ tune the strength and range of the repulsion, $\mathbf{n}_{i,j}(t) = \frac{\mathbf{p}_i(t) - \mathbf{p}_j(t)}{\left\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\right\|}$ is the direction from the opponent position, $r_i$ and $r_j$ are the radii of the personal

spaces and $F_{i,j}^{fov}(t) = \lambda + (1-\lambda)\frac{1+\cos\gamma_{i,j}(t)}{2}$ represents a time-varying anysotropic factor that capture the effect of the limited field of view [25] ($\gamma$ is the angle between the actual direction of motion of $i$ and the segment joining the positions of $i$ and $j$, while $\lambda \in [0,1]$).

- **Obstacles repulsive forces:** since the obstacle avoidance is another central task that the model has to achieve, similar considerations can be done to the force generated by the opponents in the environment. Therefore the force sensed by the agent due to the nearest obstacle can be expressed as:

$$\mathbf{F}_i^{obs}(t) = \exp\left\{1 - \frac{\|\mathbf{p}_i(t) - \mathbf{p}_{obs}(t)\|}{R_0}\right\} F_{i,obs}^{fov}(t)\mathbf{n}_{i,obs}(t) \tag{4.3}$$

  where $\mathbf{p}_{obs}$ is the nearest obstacle position, $R_0$ is the minimum acceptable distance, $F_{i,obs}^{fov}(t)$ is the anysotropic factor defined in the same way as the previously analyzed and $\mathbf{n}_{i,obs}(t) = \frac{\mathbf{p}_i(t) - \mathbf{p}_obs(t)}{\|\mathbf{p}_i(t) - \mathbf{p}_obs(t)\|}$ is the direction from the obstacle position.

### 4.1.3 Motion generation

Considering all the forces acting on the generic agent, applying the superposition principle it is possible to compute the resultant force $\mathbf{F}_i^{res}(t) = \mathbf{F}_i^{goal}(t) + \sum_j \mathbf{F}_{i,j}^{rep}(t) + \mathbf{F}_i^{obs}(t)$. From this resultant, assuming unitary mass and applying the laws of Newtonian mechanics, we can find out the equations governing the agent's motion.

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i^{res}(t) \tag{4.4}$$

$$\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t) \tag{4.5}$$

Equation 4.4 computes the velocity in the next step of the simulation, while equation 4.5 updates agent's position in the space.

## 4.2 Optimal Reciprocal Collision Avoidance

The Optimal Reciprocal Collision Avoidance (ORCA) algorithm has been introduced in 2011 by Van Den Berg et al. [9]. It is based on the assumption that each moving entity in the environment can be translated into a Velocity Obstacles (VO) for the agent, which then computes a feasible velocity to assume in order to avoid all the possible collisions. Over the years this approach has become increasingly important, gaining the status of a state-of-the-art method. What is more, in recent years, it is used as comparative baseline for algorithm evaluation [27] and as collision

avoidance feature in more complex frameworks. For instance, in [44] ORCA is used in addition to Deep Reinforcement Learning (DRL) to solve the collision avoidance problem in complex scenarios full of interactive obstacles and in [45] it is combined with MPC, reducing velocity vibrations.

Here are some of the algorithm's principal characteristics:

- Decentralized: ORCA is a fully decentralized navigation algorithm, which means that each robot or agent computes its own collision-free velocity based only on its own sensor data and without explicit communication with other robots.

- Velocity-based: ORCA is a velocity-based algorithm, which means that it calculates collision-free velocities for the robot rather than specific paths or trajectories. This allows the robot to adjust its movement in real-time based on changing environmental conditions.

- Optimization-based: ORCA is an optimization-based algorithm, which means that it computes an optimal solution to the navigation problem by minimizing the risk of collisions between agents. In our reformulation, it uses a quadratic program to calculate the optimal collision-free velocity for each agent.

- Reciprocal: ORCA is a reciprocal algorithm, which means that it takes into account the velocity and trajectory of other agents in the environment when computing its own collision-free velocity. It ensures that the velocities of all agents are compatible, which helps to avoid deadlock situations where two or more agents are blocked.

The core assumption behind the creation of a socially-aware motion planner using this algorithm is that pedestrians facing the robot will react to its presence by modifying their velocities. This modification follows a strategy comparable to the one employed by the robot, which is based on selecting a velocity that avoids collisions within a predefined time horizon.

The decentralized peculiarity of this algorithm make possible his direct translation as a planner into the navigation framework built in ROS. The computational complexity can be tackled by the internal PC of the robot, the Locobot WX250s [10], and there is no need to modify the main elements.

ORCA is used originally as a planner for a multi-agent navigation framework, but in our case, the only agent that uses the ORCA logic is the robot, while the pedestrians are assumed to be a kind of reactive dynamical obstacle. The prevision of future human movement is done using the same strategy used by the robot to select the next step velocity, but it is only a simple collision check. Therefore this approach falls into the previously analyzed group of reactive-based planners.

### 4.2.1 Velocity Obstacles

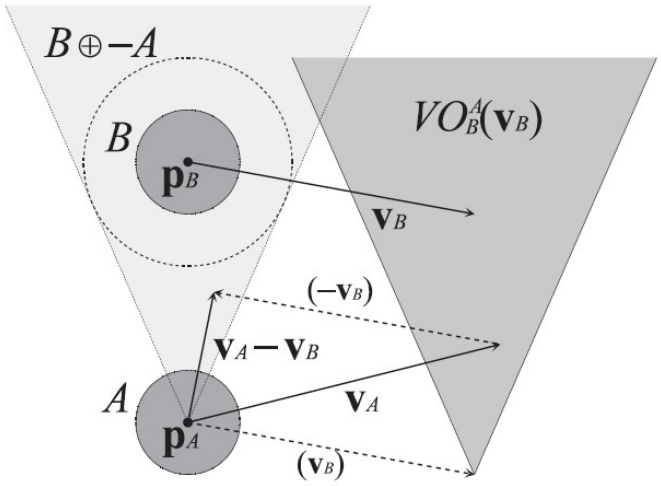


**Figure 4.1:** The Velocity Obstacle $\mathrm{VO}_B^A(\mathbf{v}_B)$ of a disc-shaped obstacle B to a disk-shaped agent A [32].

The basic geometric construct that is used by the algorithm to build the mathematical description of the navigation problem is the velocity obstacle [30] (VO), as shown in Figure 4.1. Such figure represents the set of velocities that would result in a collision with the indicated obstacle (B in the figure), if the agent (A in the figure) will maintain constant its actual velocity, given a certain time horizon. Practically the set is defined by means of the relative velocity $\mathbf{v}_{rel}$ between the agent and the obstacle, computed as the difference between their actual velocities.

The same concept can be generalized considering two agents i and j. The VO between them is defined as the set of all velocities $\mathbf{v}_i$ of agent i that would result in a collision with agent j, if its velocity $\mathbf{v}_j$ will be maintained through the given time horizon. The mathematical definition is expressed in Equation 4.6, where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the positions of the agents and $\mathrm{r}_{i,j}^2$ is the squared sum of their radii.

$$\mathrm{VO}_j^i(\mathbf{v}_j) = \left\{\mathbf{v}_i | (\mathbf{v}_i - \mathbf{v}_j)(\mathbf{p}_i - \mathbf{p}_j) \geq 0, |\mathbf{v}_i - \mathbf{v}_j|^2 \leq \mathrm{r}_{i,j}^2\right\} \tag{4.6}$$

The first equation defining the VO ensures that the relative velocity is in the direction of the relative position between the two agents. This means that the two agents are moving towards each other. The second equation ensures that the distance between the two agents at the time of collision is less than or equal the sum of their radii, including in the set all the velocities that will result in a collision.

This concept can be used for multi-agent or single-agent navigation, when each agent regards the other agents as moving obstacles, choosing at each step a velocity

that lies outside any of the velocity obstacles included by the other agents or obstacles, but this approach results in undesirable oscillatory motions [32].

### 4.2.2 Reciprocal Velocity Obstacles

Reciprocal Velocity Obstacles (RVOs), represented in Figure 4.2, are a generalization of VOs that can handle two-way interactions between agents [32]. VOs define a set of velocities that would result in a collision between two agents if one agent maintained its current velocity and the other agent moved with a velocity in the VO. RVOs, on the other hand, consider both agents' velocities and define a set of not allowable velocities for both agents that would result in a collision scenario. This means that RVOs take into account the "reciprocal" nature of the collision avoidance problem.

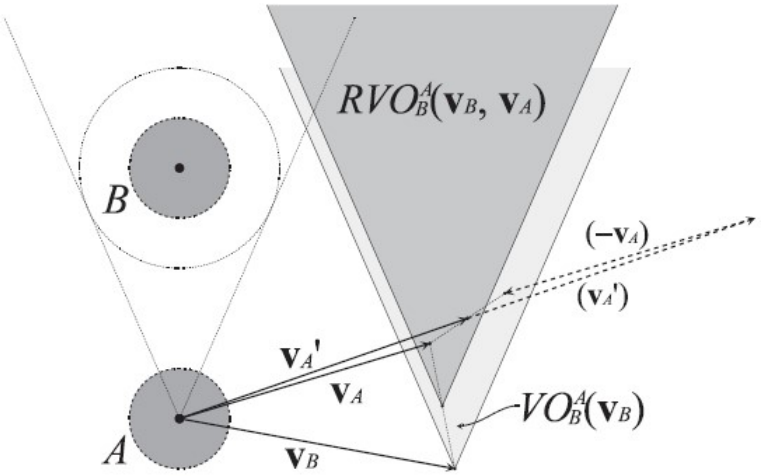


**Figure 4.2:** The Reciprocal Velocity Obstacle $\mathrm{RVO}^A_B(\mathbf{v}_B, \mathbf{v}_A)$ of agent B to agent A [32].

The RVO takes into account the potential actions of both agents in a collision avoidance scenario, which makes it more sophisticated than the original VO concept. The mathematical definition is expressed in Equation 4.7, where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the positions of the agents and $\mathrm{r}^2_{i,j}$ is the squared sum of their radii. The meaning of the two equations is the same that was described in the previous subsection.

$$\mathrm{RVO}^i_j(\mathbf{v}_j, \mathbf{v}_i) = \Big\{(\mathbf{v}_i, \mathbf{v}_j) | (\mathbf{v}_i - \mathbf{v}_j)(\mathbf{p}_i - \mathbf{p}_j) \geq 0, |\mathbf{v}_i - \mathbf{v}_j|^2 \leq \mathrm{r}^2_{i,j}\Big\} \tag{4.7}$$

Based on this concept, a multi-agent motion planner has been developed by Van Den Berg et al. [32] and tested in simulation. The main steps that compose the base algorithm are the following:

1. Sense positions and velocities of all the agents in the scene and compute the RVOs between each pair of agents in the environment.

2. Select as new velocity of each agent the vector pointing towards its goal and is farthest from the union of all the RVOs computed with the other agents. This is done by giving a score to each possible velocity and selecting as the optimal one the velocity with maximum score [46].

3. The RVOs computation is repeated after a predefined time step and the process continues until each goal position is reached.

Ideally, the multi-agent motion planner can be used also as a single-agent motion planner, considering only one agent in the scene as the moving robot and assuming that all the possible encountered pedestrians use a similar strategy of collision avoidance, being considered as the other agents in the problem formulation.

This algorithm is fully decentralized and can handle complex environments with many moving agents, using the local information sensed by itself. It ensures that each agent moves towards its goal while avoiding collisions. Moreover, it guarantees that motions are oscillations free. A further improvement has been introduced after some years by the same authors, adding a completely reformulated optimization computation that reached higher performance [9].

### 4.2.3 Optimization basics

Optimization is a technology that can be used to make decisions or predictions in various contexts. It is based on a mathematical model of the problem that needs to be analysed and then it is solved using suitable numerical algorithms. A model requires the definition of a quantitative objective criterion of goodness in order to make possible the decision by means of a mathematical computation: normally it is expressed in the form of a cost function, that must be minimized, or as a score function, that must be maximized. The criterion is completed with the definition of constraints, which represents physical limits on the decision actions. Solve the optimization means finding the best possible value of the quantitative objective criterion of goodness while satisfying all the problem constraints.

From a verbal description of the problem, some decision variables $x = (x_1, ..., x_i)$ are selected and a cost function is built as a function of these variables $f(x)$, while the constraints are expressed as equality or inequality expressions on the same variables. The standard form of an optimization problem is reported in Equation 4.8 [47].

$$\begin{aligned} &\min_x f(x) \\ &\text{subject to } g_i(x) \leq 0, i = 1, \ldots, m \\ &\qquad\qquad h_j(x) = 0, j = 1, \ldots, p \end{aligned} \tag{4.8}$$

where:

- $f(x)$ is the objective function to be minimized over the n-dimensional vector $x$ of decision variables $x_i$
- $g_i(x) \leq 0$ are m inequality constraints
- $h_j(x) = 0$ are p equality constraints.

If $m$ and $p$ are equal to zero, the problem is said unconstrained. In a constrained problem $m \geq 0$ and $p \geq 0$. The set of vectors $x$ that satisfies all the constraints is called feasible set. If this set is empty, the problem has not a finite solution and is said infeasible. The set of feasible points for which the objective function achieves the optimal value is called optimal set. If the set is composed by a single point, it is called minimizer and the value of the objective function correspondent is the optimal value.

Among all the possible problem reformulations, the most interesting ones are the convex ones [47]: these problems have a bowl-shaped graph that gives the particular property that any local minimum found is a global one.

### 4.2.4 ORCA problem formulation

A further improvement in the RVO algorithm was done by introducing an optimization reformulation of the collision avoidance problem as the core element to select the velocities of the agents. The name that is associated with this new approach is Optimal Reciprocal Collision Avoidance (ORCA). In 2011, the year of the first scientific publication, J. Van Der Berg et al. introduced a method that represents the first that can guarantee local collision-free motion for a large number of robots in a cluttered workspace [9].

#### Overall description of ORCA

Here for simplicity, the problem will be described following the original purpose of a multi-agent collision avoidance algorithm, and then there will be a detailed translation of the main concepts to a single-agent collision avoidance algorithm.

Let there be a set of $n$ disk-shaped agents sharing a 2D environment. Each agent $i$ is described by means of:

- *External state*: current position $\mathbf{p}_i$ (the center of its disk), current velocity $\mathbf{v}_i$ and radius $r_i$
- *Internal state*: maximum speed $v_i^{max}$, preferred velocity $\mathbf{v}_i^{pref}$ (velocity directed towards the robot's goal with a magnitude equal to the robot's preferred speed).

The only parameters that can be sensed by the other agents in the environment are the external ones, the others can be only assumed. Implicitly, all the agents assume that the others are using the same collision avoidance strategy.

The objective of each individual $i$ is to independently (and simultaneously) select a new velocity $v_i^{new}$ for itself such that all the participants are guaranteed to be collision-free for at least a preset amount of time $\tau$ (time horizon of the movement prevision), when they would maintain their selected new velocity. As a secondary objective, the robots should select their new velocity as close as possible to their preferred velocity.

### Geometrical interpretation

Considering two robots A and B, the velocity obstacle, detailed in the previous sections, for A induced by the presence of B considering a time window $\tau$ can be expressed as in Equation 4.9, denoting with $D(\mathbf{p}, r) = \{\mathbf{q} | \, \|\mathbf{q} - \mathbf{p}\| \leq r\}$ an open disc of radius r centered at $\mathbf{p}$. The geometrical representation can be found in Figure 4.3.

$$\mathrm{VO}^{\tau}_{A/B} = \{\mathbf{v} | \exists t \in [0, \tau] : t\mathbf{v} \in D(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B)\} \tag{4.9}$$

The VO is interpreted as a truncated cone with its apex at the origin of the velocity space $(v_x, v_y)$ and its legs tangent to the disc of radius $r_A + r_B$ and centered at $\mathbf{p}_B - \mathbf{p}_A$. The truncating arc is part of a disk of radius $(r_A + r_B)/\tau$ and centered at $(\mathbf{p}_B - \mathbf{p}_A)/\tau$. The collision between the agents will happen if $\mathbf{v}_A - \mathbf{v}_B \in \mathrm{VO}^{\tau}_{A/B}$, or equivalently $\mathbf{v}_B - \mathbf{v}_A \in \mathrm{VO}^{\tau}_{B/A}$.

Recalling that the VO are defined using the relative velocities and defining the Minkowski sum of two generic sets X and Y as is shown in Equation 4.10, it is possible to define the set of collision-avoiding velocities $\mathrm{CA}^{\tau}_{A/B}(V_B)$ that A can assume in order to avoid the collision with B (Equation 4.11), given that B assumes a velocity from a given set $V_B$ (see Figure 4.3).

$$X \oplus Y = \{\mathbf{x} + \mathbf{y} | \mathbf{x} \in X, \mathbf{y} \in Y\} \tag{4.10}$$

$$\mathrm{CA}^{\tau}_{A/B}(V_B) = \left\{\mathbf{v} | \mathbf{v} \notin \mathrm{VO}^{\tau}_{A/B} \oplus V_B\right\} \tag{4.11}$$

A pair of sets $V_A$ and $V_B$ are called *reciprocally collision-avoiding* if $V_A \subseteq \mathrm{CA}^{\tau}_{A/B}(V_B)$ and $V_B \subseteq \mathrm{CA}^{\tau}_{B/A}(V_A)$. If the strict equality holds, they are called

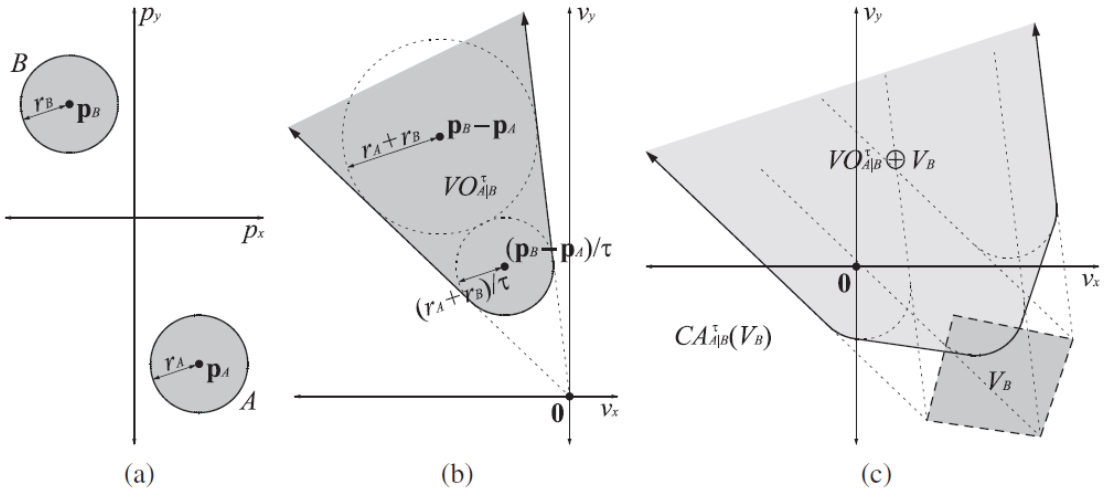


**Figure 4.3:** (a) A configuration of two agents. (b) The velocity obstacle $\mathrm{VO}^{\tau}_{A/B}$. (c) The set of collision-avoiding velocities $\mathrm{CA}^{\tau}_{A/B}(V_B)$ for agent A given that B selects its velocities from some set $V_B$ [9].

*reciprocally maximal*. Selecting as the only possible velocities sets for agents two reciprocally maximal sets, it is theoretically guaranteed the collision avoidance between them.

Among the infinite possible pair of reciprocally maximal sets, we are interested on those sets that maximizes the amount of permitted velocities near some optimization velocity $\mathbf{v}_i^{opt}$, for each agent. Such a pair of sets is defined formally in Equation 4.12, meaning that the two sets are reciprocally collision-avoiding and maximal, and Equation 4.13, meaning that they are the sets whose elements are nearest to $\mathbf{v}_A^{opt}$ and $\mathbf{v}_B^{opt}$.

$$\mathrm{CA}^{\tau}_{A/B}(\mathrm{ORCA}^{\tau}_{B/A}) = \mathrm{ORCA}^{\tau}_{A/B} \text{ and } \mathrm{CA}^{\tau}_{B/A}(\mathrm{ORCA}^{\tau}_{A/B}) = \mathrm{ORCA}^{\tau}_{B/A} \tag{4.12}$$

$$\begin{aligned}\left|\mathrm{ORCA}^{\tau}_{A/B} \cap D(\mathbf{v}_A^{opt}, r)\right| = \left|\mathrm{ORCA}^{\tau}_{B/A} \cap D(\mathbf{v}_B^{opt}, r)\right| \geq \\ \min(\left|V_A \cap D(\mathbf{v}_A^{opt}, r)\right|, \left|V_B \cap D(\mathbf{v}_B^{opt}, r)\right|)\end{aligned} \tag{4.13}$$

The two sets can be geometrically constructed as shown in Figure 4.4.

Assuming that the two agents A and B are travelling with their optimization velocities and a collision event will happen ($\mathbf{v}_A^{opt} - \mathbf{v}_B^{opt} \in \mathrm{VO}^{\tau}_{A/B}$), let $\mathbf{u}$ be the vector from $\mathbf{v}_A^{opt} - \mathbf{v}_B^{opt}$ to the closest point on the boundary of the velocity obstacle (Equation 4.14). Let $\mathbf{n}$ be the outward normal of the boundary of $\mathrm{VO}^{\tau}_{A/B}$ at the point $\mathbf{v}_A^{opt} - \mathbf{v}_B^{opt} + \mathbf{u}$.

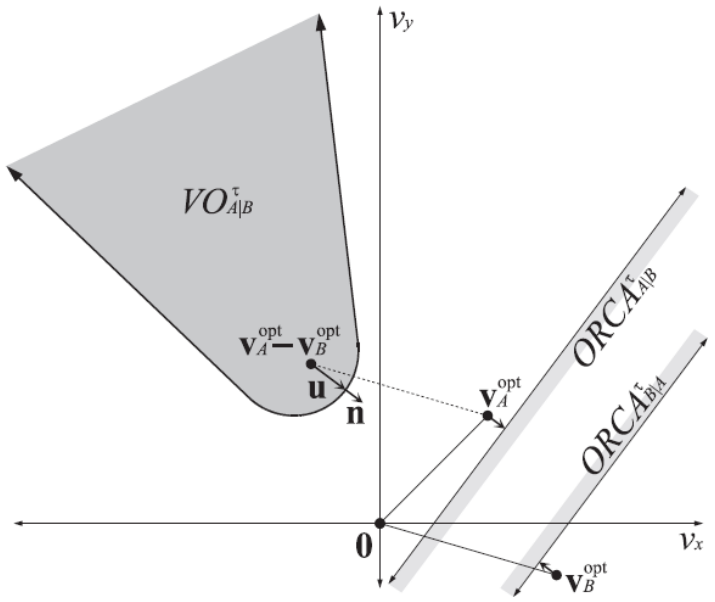


**Figure 4.4:** Geometrical construction of $\mathrm{ORCA}^{\tau}_{A/B}$ and $\mathrm{ORCA}^{\tau}_{B/A}$ [9].

$$\mathbf{u} = (\arg \min_{\mathbf{v} \in \partial \mathrm{VO}^{\tau}_{A/B}} \left\| \mathbf{v} - (\mathbf{v}_A^{opt} - \mathbf{v}_B^{opt}) \right\|) - (\mathbf{v}_A^{opt} - \mathbf{v}_B^{opt}) \tag{4.14}$$

The vector $\mathbf{u}$ can be interpreted as the smallest change required to the relative velocity to avoid the collision within the time horizon $\tau$. In order to share the responsibility of avoiding collisions among the agents is possible to insert a multiplicative parameter $\gamma$ in the set definition. Considering as reference Figure 4.4, the set $\mathrm{ORCA}^{\tau}_{A/B}$ is formally expressed as the half-plane pointing in the direction of $\mathbf{n}$ and passing through the point $\mathbf{v}_A^{opt} + \gamma \mathbf{u}$ (Equation 4.15). The set $\mathrm{ORCA}^{\tau}_{B/A}$ can be constructed on a symmetrical way.

$$\mathrm{ORCA}^{\tau}_{A/B} = \left\{ \mathbf{v} | (\mathbf{v} - (\mathbf{v}_A^{opt} + \gamma \mathbf{u}) \cdot \mathbf{n}) \geq 0 \right\} \tag{4.15}$$

In multi-agent motion planning $\gamma = 1/2$ in order to equally share the responsibility among all the agents [9]. In our case, we are developing a planner for a robot moving in an environment populated by humans. The value of $\gamma$ has been increased to 0.7-0.8 after some practical trials.

What is more, the optimization velocity presented in the previous equations is a tool that allow to generalize the calculations. Developing the planner, it is set equal to the actual velocity of the robot, in order to minimize the variation of the velocity required through two subsequent steps of calculations. It automatically adapts to the situations: it is near the preferred velocity in absence of obstacles and

near zero in high-density conditions. Further details are discussed in the following sections and can be found in [9].

### Optimization problem formulation

The collision avoidance for each agent is performed following four main steps, and it is repeated until the goal position is reached (Figure 4.5):

1. Sensing: the agent acquires the radius, the current position and the current velocity of all the other individuals present into the environment and of itself
2. $\text{ORCA}^{\tau}_{A/B}$ computation with respect to each of the other agents present into the shared environment
3. $\mathbf{v}_A^{new}$ selection, by means of solving an optimization problem constrained using the computed half-planes
4. Acting: apply the new velocity to the robot actuators updating the actual position.

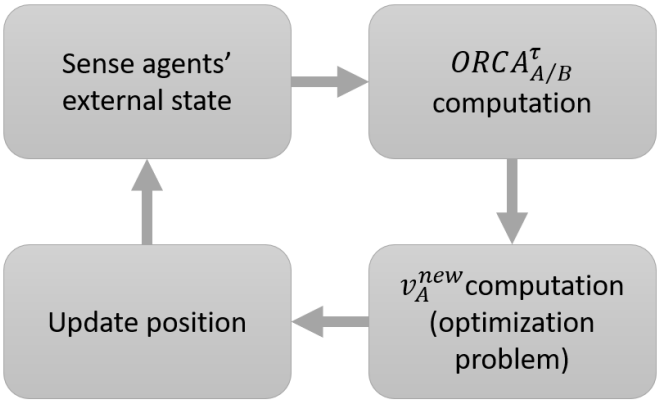


**Figure 4.5:** A schematic overview of the sensing-acting loop performed by the actor during a step [9].

After the sensing step, the acquired information are used to compute the half-panes of permitted velocities, recalling that each opponent determines a constraint on the velocity selection in the form of an half-plane in the velocity space, expressed as $\text{ORCA}^{\tau}_{A/B}$. Considering all the computed constraints and an additional one concerning the maximum attainable velocity by the robot, it is possible to define the set of the permitted velocity for A at the time of the computation $\text{ORCA}^{\tau}_{A}$ (Equation 4.16).

$$\mathrm{ORCA}_A^\tau = D(0, v_A^{max}) \cap \bigcap_{B \neq A} \mathrm{ORCA}_{A/B}^\tau \tag{4.16}$$

The agent A under consideration selects a new velocity $\mathbf{v}_A^{new}$ for itself that is closest to the preferred one amongst all the allowed velocities, by means of an optimization problem resolution, as it can be seen in Equation 4.17.

$$\mathbf{v}_A^{new} = \arg \min_{\mathbf{v} \in \mathrm{ORCA}_A^\tau} \left\| \mathbf{v} - \mathbf{v}_A^{pref} \right\| \tag{4.17}$$

Assuming that we are using a sufficiently small time step $\Delta t$, we can simplify the computations using the uniform rectilinear motion to update the robot position (Equation 4.18).

$$\mathbf{p}_A^{new} = \mathbf{p}_A + \mathbf{v}_A^{new} \Delta t \tag{4.18}$$

In our reformulation, the time step used is 0.1 s.

### Optimization problem solution

The fundamental step in the ORCA motion planner algorithm is the solution of the optimization problem for the selection of the new velocity for the considering agent. This is done by the authors of [9] reformulating the problem as a linear program (LP) and following the numerical algorithms detailed in [48].

In order to implement the algorithm into a real robot there is the need of a different reformulation, that can be solved using a programming language that is compatible with ROS (robot operating system), that is the middle-ware used by the robot to communicate between its components (further details can be found in the following chapters). Using as a reference the results presented in [9] [48] and the convex optimization concept revised in [47], it is found out that an efficient way of solve the problem is to reformulate the model as a quadratic program (QP) [49]. Even if it is a more complex reformulation than the presented one in the original article [9], the running time of the code developed is compatible to the time step used during the practical experiments, and no problems arises for what concerns computational speed or complexity.

The cost function of the ORCA optimization problem (Equation 4.17) is an euclidean distance between the velocity that we want to compute and the preferred velocity, that is recomputed at each iteration before entering into the optimization resolution. It is a nonlinear function, that can be rewritten in a quadratic one, as it is requested by a QP formulation. The fundamental passages are reported in Equation 4.19. The function is squared, since this operation will not change the argument of the optimization problem [47], and the norm is exploited. Note

that the last element of the equation $(\mathbf{v}_A^{pref})^T\mathbf{v}_A^{pref}$ will be eliminated in the next expressions, since it is a constant and does not affect the solution of the problem.

$$\begin{aligned}\left(\left\|\mathbf{v}-\mathbf{v}_A^{pref}\right\|_2\right)^2 &= \mathbf{v}^T\mathbf{v} - 2(\mathbf{v}_A^{pref})^T\mathbf{v} + (\mathbf{v}_A^{pref})^T\mathbf{v}_A^{pref} \\ &= \frac{1}{2}\left(\mathbf{v}^T\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}\mathbf{v} + \begin{bmatrix}-2v_{xA}^{pref}\\ -2v_{yA}^{pref}\end{bmatrix}\mathbf{v}\right) + (\mathbf{v}_A^{pref})^T\mathbf{v}_A^{pref}\end{aligned} \tag{4.19}$$

The preferred velocity $\mathbf{v}_A^{pref}$ can be expressed as a vector directed from the current position $\mathbf{p}_A$ to the goal position $\mathbf{p}_G$, with modulus equal to a percentage of the maximum attainable velocity, expressed by means of the parameter $\alpha$. The complete expression can be found in Equation 4.20, taking inspiration from [32] [46] [33].

$$\mathbf{v}_A^{pref} = \frac{\mathbf{p}_G - \mathbf{p}_A}{\|\mathbf{p}_G - \mathbf{p}_A\|_2}\mathbf{v}_A^{max}\alpha \tag{4.20}$$

The constraints of the ORCA optimization problem (Equation 4.17) define a feasible set that is the union of two convex sets (Equation 4.16). Therefore, as a consequence, the feasible set is convex for construction. The union of the half-planes can be directly transformed into a series of linear constraints [9] [48], one for each other agent or pedestrian present into the environment.

$$a_i^T\mathbf{v} \leq b_i i = 1, \ldots, n_{ped} \tag{4.21}$$

The constraint on the maximum attainable velocity is a quadratic constraint. In principle it transforms the problem into a Quadratic Constrained Quadratic Program (QCQP), but using simple geometrical considerations it is possible to approximate it with a certain amount of linear constraints, giving up to some accuracy in calculation for the sake of faster computation. As it can be seen in Figure 4.6, a circular set in two dimensions can be approximated using an $n_{edg}$ regular polygon, delimited in the space by means of $n_{edg}$ lines, or in terms of inequalities by means of $n_{edg}$ half-planes. Each half-plane is delimited by the line passing through the points expressed in Equation 4.22, whose explicit formulation is reported in Equation 4.23.

$$\begin{aligned}&(x_1, y_1) = (\rho\cos i\theta, \rho\sin i\theta) \quad (x_2, y_2) = (\rho\cos(i+1)\theta, \rho\sin(i+1)\theta) \\ &i = 0, \ldots, n_{edg}\end{aligned} \tag{4.22}$$

$$\begin{bmatrix}1\\ -\frac{y_1-y_2}{x_1-x_2}\end{bmatrix}\mathbf{v} = \frac{x_1y_2 - x_2y_1}{x_1 - x_2} \rightarrow a_c^T\mathbf{v} = b_c \tag{4.23}$$

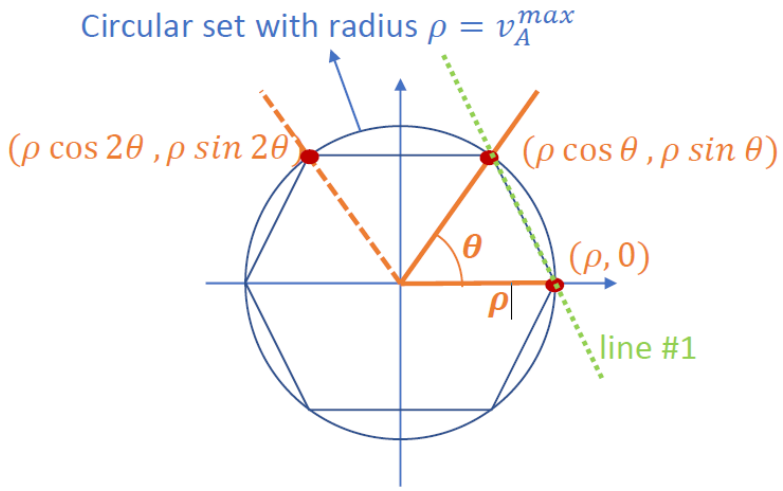


**Figure 4.6:** A scheme explicating the approximation of a circular set by means of $n_{edg}$ half-planes.

Considering all the previously explained reformulations, the optimization problem is finally rewritten as a QP, directly tractable with a solver library for a selected programmable language (Equation 4.24).

$$\begin{aligned} \mathbf{v}_A^{new} = \arg\min_{v} \frac{1}{2} \left( \mathbf{v}^T \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \mathbf{v} + \begin{bmatrix} -2v_{xA}^{pref} \\ -2v_{yA}^{pref} \end{bmatrix} \mathbf{v} \right) \\ \text{subject to: } a_i \mathbf{v} \leq b_i, i = 1, \ldots, n_{ped} \\ (a_c)_j \mathbf{v} = (b_c)_j, j = 1, \ldots, n_{edg} \end{aligned} \tag{4.24}$$

### 4.2.5 Implemented ORCA algorithm

The aim of the development is to use the algorithm as a planner in the simulation of a real robot that uses ROS as communication middleware. Thus, the algorithm must be written into a programming language that is completely compatible and that can exploit all its functionalities. The choice process ends with the decision of utilize C++, since it is completely compatible with ROS and can directly be integrated into the already developed navigation framework of ROS. Considering as a reference the public GitHub repository [50] and the original article [9], the algorithm has been developed integrating in it the ALGLIB library [51], which is a cross-platform numerical analysis and data processing library able to solve efficiently QP. Further details can be found on the official documentation [51].

Trying to be as close as possible to the overall description of the problem written in the first subsection, the algorithm has been summarized in the form of pseudo-code as can be seen in Algorithm 1.

**Algorithm 1** ORCA algorithm

$ProblemInitialization$
$\mathbf{M} \leftarrow MapInitialization$
$t \leftarrow 0$
**for** $i = 1 : n\_agents$ **do**
    $\mathbf{p}_i, \mathbf{v}_i, \mathbf{p}_i^{goal} \leftarrow AgentsInitialization$
    $\mathbf{traj}(t) \leftarrow \mathbf{p}_i$
**end for**
$end\_sim \leftarrow False$
**while** not($end\_sim$) and ($t \leq t\_sim$) **do**
    **for** $i = 1 : n\_agents$ **do**
        $\mathbf{n}_i^{goal} \leftarrow GoalDirectionComputation(\mathbf{p}_i, \mathbf{p}_i^{goal})$
        $\mathbf{v}_i^{pref} \leftarrow VelPrefComputation(\mathbf{n}_i^{goal}, v^{max})$
        $\mathbf{p}_i^{obs} \leftarrow ObsPosComputation(\mathbf{p}_i, M)$
    **end for**
    **for** $agent = 1 : n\_agents$ **do**
        **if** $SingleGoalReachingCheck(agent) == False$ **then**
            $\mathbf{v}_{agent}^{new} \leftarrow VelocityComputation(\mathbf{p}, \mathbf{v}, \mathbf{v}^{pref}, v^{max}, \mathbf{p}^{obs}, agent)$
            $\mathbf{v}_{agent} \leftarrow \mathbf{v}_{agent}^{new}$
            $\mathbf{p}_{agent} \leftarrow \mathbf{p}_{agent} + \mathbf{v}_{agent}\Delta t$
        **end if**
    **end for**
    $t = t + \Delta t$
    **for** $i = 1 : n\_agents$ **do**
        $\mathbf{traj}(t) \leftarrow \mathbf{p}_i$
    **end for**
    $end\_sim \leftarrow GoalReachingCheck$
**end while**

Observing the code, it is possible to notice a division into functions, whose purpose is detailed below:

- ***ProblemInitialization*** sets the parameters describing the problem and tuning the behaviour of the agents. The most important ones are reported in Table 4.1, along with a synthetic description.

| Parameter Name | Value | Description |
|---|---|---|
| $\Delta t$ | 0.1 s | Time step of the velocity computation. |
| $n\_edg$ | 20 | Number of edges that are used in the circular constraint approximation. |
| $\tau$ | 3 s | Time horizon considered into the constraints computation. |
| $\gamma$ | 0.8 | Responsibility parameter. |
| $n\_ped$ | 3 | Number of pedestrians in the simulated environment. |
| $n\_agents$ | 4 | Defined as the number of pedestrians plus one, representing the robot that we want to control. |
| $r_{robot}$ | 0.25 m | Radius of the disk-shape representation of the simulated robot. |
| $r_{ped}$ | 0.6 m | Radius of the disk-shape representation of the pedestrians, representing their personal space. |
| $r_{obs}$ | 0.3 m | Minimum allowable distance between the moving robot and static obstacles. |
| $v_{robot}^{max}$ | 0.5 | Maximum attainable velocity by the robot. |
| $v_{ped}^{max}$ | 0.8 | Maximum attainable velocity by the pedestrians. |
| $goal\_tolerance$ | 0.2 m | Tolerance distance considered in the goal-reaching check procedures. |
| $x_{dim}$ | 8.5 m | Dimension of the map along the $x$ axis of the fixed reference frame. |
| $y_{dim}$ | 5.5 m | Dimension of the map along the $y$ axis of the fixed reference frame. |
| $t\_sim$ | 40 s | Fixed upper-bound on the duration of each simulation. |

**Table 4.1:** Parameters used in the ORCA algorithm.

- ***MapInitialization*** generates a virtual representation of the $\mathbb{R}^2$ space in which the agents are supposed to move, that is translated into a matrix data structure named $\mathbf{M}$. This structure is composed by a discrete set of points, aimed to mimic a sensor acquisition data structure. The generated map is a rectangular room 8.5 m X 5.5 m, that has a square footage comparable to those used in experiments found in the literature [52].

- The time $t$ is set to zero and the ***AgentInitialization*** function is responsible for the agents' starting point $\mathbf{p}_i$ and goal point $\mathbf{p}_i^{goal}$ attribution. Therefore, the starting and goal points in numerical simulations are randomly selected. What is more, the initial positions are used to initialize the **traj** data structure, whose aim is to store in memory the generated trajectories.

- Then, a while loop ensures the correct simulation procedure. ***GoalDirectionComputation*** computes the unitary vector pointing from the current position $\mathbf{p}_i$ to the goal position $\mathbf{p}_i^{goal}$ for each agent, by means of the fraction present into the equation 4.20.

- ***VelPrefComputation*** exploits the preferred velocity $\mathbf{v}_i^{pref}$ for each agent. It also uses equation 4.20.

- ***ObsPosComputation*** computes the nearest static obstacle point as the point in the set $\mathbf{M}$ that has the closest euclidean distance with the current position $\mathbf{p}_i$. This is performed for all the agents.
- At this point all the information necessary for the velocity computation are stored in memory. A for loop ensures the computation for all the agents. Therefore, the function ***VelocityComputation*** is called for the individuals that has not reached their personal goal yet. A deeper explanation of the central function of the ORCA algorithm is later reported, along with its pseudo-code (Algorithm 2).
- Every time a new velocity for an agent is computed, its actual position is updated following equation 4.18, representing a rectilinear uniform constant velocity movement. Once all the actors in the scenario have updated their positions, the code exits from the for loop and a time update is set. The computed new positions are then inserted into the trajectory storing data structure **traj**, and finally the ***GoalReachingCheck*** function performs the goal-reaching check for all the simulated pedestrians, ending the simulation if all of them are close enough to their goal positions, considering a given *goal_tolerance* value.

**Algorithm 2** VelocityComputation function

1: **function** $VelocityComputation(\mathbf{p}, \mathbf{v}, \mathbf{v}^{pref}, v^{max}, \mathbf{p}^{obs}, agent)$
2: $\quad \mathbf{v}^{new} \leftarrow 0$
3: $\quad \mathbf{C}_{vel} = CircularSetApproximation(v^{max}, n\_edg)$
4: $\quad \mathbf{C}_{hyp} = ConstraintComputation(\mathbf{p}, \mathbf{v}, \mathbf{v}^{pref}, v^{max}, \mathbf{p}^{obs}, agent)$
5: $\quad \mathbf{v}^{new} \leftarrow AglibSolverCall(\mathbf{C}_{vel}, \mathbf{C}_{hyp}, \mathbf{v}^{pref}_{agent})$
6: $\quad \mathbf{v}^{new} \leftarrow VelocityCheck(\mathbf{v}^{new})$
7: $\quad$ **return** $\mathbf{v}^{new}$
8: **end function**

The main functionalities of the algorithm are inserted into a function named ***VelocityComputation*** (See Algorithm 2). Such function receives as inputs the current positions $\mathbf{p}$, the current velocities $\mathbf{v}$, the preliminarily computed preferred velocities $\mathbf{v}^{pref}$, the maximum velocity $v^{max}$, the nearest obstacle positions $\mathbf{p}^{obs}$, one with respect to each agent, and the discriminant index *agent*, used to point out the current considered individual for the next-step velocity computation. The only output of the function is the velocity that has to be applied in order to avoid collisions $\mathbf{v}^{new}$. A detailed outlook of the code is given below:

- The target velocity $\mathbf{v}_{new}$ is preliminarly set to zero (line 2). Subsequently, two distinct functions perform the optimization problem constrains' computation.

***CircularSetApproximation*** (line 3) performs the mathematical procedure represented in figure 4.6, by means of the equations 4.22 and 4.23, aimed to linearize the quadratic constraint on the maximum attainable velocity. What is more, the ***ConstraintComputation*** function (line 4) performs the feasible velocity sets $ORCA^{\tau}_{A/B}$ computation and translation into half-planes, as they have been described into equation 4.15.

- The constructed constraints, memorized into the matrix data structures $\mathbf{C}_{vel}$ and $\mathbf{C}_{hyp}$, are then fed into ***AlglibSolverCall*** (line 5), along with the target agent's preferred velocity. The latter cited application translates the constraints into a compatible ALGLIB library [51] format, as expressed in equation 4.24. Subsequently, the library optimization solver is called and the computed solution is stored in $\mathbf{v}^{new}$.

- The final step consists on a check procedure performed by the ***VelocityCheck*** function (line 6). If the solver finds out that the problem is infeasible, due to a very crowded navigation situation or the presence of too close pedestrians in the environment, the solution is set to $NaN$ by the solver. It results into code execution issues. In order to solve this problem, is such cases the velocity of the agent is set to zero. Finally, the solution is returned by the function (line 7).

This section explains the code used to run performance tests before integrating it into our simulated robot's navigation system. The code serving as a local planner for experimental simulated tests follows similar procedures. However, it directly gathers information about positions, speeds, and nearby obstacles from the robot's simulated sensors via the ROS network, without prior computation. More details about the general structure of the ROS navigation stack will be discussed in Chapter 5.

## 4.3 Game Theory Planner

The majority of the approaches reviewed in the literature, as discussed in Chapter 3, primarily focus on predicting human motion. However, these approaches often neglect the interaction among humans themselves, concentrating solely on the interaction between the unmanned agent and individuals within the shared environment. The incorporation of game theory, as outlined by Galati in his thesis [53], facilitates the modeling of human motion by considering interactions between our robot, pedestrians, and interactions among pedestrians. In contrast to prevalent reactive behaviors employed by many state-of-the-art motion planning methods, which prioritize obstacle avoidance without effective trajectory prediction, our work adopts a different paradigm. We are developing a model capable of addressing

these peculiarities, being then integrated into a predictive-based path planner. This approach seeks to overcome the challenges associated with reactive-based planners, as discussed in detail in Chapter 3.

The concept of studying human motion through a game-theoretic point of view, considering only two players and incorporating human-like obstacle avoidance, originated from [4] and [5]. This approach, regarded as one of the most compelling in the literature, underwent an attempt of reformulation in our department at Politecnico di Torino. In a prior Master Thesis by Galati [53], the model was extended to include a different cost function, accommodating multiple individuals and detecting groups. Recently, this approach has been evaluated in terms of social acceptability [42] through a modified Turing test [54] [55], administered as a survey questionnaire to 691 participants. The test aimed to discern between game-theoretical trajectories and human trajectories, with little distinctions emerging. In this thesis, we are taking a further step by implementing the game theoretical predictive approach into a motion planner, referred to as the Game Theory Planner (GTP). This implementation is applied to a simulated robot, enabling a direct evaluation of its performance through a practical simulation experiment, as detailed in Chapter 7. Importantly, the developed planner is designed for potential use in a real-world version of the robot, paving the way for future trials to assess social acceptability in real-world scenarios.

### 4.3.1 Game theory basics

Game theory is a branch of economics, mathematics, and social sciences that studies the strategic behavior of individuals or decision-making agents who interact with each other, being independent and self-interested [56]. This theory focuses on situations where the choices of one individual affect the outcomes obtained by other participants, and vice versa. In other words, game theory seeks to analyze the decisions and interactions among people when each of them seeks to maximize their objectives, taking into account the responses of the other actors.

One of the first scientific articles published about modern game theory is [57], where J. Von Neumann in 1928 proposed a mathematical solution for the zero-sum cooperative game. The work is has been improved in 1944 with [58], then J. Nash in 1951 published a solution for non-cooperatives games [59] that established one of the most famous solving approach, note as Nash equilibrium theory. This method, which will be more deeply explained later, is the decisional strategy adopted by the game theory planner (GTP) that is developed in this thesis.

In order to have a clearer comprehension of the problem description, it is necessary to point out the terminology adopted. In the next subsection, there will be a concise introduction to the words that will be used here.

### Description of a game

Game theory studies what happens when self-interested agents interact [56]. These agents in the GTP problem are both the pedestrians or the mobile robot moving following the implemented motion planner, that in a game description are called *players*. They are considered *rational* individuals, aimed to maximize their profit that can be modeled using an *utility function* (i.e. cost function) that maps characteristics of the environment, like the distance with the nearest obstacle, with a number that quantify a gain or a cost.

During the game, each player follows a *strategy*, that is a procedure in which the agent decides what to do at each possible situation of the game: it must specify an *action* for each possibility, taken one from the *action set*, that is a discrete set of possible outcomes. We can distinguish two main types of strategies (formal definitions can be found in [56]):

- **Pure strategy**: the action specification is deterministic. At each stage an action is selected and played as a response to the particular situation.
- **Mixed strategy**: it is specified a probability distribution for each action in the action set. At each stage an action is selected randomizing over the set of available actions according to the given distributions.

A *stage* is an event for which the player must take a decision. The *state of the game* includes all the information about all the players at a particular stage.

The games can be classified according to their main characteristics, like the number of players or actions and the way in which the information is shared among them. The most important discriminant characteristics are listed below [56].

- **Non-cooperative or cooperative**: in non-cooperative game theory the basic modeling unit is the individual, while in cooperative game theory the basic modeling unit is the group. This means that in non-cooperative games each agent put its own interest before the group interest, and a collaboration may happen, but with the goal of maximize its individual profit. On the other hand, in cooperative games the group's interest is always put over the individual one.
- **Zero-sum or non-zero-sum**: in zero-sum games the profit of a player is equal to the loss of the other participants. Meanwhile, in non-zero-sum games this may not happen and the sum of the payoffs can be different from 0.
- **Static or dynamic game**: in a static game the players make decisions simultaneously, without theoretically know the current move of the opponents, while in a dynamic game they act sequentially, having asymmetric information about the opponents' moves.

Other two important concept useful to describe our problem are:

- **Perfect information game**: each player during a decision knows all the previous step actions of all the opponents.
- **Finite game**: the number of players and the number of actions are both taken from a finite set.

#### Nash equilibrium

The aim of each player as a rational agent is to find an optimal strategy that maximize his payoff for a given environment and utility, or cost, function. The most used approaches are the *Pareto optimality criterion* [56] and the *Nash equilibrium* [56] [59]. Our navigation problem can be treated as a non-cooperative static non-zero-sum game, with perfect information and finite number of players and actions set. The most famous and used approach for solving a non-cooperative game is the Nash equilibrium concept. It is a combination of strategies where no agent can reduce its own cost by changing its action if the other agents stick to their actions [5]. In other words, it is a condition in which the considered agent cannot increase its profit alone, but only if the opponents will make a decision that changes the environment conditions.

In our context the Nash equilibrium is the discriminant situation that we consider in order to choose the action that the robot has to perform in the considered time step. Further details can be found in the following sections, where a complete description of the algorithm is done.

A formal definition of the concept can be found in [4]. The Nash equilibrium is a N-tuple (group of N elements) of strategies $\left(s_1^{j*}, s_2^{j*}, \ldots, s_N^{j*}\right) \in S_i$ such that the inequalities in 4.25 hold.

$$\begin{aligned} &J_1\left(s_1^{j*}, s_2^{j*}, \ldots, s_N^{j*}\right) \leq J_1\left(s_1^{j}, s_2^{j*}, \ldots, s_N^{j*}\right) \\ &J_1\left(s_1^{j*}, s_2^{j*}, \ldots, s_N^{j*}\right) \leq J_1\left(s_1^{j*}, s_2^{j}, \ldots, s_N^{j*}\right) \\ &\ldots \\ &J_1\left(s_1^{j*}, s_2^{j*}, \ldots, s_N^{j*}\right) \leq J_1\left(s_1^{j*}, s_2^{j*}, \ldots, s_N^{j}\right) \end{aligned} \tag{4.25}$$

Where the subscript $i$ refers to the players, the subscript $j$ refers to the considered stage and $J_i$ are the utility (or cost) functions correspondent to each player in the game.

### 4.3.2 GTP game formulation

The proposed model for pedestrian motion is a non-cooperative, static, finite, and general-sum game with many players [42]. What is more, the game is based on

perfect information, that means that each player know the last decisions made by the opponents. This is primary a multi-agent motion planner, but focusing on one agent as the real robot that we want to control and assuming that the pedestrians use a similar collision avoidance method to move in the environment, this model can be used to create a planner that can be uploaded into our simulated locobot [10]. For simplicity we does not perform group recognition, that has been considered instead in the original work developed by Galati et al. [42].

Each agent $i$, belonging to a finite agent set $\mathcal{N}$, performs his action $\theta(t)$ at each time $t$ choosing from a finite action set $\Theta$, that is equal for all the agents. The navigation is modeled as a movement into the $\mathbb{R}^2$ space, where $\mathbf{p}_i$ denotes the agent position, and the action $\theta(t)$ corresponds to a rectilinear movement at a constant velocity $v$ along the direction represented by $\theta(t)$, that is interpreted as an heading.

In order to find the best response that each agent should have at each stage we consider an approach similar to [4] and [5]. The best possible action that an agent can perform at each time $t$ is the action that allow him to be in a *Nash equilibrium* situation [59], where every possible action change will increase the correspondent cost function value if the opponents will maintain their selected action. In our situation the existence and uniqueness of a Nash equilibrium is not guaranteed [42], thus it is necessary to use numerical approaches for an approximation. Therefore, in order to compute the equilibrium in each stage the *Sequential Best Response Approach* (SBRA) presented in [60] is used, requiring the recursive resolution of an optimization problem.

In the next sections a deeper analysis of the optimization problem formulation and resolution is given, along with a practical description of the SBRA.

### Optimization problem formulation

All players seek for the Nash equilibrium applying the SBRA, solving their independent optimization problem considering the observed behavior of the opponents. Solve these optimization problems, whose basic considerations are presented in section 4.2, means to find for each agent $i \in \mathcal{N}$ the best sequence of actions $\boldsymbol{\theta}_i^* = [\theta_i(t), \theta_i(t+\Delta t), \theta_i(t+2\Delta t), \dots, \theta_i(t+T\Delta t)]$ over the specified finite prediction horizon $T\Delta t$. Considering $\Delta t = 1$ for a better readability and taking as reference [42], the optimization problem that need to be solved can be expressed as

$$\begin{aligned} \boldsymbol{\theta}_i^* &= \min_{\theta_i} J(\boldsymbol{\theta}_i) \\ \text{subject to } &\left\| \mathbf{p}_i(t, \theta_i(t)) - \mathbf{p}_j(t) \right\|_2 \geq \beta \quad \forall t, \forall i, j \in \mathcal{N}, i \neq j \\ &\mathbf{p}_i(t, \theta_i(t)) \notin \mathcal{O}_{obs} \quad \forall t, \forall i \in \mathcal{N} \end{aligned} \tag{4.26}$$

where:

- $\mathbf{p}_i(t, \theta_i(t)) = \mathbf{p}_i(t-1, \theta_i(t-1)) + \Delta\mathbf{p}(\theta_i(t), v)$ is the position of the agent at time $t$, defined as the position of the agent in the previous time step plus the movement done in one step as straight motion at a constant velocity $v$ in the selected optimal direction $\theta_i(t)$.
- $\mathbf{p}_j(t)$ is the position of the opponent $j$ at time $t$. An opponent is any other agent with respect to the one that is performing the computation. These positions are sensed by the agent, recalling the principle of perfect information game.
- $J(\boldsymbol{\theta})_i$ is the cost function correspondent to agent $i$, that is composed by three addends $J(\boldsymbol{\theta}_i) = \Phi_{goal}(\boldsymbol{\theta}_i) + \Phi_{smoothness}(\boldsymbol{\theta}_i) + \Phi_{obstacle}(\boldsymbol{\theta}_i)$, that have different meanings that are discussed later.

The optimization problem has two constraints. The first one is an hard constraint responsible for the collision avoidance, imposing a minimum bound on the distance between two distinct agents, in order to respect a circular region around them as their personal space, recalling the theory of *Proxemics* [1] [16]. The second one ensures that the agent position is coherent with the space that is free in the 2D environment in which it navigates.

The cost function $J(\boldsymbol{\theta})_i$ is composed by three terms. The first one is expressed in Equation 4.27 and it models the goal-oriented attitude of the player. The objective of this part of the cost function is to reduce the total path length.

$$\Phi_{goal}(\boldsymbol{\theta}_i) = \sum_{t=1}^{T} \gamma(t) \left\| \mathbf{p}_i(t, \theta_i(t)) - \mathbf{p}_i^* \right\| \tag{4.27}$$

The parameter $\gamma$ is a time-varying weighting factor, $\mathbf{p}_i(t, \theta_i(t))$ is the position of the agent at time $t$ that is computed considering the selected optimal direction and $\mathbf{p}_i^*$ is the agent's goal position. The goal of the pedestrian, when it is not explicated, is assumed as the point that he would reach if his velocity and his direction remains constant in the considered time horizon $T$.

The second addend is expressed in 4.28 and it is responsible for the generation of a smoother trajectory, penalizing excessive rotations during the motion. It is a behaviour followed by humans in order to minimize their energy consumption [61].

$$\Phi_{smoothness}(\boldsymbol{\theta}_i) = \sum_{t=1}^{T} (1 - \gamma(t)) \left| \theta_i(t) - \theta_i(t-1) \right| \tag{4.28}$$

The terms $\theta_i(t)$ and $\theta_i(t-1)$ represent the orientations of the agent at time $t$ and $t-1$. What is more, also in this term a weighting factor has been inserted, whose value is complementary to the one previously observed in equation 4.27. This is the results of the assumption that attraction of the goal and smoothness of

the generated trajectory are two navigation peculiarities that assumes different importance during the completion of the task: approaching the goal point, the attraction became more important than the smoothness of the trajectory, and vice-versa.

The third element is expressed in Equation 4.29 and it is responsible for the behaviour of the agent facing a static obstacle. Like a human, it is imposed that the player must not travel too close to obstacles, unless it is necessary. It is done by using a soft constraint that penalizes short distances between the agent and the nearest obstacle, which is expressed as a point in the space.

$$\Phi_{obstacle}(\boldsymbol{\theta}_i) = \sum_{t=1}^{T} \frac{\rho}{\|\mathbf{p}_i(t, \theta_i(t)) - \mathbf{p}_{obs}\|} \tag{4.29}$$

The parameter $\rho$ is a weighting factor representing the minimum allowed distance and $\mathbf{p}_{obs}$ is the position of the closest static obstacle at time $t$.

### Optimization problem solution

As previously explained, the navigation of each agent is performed by means of the Nash equilibrium computation in each time instant in order to find the optimal direction to be followed. The solution is reached using the SBRA [60]. This is formulated as the solution of many optimization problems as the one expressed in the previous subsection. Consequentially, it is important to find a solution for the optimization in the more efficient way and in the faster way possible.

The first step in whatever optimization problem solution is the problem classification. In our case the problem expressed in equation 4.26 is overall a nonlinear and non convex one. This means that we cannot use efficient convex optimization algorithms as it has been done in the ORCA algorithm's implementation. In principle, the solution can be found considering nonlinear and non-convex solvers, but the time required for each problem solution would bee too large to be compliant with the time step required into a real time simulation of the navigation scenario. After some initial reformulation trials, the solution considered follows the approach presented in [53] and tested in [42], that represent a discrete approximation of the exact continuous solution.

As previously stated, the agent picks up its optimal action, here on denoted as $u_i(t)$, from a finite set $\Theta$. The solution of the proposed problem is a sequence of headings $\boldsymbol{\theta}_i^*$, whose i-th element is the heading of the agent at time $t$ and its update follows the equation 4.30.

$$\theta_i(t) = \theta_i(t-1) + u_i(t-1) \tag{4.30}$$

The values populating the action set has been chosen considering a limited field of view of 180° comprehensively, recalling the principle of the *Information Process*

*Space* [25]. This angular sector is then further subdivided in four parts, obtaining an action set equal to $\Theta = \{-\pi/2, -\pi/4, 0, \pi/4, \pi/2\}$.

A generic sequence of headings $\boldsymbol{\theta}_i$, addressed to be the optimal one during the optimization problem solution, generates a trajectory in the $\mathbb{R}^2$ space describing the environment. Starting from the actual position of the agent $\mathbf{p}_i$, $n\_ac^{step\_hor}$ trajectories are possible considering the number of possible actions $n\_ac$ and the number of time steps aiming to be predicted $step\_hor = \frac{T}{\Delta t}$. The best sequence $\boldsymbol{\theta}_i^*$ is chosen among this possible ones, computing the cost of the feasible trajectories by utilizing the utility function $J(\boldsymbol{\theta}_i)$ explained in equation 4.26.

Although measures have been taken into consideration while writing the code to reduce computational complexity, such as dynamically eliminating infeasible trajectories during the cost computation phase, the time required for the discrete resolution of the optimization problem is, however, incompatible with the time step required for real-time simulation or implementation. Therefore, preliminary simulations concerning the predictive model have been performed using Matlab and the results has been summarized in Chapter 6. Subsequently, these simulations have been utilized to generate a training dataset for a fully-connected NN using the Tensorflow application [62]. In this way ML is employed to speed up the decision-making process within the algorithm, making possible its subsequent integration into the real-time robot simulation model. However, a more accurate explanation is given in Chapter 7, where the test procedure and framework are introduced.

### Nash equilibrium computation

After discussing the solution to the optimization problem, this procedure is employed to calculate the Nash equilibrium, determining the action that the robot should take in the step following the computation. As previously asserted, the Nash computation is performed using the SBRA [60], whose practical procedure is now explained.

In summary, each agent solve its optimization problem in order to find its own best strategy, given the last actions chosen by the opponents. Then a control action is performed, checking if the action selected for each player is equal to the one computed at the previous iteration. If the equality holds, we have found a Nash equilibrium, otherwise the computed action is chosen for the considering agent and the optimization problem is solved by another agent. This is repeated for all the agents until the equilibrium situation is met, and the actions correspondent to the equilibrium are selected as the best response in this particular stage. Then, a time step action is imposed and the procedure restarts considering the new positions reached by the agents. This is repeated until the goal position for all the agents is reached.

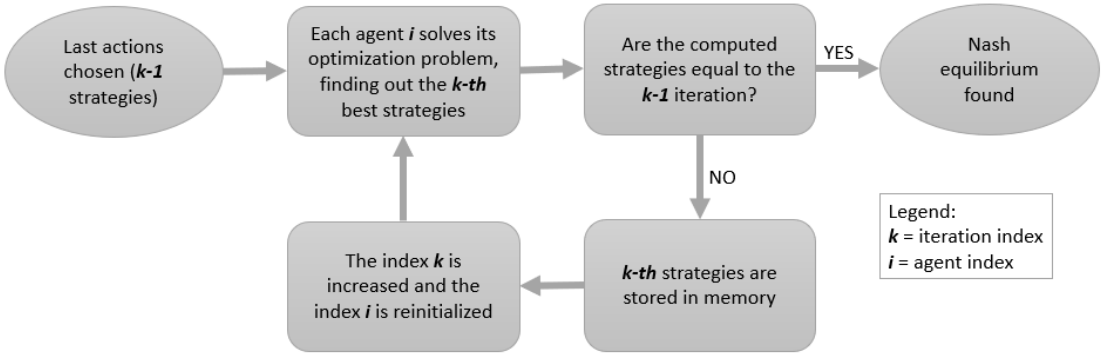


**Figure 4.7:** Simple scheme explaining the Sequential Best Response Approach applied in our context.

#### Motion generation

Once the Nash equilibrium is numerically computed, the best sequence of actions $\boldsymbol{\theta}_i^*$ is used to perform the motion. In particular, the direction chosen for the one-step movement is the first predicted entry of the vector. Therefore, the position update of each player $i$ follows the Equation 4.31, that represent a rectilinear movement at constant velocity $v$.

$$\mathbf{p}_i(t+1, \theta_i(t+1)) = \mathbf{p}_i(t, \theta_i(t)) + \mathbf{v}_i(t)\Delta t \tag{4.31}$$

In the equation above $\mathbf{v}_i(t)$ express the velocity vector that has constant module $v$ during the step time $\Delta t$ and direction expressed by the chosen action $\theta_i(t)$ by means of the Nash equilibrium just solved.

### 4.3.3 Implemented GTP algorithm

Recalling the verbal description presented in the previous subsections, the complete algorithm used in trajectories generation, developed in Matlab, is reported in the form of pseudo-code in Algorithm 3.

A complete overview of the code is given below, following its division in functions:

- The ***ProblemInitialization*** module (line 1) is responsible for configuring the parameters that define the problem and fine-tune the behavior of the agents. The critical parameters are summarized in Table 4.2, accompanied by a concise description.
- ***MapInitialization*** (line 2) is responsible for creating a virtual representation of the $\mathbb{R}^2$ space in which the agents operate. This representation is translated into a matrix data structure named $\mathbf{M}$. The structure consists of a discrete

**Algorithm 3** GTP algorithm

$ProblemInitialization$
$\mathbf{M} \leftarrow MapInitialization$
$\mathbf{U} \leftarrow ComputePossibleStrategies$
$t \leftarrow 0$
**for** $i = 1 : n\_pl$ **do**
    $\mathbf{p}_i, \theta_i, \mathbf{p}_i^{goal} \leftarrow PlayersInitialization$
    $\boldsymbol{\tau}(t) \leftarrow \mathbf{p}_i, \theta_i$
**end for**
$end\_sim \leftarrow False$
**while** not($end\_sim$) and ($t \leq t\_sim$) **do**
    $nash\_iter \leftarrow 0$
    $\mathbf{P}, \Theta \leftarrow FirstEstimation(\mathbf{p}, \theta)$
    $nash\_iter \leftarrow NashEquilibriumComputation(\mathbf{P}, \Theta, \mathbf{M}, \mathbf{U})$
    $t = t + \Delta t$
    $\boldsymbol{\tau}(t) \leftarrow nash\_iter[1]$
    $end\_sim \leftarrow GoalReachingCheck$
    **for** $i = 1 : n\_pl$ **do**
        $\mathbf{p}_i, \theta_i \leftarrow \boldsymbol{\tau}(t)$
    **end for**
**end while**

| Parameter Name | Value | Description |
|---|---|---|
| $\Delta t$ | 0.3 s | Time step of the position update. |
| $T$ | 1.8 s | Time horizon of the predictions. |
| $step\_hor$ | 6 | Number of predicted time instants. |
| $max\_iter$ | 30 | Maximum allowable iterations in the Nash equilibrium computation. |
| $n\_ped$ | 3 | Number of pedestrians in the simulated environment. |
| $n\_players$ | 4 | Defined as the number of pedestrians plus one, representing the robot that we want to control. |
| $r_{robot}$ | 0.25 m | Radius of the disk-shape representation of the simulated robot. |
| $r_{ped}$ | 0.6 m | Radius of the disk-shape representation of the pedestrians, representing their personal space. |
| $r_{obs}$ | 0.3 m | Minimum allowable distance between the moving robot and static obstacles. |
| $v_{robot}^{max}$ | 0.5 | Maximum velocity attainable by the simulated robot. Upper-bound used only in Gazebo simulations. |
| $v^{max}$ | 0.8 | Maximum attainable velocity by the simulated players in Matlab. |
| $n\_ac$ | 5 | Number of actions among which the agents can choose at each stage, in their FOV. |
| $fov$ | $\pi$ | Agents' FOV. |
| $\beta$ | 0.7 | Minimum allowable distance between pedestrians. |
| $\gamma$ | $0.6 - 0.9$ | Weighting factor between goal-reaching and smoothness part of the cost function. The value is increased while approaching the goal. |
| $\rho$ | 0.6 | Weighting factor of the distance from obstacle part of the cost function. |
| $goal\_tolerance$ | 0.1 m | Tolerance distance considered in the goal-reaching check procedures. |
| $x_{dim}$ | 8.5 m | Dimension of the map along the $x$ axis of the fixed reference frame. |
| $y_{dim}$ | 0.1 m | Dimension of the map along the $y$ axis of the fixed reference frame. |
| $t\_sim$ | 40 s | Fixed upper-bound on the duration of eaach simulation. |

**Table 4.2:** Parameters used in the GTP algorithm.

set of points designed to simulate a sensor acquisition data structure. The generated map depicts a rectangular room measuring 8.5 m x 5.5 m, offering a square footage comparable to those used in experiments documented in the literature [52].

- In the problem settings, there exists a constant number of potential choices at each stage denoted as $n\_ac$, and a fixed number of predicted steps denoted as $step\_hor$. Consequently, at each decisional stage, we have the same $n\_ac^{step\_hor}$ possible strategies. These strategies are generated by the application of ***ComputePossibleStrategies*** and stored in a data structure named **U** (line 3). Later in the code, when a decision needs to be made, the possible trajectories are derived by considering these pre-computed strategies, thereby optimizing computational efficiency.

- At the outset, the time $t$ is set to zero (line 4). Subsequently, within a for loop, the ***PlayersInitialization*** function is responsible for assigning the agents' starting points $\mathbf{p}_i$ and goal points $\mathbf{p}_i^{goal}$ (lines 6). In numerical simulations, the initial selection of starting and goal points is random. However, these points undergo subsequent modifications to more accurately depict navigation behavior. This adjustment is crucial as the generated data serves as a training dataset for a neural network intended to replicate navigation functionalities.

Additionally, the initial positions are used to initialize the $\boldsymbol{\tau}$ data structure (line 7), designed to store the generated trajectories in memory. These trajectories encompass both position $\mathbf{p}_i$ and orientation $\theta_i$.

- Subsequently, a while loop governs the progression of the simulation, regulated by the flag variable *end_sim*, initially set to $False$ (line 9). Additionally, a time limit *t_sim* is established to halt the simulation in the event of issues during parameter tuning. The *nash_iter* matrix data structure is initialized with all zero values (line 11). This structure is designed to store the positions and orientations of the agents during the Nash equilibrium computation, executed through the Sequential Best Response Approach, as depicted in section 4.3.2. In particular, this structure stores both the current and predicted time instants.

- As an initial step, preceding the actual equilibrium computation, the anticipated future movements of the agents within the environment are approximated using ***FirstEstimation***. This involves projecting rectilinear movements in the current direction that agents have over the time horizon (line 12). The function takes as input the positions and orientations of all agents stored in the structures $\mathbf{p}$ and $\theta$, respectively, and records the estimations into $\mathbf{P}$ and $\boldsymbol{\Theta}$, containers for position and orientation trajectories of all agents over the time horizon. These data structures are implemented in a dense format for ease of translation into a *.csv* format dataset.

- The core functionality of this code is encapsulated in the function ***NashEquilibriumComputation***, primarily designed to efficiently compute an approximation of the Nash equilibrium using the SBRA (line 13). The key inputs for this function include the initial trajectory estimations stored in $\mathbf{P}$ and $\boldsymbol{\Theta}$, the map data generated in $\mathbf{M}$, and the pre-computed possible strategies $\mathbf{U}$. The output, which contains all trajectories (strategies) of positions and orientations of the agents at Nash equilibrium, is returned in the matrix *nash_iter*. A comprehensive overview of the function is presented in Algorithm 4, followed by a verbal description.

- The simulation time is updated (line 14), followed by the upgrade of trajectories stored in $\boldsymbol{\tau}$. This procedure takes into account the first estimated step stored in the *nash_iter* structure (line 15).

- Finally, the ***GoalReachingCheck*** function executes a goal-reaching check for each individual, considering a specified *goal_tolerance* (line 16). If all players have reached their goal positions, the simulation is halted by changing the *end_sim* flag to $True$. If not, the final for loop reinitializes the initial

positions $\mathbf{p}_i$ and orientations $\theta_i$ (line 17). The simulation procedure will then restart and continue iterating until all agents have reached their goal positions.

**Algorithm 4** NashEquilibriumComputation function

```
1: function NashEquilibriumComputation(P, Θ, M, U)
2:     nash_iter ← P, Θ
3:     nash_prec_iter ← 0
4:     nash_reached ← False
5:     iter = 1
6:     while (iter ≤ max_iter) and not(end_sim) and not(nash_reached) do
7:         for agent = 1 : n_pl do
8:             if SingleGoalReachingCheck(agent) == True then
9:                 nash_iter ← FreezePosition(nash_iter)
10:            else
11:                nash_iter ← EvaluatePossibleTraj(nash_iter, M, U)
12:            end if
13:        end for
14:        if NashReachingCheck(nash_prec_iter, nash_prec) == True then
15:            nash_reached ← True
16:        else
17:            iter = iter + 1
18:            nash_prec_iter ← nash_iter
19:        end if
20:    end while
21:    return nash_iter
22: end function
```

In Algorithm 4, you can find the pseudocode for the function associated with Nash equilibrium computation. A detailed description of the main steps is provided below:

- Initially, the first-try estimations of future positions and orientations, stored in $\mathbf{P}$ and $\mathbf{\Theta}$, are employed to initialize the previously described *nash_iter* structure (line 2). A similar structure, named *nash_prec_iter*, is created and set to all zeros to store predictions from the previous iteration (line 3). The index *iter* is initially set to one (line 4).

- The while loop, which implements the SBRA, is initiated with a three-way condition: the flag *nash_reached* needs to be kept as $False$, the global boolean *end_sim* must remain $False$, and an upper limit on possible iterations is set to *max_iter* to restrict the number of interactions (line 6).

- An inner for loop instantiates the *agent* index, used for individual goal-reaching checks performed by ***SingleGoalReachingCheck*** (line 8). If the goal is reached by the target individual, the ***FreezePosition*** function is called, setting the predicted position equal to the actual one. This results in a trajectory composed of identical points saved in *nash_iter* (line 9). If the goal is not reached, the ***EvaluatePossibleTraj*** function is called (line 11). It takes the map **M**, the *nash_iter* structure, and the possible strategies **U** as inputs. Initially, possible trajectories are computed by extracting actual positions from *nash_iter* and considering the strategies **U**. Each feasible trajectory, meaning those without collisions with predicted movements of other agents or static obstacles in the map **M**, is then evaluated by computing the associated cost based on the optimization function presented in Equation 4.26. Finally, the trajectory with the minimum cost is saved in *nash_iter* for each agent (line 11).

- Subsequently, the control action, aiming to check the Nash equilibrium reaching, is performed by the ***NashReachingCheck*** function (line 14). It precisely compares the *nash_iter* structure with its counterpart, *nash_prec_iter*, where the computed trajectories of the last iteration are stored. If equality holds, the simulation is stopped by changing the boolean *end_sim* to $True$ (line 15). If not, the index *iter* is incremented (line 17), and the currently computed trajectories are inserted into *nash_prec_iter* (line 18). Finally, the *nash_iter* structure is returned as the sole output of the overall function (line 21).

Simulations executed using this algorithm, along with corresponding comments on the overall performance, are documented in Chapter 6. As previously mentioned, this code is employed to generate trajectories for constructing a dataset referred to as ***datasetGTP***. The purpose of this dataset is for training a fully-connected neural network aimed to replace the trajectory planning of the ROS navigation stack, as detailed in Chapter 5.

# Chapter 5

# Software tools used and hardware description

Taking into consideration the objectives expressed in the introductory chapter of this thesis, it was necessary to carefully choose the simulation and implementation tools to be used during the project. As previously anticipated, ROS [63] was used as a nerve center for communication between the various simulation and implementation components. In addition to this, Gazebo [6] was chosen as a virtual testing environment to be able to evaluate the performance of the algorithms and to carry out preliminary debugging. As regards the actual implementation of the algorithms, the basic navigation stack of ROS was used. The default local planner plugin, which uses an algorithm attributable to the Dynamic Window Approach (DWA) [28], is replaced with the planners implemented using the C++ programming language. Furthermore, some preliminary tests and some phases of the development of the algorithms were possible thanks to the use of the Matlab application.

In the sections below the main tools used will be analyzed with a brief introduction to the concepts necessary to understand the simulation and implementation methods followed. In the end, a brief overview of Tensorflow is given since it is used to overcome the time-consuming issue that came out on GTP decision-making process.

## 5.1 ROS

ROS (Robot Operating System) [63] is an open-source software middleware developed by Open Robotics that controls the communications between different functionalities of a robotic project, providing libraries and tools to help software developers. For instance, it provides hardware abstraction for simulations, low-level

device drivers in order to act on an actuator or receive data from a sensor, software libraries to develop personalized applications, message transfer management and many more features. The ROS framework is based on a peer-to-peer network of processes, that can be distributed among different machines, which are linked using the ROS communication infrastructure. In the following subsections, the main components of ROS are analyzed, focusing on the most important functionalities that have been used in the development of this project, taking as reference the official documentation [64].

### 5.1.1 Basic concepts

The ROS communication infrastructure, organized in the file-system level into software packages, relies on a few essential concepts, summarized as follows:

- **Nodes**: These are processes that perform computations. Each node encapsulates a single functionality, and a typical control system is composed of multiple nodes. A ROS node is essentially code programmed in common languages like C++ or Python using the corresponding ROS client library. Nodes can be grouped into *packages*, and packages can, in turn, be organized into *stacks*.

- **ROS master**: This is a crucial process that provides name registration and facilitates communication among different nodes. It includes a parameter server where fixed data can be stored by key and retrieved within the framework.

- **Messages**: These are data packages exchanged between nodes. Messages are essentially data structures specified in a *.msg* text file within the package where they are used.

- **Topics**: These are names used to identify the content of a transferred message, analogous to specific types of buses. Communication between nodes is based on a publish-subscribe mechanism, where a node sends a message by publishing it to a certain topic, and another node receives the message by subscribing to the same topic.

- **Services**: These involve pairs of message structures defining a slightly different communication paradigm based on request-response interactions. A node, called the server, provides a service response message when a node, called the client, sends a request message.

- **Bags**: This file format enables the storage of a data stream in the network, encompassing all moving messages or only specific ones. This storage method is crucial for testing and evaluation purposes.

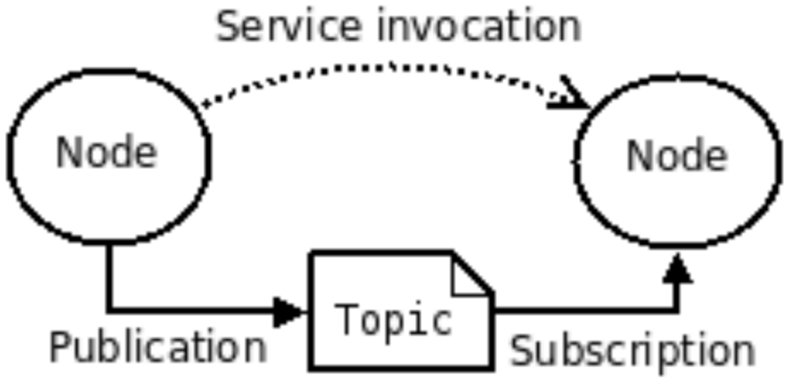


**Figure 5.1:** ROS communication paradigm [63].

A summary scheme of the communication paradigm is represented in Figure 5.1.

Throughout the project, the ROS1 version, specifically the ROS Noetic distribution, was utilized, running on Linux Ubuntu 20.04. This choice was driven by the abundance of online resources and documentation regarding modifications to the navigation framework for ROS1 compared to ROS2. Furthermore, the decision was influenced by the fact that the robot employed for simulation testing is equipped with a set of drivers and ROS wrappers specifically designed for compatibility with ROS1.

### 5.1.2 Navigation stack

The navigation pipeline, detailed in Chapter 2, is implemented in the ROS framework through several functional nodes, which are organized for clarity into a stack named the *Navigation Stack* [65]. The Navigation Stack exhibits notable conceptual simplicity. It receives data from odometry and sensor streams, then generates velocity commands for transmission to a mobile base. While utilizing the stack in a generic robot can be challenging without proper low-level software integration, Trossen Robotics provides its Locobot-WX250s [10] with pre-installed low-level wrappers [66] that alleviate this issue. The following subsection provide a brief and high-level overview of the main component of the stack.

#### ***move_base*** **package**

The *move_base* package [67] is comprised of multiple interconnected nodes, as depicted in Figure 5.2, where the utilized topics and messages are specified. Essentially, it implements the goal-reaching functionality of a mobile robot within the ROS framework. The goal pose, consisting of the desired position in space and orientation, is provided to the *move_base* node. This node interacts with sensors

to determine its actual position and, employing a costmap representation of the environment, computes a feasible path using a global planner. Subsequently, it generates the necessary movement commands using a local planner.

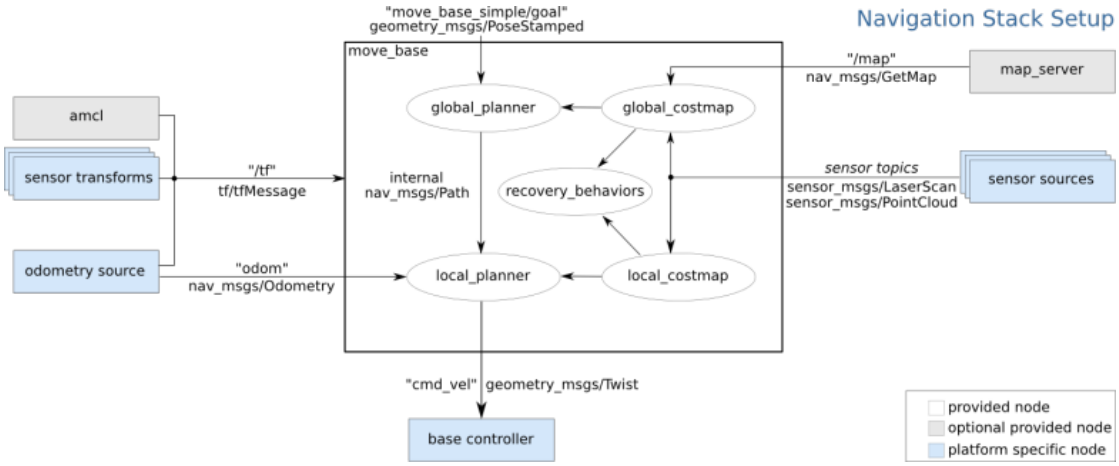


**Figure 5.2:** *move_base* package schematic overview [67].

Both the global and local planners can be modified by creating a ROS plugin to be imported into the *move_base* node. ROS plugins are dynamically loadable classes loaded from a runtime library [68]. In simpler terms, these are pieces of code that can be loaded into a node by specifying a parameter in the relevant configuration file within the *move_base* package. Each algorithm presented in Chapter 4 is translated into a plugin following the procedure outlined in the official ROS tutorial [69].

### 5.1.3 Rviz

*RViz* [70], short for ROS Visualization, is a robust 3D visualization tool that enables users to visualize and interact with various types of data from a robot's sensors and state. It offers a graphical interface for displaying sensor data, robot models, and other information in a three-dimensional space. Additionally, RViz can communicate with a real robot in practical experiments and with the virtual implementation built using Gazebo, whose functional description can be found in the next section. The tool's layout is fully customizable, allowing users to enable or disable several possible display channels.

Throughout the project, this tool is utilized to:

- Visualize the robot into the self-generated environment map, with the possibility to enable the vision of the costmap used by the global planner.

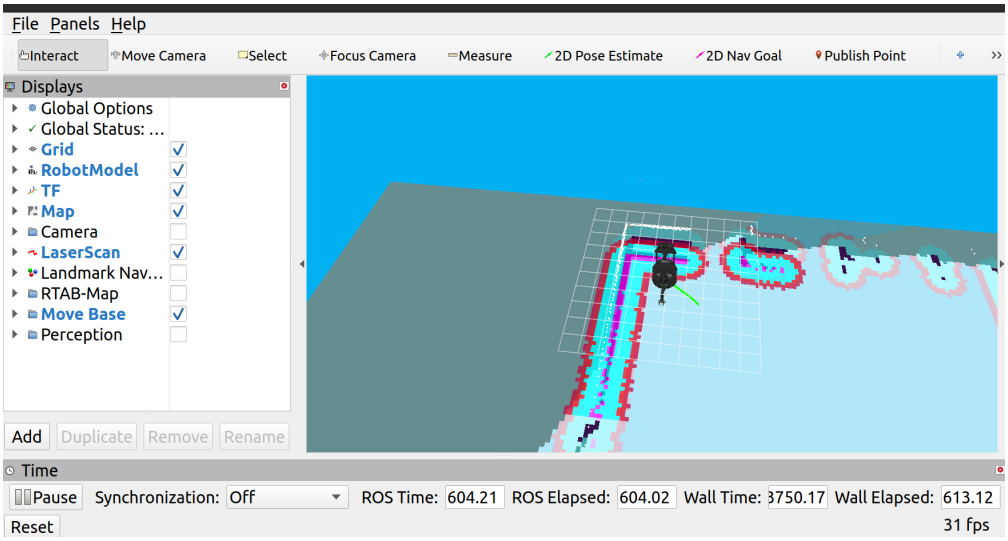


**Figure 5.3:** RViz interface.

- Visualize the frame of reference tree, a crucial element for defining the data transformations required in each algorithm implementation, especially as a local planner ROS plugin for the *move_base* package.

- Conduct point-goal tests using built-in interactive interfaces.

## 5.2 Gazebo

Simulation plays a crucial role in robotics research. Instead of dedicating engineering effort directly to creating application prototypes, modern computational capabilities allow researchers to conduct simulations in environments that replicate the operating conditions where the device being developed will ultimately function. Among the numerous tools available in the literature [71], one of the most widespread and commonly used in recent years is Gazebo [6] [72], an open-source three-dimensional simulator that enables the recreation of dynamic environments, considering the physical properties of individual elements within it. In Gazebo, the robot base, along with all the sensors and actuators that constitute its body, is loaded into this virtual world, allowing interaction with objects and moving entities.

The standout feature of Gazebo is its seamless integration with ROS. All nodes, topics, and message streams function in the same way as with a real robot. This characteristic provides an ideal environment for preliminary testing and, importantly, enables data generation for numerical validation.

### 5.2.1 SFM Gazebo plugin

To replicate real operational conditions as closely as possible, it is essential to adopt realistic human pedestrian modeling in simulations. Gazebo provides the option to insert dynamic objects in the environment and also includes human representations referred to as "Actors." These Actors can be configured to move from a starting point to a goal point, passing through a sequence of waypoints. While this solution serves as an initial debugging framework, the generated trajectories do not inherently consider the presence of the robot or obstacles in the environment. To overcome this issue and introduce a reactive behaviour to the model of pedestrian used in simulation, we have adopted the Gazebo SFM plugin [73]. The plugin is based on the original formulation presented in [8]. The trajectories are generated following the algorithm briefly explained in Chapter 4. What is more, the parameters that regulate the SFM are different for each pedestrian, aiming to emulate the variability of human behaviours in real scenarios.

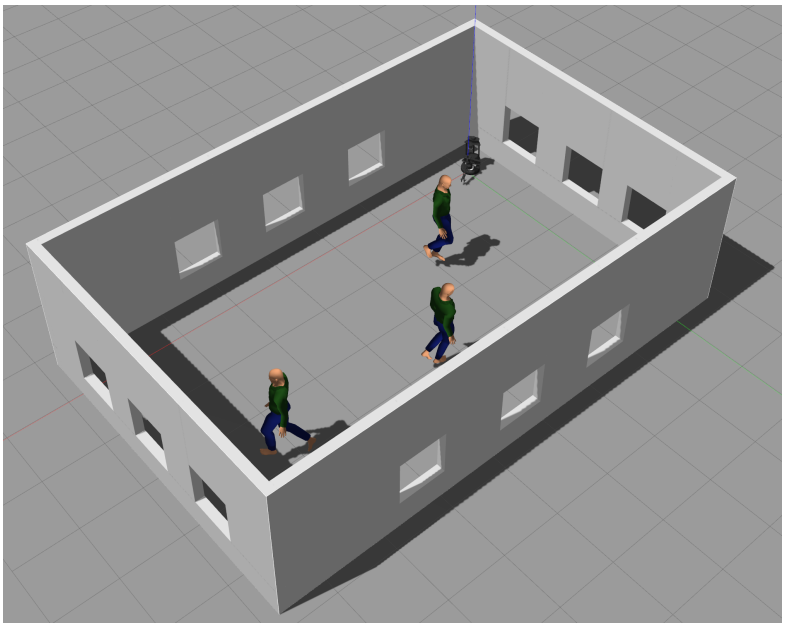

**Figure 5.4:** Gazebo simulation with pedestrian moving using SFM plugin.

## 5.3 Tensorflow

TensorFlow is an end-to-end open-source platform for machine learning maintained by Google Brain [62]. It provides a comprehensive set of tools, libraries, and community resources for quickly building machine learning models. The application

boasts a flexible and scalable architecture, enabling computations to be deployed on various platforms, including CPUs, GPUs, and mobile devices. This adaptability makes it suitable for a broad spectrum of applications. High-level APIs like Keras are integrated into it, simplifying the process of building and training deep learning models. The platform benefits from a large and active community that contributes to its extensive ecosystem, which includes pre-trained models, additional libraries and various tools for model interpretation and deployment. What is more, TensorFlow is open source, allowing developers to access and contribute to its codebase. This openness has played a significant role in its popularity and widespread adoption in both research and industry.

As previously mentioned, the GTP algorithm requires a computational time for decision-making that exceeds the admissible time interval for real-time implementations. To address this issue and achieve the time step required by the chosen prediction interval in the algorithm design, which is $\Delta t = 0.3$s, the TensorFlow library is selected and employed to train a fully-connected neural network, whose objective is to replace the decision-making process within the algorithm itself.

In the following subsections, a concise overview of the key elements comprising TensorFlow is provided, with a focus on the aspects utilized in our GTP implementation.

### 5.3.1 Basics

In TensorFlow, numerical computations are executed on specific multidimensional arrays known as *tensors*. These tensors are characterized by:

- **Data type**: The category of data stored in it, such as float numbers or strings.
- **Rank**: The number of axes that compose the tensor. For instance, a scalar has rank 0, a vector has rank 1 and a matrix is rank 2.
- **Shape**: The lengths of the axes, summarizing the data organization. It is usually expressed as a vector.
- **Size**: The number of items in the tensor, expressed as the product of the elements in the shape vector.

Normally a tensor has a fixed structure. A modifiable data structure is recalled as a **variable**. There are multiple ways to visualize a tensor, and an example of a three-dimensional tensor visualization can be seen in Figure 5.5.

These tensors undergo modifications and multiple computations during the definition and training of a machine learning model. To expedite these procedures, each computation is structured within a TensorFlow graph, separating TensorFlow computations from other platform-related processes, such as Python functions.

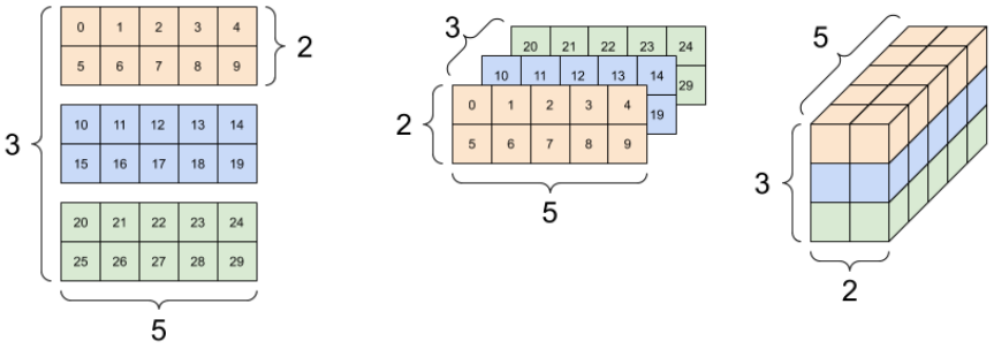


**Figure 5.5:** Different visual interpretations of a three axis tensor, with shapes specified as [3, 2, 5] [62].

A graph is a data structure that includes *operation* objects, representing computational units, and *tensor* objects, representing the data streams between the operations. This structure enables the application to execute machine learning-related procedures quickly, in parallel and efficiently. A *function* is a link between an actual python function and its graph representation.

### 5.3.2 Keras

In TensorFlow, when engaging in machine learning (ML) tasks, it's common to define, save, and restore a *model*. A model comprises a set of variables that can be updated during *training*, and a function that computes something on input tensors based on these variables. Due to the complexity of the underlying mathematical procedures, manual execution is not practical. Keras, serving as the high-level API of the TensorFlow platform, offers a user-friendly and highly productive interface for addressing ML challenges. Covering every aspect of the ML workflow, from data processing to hyperparameter tuning and deployment, Keras is designed to facilitate rapid experimentation.

A *Keras model* is an object that organizes layers and can undergo training on provided data. The data needs to be in tensor format and can be imported using specific functions from *.csv* files. In our scenario, the trajectories generated by the GTP algorithm are stored in a .csv format dataset, which is then imported into a Keras model as the training dataset. For our purposes, a *fully-connected* neural network is instantiated using the sequential model, where each layer has exactly one input tensor and one output tensor that may have different shapes. This Keras model is employed to train the neural network, evaluate its performance on a validation dataset, and subsequently save the neural network parameters. This allows loading the model onto the implemented local planner on the robot to expedite computations during navigation.

### 5.3.3 Details about GTP real-time implementation

The *datasetGTP*, obtained by processing the trajectories generated by the Matlab formulation of the GTP algorithm presented in Chapter 4, is employed to train a *fully-connected neural network* (FCNN). This neural network is then utilized by the navigation algorithm to perform the Nash equilibrium computation, enabling the selection of the optimal action at each stage of the game, within a time step that aligns with the real-time requirements for implementation into the locobot WX250s, as described earlier.

A FCNN, also known as a *feed-forward neural network*, is a type of artificial neural network architecture. In this architecture, neurons are organized in layers, and each neuron in one layer is connected with all the neurons in the subsequent layer. The layers are typically divided into an input layer, hidden layers, and an output layer. Hidden layers are responsible for learning complex patterns and representations from the input data. Each connection between neurons is associated with a multiplicative weight applied to the information passing through it. Additionally, each neuron has its own activation function that is applied to its input data. During the learning process, these weights are adjusted based on the error between the predicted output and the actual target. This process, known as back-propagation, utilizes optimization algorithms such as gradient descent to minimize the error and enhance the network's performance. For more information about FCNN and the learning process, refer to [74].

Our neural network involves the usage of 22 input quantities, which include:

- Current positions and current orientations of the robot and the other three simulated pedestrians.
- Current position of the closest static obstacle to the robot.
- Current velocity of the robot.
- Information about the map, in the form of coordinate limits $\{x_{min}, x_{max}, y_{min}, y_{max}\}$.
- Goal position given to the robot.

The outputs of the system are the predicted positions, orientations, and velocities of the robot at six future time instants from the current time instant, resulting in a total of 30 outputs.

The structure of the implemented FCNN comprises an input layer composed of 22 neurons, which is fed by inputs that undergo a normalization procedure to achieve higher performance metrics. This normalization is implemented because the ranges of the values assumed by the inputs are quite different. For instance, the velocity values and the map-related information differ by one order of magnitude,

which could pose challenges during the training process. Normalization helps correctly weight the influence of the input values and facilitates effective learning during the training phase. Additionally, the internal structure consists of four hidden layers with a variable number of neurons, organized as 600-500-260-180-80. These values were chosen through a trial and error procedure, revealing that using this configuration enables higher performance in prediction. The neurons in these layers use the Sigmoid function as an activation function to combine the data from the previous layer. The total number of trained parameters is 508450. A representation of a simplified version of a FCNN can be seen in Figure 5.6

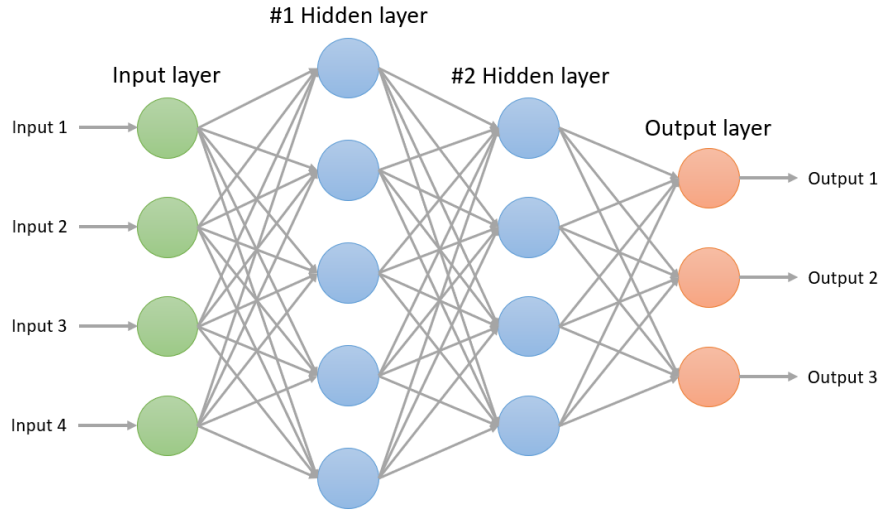


**Figure 5.6:** Representation of a simplified Fully-connected Neural Network (FCNN). It is composed by four input neurons, two hidden layers of five and three neurons and three output neurons.

The training phase is conducted using the pre-built *adam* optimizer of TensorFlow, which implements a stochastic gradient descent method to solve the optimization processes needed. The loss function employed for the training phase is Mean Squared Error (MSE), while the performance evaluation is carried out using both this metric and the Mean Absolute Error (MAE). The dataset comprises 31519 labeled data, and the training epochs are set to 80. The final performance metrics obtained are a value of 0.0180 for MSE and 0.0647 for MAE.

A scheme that summarize the procedure followed in order to realize the GTP real-time implementation is reported in Figure 5.7.

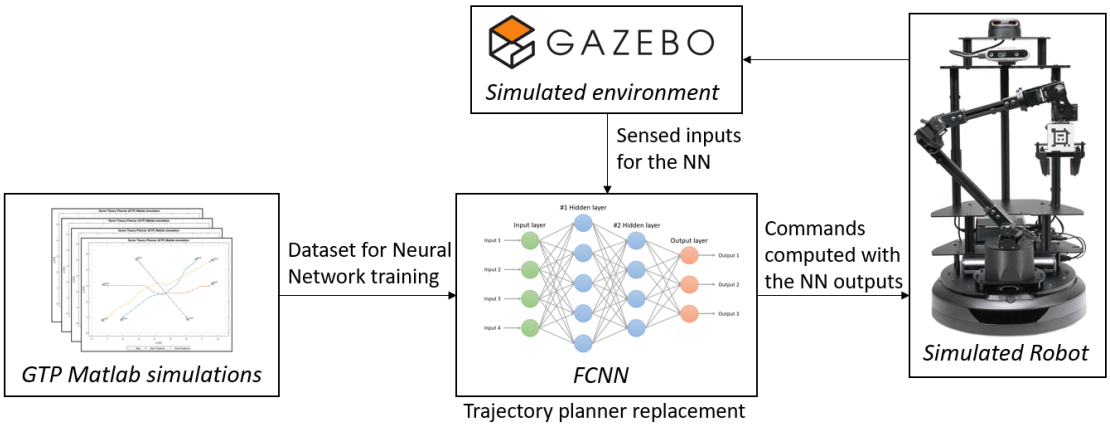


**Figure 5.7:** A schematic representation summarizing the procedure followed for the GTP real-time implementation. Preliminarily a large number of GTP Matlab simulations were conducted to generate a dataset. This dataset was then utilized to train the FCNN, intended to replace the trajectory planner of the simulated robot. The FCNN takes inputs from direct sensing of the simulated environment, and its outputs enable the motion planner to compute the commands to be sent to the simulated robot. Subsequently, the Gazebo information is updated, and the loop restarts.

## 5.4 Robot description: Locobot-WX250s

Before delving into the details of the algorithms' testing and validation performed in simulation, let's provide a general overview of the utilized robot model. Additionally, it's worth noting that the described agent is present in the facilities of the robotics lab, allowing for some simple real-world trials. The utilized agent in this context is the Locobot WX250s, a ground-wheeled robot developed and maintained by Trossen Robotics [10]. It is equipped with a well-configured set of sensors, enabling a comprehensive perception of the surrounding environment. Furthermore, it features a 6-degree-of-freedom (6DOF) robotic arm, which, in principle, can be used for simple grasping operations.

### 5.4.1 Main components

In the subsequent subsections, a high-level description of the components is provided, with a specific focus on the functionalities they offer for the navigation task.

**Figure 5.8:** Locobot WX250s [10].

### Create3 mobile base

The hardware component responsible for the robot's movement is the mobile base, specifically the iRobot® Create® 3 Educational Robot [75]. This base is equipped with a comprehensive suite of onboard sensors and actuators and operates within a software environment built entirely on the ROS2 framework. The paradigms of ROS2 are utilized for data transfer and command reception. The robot is wired to the NUC computer using an Ethernet cable, and Trossen Robotics [10] has successfully interconnected ROS1 with ROS2 using a software bridge, enabling ROS1 development.

The movement of the robot is controlled by two independent drive wheels, and a front caster functions as a balancing element. Both wheels are equipped with proper odometry sensors, allowing for the estimation of the robot's position in space during motion. Additionally, optical and infrared sensors provide the base with the capability to perceive the nearest obstacles.

### RPLIDAR A2M8

The RPLIDAR A2M8 is an indoor 360-degree 2D LIDAR (Light Detection and Ranging) sensor. Its primary role is to perceive the environment through laser


**Figure 5.9:** Create3 mobile base [10].

scanning, achieved by capturing a high rate of point-wise measurements during a rotational movement. It can acquire up to 8000 samples of laser ranging per second. Moreover, the onboard system can perform scans within a range of 12-18 meters. The generated 2D point cloud data can be utilized for mapping, localization, and object/environment modeling [10].


**Figure 5.10:** RPLIDAR A2M8 [10].

During the experiments, the LIDAR enables the generation of the environment static map, a crucial task for the testing phase. The generated 2D point cloud, directly visualizable in RViz, is processed in each algorithm implementation to compute the nearest obstacle, a fundamental data for the functionalities to be

provided.

### Intel Realsense Depth Camera D435

Another crucial built-in sensor used complementarily to the LIDAR for SLAM is a stereocamera, the RealSense Depth Camera D435, developed by Intel Corporation [76]. Its fundamental role is to reconstruct the three-dimensional environment and provide real-time depth data. This is achieved by capturing data from two cameras positioned at slightly different positions, enabling the reconstruction of the third dimension through specific mathematical procedures. Details about the specific algorithms implemented by the device are left uncovered, as they are beyond the scope of this thesis.

**Figure 5.11:** Intel Realsense Depth Camera D435 [10].

The stereocamera is positioned right in front of the Locobot, allowing it to perceive nearby obstacles or moving entities. During our testing phases, its primary usage is in mapping operations.

### Intel NUC 8i3BEH Mini PC

The central computational unit of the robot is a mini-computer developed by Intel Corporation, specifically the Intel NUC 8i3BEH [77]. It features the following specifications: 8th Gen Intel Dual-Core i3, 8GB DDR4 RAM, 240GB Solid State Drive (SSD), Intel Iris Plus Graphics 655, WiFi, Bluetooth 5.0, Gigabit Ethernet, 4k Support, Card Reader, Dual Monitor Capability, HDMI, USB, Thunderbolt 3, and runs on Ubuntu 20.04 [10].

All the sensors are directly connected to the NUC by means of USB cables. The Create3 base is connected over an Ethernet cable. Sensors, NUC, and robotic arm are powered by an external 5000 mAh battery pack, while the Create3 has its own separate power source.

**Figure 5.12:** Intel NUC 8i3BEH [10].

### WidowX 250 Robot Arm

The WidowX 250 Robot Arm is a 6DOF manipulator driven by the dynamic DYNAMIXEL-X Series actuators [78]. It is designed for education and research, with support for ROS Noetic, Gazebo, and MATLAB. The robot has a 75cm horizontal reach from the center of the base to the gripper, with a total span of 150cm. The working payload is 250g. It represents the weight that the arm should not exceed under normal working circumstances and is measured by the arm's ability to repeatedly lift an object at roughly half extension without failure. Additionally, gripper carriages are designed for users to quickly and easily change the gripper fingers for different projects.

Given that grasping and pick-and-place procedures are beyond the scope of our research, the arm is not utilized in our experimentations. However, it could be an interesting feature to be employed in addition to the mobile base navigation, enabling the robot to accomplish more complex tasks involving manipulation.

**Figure 5.13:** WidowX 250 Robot Arm [78].

# Chapter 6

# Matlab simulations

The effectiveness of the proposed human prediction method has been tested and validated by Galati et al. [42]. In our work, we implemented the same procedure in Matlab and demonstrated that performances can be enhanced through proper parameter tuning. This chapter provides a brief analysis of the attained performances, highlighting the key features of the algorithm and identifying potential areas for improvement.

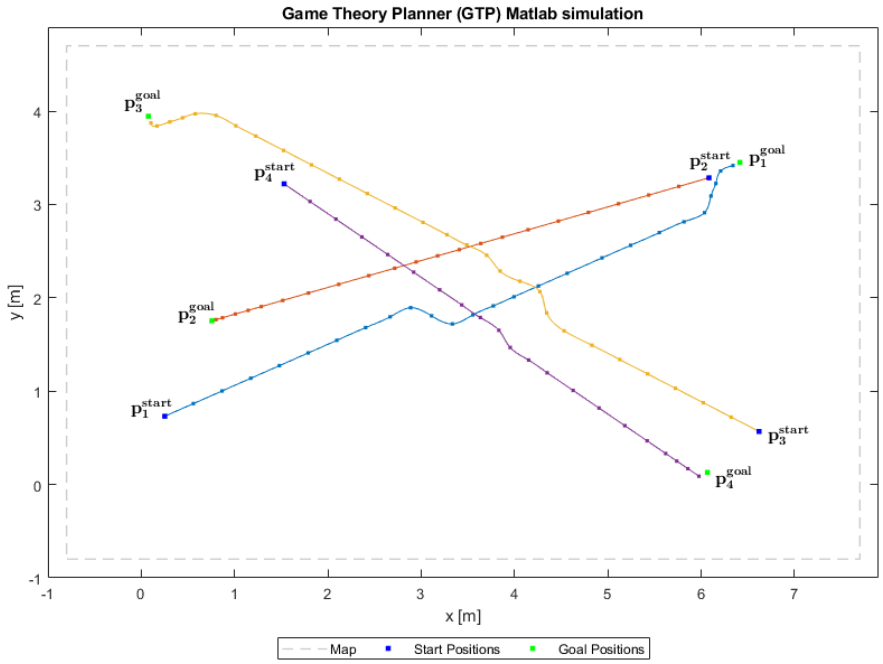


**Figure 6.1:** Generated trajectories obtained considering four agents moving with the proposed Game Theory prediction model.

The detailed description of the code used can be found in Chapter 4, along with the primary parameters used (reported in Table 4.2). Figure 6.1 illustrates the generated trajectories obtained by considering four agents, each having five possible actions to choose from in order to navigate in the shared environment.

Upon observing the trajectories, the initial impression is positive: the agents successfully avoid collisions with each other, generating relatively smooth trajectories, even though the movement modeling considers only five possible actions that agents can choose at each stage of the game. In Figure 6.2, the number of simulated pedestrians is increased to six. The overall behavior highlights that the model can effectively predict human movements even in highly crowded environments without significant difficulty.

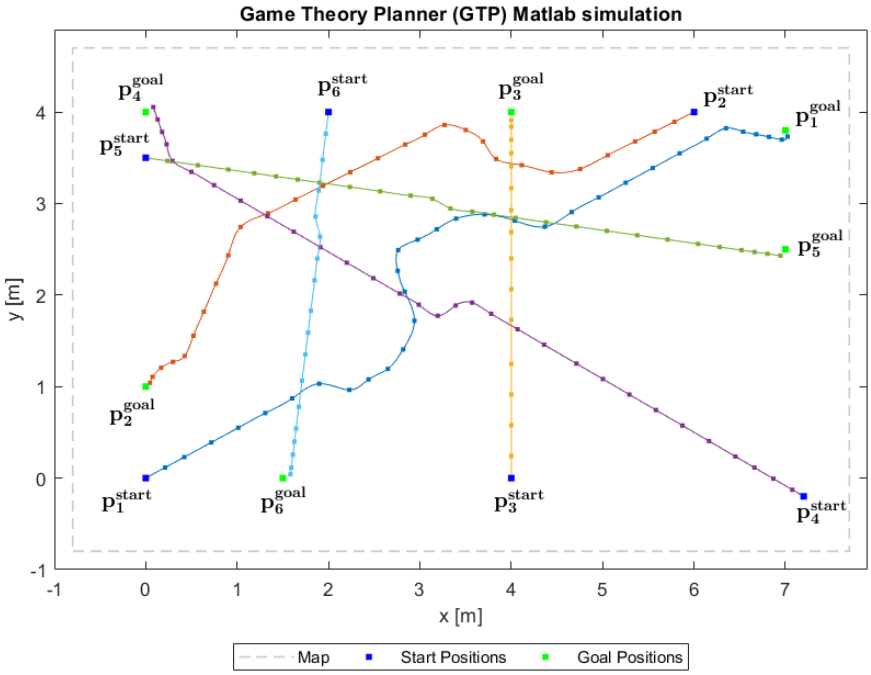


**Figure 6.2:** Generated trajectories obtained considering six agents moving with the proposed Game Theory Planner.

Analysing the model via numerous simulations with variations in all pertinent parameters, we have identified that the parameter predominantly influencing performance is the number of actions, denoted as $n\_ac$. Increasing its value results in smoother trajectories and heightened precision in predicting movements. Figure 6.3 illustrates a simulation with four actors and seven actions, while Figure 6.4 depicts another simulation with the same number of actors and nine actions. The trajectories generated exhibit a visually smoother quality and are subjectively perceived as

more natural and human-like with an augmented number of actions.

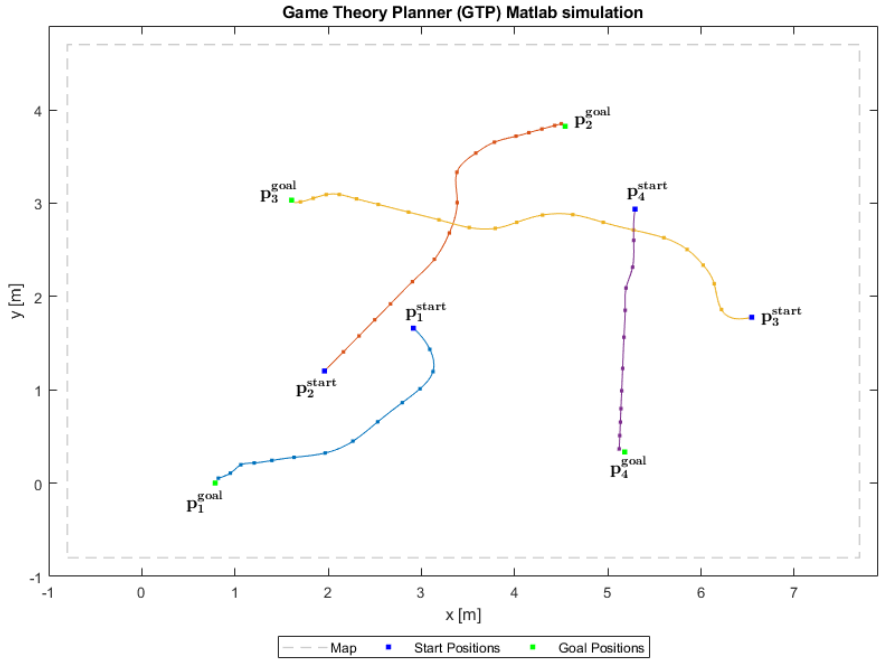


**Figure 6.3:** Generated trajectories obtained considering four agents moving and seven possible actions.

Starting from the outlined predictive model and following the procedure mentioned in the preceding chapters, we have implemented our Game Theory Planner (GTP) as a C++ ROS plugin for the *move_base* package, as detailed in Chapter 5. The primary challenge encountered in real-time implementation is associated with the considerable time required by the code to solve the Nash Equilibrium, as elaborated in Chapter 4. To address this issue, we employ a Fully-connected Neural Network trained with a dataset comprising trajectories generated by the aforementioned predictive model. To generate a relatively high number of trajectories necessary for the neural network training, the parameter *n_ac* is maintained at a value of five. While this choice may impose limitations on achievable performance, it is considered a preliminary starting point, recognizing the potential for higher performance levels with further optimization in the code.

The implemented planning algorithm will undergo simulation in Gazebo and evaluation based on criteria such as the naturalness of motion and perceived comfort from the human perspective, as delineated in Chapter 7. This evaluation aims to demonstrate the feasibility of utilizing Game Theory as the foundational predictive model for Human-Robot Interaction within a path planner designed for a robot navigating among pedestrians.

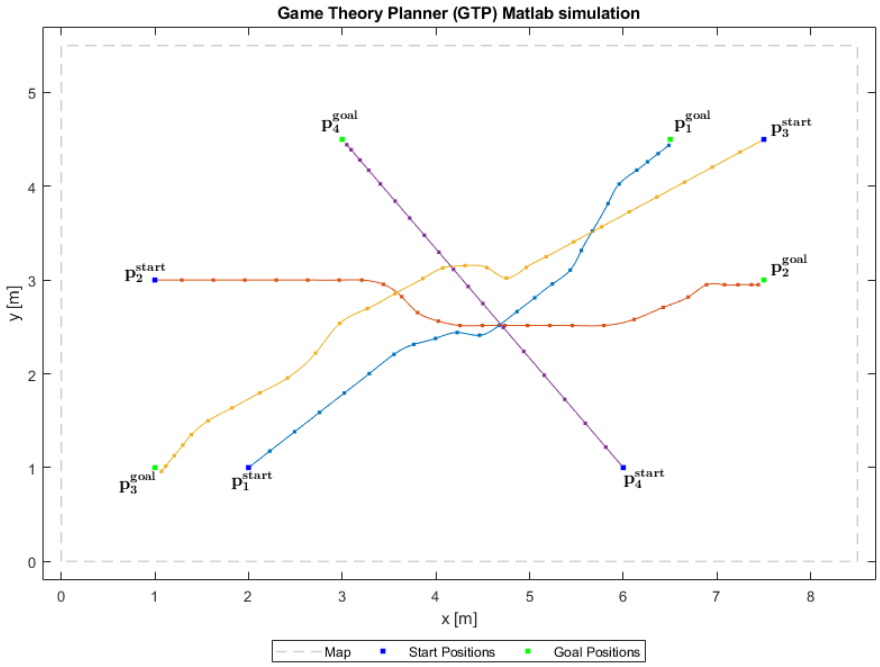


**Figure 6.4:** Generated trajectories obtained considering four agents moving and nine possible actions.

# Chapter 7

# Gazebo simulations and performance evaluation

A simulated testing procedure has been devised to compare the proposed GTP approach for socially-aware motion navigation with two state-of-the-art methods, namely SFM and ORCA algorithms.

In the following sections, a comprehensive overview of the chosen metrics, the procedure for simulative testing, and the obtained results are provided.

## 7.1 Evaluation procedure and metrics

As mentioned in Chapter 3, a socially-aware path planner algorithm must generate trajectories that are natural and ensure user comfort during interactions. To assess these aspects, it is crucial to identify the most suitable metrics used in state-of-the-art evaluation frameworks. In this section, a concise overview of evaluation methods is provided, highlighting the chosen metrics for practical experiments in Gazebo.

While research on methods used in socially-aware robot navigation has expanded over the last years, there are no established standards for evaluation protocols [52]. Each proposed approach tends to be evaluated using scenarios and metrics chosen by the researchers based on their specific implementations, making it challenging to compare different approaches in an objective and quantifiable way [27]. The motivations for this can be connected to the subjective concept of success in evaluating a proposed functionality and the heavy dependency on contextual factors, such as location, available space, variable human behaviors, robot body conformation and cultural biases.

Considering the extreme variety of test procedures found in the literature [52], Biswas et al. [27] proposed the concept of an *ideal test*. In such a test, given

infinite resources, the strategy would involve running thousands of trials of a robot navigating through crowds in many different real-world locations. The *ideal metric*, directly connected to the previous concept, would involve the people who interacted with the robot rating their experiences using an HRI state-of-the-art questionnaire. Some examples of already tested and used questionnaires can be found in [79] [80] [81] [82]. However, the cost of running such a study is impractical, so researchers have used various testing strategies and metrics to approximate the ideal test.

One of the latest developed frameworks proposed in the literature is the SocNavBench simulation framework [27]. It reproduces a virtual representation of different real-world scenarios and tests the robot algorithms by implementing them on a fictional robot moving into such environments, where pedestrians move following real-world recorded trajectories. This approach addresses the variability of human behaviors and provides the powerful possibility of generating a large number of simulated test cases. Another useful aspect of this software tool is the automatic computation of metrics, covering a wide range of evaluation parameters that also address our central concepts of naturalness and comfort. The main drawback of SocNavBench is that its agents do not consider the mutual interaction of pedestrians with the robot presence, as the recorded data are fixed, and the pedestrians move in fixed trajectories. Furthermore, the integration with the ROS framework is challenging, making a direct connection between our framework and this evaluation tool impractical.

Essentially, we have selected our metrics based on the diverse library offered by SocNavBench, choosing the ones most relevant to our objectives. We generated the required data through Gazebo simulations, recording information about the positions of pedestrians as well as the positions and orientations of the robot. The data was saved as ROS bag files and later analyzed using the Matlab ROS toolbox [83].

### 7.1.1 Metrics used

The chosen metrics computed for a comprehensive evaluation of the naturalness and comfort assurance of the algorithms are:

- **Path Length Ratio (PLR)**. It is defined as the ratio between the minimum travel trajectory, represented by the straight line between the initial and final position of the robot, and the actual travelled path length, computed as a discrete series of straight lines between the recorded positions. The formal definition can be summarized as:

$$PLR = \frac{\|\mathbf{p}_{robot}(T_{goal}) - \mathbf{p}_{robot}(0)\|}{\sum_{t=1}^{T_{goal}} \|\mathbf{p}_{robot}(t) - \mathbf{p}_{robot}(t-1)\|} \tag{7.1}$$

where $T_{goal}$ is the number of time steps needed to reach the goal.

This can be considered as an overall performance metric, describing the efficiency of the path planner in reaching the goal position. It takes values from 0 to 1. A high value indicates that the planner tends to minimize the length of the path, resulting in an efficient goal-oriented behavior.

- **Closest Pedestrian Distance (CPD)**. It is the measure of the closest distance to the robot attained by one of the pedestrians involved in the simulated tests. Formally it can be stated as:

$$CPD = \min_{t,i} \|\mathbf{p}_{robot}(t) - \mathbf{p}_i(t)\| \tag{7.2}$$

It can be considered a comfort-related evaluation metric. Following the previously mentioned theory of *proxemics*, a higher value of CPD ensures a higher level of comfort perceived by the facing humans.

- **Average Speed (AS)**. It is the average speed of the robot over the recorded trajectory, calculated as the sum of the attained velocities divided by the number of steps taken to traverse the trajectory:

$$AS = \frac{\sum_{t=1}^{T_{goal}} \mathbf{v}_{robot}(t)}{T_{goal}} \tag{7.3}$$

The metric can be considered both an overall performance metric and a comfort-related metric. A higher average speed indicates that the robot is moving more efficiently and reaching its goal in less time. However, a faster-moving robot might be perceived as more aggressive, potentially leading to discomfort. Therefore, a balance between these two aspects must be considered in the final evaluation.

- **Path Regularity (PR)**. It is a measure of the smoothness of the generated trajectory, defined as one minus the normalized sum of the variations in orientation over the considered path:

$$PR = 1 - \frac{\sum_{t=1}^{T_{goal}} |\theta(t) - \theta(t-1)|}{\max_{alg} \sum_{t=1}^{T_{goal}} |\theta(t) - \theta(t-1)|} \tag{7.4}$$

where $\theta(t)$ represents the robot orientation and the denominator is the maximum value found of the sum over the trials performed using the three considered algorithms.

Such metric is directly connected to the naturalness of the generated trajectory. Taking values from 0 to 1, it is preferred to reach a high value, meaning that the robot has not performed an excess of rotations.

## 7.2 Simulation test design and procedures

The metrics outlined in the preceding section are designed to assess the performance of the proposed GTP path planner concerning naturalness and comfort. This evaluation aims to provide an initial, comprehensive assessment of social acceptability. To establish a reference baseline, two additional algorithms, namely the Social Force Model (SFM) [8] and the Optimal Reciprocal Collision Avoidance (ORCA) [9], have been implemented and employed in these tests.

Our performance evaluation campaign is executed through a *Monte Carlo numerical simulation* within a virtual environment established in the previously mentioned Gazebo simulator [6] [72]. The simulation encompasses 180 point-goal trials for each method under consideration. These trials are further categorized into two sets, with 90 trials each, featuring varying numbers of simulated pedestrians: the first set involves three actors within the simulated environment, while the second set comprises four pedestrians. The environmental dimensions replicate real-world trial ones found in the literature [52]. What is more, to enhance the realism of pedestrian behavior, as detailed in Chapter 5, the SFM Gazebo plugin is utilized. This plugin imparts a reactive behavior to the simulated pedestrians, aligning with a fundamental assumption guiding the design of the GTP algorithm.

The outcomes are presented and discussed in the concluding section of this chapter.

### 7.2.1 Virtual environment description

All trials were conducted within a virtual room measuring 8.5m x 5.5m. To ensure the smooth execution of the automated simulation procedure and avoid potential issues arising from the robot spawning too close to pedestrians or two pedestrians spawning in close proximity, the environment was divided into six zones, as illustrated in Figure 7.1. Zone F serves as the spawning area for the robot, while zones A, B, C, and D are designated for pedestrian spawning. Each zone can be occupied by only one agent at a time. Additionally, goal points are selected among two possible alternatives, determined by the starting zone of each agent in the environment. For instance, an agent spawning in zone A can randomly choose between D and E as goal zone. The other starting-goal zones associations are $(B \rightarrow D, F)\,, (C \rightarrow E, F)\,, (D \rightarrow A, B)$ and $(F \rightarrow C)$. Spawn and goal positions are deliberately set on opposite sides of the room, a choice aimed at encouraging interactions between the robot and the moving humans.

Both spawning and associated possible goal zones are randomly selected considering a uniform distribution. Defining the zones by coordinates limits as $\mathcal{Z}_i = \{x_i^{min}, x_i^{max}, y_i^{min}, y_i^{max}\}$ and selecting one of them as $\mathcal{Z}_i$, the initial and

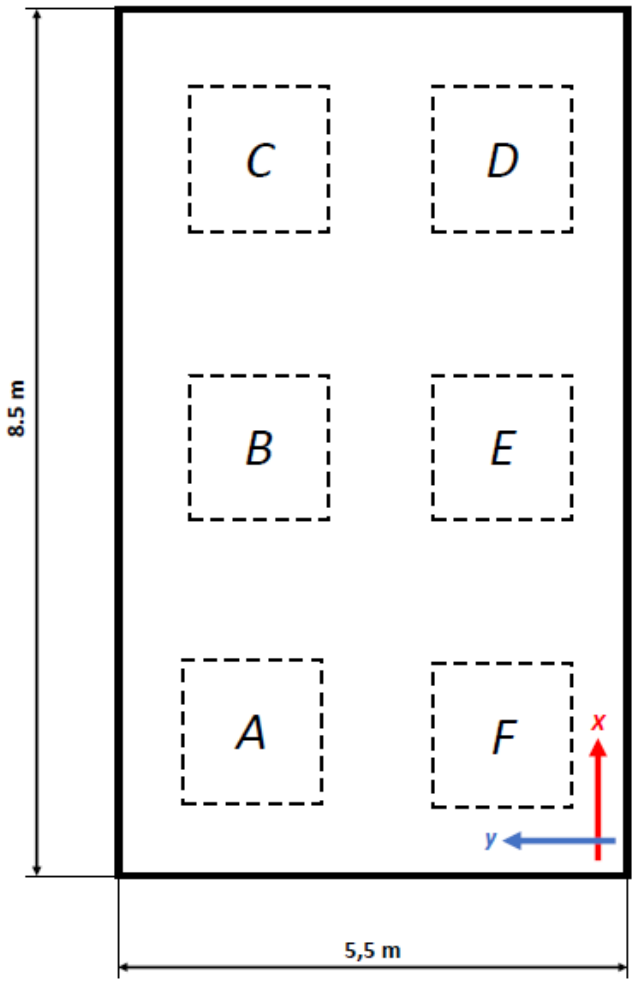


**Figure 7.1:** Spawn/goal zones in the simulation environment.

goal positions are set using the following equations:

$$\begin{aligned} x_{spawn/goal} &= x_i^{min} + rand(0{,}10)\frac{x_i^{max} - x_i^{min}}{10} \\ y_{spawn/goal} &= y_i^{min} + rand(0{,}10)\frac{y_i^{max} - y_i^{min}}{10} \end{aligned} \tag{7.5}$$

where $rand(0{,}10)$ represents a uniform random number between 0 and 10. This procedure sets random spawn or goal coordinates within the defined zones.

## 7.3 Obtained results

Subsequent to the experiments conducted in Gazebo, the data processed through Matlab's ROS toolbox yielded the results depicted in Figure 7.2.

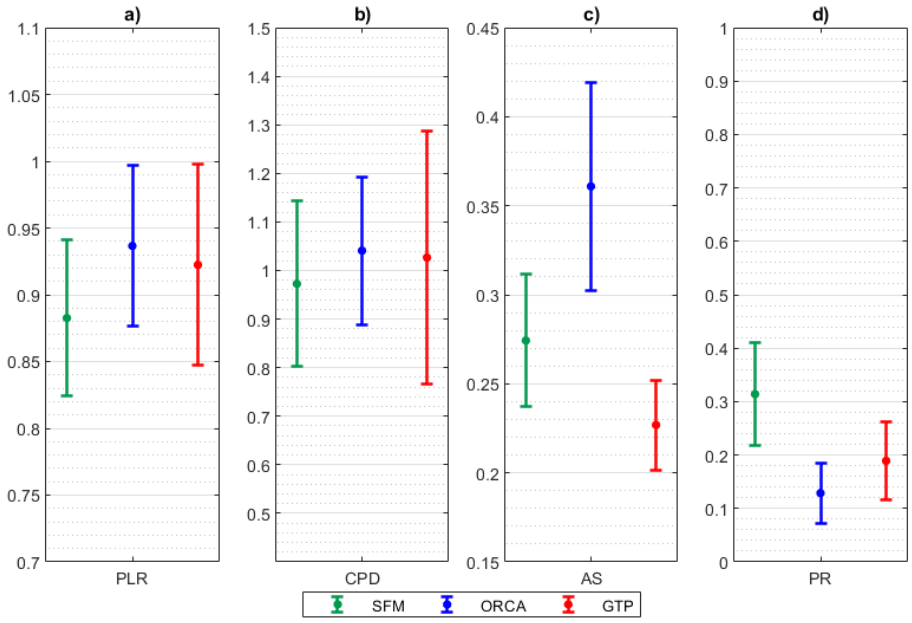


**Figure 7.2:** Results obtained expressed in terms of mean value and standard deviation of the considered metrics. The tested algorithms are SFM (Social Force Model), ORCA (Optimal Reciprocal Collision Avoidance), GTP (Game Theory Planner). The performance metrics are a) PLR (Path Length Ratio), b) CPD (Closest Pedestrian Distance), c) AS (Average Speed) and d) PR (Path Regularity).

Analyzing the graph, it can be concluded that our implementation of the GTP achieves comparable performances in terms of Path Length Ratio (PLR) and Closed Pedestrian Distance (CPD) with the state-of-the-art SFM and ORCA algorithms. Specifically, considering the data summarized in Table 7.1, GTP attains a PLR mean value higher than SFM and only slightly lower than ORCA. The same trend is observed with CPD values. This suggests that the overall performance in terms of goal-oriented behavior and efficiency is quite satisfactory, resulting in a theoretical comfort level for encountered pedestrians that surpasses SFM and is slightly below ORCA. While this marks a positive starting point, it cannot be definitively asserted that our approach has surpassed the presented state-of-the-art methods. This statement is further emphasized by the standard deviation values of GTP, which remain higher than those of the state-of-the-art approaches. This implies that the

overall behavior is satisfactory but is highly dependent on the specific scenario addressed.

Examining the third metric reported in Table 7.1, GTP achieves the lowest average speed, displaying a more cautious and deliberate behavior, particularly when compared to ORCA, whose overall behavior can be perceived as aggressive. In principle, this is advantageous in terms of comfort, as encountered pedestrians will perceive a robot that moves slowly and exhibits a non-aggressive behaviour. However, in terms of overall efficiency, the time required to reach the goal is indeed greater than the other approaches. Analyzing the standard deviations, it is noteworthy that the lowest value is associated with GTP. This indicates that the velocity values during path traversal will remain within a short range, resulting in a natural and consistent feature. One thing that is worth mentioning is that all velocity-related data are influenced by the physical limits of the robot, which is unable to move faster than 0.5 m/s, making these values noticeably lower than average human speeds.

In terms of path regularity, our GTP approach demonstrates a slight improvement over ORCA in terms of the mean value, attributed to its intrinsic predictive behavior. However, the PR value is slightly lower than SFM, indicating that there is no tangible increase in performance concerning the naturalness of movements. However, considering the standard deviations, GTP has a lower value than SFM, suggesting that its reached level of smoothness is lower but more reliable. The lowest value presented by the ORCA method is explainable by its tendency to enter deadlock situations and rotate in order to find a feasible velocity for the optimization problem that governs its movements.

| **Algorithm** | **Metric** | **Mean** | **Standard deviation** |
|---|---|---|---|
| SFM | PLR | 0.8825 | 0.0582 |
| ORCA | PLR | 0.9366 | 0.0600 |
| GTP | PLR | 0.9224 | 0.0754 |
| SFM | CPD | 0.9727 | 0.1697 |
| ORCA | CPD | 1.0408 | 0.1523 |
| GTP | CPD | 1.0269 | 0.2606 |
| SFM | AS | 0.2743 | 0.0370 |
| ORCA | AS | 0.3609 | 0.0584 |
| GTP | AS | 0.2264 | 0.0252 |
| SFM | PR | 0.3140 | 0.0960 |
| ORCA | PR | 0.1287 | 0.0557 |
| GTP | PR | 0.1890 | 0.0738 |

**Table 7.1:** Mean and standard deviation values for the computed metrics.

These results align with our expectations. The selected parameters of the GTP, used to generate the training data for the neural network, intentionally reduce the algorithm performance by selecting a lower value of possible actions during game solution to expedite trajectory computation. It is anticipated that enhancing the complexity of the model and expanding the range of possible actions for the players in the scenario will contribute to an overall performance improvement.

For the purpose of this thesis, these tests are regarded as a foundational step, serving as a starting point for future improvements. The directions for these enhancements will be elucidated in the final chapter, where a comprehensive discussion will be presented.

# Chapter 8

# Conclusions and future works

In this thesis, we have examined the concept of autonomous robot navigation, with a specific focus on socially-aware navigation. Social path planners aim to generate trajectories that are perceived as socially acceptable by humans. Recent research efforts have prominently concentrated on algorithms capable of predicting future pedestrian movements, emphasizing the importance of Human-Robot Interaction. Furthermore, the growing interest in Artificial Intelligence and Machine Learning, fueled by the increasing computational power of modern devices, has prompted numerous research groups to propose socially-aware navigation algorithms based on these topics. The prevailing trend is to increasingly center on models of this nature. Within this context, we explored a promising human behavior modeling approach based on game theory [53]. Our objective was to take a step further by creating a real-time implementation of a path planner that integrates this idea into a simulated environment, employing a Fully-Connected Neural Network to address the computational complexity associated with such approach.

Our predictive path planner, named the Game Theory Planner (GTP), underwent evaluation through the computation of state-of-the-art metrics based on 180 trials conducted in a Monte Carlo numerical simulation using the Gazebo software. The results affirm that our real-time planner exhibits satisfactory behaviors in terms of naturalness and comfort. It achieves values of Path Length Ratio and Closest Pedestrian Distance that are comparable with the state-of-the-art algorithms evaluated, namely the Social Force Model (SFM) [8] and the Optimal Reciprocal Collision Avoidance (ORCA) [9]. Therefore, GTP can be considered a promising path planner, especially considering that this initial real-time implementation represents just a preliminary effort, and there are ample opportunities for further enhancements and refinements.

A comprehensive analysis of the different facets of the design and implementation process reveals that the primary bottleneck in this modeling method is the computation of Nash Equilibrium, particularly in the prediction of pedestrian movements. This involves evaluating a substantial number of potential traversable trajectories using a high number of actions within the action set of the game that models navigation. To address this challenge, the current implementation employs a relatively low value of five possible actions from which each player can choose during the decision-making process of their strategy. An apparent avenue for improvement in future upgrades would be to increase the value of the possible actions, which could enhance the modeling accuracy and overall performance of the system.

Another potential direction for extending our work is to explore the utilization of a different type of end-to-end Machine Learning system to simulate the Nash Equilibrium computation. Alternatively, one could investigate other methods for finding solutions to the game that models the navigation process, such as the Pareto Optimal approach [58].

In conclusion, the natural progression of this work could involve the direct implementation of the GTP into a physical robot. The procedures followed in this thesis are designed to create a real-time path planner capable of functioning in the real-world version of the simulated locobot WX250s [10]. By doing so, a qualitative evaluation of the algorithm's performance can be conducted through a real-world experiment that imitates the procedures of our simulated trials. To assess social acceptability in terms of perceived comfort and the naturalness of the robot's behavior, a questionnaire could be administered to volunteers. Established questionnaires in the literature, such as Godspeed [79], RoSAS [81], or HRIES [82], could serve as a framework for this assessment.

# Bibliography


[1] J. Rios-Martinez, A. Spalanzani, and C. Laugier. «From proxemics theory to socially-aware navigation: a survey». In: International Journal of Social Robotics, 7 (Sept. 2014), pp. 137–153 (cit. on pp. 1, 11–14, 44).

[2] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch. «Human-aware robot navigation: A survey». In: Robotics and Autonomous Systems, 61 (Dec. 2013), pp. 1726–1743 (cit. on pp. 1, 12, 14, 15).

[3] A. Waytz, J. Cacioppo, and N. Epley. «Who sees human? The stability and importance of individual differences in anthropomorphism». In: Perspectives on Psychological Science, 5 (May 2010), pp. 219–232 (cit. on pp. 2, 14).

[4] A. Turnwald, D. Althoff, D. Wollherr, and M. Buss. «Understanding human avoidance behavior: interaction-aware decision making based on game theory». In: International Journal of Social Robotics, 8 (Apr. 2016), pp. 331–351 (cit. on pp. 2, 18, 21, 40, 42, 43).

[5] A. Turnwald and D. Wollherr. «Human-like motion planning based on game theoretic decision making». In: International Journal of Social Robotics, 11 (July 2019), pp. 151–170 (cit. on pp. 3, 18, 40, 42, 43).

[6] N. Koenig and A. Howard. «Design and use paradigms for Gazebo, an open-source multi-robot simulator». In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 3 (Sept. 2004), pp. 2149–2154 (cit. on pp. 3, 53, 57, 76).

[7] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache. «Simulation environment for mobile robots testing using ROS and Gazebo». In: International Conference on System Theory, Control and Computing (Oct. 2016), pp. 96–101 (cit. on p. 3).

[8] D. Helbing and P. Molnar. «Social force model for pedestrian dynamics». In: Physical Review E, 51 (May 1995) (cit. on pp. 3, 16, 21, 22, 58, 76, 81).

[9] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. «Reciprocal n-Body Collision Avoidance». In: Springer Tracts in Advanced Robotics, 70 (2011), pp. 3–19 (cit. on pp. 3, 17, 23, 27, 28, 30–35, 76, 81).

[10] Trossen Robotics. *Documentation for locobot-WX250s*. Available on line, accessed in nov. 2023. URL: `docs.trossenrobotics.com/interbotix%5C_xslocobots%5C_docs` (cit. on pp. 3, 24, 43, 55, 63–67, 82).

[11] H. S. Hewawasam, M. Y. Ibrahim, and G. K. Appuhamillage. «Past, present and future of path-planning algorithms for mobile robot navigation in dynamic environments». In: IEEE Open Journal of the Industrial Electronics Society, 3 (June 2022), pp. 353–365 (cit. on pp. 6, 9, 10).

[12] K. J. Singh, D. S. Kapoor, K. Thakur, A. Sharma, A. Nayyar, S. Mahajan, and M. A. Shah. «Map making in social indoor environment through robot navigation using active SLAM». In: IEEE Access, 10 (Dec. 2022), pp. 134455–134465 (cit. on p. 6).

[13] X. Xiao, B. Liu, G. Warnell, and P. Stone. «Motion planning and control for mobile robot navigation using machine learning: a survey». In: Autonomous Robots, 46 (Mar. 2022), pp. 569–597 (cit. on pp. 7, 8, 17, 20).

[14] O. Khatib. «Real-time obstacle avoidance for manipulators and mobile robots». In: Proceedings. 1985 IEEE International Conference on Robotics and Automation (Mar. 1985), pp. 500–505 (cit. on p. 9).

[15] D. Bodhale, N. Afzulpurkar, and N. T. Thanh. «Path planning for a mobile robot in a dynamic environment». In: 2008 IEEE International Conference on Robotics and Biomimetics (Feb. 2009), pp. 2115–2120 (cit. on p. 9).

[16] R. Möller, A. Furnari, S. Battiato, A. Härmä, and G. M. Farinella. «A survey on human-aware robot navigation». In: Robotics and Autonomous Systems, 145 (Nov. 2021) (cit. on pp. 10, 12, 44).

[17] A. Pandey and D. R. Parhi. «Optimum path planning of mobile robot in unknown static and dynamic environments using Fuzzy-Wind Driven Optimization algorithm». In: Defence Technology, 13 (Feb. 2017), pp. 47–58 (cit. on p. 10).

[18] Boston Dynamics. *Boston Dunamics home page*. Available on line, accessed in nov. 2023. URL: `bostondynamics.com` (cit. on p. 11).

[19] S. F. Ramadhan, M. Taufik, D. Novita, and A. Turnip. «Design of 2D LiDAR-based Indoor SLAM for Medical Robot Covid-19». In: AIMS 2021 - International Conference on Artificial Intelligence and Mechatronics Systems (Apr. 2021) (cit. on p. 11).

[20] H. Q. T. Ngo, V. N. Le, V. D. N. Thien, T. P. Nguyen, and H. Nguyen. «Develop the socially human-aware navigation system using dynamic window approach and optimize cost function for autonomous medical robot». In: Advances in Mechanical Engineering, 12 (Dec. 2020), pp. 1–17 (cit. on p. 11).

[21] R. Pinillos, S. Marcos, R. Feliz, E. Zalama, and J. Gómez-García-Bermejo. «Long-term assessment of a service robot in a hotel environment». In: Robotics and Autonomous Systems, 79 (May 2016), pp. 40–57 (cit. on p. 11).

[22] S. Ivanov and C. Webster. «Restaurants and robots: public preferences for robot food and beverage services». In: Journal of Tourism Futures, 9 (May 2023), pp. 229–239 (cit. on p. 11).

[23] D. Ruiz-Equihua, J. Romero, S. M. C. Loureiro, and M. Ali. «Human–robot interactions in the restaurant setting: the role of social cognition, psychological ownership and anthropomorphism». In: International Journal of Contemporary Hospitality Management, 36 (May 2023), pp. 1966–1985 (cit. on p. 11).

[24] N. Koceska, S. Koceski, P. B. Zobel, V. Trajkovik, and N. Garcia. «A telemedicine robot system for assisted and independent living». In: Sensors (Switzerland), 19 (Feb. 2019) (cit. on p. 11).

[25] K. Kitazawa and T. Fujiyama. «Pedestrian vision and collision avoidance behavior: Investigation of the information process space of pedestrians using an eye tracker». In: Pedestrian and evacuation dynamics (2009), pp. 95–108 (cit. on pp. 13, 23, 46).

[26] M. Mori. «The uncanny valley». In: Energy, 7 (1970), pp. 33–35 (cit. on p. 14).

[27] A. Biswas, A. Wang, G. Silvera, A. Steinfeld, and H. Admoni. «Socnavbench: A grounded simulation testing framework for evaluating social navigation». In: ACM Transactions on Human-Robot Interaction, 11 (July 2022), pp. 1–24 (cit. on pp. 16, 21, 23, 73, 74).

[28] D. Fox, W. Burgard, and S. Thrun. «The dynamic window approach to collision avoidance». In: IEEE Robotics and Automation Magazine, 4 (Mar. 1997), pp. 23–33 (cit. on pp. 16, 53).

[29] S. M. LaValle and J. J. Kuffner. «Randomized Kinodynamic Planning». In: The International Journal of Robotics Research, 20 (May 2001), pp. 378–400 (cit. on p. 16).

[30] P. Fiorini and Z. Shiller. «Motion planning in dynamic environments using velocity obstacles». In: The International Journal of Robotics Research, 17 (July 1998), pp. 760–772 (cit. on pp. 16, 25).

[31] P. Trautman and A. Krause. «Unfreezing the robot: Navigation in dense, interacting crowds». In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (Oct. 2010) (cit. on p. 16).

[32] J. Van Den Berg, M. Lin, and D. Manocha. «Reciprocal velocity obstacles for real-time multi-agent navigation». In: 2008 IEEE International Conference on Robotics and Automation (May 2008) (cit. on pp. 17, 25, 26, 34).

[33] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha. «The hybrid reciprocal velocity obstacle». In: IEEE Transactions on Robotics, 27 (Aug. 2011), pp. 696–706 (cit. on pp. 17, 34).

[34] C. M. Sánchez, M. Zella, J. Capitán, and P. J. Marrón. «From perception to navigation in environments with persons: An indoor evaluation of the state of the art». In: Sensors, 22 (Feb. 2022) (cit. on pp. 17, 18).

[35] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How. «Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns». In: Autonomous Robots, 35 (May 2013), pp. 51–76 (cit. on p. 18).

[36] A. Bera, T. Randhavane, R. Prinja, and D. Manocha. «Sociosense: Robot navigation amongst pedestrians with social and psychological constraints». In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (Dec. 2017) (cit. on pp. 18, 20).

[37] Y. F. Chen, M. Liu, M. Everett, and J. P. How. «Decentralized non communicating multiagent collision avoidance with deep reinforcement learning». In: 2017 IEEE international conference on robotics and automation (2017), pp. 285–292 (cit. on p. 20).

[38] Y. F. Chen, M. Everett, M. Liu, and J. P. How. «Socially aware motion planning with deep reinforcement learning». In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (Dec. 2017), pp. 1343–1350 (cit. on p. 20).

[39] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. «Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning». In: 2019 International Conference on Robotics and Automation, 26 (Aug. 2019) (cit. on p. 20).

[40] A. Pokle, R. Martín-Martín, P. Goebel, V. Chow, H. Ewald, J. Yang, Z. Wang, A. Sadeghian, and D. Sadighand S. Savarese et al. «Deep local trajectory replanning and control for robot navigation.» In: 2019 international conference on robotics and automation (2019), pp. 5815–5822 (cit. on p. 20).

[41] N. Perez-Higueras, F. Caballero, and L. Merino. «Human-aware robot navigation: A survey». In: 2018 IEEE International Conference on Robotics and Automation (2018), pp. 5897–5902 (cit. on p. 20).

[42] G. Galati, S. Primatesta, S. Grammatico, S. Macrì, and A. Rizzo. «Game theoretical trajectory planning enhances social acceptability of robots by humans». In: Scientific Reports, 12 (Dec. 2022) (cit. on pp. 21, 40, 42, 43, 45, 69).

[43] F. Zanlungo, T. Ikeda, and T. Kanda. «Social force model with explicit collision prediction». In: Europhysics Letters, 93 (Mar. 2011), p. 68005 (cit. on p. 22).

[44] R. Han, S. Chen, S. Wang, Z. Zhang, R. Gao, Q. Hao, and J. Pan. «Reinforcement learned distributed multi-robot navigation with reciprocal velocity obstacle shaped rewards». In: IEEE Robotics and Automation Letters, 7 (July 2022), pp. 5896–5903 (cit. on p. 24).

[45] H. Cheng, Q. Zhu, Z. Liu, T. Xu, and L. Lin. «Decentralized navigation of multiple agents based on ORCA and model predictive control». In: IEEE International Conference on Intelligent Robots and Systems (Sept. 2017), pp. 3446–3451 (cit. on p. 24).

[46] J. Van Den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin. «Interactive navigation of multiple agents in crowded environments». In: Proceedings of the 2008 symposium on Interactive 3D graphics and games (Feb. 2008), pp. 139–147 (cit. on pp. 27, 34).

[47] S. P. Boyd and L. Vandenberghe. «Convex optimization». In: Book (2004) (cit. on pp. 27, 28, 33).

[48] M. De Berg. «Computational geometry algorithms and applications». In: Book, Springer (2000) (cit. on pp. 33, 34).

[49] A. Turlach and S. J. Wright. «Quadratic programming». In: Wiley Interdisciplinary Reviews: Computational Statistics, 7 (Mar. 2015), pp. 153–159 (cit. on p. 33).

[50] M. Nazecic-Andrlon. *Public Github repository - PyORCA*. Available on line, accessed in nov. 2023. URL: `github.com/Muon/pyorca` (cit. on p. 35).

[51] S. Bochkanov et al. *Documentation for ALGLIB - C++ library*. Available on line, accessed in nov. 2023. URL: `alglib.net/docs.php` (cit. on pp. 35, 39).

[52] Y. Gao and C. M. Huang. «Evaluation of socially-aware robot navigation». In: Frontiers in Robotics and AI, 8 (Jan. 2022) (cit. on pp. 37, 49, 73, 76).

[53] G. Galati. «Learning from humans to improve socially-aware motion planning». In: Marter of Science Thesis, Politecnico di Torino (2019) (cit. on pp. 39, 40, 45, 81).

[54] A. M. Turing. «Computing machinery and intelligence». In: Mind, 59 (1950), pp. 433–460 (cit. on p. 40).

[55] A. P. Aygin, I. Cicekli, and V. Akman. «Turing test: 50 years later». In: Minds and Machines, 10 (Nov. 2000), pp. 463–518 (cit. on p. 40).

[56] K. Leyton-Brown and Y. Shoham. «Essentials of game theory: a concise multidisciplinary introduction». In: Synthesis lectures on artificial intelligence and machine learning, 2 (2008), pp. 1–88 (cit. on pp. 40–42).

[57] J. Von Neumann. «On the theory of games of strategy». In: Annals of mathematics (1928), pp. 295–320 (cit. on p. 40).

[58] J. Von Neumann and O. Morgenstern. «Theory of games and economic behavior». In: Book (1944) (cit. on pp. 40, 82).

[59] J. Nash. «Non-cooperative games». In: Annals of mathematics (1951), pp. 286–295 (cit. on pp. 40, 42, 43).

[60] S. Sagratella. «Algorithms for generalized potential games with mixed-integer variables». In: Computational Optimization and Applications, 68 (July 2017), pp. 689–717 (cit. on pp. 43, 45, 46).

[61] A. R. MaNeill. «Energetics and optimization of human walking and running: The 2000 Raymond Pearl Memorial Lecture». In: American Journal of Human Biology, 14 (Sept. 2002), pp. 641–648 (cit. on p. 44).

[62] Google Brain. *TensorFlow official guide*. Available on line, accessed in nov. 2023. URL: `tensorflow.org/guide/basics` (cit. on pp. 46, 58, 60).

[63] Open Robotics. *ROS overview*. Available on line, accessed in nov. 2023. URL: `ros.org` (cit. on pp. 53, 55).

[64] Open Robotics. *Documentation for ROS*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org` (cit. on p. 54).

[65] Open Robotics. *Documentation for ROS navigation stack*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org/navigation` (cit. on p. 55).

[66] Trossen Robotics. *Github repository for interbotix-ros-rovers*. Available on line, accessed in nov. 2023. URL: `github.com/Interbotix/interbotix%5C_ros%5C_rovers` (cit. on p. 55).

[67] Open Robotics. *ROS package - move_base*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org/move%5C_base?distro=noetic` (cit. on pp. 55, 56).

[68] Open Robotics. *ROS package - pluginlib*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org/pluginlib` (cit. on p. 56).

[69] Open Robotics. *ROS navigation tutorials*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org/navigation/Tutorials` (cit. on p. 56).

[70] Open Robotics. *Documentation for RViz*. Available on line, accessed in nov. 2023. URL: `wiki.ros.org/rviz` (cit. on p. 56).

[71] P. Kaur, Z. Liu, and W. Shi. «Simulators for mobile social robots: state-of-the-art and challenges». In: 2022 Fifth International Conference on Connected and Autonomous Driving (MetroCAD) (Apr. 2022), pp. 47–56 (cit. on p. 57).

[72] Open Robotics. *Documentation for Gazebo*. Available on line, accessed in nov. 2023. URL: `gazebosim.org/home` (cit. on pp. 57, 76).

[73] Open Robotics. *Documentation for Gazebo SFM plugin*. Available on line, accessed in nov. 2023. URL: github.com/robotics-upo/gazebo%5C_sfm%5C_plugin (cit. on p. 58).

[74] V. K. Ojha, A. Abraham, and V. Snášel. «Metaheuristic design of feedforward neural networks: A review of two decades of research». In: Engineering Applications of Artificial Intelligence, 60 (Apr. 2017), pp. 97–116 (cit. on p. 61).

[75] Intel Corporation. *Documentation for Create3 mobile base*. Available on line, accessed in nov. 2023. URL: iroboteducation.github.io/create3%5C_docs (cit. on p. 64).

[76] Intel Corporation. *Documentation for depth-camera-d435*. Available on line, accessed in nov. 2023. URL: intelrealsense.com/depth-camera-d435 (cit. on p. 66).

[77] Intel Corporation. *Documentation for nuc8i3beh*. Available on line, accessed in nov. 2023. URL: intel.com/content/www/us/en/products/sku/126150/intel-nuc-kit-nuc8i3beh/specifications.html (cit. on p. 66).

[78] Trossen Robotics. *Documentation for widowx-250-robot-arm-6dof*. Available on line, accessed in nov. 2023. URL: trossenrobotics.com/widowx-250-robot-arm-6dof.aspx (cit. on pp. 67, 68).

[79] C. Bartneck, D. Kulić, E. Croft, and S. Zoghbi. «Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots». In: International Journal of Social Robotics, 1 (Nov. 2008), pp. 71–81 (cit. on pp. 74, 82).

[80] M. Joosse, A. Sardar, M. Lohse, and V. Evers. «BEHAVE-II: The Revised Set of Measures to Assess Users' Attitudinal and Behavioral Responses to a Social Robot». In: International Journal of Social Robotics, 5 (June 2013), pp. 379–388 (cit. on p. 74).

[81] C. M. Carpinella, A. B. Wyman, M. A. Perez, and S. J. Stroessner. «The robotic social attributes scale (RoSAS) development and validation». In: Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction (Mar. 2017), pp. 254–262 (cit. on pp. 74, 82).

[82] N. Spatola, B. Kühnlenz, and G. Cheng. «Perception and evaluation in human–robot interaction: The Human–Robot Interaction Evaluation Scale (HRIES) — A multicomponent approach of anthropomorphism». In: International Journal of Social Robotics, 13 (Jan. 2021), pp. 1517–1539 (cit. on pp. 74, 82).

[83] MathWorks. *Documentation for ROS toolbox*. Available on line, accessed in nov. 2023. URL: mathworks.com/help/ros/index.html?s%5C_tid=CRUX%5C_lftnav (cit. on p. 74).