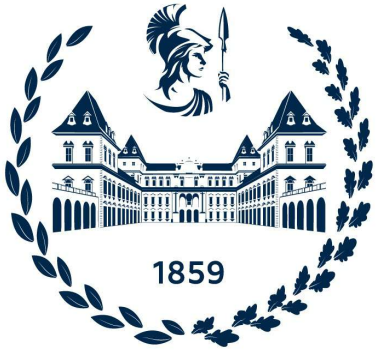
POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering



Master's Degree Thesis

Modelling and simulation of mobile robot motion and its interaction with humans

Supervisors

Prof. Alessandro RIZZO

Dr. Giada GALATI

Candidate

Andrea USAI

October 2023

"Quel *destino* di cui parli io *lo posso cambiare*!"

Acknowledgements

Innanzitutto, desidero esprimere la mia profonda gratitudine al Professor Rizzo per avermi concesso l'opportunità di svolgere questa tesi e per la costante disponibilità che ha dimostrato nei miei confronti.
Ringrazio la Dottoressa Galati, la mia correlatrice, per il suo continuo sostegno e la sua guida preziosa nelle diverse fasi del progetto.
Desidero ringraziare anche Giacomo, collega e collaboratore, per il prezioso contributo che ha fornito alla realizzazione di questa tesi.

Un sentito ringraziamento va ai miei genitori, che sin da bambino mi hanno insegnato l'importanza di porsi degli obiettivi e di lavorare costantemente per raggiungere i propri sogni.
Ringrazio sinceramente mio fratello Luca, in quanto negli anni è stato per me un costante punto di rifermento da raggiungere e superare.
Un ringraziamento speciale va ai miei amici Luca, Ilenia, Daniel e Sara, che sono stati presenti in un momento per me molto difficile. La loro amicizia ha reso quel periodo meno doloroso.
Non posso non ringraziare Emanuele, Lorenzo, Lorenzo e Davide, amici fin dagli anni di scuola superiore, in quanto hanno dimostrato che l'amicizia può trascendere la distanza, lo scorrere del tempo o le scelte di vita differenti.
E infine, ultima ma assolutamente non per importanza, ringrazio profondamente Martina, la mia fidanzata. Il suo ingresso improvviso in un momento molto buio della mia vita si è rivelato come un faro di luce in mezzo alle tenebre.

Tutte queste persone hanno sempre creduto in me. La loro presenza nella mia vita è stata fondamentale nel rendermi la persona che sono oggi. Sono grato di averli avuti al mio fianco in questo percorso.

Summary

In recent years, technological advances have allowed robots to become a part of our everyday lives. The integration of such robots in human-populated environments presents new challenges, particularly in the context of navigation. To operate efficiently in these scenarios, robots must consider the physical and psychological safety of pedestrians. In this way, they can move safely and in a socially acceptable manner. To achieve these goals, it is essential to develop socially aware navigation algorithms able to perceive humans as social entities, predict their movements, and perform mutual avoidance maneuvers. Although existing approaches ensure safe robot navigation even in crowded environments, many of them exhibit reactive behavior and treat pedestrians as mere dynamic objects without predicting their future movements. Recently, various machine learning techniques have been used to predict human motion with optimal results. However, most of them focus on predicting human movements individually, neglecting potential interactions between pedestrians during navigation. Therefore, modeling social behaviors is crucial for designing socially-aware robots that can predict future pedestrians' motions and adjust their decisions accordingly. To face these challenges, this thesis proposes a navigation algorithm that combines game theory with the well-known Social Force Model (SFM). Unlike previous approaches, game theory allows to explicitly model the decision-making process typical of human beings by considering pedestrians and the robot as rational agents capable of influencing each other's decisions. Here, navigation is modeled as a non-cooperative game. Each agent has a set of possible actions represented by trajectories generated from different sets of Social Force Model parameters. Each agent aims to find optimal trajectories considering potential interactions with other players. The solution of the game is established by reaching Nash equilibrium. To ensure a higher level of naturalness and comfort, it is necessary to solve the game by considering trajectories that are as real as possible. However, the manual definition of the SFM parameters can be very complex. This is related to the high sensitivity of the model and the significant variability of the behaviors they determine. Therefore, a Differential Evolution (DE) algorithm has been used to estimate the best SFM parameters that approximate real human trajectories from a public dataset called "Thör". However, the excessive

time required for estimation makes DE not suitable for real-time applications. To overcome this problem, a neural network has been employed to mimic the behavior of the Differential Evolution. In this context, DE has been used to create a training dataset, where the estimated best parameters have been exploited to label specific features of the analyzed real trajectory. The proposed algorithm has been quantitatively validated through Monte Carlo numerical simulations. Specifically, the algorithm has been compared with two other state-of-the-art approaches: the standard SFM and ORCA (Optimal Reciprocal Collision Avoidance). The performance evaluation has been carried out using 4 state-of-the-art performance metrics: Path Length Ratio, Closest Pedestrian Distance, Average Speed, and Path Regularity. Our numerical results,performed in different scenarios with different number of pedestrians, indicate that the proposed algorithm outperforms the standard SFM and has significant improvements over ORCA, especially concerning Path Regularity.

Table of Contents

List of Tables

List of Figures

Acronyms

SFM
Social Force Model

GT
Game Theory

SLAM
Simultaneous Localization And Mapping

DL
Deep Learning

DNN
Deep Neural Network

RL
Reinforcement Learning

DRL
Deep Reinforcement Learning

IRL
Inverse Reinforcement Learning

Chapter 1

Introduction

1.1 Socially-aware navigation

Nowadays, robots are employed not only in industry, but also in a wide variety of everyday applications to assist humans in performing various types of very simple tasks. Some examples of such applications are the use of mobile robots as "guides in museums" [1] or "service robots" in offices [2], hospitals [3], or hotels [4].
In the near future, the growing diffusion of this technology will inevitably lead to the sharing of physical space between humans and robots. This perspective confronts us with issues and challenges that need to be addressed in order to ensure proper interaction between them. This is especially important in the context of navigation.
In general, robots are already capable of performing autonomous tasks efficiently when they operate in environments with simple static and dynamic objects. However, humans cannot be considered as mere moving objects but must be regarded as real social entities capable of interacting with each other.
In fact, human society is inherently characterized by social conventions, i.e., behaviors dependent on the environment and culture that help humans understand and predict the intentions of other people [5].
Therefore, to be naturally incorporated in human populated environments, mobile robots must be able to detect such behaviors, integrate them into their movement, and adapt their behavior according to the social expectations of other pedestrians involved in the navigation [5].
Overall, these considerations determine the need to develop mobile robots that are not only safe but also socially acceptable. Hence, by guaranteeing a *socially aware navigation*, it will be possible to ensure an easier coexistence between robots and humans.

In [5], Martinez et al. define socially-aware navigation 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*".

To achieve such behavior and enhance social acceptability, Kruse et al. [6] highlight three main features that the robot should ensure:

- ***Human comfort***
  It corresponds to the psychological safety perceived by humans during navigation. Indeed, the robot must avoid causing annoyance, stress, or general negative emotions toward the humans whom it interacts with.
  Therefore, the robot must be able to move in a physically safe manner, i.e., avoiding possible collisions, but also in a way that makes the pedestrians involved in navigation feel safe.
  One of the main aspects affecting comfort is the robot's ability to respect each pedestrian's personal space, which is a region of space around humans that they actively try to maintain and into which other people cannot enter without causing some level of discomfort [6]. An example of human personal space is shown in Fig. 1.1.

- ***Naturalness***
  This feature represents the robot's ability to mimic nature and human-like motion by determining trajectories similar to those generated by pedestrians. Ensuring a natural or human-like motion increases the robot's behavior "readability" and "reliability" during navigation.
  Humans, being social entities, unconsciously interpret the movements of other pedestrians (including the robot) to anticipate their future movements and perform their actions accordingly [6]. This behavior is related to the concept of "*anthropomorphism*", which refers to the tendency of humans to attribute human qualities, such as consciousness, beliefs, and intentions to objects [7]. If the robot is able to move like a human being, it will be perceived by humans as an entity similar to them due to the process of *anthropomorphizing*. Consequently, this will result in increased social acceptability, thus ensuring easier integration between robots and humans.
  In general, naturalness of motion can be achieved by acting on low-level parameters of the robot's behavior, such as the dynamics, shapes and velocities of the motion. In fact, one of the main aspects that must be considered to ensure naturalness is the smoothness of the trajectory, from both geometric and velocity profile perspective [6].

- ***Sociability***

It is the ability to adhere to explicit high-level cultural conventions. This means that the robot must be able to use known socio-cultural norms of human behavior to take high-level human-like decisions [6].
Examples of this behavior include walking on the right side of a hallway or asking permission to pass when passing through another individual's personal area. These behaviors generally represent socially imposed constraints and can be used to solve conflicts or establish social order in navigation.
However, sociability is an extremely complex characteristic to ensure, as it also depends on the target culture and the context in which the robot is deployed.

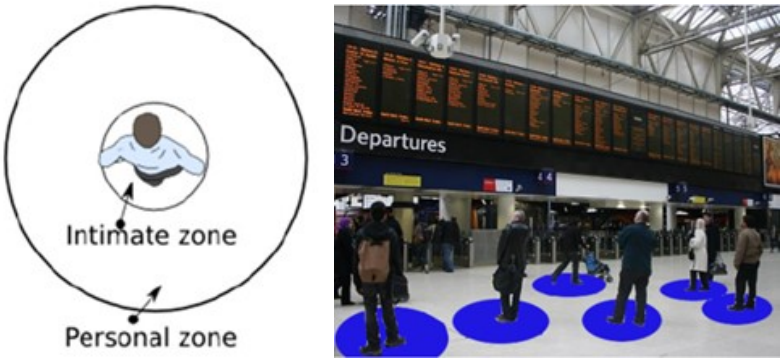


Figure 1.1: Personal space respected by humans during navigation.

Therefore, we can summarize the basic requirements that socially-aware navigation should have [6]:

- *respect personal spaces;*
- *human-like trajectory;*
- *physical safe motion;*
- *interaction awareness and mutual avoidance;*
- *avoid movements that can be causes of discomfort or negative emotions;*
- *reduce velocity when approaching a human;*
- *avoid culturally inappropriate behaviors;*

To ensure these characteristics, it is essential to develop a predictive model of human motion. By integrating such a model into the navigation framework, the robot will be able to anticipate the future trajectories of surrounding pedestrians and determine its actions accordingly. In this way, the robot will be able to perform

its task efficiently without causing discomfort to the human.

The next section describes the human motion model used in this thesis and the considerations made to ensure the characteristics of socially aware navigation described above.
This thesis proposes a human motion prediction model which combines Game Theory and the Social Force Model (SFM). To enhance readability, since the model is used as the basis for the robot's navigation system, we will use the term "agents" or "players" to refer to any entity capable of moving in the environment, both pedestrians and the robot. The terminology related to game theory and the operation of the SFM will be explained in detail in Chapter 3.

1.2 Our approach

Game theory is a mathematical tool for modeling situations in which players make interdependent decisions that may influence those of other players [8]. In fact, through the game resolution it is possible to determine what are the optimal decisions for each player while also considering the choices of all the other players. Since the robot's decision can influence the humans ones, it is necessary to make multiple predictions for each alternative decision that the robot can make in order to ensure a reliable estimation of the future human motion.
However, this approach is more complex to describe and implement [6]. For this reason, most state-of-the-art predictive approaches first perform human motion prediction individually. Then, based on that prediction, they establish the robot's motion, as if there is no mutual influence between the two [6].
The use of game theory allows overcome this limitation since the robot is seen as a rational agent whose decisions also influence those of the pedestrians. This ensures the integration of the concept of interaction-awareness within the robot's navigation system, i.e., the mutual influence between humans and robots [7], which is an essential feature to ensure the social acceptability of the robot, as described above.

In the proposed predictive model, the decision-making process during navigation is modeled through the concept of a non-cooperative game, where all players are considered as rational agents. This means that they will try to minimize their own cost without cooperating with other players.
In a game, each agent needs a set of possible actions. The Social Force Model is characterized by a set of parameters that described the behaviours and the intensities of pedestrians interactions. Different sets of parameters result in different type of trajectories.

In this context, a fixed number of SFM parameter sets are used to generate different types of trajectories. Such trajectories are used as possible actions of the various agents.
For each trajectory a cost is associated, determined by means of a cost function. Such a cost function is characterized by three different terms: the first term, tends to minimize the distance from the final goal; the second term, tends to guarantee a certain smoothness of the trajectory (avoiding movements that are too weird or sudden); the third term, tends to encourage the maximization of the distance between pedestrians.
The solution of the game is established by achieving Nash equilibrium, which ensures optimal action for each player.

Furthermore, to ensure a higher level of human-likeness for the trajectory of the robot, it is necessary to apply game theory to trajectories that are as realistic as possible. This necessarily results in the definition of sets of Social Force Model parameters that allow for the generation of such types of trajectories.
However, the high sensitivity of the model to the values assumed by the parameters results in high variability in the behaviors and final trajectories. Therefore, manually establishing such parameters for each player is significantly complex.
To overcome this limitation, an evolutionary algorithm has been used to extract the best parameters of the SFM approximating the trajectories contained in Thör [9], a public dataset of real human trajectories. In this context, a Differential Evolution (DE), a very simple and efficient optimization algorithm, has been employed.

The parameters of the SFM represent the way of modeling the behavior of the individual person. Such behavior tends to vary and adapt according to how the navigation scenario evolves. Thus, we need to estimate time-varying sets on the basis of specific features that describe the changing scenario.
Nevertheless, the excessive time required to perform parameter estimation makes DE unsuitable for real-time applications. Therefore, the operation of Differential Evolution has been approximated by the use of a neural network for both the robot and pedestrians.
To ensure the generation of realistic and variable parameters, according to the state of the agent in the environments, the neural network has been trained in a supervised manner by exploiting a labelled dataset.
Such dataset has been created by associating the best parameters, obtained through the DE algorithm, with specific features that describe how the surrounding environment evolves along the considered human trajectory (such as minimum distances from obstacles or other pedestrians).

Therefore, the main objectives of this thesis are:

1. develop a social navigation algorithm for mobile robots that predicts human motions by exploiting a model that combines Game Theory and Social Force Model;

2. enhance social acceptability through real-time estimation of realistic SFM parameters. Such estimation is performed by a neural network that mimics the behaviour of the Differential Evolution algorithm;

1.3 Thesis organization

The rest of the thesis is organized as follows.

Chapter 2 reviews social navigation algorithms and Differential Evolution's applications in the state-of-the-art. Chapter 3 presents the theoretical concepts necessary to understand the two models underlying the navigation algorithm used in this thesis, namely SFM and game theory. Chapter 4 briefly describes the working principles of the *Game-theoretic Social Force Model* focusing more on the *Differential Evolution*. Chapters 5 and 6 introduce the main hardware characteristics of the mobile robot employed in the experimental session and the ROS framework used for the implementation of the navigation algorithm, respectively. Section 7 outlines the methodologies employed for conducting the experimental campaign and the state-of-the-art metrics used for quantitative evaluating the tested algorithms. In Chapter 8 the experimental results are discussed. Finally, Chapter 9 presents the conclusions of this dissertation and outlines future research directions.

Chapter 2

Related Works

In this chapter the basic concepts of mobile robot navigation (such as global and local planners) are provided. Then, an analysis of the main types of algorithms used in state-of-the-art social navigation are presented. The chapter concludes with an introduction to the state-of-the-art of Differential Evolution algorithm, the optimization approach chosen for parameters estimation.

2.1 Autonomous navigation

In general, it is possible to identify four basic phases that characterize the autonomous navigation of a mobile robot in the environment:

- ***Mapping:*** it is the method by which the robot constructs the map of the environment in which it operate and move;
- ***Localization:*** it is the process by which the robot identifies its position within the map;
- ***Trajectory planning:*** it is the phase in which the robot computes a sequence of feasible poses (positions and velocities) to reach the final position;
- ***Actuator control:*** it is the stage where the robot controller computes the precise commands that will be sent to the actuators in order to guide the robot along the path obtained from the planning phase.

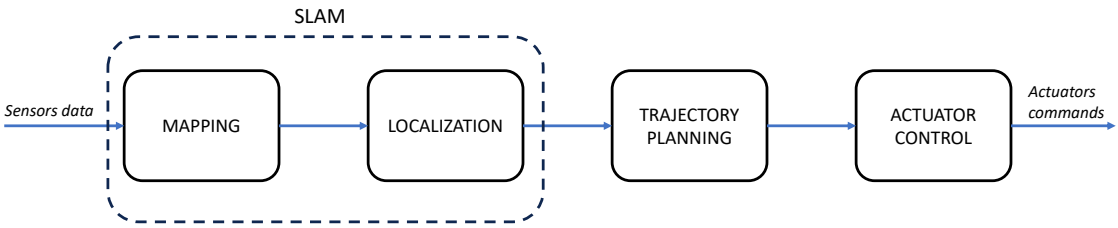


Figure 2.1: Main phases of autonomous navigation.

In order to develop human-awareness algorithms, it is necessary to focus on how the robot's trajectories are generated.

Typically, the trajectory planning process can be divided into two sub-phases:

- *Global planning*: it has the task of providing the robot with an optimal and collision free path that allows it to move freely from the initial position to the goal position taking into account only static obstacles;
- *Local planning*: it is responsible for making local changes to the path to be followed based on the information received from the sensors;

This division emphasizes the importance of focusing on the development of local planning algorithms since they are the ones that effectively avoid dynamic and unpredictable obstacles such as pedestrians.
Such algorithms must therefore be able to integrate information about pedestrians, their behaviour and social conventions within trajectory computations to ensure the natural, safety and social acceptability of the robot's navigation.

In the following sections, the state-of-the-art of Social-aware navigation algorithms and the state-of-the-art of Differential Evolution algorithm are presented.

2.2 State-of-the-art Social navigation

Currently, in the field of autonomous navigation, there are numerous classifications of the various algorithms developed over the years. Some examples of this classifications are presented in some reviews such as: [10] or [11].
Although some of the classes proposed in these works also include approaches used for navigation in human-shared environment, they do not represent an effective classification of the algorithms required for social navigation.

In general, it is challenging to define a clear and explanatory classification for these types of algorithms due to their many features. For this reason, we propose a classification based on two criteria defined as follows:

- ***Type of motion planning:*** it takes into account how robot trajectories are computed;
- ***Integration of a human-motion prediction model:*** it considers whether the algorithm uses a predictive model to determine the future movements of humans and integrate them into the trajectories computation;

Using these two criteria, it is possible to classify social navigation algorithms in four categories [12][13][14]: *model-based*, *learning strategies*, *reactive* and *predictive*. This classification is described in fig. 2.2.

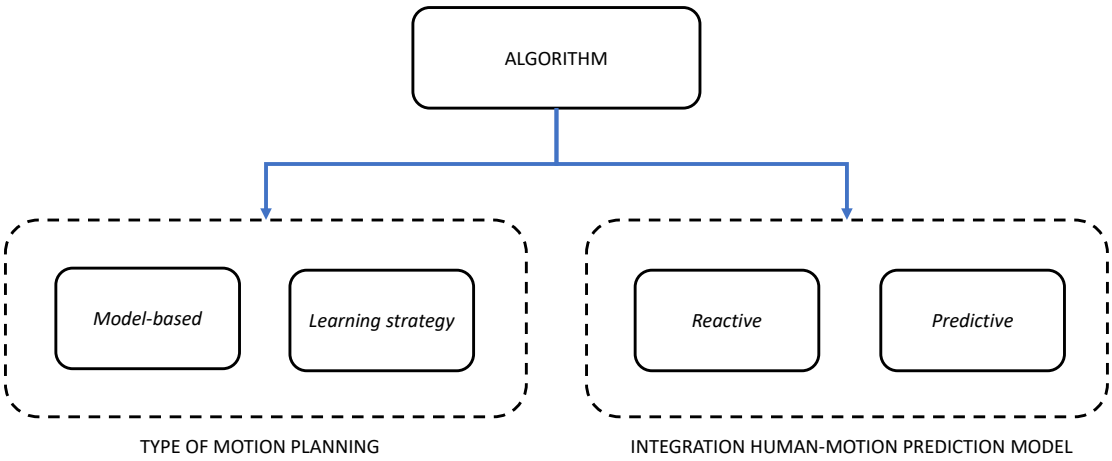


Figure 2.2: Classification of Social navigation algorithms: the dotted line denotes the algorithm's classification based on a particular criteria. The continuous line inside the classification represents the category related to the same criteria.

In general, categories within the same class are *typically* complementary. Depending on the criterion considered, an algorithm may belong to different classification categories at the same time. For example, an approach may be considered as model-based if analysed from the point of view of motion planning, but at the same time also reactive or predictive with respect to the use of a human-motion predictive model.
However, especially in recent years, several studies attempt to merge model-based algorithms with learning strategies to develop hybrid methods that exploit the advantages of both categories and share many of their characteristics.

In the following sections, the 4 categories presented in fig. 2.2 are described individually, analyzing both their benefits and drawbacks.

2.2.1 Model-based algorithms

As the name suggests, these algorithms try to describe the dynamics of navigation by means of a mathematical model, which are typically derived from geometric relationships that characterize the motion [13] or through physics-based equations. By using these models, it is possible to explicitly define different types of interactions between the individual agent and the environment that occur during navigation (such as agent-agent or agent-obstacle interactions). In this way, the pedestrian or the robot will be able to establish the action to be performed.
In general, these interactions and resulting actions are individually regulated by a set of parameters, which must be adjusted according to the navigation context.
This approach not only ensures great flexibility and generalization but also enables the model to incorporate new interactions and behaviours that were not initially considered. As a result, the descriptive capabilities of the model increases.
Furthermore, model-based algorithms are easier to implement than learning strategies (which will be described below) because they do not require a training phase or advanced knowledge of robotics.
Such features make model-based approaches the most widely used implementation strategies for developing human-aware navigation systems [14].
One of the most famous algorithms in this category is the Social Force Model (SFM) [15]. This approach models interactions among agents by means of attractive and repulsive forces (more details in chapter 3) which cause the consequent movement of the robot according to the laws of classical mechanics.
Given the significant potential of the model, subsequent studies have attempted to extend it and mitigate its limitations in various ways. Some examples are [16] where Zanlungo et al. try to model also the social behaviour of people moving in groups or [17] in which the authors introduce a set of non-holonomic kinematics constraints in order to remove the intrinsic isotropic behaviour and model the tendency of humans to move along their gaze direction.
Another model-based algorithm is the Velocity Obstacle (VO) developed by Fiorini and Shiller [18] which uses the robot's acceleration constraints and the knowledge of moving obstacles' velocities to establish a dynamic feasible speed that allows the robot to avoid collisions. The main problem is the fact that it does not take into account the reciprocal mutual avoidance typical of human behaviour during navigation.
This limitations was solved by Berg et al. by introducing the reactive ability also for other pedestrians involved in the navigation via the Reciprocal Velocity Obstacle (RVO) [19]. Later, same authors extended the concept to n-agents through ORCA

(Optimal Reciprocal Collision Avoidance) [20].

Although these algorithms often have good computational complexity, one critical issue is their high sensitivity to the model's parameters or constraints that govern interactions.
In order to achieve the desired behaviour, a precise calibration is required according to the specific scenario in which the robot operates [14].
Furthermore, current state-of-the-art models are not able to describe the more complex mechanisms of navigation, especially in the case of large crowds. This may lead to the generation of oscillatory paths [13].

2.2.2 Learning strategies

Nowadays, the constant technological progress in the field of Deep Learning (DL) and the introduction of Deep Neural Networks (DNN) are making learning strategies increasingly popular. This is due to the possibility of approximating extremely complex behaviours (such as navigation) in a relatively simple manner. This feature enables the development of new approaches with the aim of making the autonomous movement of robots more comfortable for humans [12].
By using different machine learning models, it is possible to exploit real data to train DNNs allowing the robot to directly learn the underlying dynamics of the navigation context. In this way, it is not necessary to explicitly model individual interactions as in the previous category. This approach allows the robot to exhibit extremely natural behaviour, outperforming many state-of-the-art model-based algorithms.
Additionally, in terms of effort, the ability to perform the training phase off-line guarantees a low on-line computational intensity, since most of the calculations are executed before the actual navigation.

Basically, it is possible to categorize the main state-of-the-art machine learning techniques for social navigation into three groups [12]:

- ***Supervised learning***
  In general, pedestrians show good performance when moving in a crowd. As a result, extensive research has attempted to emulate these skills by means of supervised learning [12]. This subclass of learning strategy is characterized by algorithms that exploit various datasets consisting of real trajectories in order to learn and subsequently replicate the same social behaviour exhibited by pedestrians (*Behaviour Cloning* [21]).
  An example is the work done by Xie et al. [22]. In their study, the authors propose the use of a Deep Neural Network trained through expert demonstrations. Taking as input an early fusion of "short history" data from LiDAR and

kinematic data concerning nearby pedestrians, the network ensures a control policy that allows the robot to navigate in a socially acceptable manner.
In [23], Shing et al. divide the robot's workspace into five segments, each of 30 degrees, and exploit a Multilayer Perceptron (MLP) to determine the best direction in which the robot should move. Once trained, the MLP is able to generate a collision-free trajectory that allows the robot to reach the goal by moving within a dynamic environment.
Other studies aim to differentiate the behaviour of the robot according to the scenario in which it will operate. To provide the robot with the ability to adapt to different contexts and change navigation strategy, Banisetty et al. [24] propose a combined learning approach using Convolutional Neural Network (CNN) and Support Vector Machine (SVM) technique to develop a context-classifier and combine it with non-linear optimization based local planners.

- ***Deep reinforcement learning***
  Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) algorithms take inspiration from the natural world to try to replicate the way of learning of many animals, including humans.
  The navigation algorithms based on these two strategies, in particular DRL, can enable the robot to have an autonomous learning capability and to acquire decision-making skills [25]. They employ a reward value function, as a feedback mechanism, to evaluate the interaction of the agent with the environment.
  The training phase involves a continuous process of trial and error in which the robot learns the sequence of actions that result in the highest cumulative reward. Therefore, in contrast to supervised learning where the robot learns the correct actions through real data, this approach involves instructing the robot to recognize incorrect actions by means of a reward.
  One of the pioneers in this field is Chen et al. [26]. The authors develop one of the first decentralized multiagent collision avoidance algorithms based on DRL, namely CADRL. Although the algorithm successfully prevented collisions with pedestrians, the robot's learned cooperative behaviours did not conform to the basic social rules of human navigation. The same authors extended their previous algorithm by integrating socially aware behaviours [13].
  A further remarkable algorithm is the SARL [27] where the authors focus more on teaching the robot to navigate in crowds by trying to predict possible human-human and human-robot interactions. However, the robot's learnt navigation policy was found to be limited by the distance associated with the training process, leading to a reduction in navigation performance for goals far away from the robot's position. The problem was later solved by Li

and Xu [28] through the integration of a dynamic local goal setting mechanism.

- ***Inverse reinforcement learning***
  The implementation of DRL techniques requires a hand-crafted reward value function. This needs an advanced knowledge of robotics, sensing and motion planning [12]. Moreover, in very complex scenarios, the design of such a reward value function can be an even more difficult challenge.
  For these reasons, a possible solution is to exploit real datasets to enable the robot to do this task autonomously. Through the use of optimal expert demonstrations, Inverse Reinforcement Learning (IRL) algorithms will attempt to extract the underlying reward structure so that it can subsequently be used by Deep Reinforcement Learning algorithms to learn the appropriate policy for the social-aware navigation [29].
  An example of such an approach is [30], in which the authors propose the use of inverse reinforcement learning to teach the robot where and how to approach a person in an unstructured open area. This information is then used to generate a path that allows social norms to be respected during navigation. However, the main problem lies in the fact that the robot can only approach isolated people, making this algorithm difficult to use in medium or large crowds.
  In another work, exploiting the potential of Generative Adversarial Networks (GAN), Tai et al. [21] propose the use of a Generative Adversarial Imitation Learning (GAIL) strategy to skip the process of estimating the reward value function and directly force the generation of state-action pairs matching that from the expert demonstrations, which are subsequently used to teach the robot to navigate in dynamic environments through reinforcement learning.
  More recently, Sun et al. [31] focused on the fact that most of IRL approaches do not consider the kinematic constraints of the robot. For this reason, they developed an Inverse Reinforcement Learning-based planner capable of generating paths that respect human social rules and directly integrate the non-holonomic constraints of the robot.

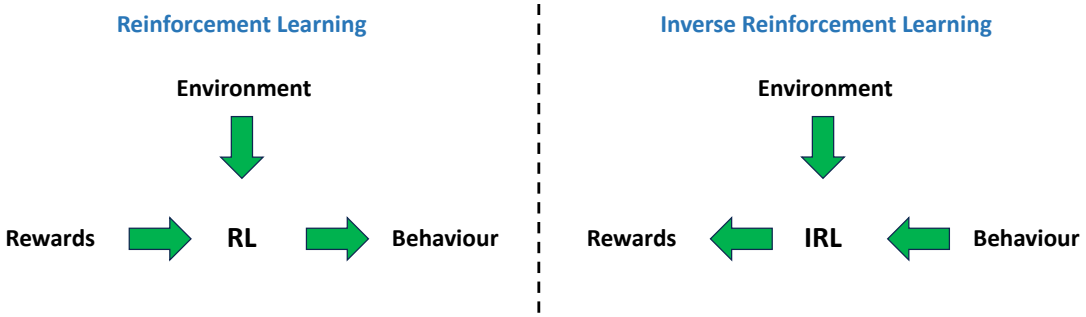


Figure 2.3: Schematic comparison between Reinforcement Learning and Inverse Reinforcement Learning [32].

While learning strategies offer impressive descriptive capabilities and excellent performance, they are also accompanied by significant drawbacks. Indeed, for their training, such models require a vast amount of data (supervised learning) or trials (DRL and IRL), resulting in a very time-consuming process.
Furthermore, this data must be able to accurately represent the specific navigation scenario in which the robot will operate in order to identify all possible situations that could occur. This feature renders these algorithms highly non-generalizable. Moreover, the limited availability of different datasets (especially the annotated ones required for supervised learning) makes the training of such models difficult, since the similarity between the data can lead to the risk of overfitting DNN parameters [33]. For this reason, some model-based algorithms can outperform these approaches in specific cases.
Finally, the use of DNN and different machine learning strategies is related to a phenomenon called "lack of explainability". When a system, such as a mobile robot, makes decisions autonomously, it is crucial to establish the process that led to that specific decision. Such a characteristic is called “explainability” [33]. Due to the implicit nature of DNNs, it is not possible to derive an a priori model to establish this decision-making mechanism. This therefore makes explainability extremely difficult or even impossible in some cases.

2.2.3 Reactive algorithms

Reactive algorithms are a subset of algorithms that do not rely on any human prediction model. For this reason, they only modify the "next step action" (action they will perform at the next time step) of the local path, completely ignoring the future states of other agents [12]. Then, when the algorithm detects an obstacle, it "*reacts*" by changing the local path and establishing new speed commands to avoid collision with the object.

For the reasons explained at the beginning of this section, many algorithms described as model-based (such as SFM, VO and ORCA) can also be considered as reactive algorithms. Over the years, many works have attempted to extend such approaches without necessarily introducing a predictive model of human behaviour. An example is [34] where the authors modify the SFM to represent the social space of pedestrians by a Gaussian distribution. In [35] Chen et al. merge the VO and the Rapidly Random Tree* (RRT*) algorithm with the aim of reducing the uncertainty of robot trajectories.
Another of the well-known models based on reactive maneuvers is the Dynamic Window Approach (DWA) [36]. In this case, the algorithm uses obstacles and pedestrians information from the environment and integrates it with the dynamic and kinematic constraints of the robot to compute a "velocity space", which consists of combinations of linear and angular velocity that do not result in any collisions and are actually achievable in the next time interval. Such combinations are used to generate a set of feasible trajectories, which are subsequently evaluated using a cost function. The robot will then follow the one with the lowest cost. The process is repeated at each time interval to take into account changes in pedestrian speed. However, the algorithm has a limited responsiveness, making it only suitable for scenarios with a very low number of pedestrians.
In recent years, one of the most common approaches is to use learning strategies due to the great advantages they offer. In [37], authors consider navigation as a classification task and use expert trajectory demonstrations to train Fully Convolutional Neural Networks. In this way, the networks can classify the generated path as feasible or not, taking into account obstacles and pedestrians in the robot's vicinity. However, using only expert demonstrations and not also the actual trajectories of pedestrians, results in an inability to incorporate information about social interactions and generalize navigation to different scenarios.
In another work, Gil et al. [38] model navigation by means of the SFM and exploit a trained Neural Network to extract the acceleration of the robot from the computed interaction forces.
Although these types of algorithms are characterized by high computation efficiency and can be well generalized [12], the inability to predict future human behaviour by means of a incorporated human-motion model could lead to the generation of unnatural or even unsafe trajectories, especially in contexts where the robot moves at speed similar to those of pedestrians [13]. Nevertheless, even when dealing with collision-free trajectories, the absence of a human-predictive model could lead to trajectories that fail to respect the individual's interpersonal space, resulting in discomfort for human beings [12].
Overall, in these types of algorithms pedestrians are commonly considered as individual dynamic passive objects rather than multiple cooperative agents, which can result in problematic situations such as the well-known "freezing robot problem".

This occurs when a robot's navigation algorithm cannot identify free and safe paths due to high crowd density, resulting in freezing of robot motion [39]. Therefore, these issues make these algorithms unsuitable for an efficient implementation in social navigation contexts.

2.2.4 Predictive algorithms

Unlike reactive algorithms, predictive algorithms exploit human-motion model to predict the future trajectories of agents involved in the navigation. They will then attempt to determine the action to be taken based on the future states of the detected pedestrians [12]. This integration ensures a significant improvement in the effectiveness of navigation in dynamic and crowded environments [40], thus, preventing potential problematic situations such as the already mentioned "freezing robot problem".
Indeed, the usage of a predictive model of human behaviour enables the integration of more social information into the navigation algorithm. Pedestrians are no longer seen as mere passive dynamic objects but as active agents capable of making autonomous decisions and cooperating with the robot to ensure mutual avoidance maneuvers. These characteristics make predictive algorithms particularly suitable for application in human-aware navigation scenarios, as they will try to respect human constraints and social rules [6].
Recent works have focused on the use of specific types of Recurrent Neural Networks (RNN), such as Long-Short Term Memory (LSTM) networks. They are neural networks capable of learning general human movement sequences.
In [41], Alahi et al. associate a Long-Short Term Memory (LSTM) network with each agents, so that specific motion properties can be identified and used to make predictions about pedestrian future positions. Since such LSTMs cannot capture the interaction between different individuals, the predictions of future states are then processed using a social pooling layer. However, this will result in a kind of "average trajectory" which can lead to incorrect predictions in some cases. Kretzschmar et al. [42] model the cooperative behaviour of agents during navigation as a mixture distribution that captures both discrete decisions, like going left or right, and the stochastic behaviour of humans, which determines the natural variance of pedestrian trajectories. By using IRL, they attempt to extract parameters values of such distributions that best match the observed expert demonstrations. This allows the robot to learn a model of human cooperative navigation behaviour that it can use to predict pedestrians movements.
Another research area that has been gaining momentum in recent years is the use of Game Theory to model human behaviour during navigation (more details in 3). In this context, the studies conducted by Turnwald et al. are particularly important. In [8] the authors modelled human navigation as a non-cooperative game,

demonstrating through real experiments how pedestrians choose trajectories that ensure the achievement of a Nash equilibrium. In the same study, five different types of cost functions are also analysed. Authors established that the best performances are guaranteed by the cost function related to the path length. Later, in [7], same considerations are employed to develop a trajectory planning algorithm able to explicitly model the human-like decision-making process by means of Game Theory. Based on similar ideas, Galati et al. [43] propose a game-theoretical social-aware navigation algorithm based on an improved cost function. This improvement is guaranteed by the integration of social information and features typically not considered in other types of approaches (including those based on game theory) such as group recognition, sequential decision making and human-obstacle interaction. This approach results in an algorithm able to generate safe and socially acceptable trajectories.
The main disadvantage of this type of algorithms is the high computational intensity required to predict the future trajectories of pedestrians, which rise significantly with increasing crowd density.
In addition, predictions of agents' future states obtained from the model may be affected by uncertainty due to the stochastic nature of human behaviour. This uncertainty tends to increase with the time horizon, resulting in significant inaccuracies in long-term predictions. Such behaviour implies that the robot must frequently recompute the various possible trajectories of pedestrians during navigation, further increasing its computational effort.

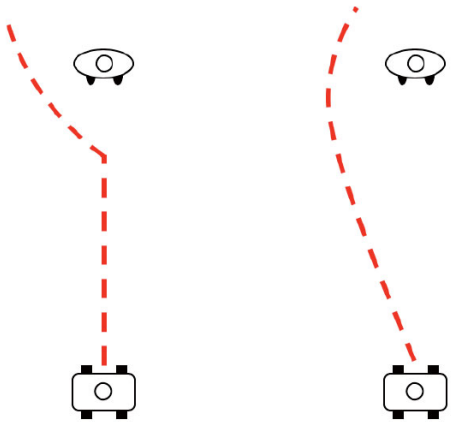


Figure 2.4: Comparison of trajectories generated by the two types of planner (from [12]): (a) Reactive: when a person appears in its way, the robot modify the local path to change direction; (b) Predictive planner: by exploting a human-motion model, the robot first predicts the future states of the person and then it computes its path considering the mutual avoidance.

2.3 State-of-the-art Differential Evolution

This thesis proposes the use of a Differential Evolution algorithm to estimate the best parameters of the Social Force model.
Differential evolution (DE) is a very powerful stochastic algorithm[1] developed by Storm and Price in 1995 [44] and used to perform global optimization of non-linear problems in a continuous search space.
In general, the DE algorithm follows the same computational procedure as a typical Evolutionary Algorithm (EA) [45]. However, differently from most EAs where new individuals are obtained through the use of different probability distributions, the DE explores the solution space by means of a scaled difference between randomly selected members of the current population. Then, through the processes of crossover and selection, it chooses the chromosomes that will be part of the next generation.

Therefore, the DE performs three main steps, which will be repeated in a cyclic manner until the last generation is reached. A schematic representation of how the algorithm works is described in the fig. 2.5:

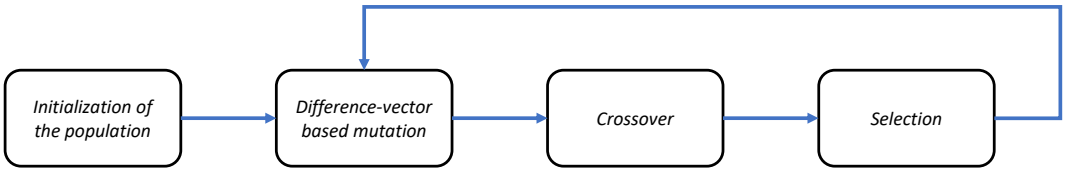


Figure 2.5: Main steps of the Differential Evolution algorithm

A detailed description of these steps will be provided in Chapter 4 in order to give the reader the necessary information to understand how the DE algorithm enables the estimation of real SFM parameters.

However, some of the most important properties that characterise the Differential Evolution algorithm are highlighted below [45]:

- ***No coding of solutions:*** in many EAs such as the Genetic Algorithm (GA), it is necessary to encode the real values of the parameters that make

[1]A stochastic algorithm is a type of optimization algorithm that attempts to find an optimal solution or a good approximation, by using random process, usually determined by probability distributions. They differ from deterministic algorithms, where the solution is instead obtained through steps that do not involve probabilistic process.

up the chromosomes as bit strings. This leads to increased complexity and computational effort;

- ***Easy implementation:*** the algorithm can be implemented in a few lines of code, without the need for advanced programming knowledge. This characteristic makes it suitable for application to various research fields;

- ***Excellent performance:*** through its operation, the DE ensures better performance in terms of accuracy, convergence speed and robustness;

- ***Reduced number of control parameters:*** the performance of the algorithm depends entirely on the calibration of three parameters, namely the number of chromosomes (NP) in the population, the mutation factor (F) and the crossover rate (CR). A detailed explanation of these parameters will be presented in Chapter 4;

Due to these characteristics, the DE has become one of the most popular and widely used optimization algorithms, both in its classical form and in its later variants developed over time [46][47][48].

One of the many fields in which DE has great potential is the systems identification. In [49], Cheng et al. approximate complex dynamic systems by means of linear models and exploit DE to estimate the transfer function parameters that guarantee the most similar time-response. A similar approach is used by Yousefi et al. [50]. In their study, the authors approximate a non-linear system (electro-hydraulic servo system with flexible load) using two different transfer functions according to the operating frequency range. In particular, they use a linear model for low frequencies and a second-order system for higher frequencies. They then exploit the DE to derive the parameters of these models using non-linear constraint functions and constrained parameter values.
In [51], Tang et al. approximate a civil building as a model consisting of n mass-spring-damping systems. Then, they estimate the actual parameters of an 8 DOF and a 20 DOF system using both DE and PSO (Particle Swarm Optimization algorithm), highlighting the better performance of the former.
However, DE is also applied in the field of automatic controls robotics. In [52], Menon et al. apply a DE algorithm for clearance of nonlinear flight control laws of high-performance aircafts. In general, high-performance aircraft are designed to be naturally unstable. Control laws are therefore required to stabilize them

during flight. Since the safety of the aircraft depends on the correct operation of these control laws, it is necessary to verify their proper functioning both during flight and malfunctions ("flight clearance problem"). Therefore, by establishing mathematical evaluation criteria, it is necessary to derive the possible combinations of aircraft configurations that determine the worst values. In their study, the authors model this process as an optimization problem. They apply DE to try to derive the combinations of aircraft configurations that result in the worst values of these criteria and they then compare its performance with GA (Genetic Algorithm). The DE guarantees better accuracy and speed, as it reduces the computational overhead by 31%.
Moreno et al. [53] propose a new solution to solve the global localization problem by means of a nonlinear evolutive filter denoted as Evolutive Localization Filter (ELF). Using the DE, the filter stochastically searches within the state space for the robot pose estimate that best matches the odometry and sensor measurements.
In [54], Aidin et al. develop an algorithm for planning time-optimal trajectories. Basically, the navigation algorithm generates a road-map (a map that presents the shortest paths between the initial position and the various possible final positions), taking kinematic constraints into account. Then, they exploit the DE to determine which is the optimal trajectory to follow.
In [55], the authors propose the use of parallel and distributed differential evolution algorithms to ensure cooperative navigation of n-robots. The DE is then used to determine the best next positions that do not result in collisions with other robots.
Another field in which DE finds application is the training of neural networks. In [56], authors succeed in achieving better convergence in the optimization of neural network weights by using a hybrid algorithm. This algorithm is obtained by combining the DE with the Levenberg Marquardt algorithm (LMA).
A further work is [57] where authors use a neural network to perform the detection of malignant regions in colonoscopy video sequences. During colonoscopy, in fact, the probe finds itself in time-varying conditions and environments characterized by variations in shadings, shadows, lighting and reflections. Such problems can lead to a loss of accuracy of the neural network and its ability to recognize tumors. Therefore, the authors propose the use of online backpropagation algorithm to train the NN and the application of DE to derive the best learning rate. This makes it possible to train the neural network according to the conditions under which the probe is operating.

In all of the applications described above, DE proves to be an excellent solution, showing great adaptability to different context and scenarios.
We will therefore use this algorithm in our case to try to obtain socially acceptable trajectories.

Chapter 3

Background Social Force Model and Game Theory

3.1 Social Force Model

3.1.1 Overview

The Social Force Model (SFM) is a mathematical model introduced in 1995 by Dirk Helbling and Péter Molnár [15]. Due to its flexibility and low computational complexity, it is one of the most widely used state-of-the-art models to describe the dynamics governing the movement of pedestrians in different crowd densities. In fact, it has been applied in various real or simulated contexts, such as museums [17] and urban environments [58], but also in hazardous situations and evacuations, for example earthquakes [59] or terrorist attacks [60].
The operating principle of the SFM is based on the assumption that a pedestrian can be represented as a particle whose movement is caused by the effect of specific types of forces, called *"social forces"*. These forces are not to be intended as actual physical phenomena, but rather as fictitious forces generated by the individual's internal motivations and their interaction with other elements of the environment in which they are moving, such as other pedestrians or obstacles. It's the combined effect of these interactions that causes the actual movement of the pedestrian in a specific direction.

The notation of the model, types of forces and pedestrian's dynamics are described in the following sections.

Notation

This section is intended to provide the reader with some notational conventions that may be useful in understanding the rest of the thesis. The sets of natural, real, real nonnegative and strictly positive real numbers will be denoted by $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ respectively. We use roman font to denote scalar quantities ($x \in \mathbb{R}$) and bold font to denote vectors in the plane ($\mathbf{x} \in \mathbb{R}^2$). Given a vector $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$, $\|\mathbf{x}\| = \sqrt{{x_1}^2 + {x_2}^2}$ indicates the Euclidean norm.

3.1.2 Model formalization and types of forces

The model consider a set $\mathcal{N} = \{1, ..., n\}$ of $n \in \mathbb{N}$ pedestrians moving in a continuous planar space $X \in \mathbb{R}^2$. Each pedestrian $i \in \mathcal{N}$ will be characterized by a goal $\mathbf{p}_i^{\mathbf{goal}} \in \mathbb{R}^2$, which represents the final position that the pedestrian wants to reach, and a state. The latter is composed by two vectors: the position $\mathbf{p}_i(t) \in \mathbb{R}^2$ and the velocity $\mathbf{v}_i(t) \in \mathbb{R}^2$ vector at the time instant $t \in \mathbb{R}_{\geq 0}$.
In the original version, the SFM includes four types of social forces that govern the motion of each pedestrian:

- ***Attractive force of the goal:*** the pedestrian in position $\mathbf{p}_i(t)$ is attracted to the goal $\mathbf{p}_i^{\mathbf{goal}}$ by a force $\mathbf{F}_i^{goal}(t)$. This force can be expressed as:

$$\mathbf{F}_i^{goal}(t) = \frac{v_i^d \hat{\mathbf{e}}_i(t) - \mathbf{v}_i(t)}{\alpha_i} \tag{3.1}$$

  where $\hat{\mathbf{e}}_\mathbf{i}(t) = \frac{\mathbf{p}_i^{\mathbf{goal}} - \mathbf{p}_i(t)}{\|\mathbf{p}_i^{\mathbf{goal}} - \mathbf{p}_i(t)\|}$ is the desired direction of the pedestrian, $\alpha_i \in \mathbb{R}_{>0}$ is a parameter called "relaxation time" that indicates how decisively the pedestrian moves towards the goal and v_i^d is the desired speed (i.e. the speed that pedestrian want to achieve).

- ***Repulsive forces from other pedestrians:*** in real-life navigation, approaching an unknown pedestrian too closely causes a reduction of the perceived comfort for both individuals involved. As a result, each pedestrian reacts by trying to move away from the other. The Social Force Model (SFM) describes this behaviour as the effect of a repulsive force related to the implicit interaction that occurs between two pedestrians. Precisely, such force works to prevent excessive approaches between the various agents involved.

  From a mathematical perspective, it is possible to express the repulsive force exerted by the pedestrian $j \in \mathcal{N} \setminus \{i\}$ on the pedestrian i as an effect of a repulsive potential that can be described using a decreasing monotonic function dependent on the vector radius connecting the two individuals $(\mathbf{p}_i(t) - \mathbf{p}_j(t))$. The definition of a potential function enables us to establish a corresponding force expression through the application of the mathematical gradient operator:

$$\mathbf{F}_{i,j}^{rep}(t) = -\nabla \mathbf{V}(\mathbf{p}_i(t) - \mathbf{p}_j(t)) \tag{3.2}$$

Although the original model developed by Helbling et al. assumes a function where elliptical equipotential lines are directed towards the pedestrian's motion, alternative forms have been proposed over time [61].

Considering the limited size of the room used in the experimental session and the expected number of pedestrians for each test, the crowd density in which the algorithms has been tested can be considered high. For this reason, in this thesis we decided to exploit a function that determines circular equipotential lines, particularly useful in cases of high-density conditions [61].
Given these assumptions, the equation of the repulsive force $\mathbf{F}_{i,j}^{rep}(t)$ between two pedestrians at distance $d_{i,j}(t) = \|\mathbf{p}_i(t) - \mathbf{p}_j(t)\|$ can be expressed as:

$$\mathbf{F}_{i,j}^{rep}(t) = A_i \exp\left\{\frac{r_i + r_j - d_{i,j}(t)}{B_i}\right\} F_{i,j}^{fov}(t)\hat{\mathbf{n}}_{i,j}(t) \tag{3.3}$$

where the terms $A_i, B_i \in \mathbb{R}_{>0}$ are constant parameters that regulate the strength and range of the repulsive force, respectively, $\hat{\mathbf{n}}_{i,j}(t) = \frac{\mathbf{p}_i(t) - \mathbf{p}_j(t)}{\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\|}$ is the unit vector of the direction between the two pedestrians and $r_i, r_j \in \mathbb{R}_{>0}$ are the radii of the personal spaces of pedestrian i and j. In addition, the term $F_{i,j}^{fov}(t) \in [0,1]$ represents a time-varying anisotropic factor that capture the effect of the limited field of view, typical of human beings. In the context of pedestrian navigation, individuals tend to focus more on objects and entities within their visual range. Consequently, the repulsive force must be adjusted by means of a quantity that is strictly dependent on the bearing angle $\gamma_{i,j}(t)$ (i.e. the angle between the actual direction of motion of agent i and the segment joining the positions of agent i and agent j) of pedestrian j measured from pedestrian i.
The scaling factor $F_{i,j}^{fov}(t)$ is defined as:

$$F_{i,j}^{fov}(t) = \lambda + (1 - \lambda)\frac{1 + \cos\gamma_{i,j}(t)}{2} \tag{3.4}$$

where $\lambda \in [0,1]$ represents the strength of the anisotropic behaviour. As can be observed, a higher lambda value will render the scaling effect insignificant, leading to a more isotropic behaviour.

- ***Repulsive forces from obstacles/walls:*** it is a repulsive force related to the proximity of the pedestrian to an obstacle or wall. The considerations for determining this force are analogous to the force between pedestrians seen above. It will then be necessary to define a monotonically decreasing

potential function as a function of the vector radius between pedestrian i and the nearest obstacle or wall. Once this potential has been defined, it will be possible to derive the force expression via the gradient.

$$\mathbf{F}_i^{obs}(t) = \exp^{\left(1-\left(\frac{d_{i,obs}(t)}{R_0}\right)\right)} F_{i,obs}^{fov}(t)\hat{\mathbf{n}}_{i,obs}(t) \tag{3.5}$$

where $d_{i,obs}(t) = \|\mathbf{p}_i(t) - \mathbf{p}_{obs}\|$ is the actual distance between the pedestrian i and the nearest obstacle and R_0 represents the minimum acceptable distance. For the same reasons discussed in the previous case, the repulsive force is scaled by a corresponding anysotropic factor $\mathbf{F}_i^{obs}(t)$ defined as:

$$F_{i,obs}^{fov}(t) = \lambda + (1-\lambda)\frac{1+\cos\gamma_{i,obs}(t)}{2} \tag{3.6}$$

where $\gamma_{i,obs}(t)$ is the bearing angle of the considered obstacle measured from pedestrian i.

- ***Attractive effects from groups or objects of interest:*** in some cases, pedestrians may be subject to "attractive effects" related to other pedestrians or individual objects of interest characterized by the position $\mathbf{p}_k(t)$. Examples include the formation of groups, or the necessity to pick up an object in a different position with respect to the final destination the pedestrian wishes to reach. Helbling et al. modelled these effects in a way that is opposite to the repulsive interaction forces seen above, i.e. by means of an attractive potential that can be expressed by a monotonic function that increases as the distance between pedestrian i and the considered point of interest k becomes greater. Therefore, as in the previous cases, by defining a suitable potential function, it is possible to derive the expression of the force by applying the gradient.

$$\mathbf{F}_i^{attr}(t) = -\nabla\mathbf{W}(\mathbf{p}_i(t) - \mathbf{p}_k(t)) \tag{3.7}$$

Although the original SFM also models this kind of interaction, the types of tests proposed in the experimental session do not include either the formation of groups (due to the limited space) or the presence of intermediate stages other than the target positions of individual pedestrians. For this reason, this type of force has not been considered in the navigation algorithm that will be proposed later.

3.1.3 Model dynamics

As expressed above, the SFM considers pedestrians as particles subject to social forces. By applying the principle of superposition, it is possible to derive the

resultant force $\mathbf{F}_i^{res}(t)$ acting on the generic pedestrian i at a given time instant t as:

$$\mathbf{F}_i^{res}(t) = \mathbf{F}_i^{goal}(t) + \sum_{j \epsilon \mathcal{N} \setminus \{i\}} \mathbf{F}_{i,j}^{rep}(t) + \mathbf{F}_i^{obs}(t) \tag{3.8}$$

Since the motion of pedestrian i can be described using the laws of Newtonian mechanics, it's possible to express its dynamics by a system of ordinary differential equations. Assuming, without loss of generality, that forces are re-scaled so that pedestrians have a unitary mass and considering the discretized case with time step Δ_t, we will have:

$$\begin{aligned} \mathbf{v}_i(t + \Delta_t) &= \mathbf{v}_i(t) + \Delta_t \mathbf{F}_i^{res}(t) \\ \mathbf{p}_i(t + \Delta_t) &= \mathbf{p}_i(t) + \Delta_t \mathbf{v}_i(t + \Delta_t) \end{aligned} \tag{3.9}$$

The table 3.1 summarizes all the relevant parameters of the SFM.

3.1.4 SFM drawbacks

The SFM is a useful and adaptable tool for representing human behavior in various situations. The main advantage is its great *description ability* i.e. the ability to describe pedestrian movement processes and phenomena [62]. However, it also presents certain limitations and issues that limit this ability. Below, some of them will be briefly outlined.

Parameters homogeneity and calibration

In the SFM model, the behaviour of the pedestrian is strongly influenced by the choice of parameters, which influence the intensity of the interaction with the environment. In real navigation scenarios, individuals respond differently to external stimuli due to their personal feelings and internal motivations. This can be translated as a large heterogeneity of model's parameter values, which will be different for each pedestrian. However, the original SFM assumes that all individuals are characterised by the same set of parameters, resulting in a certain homogeneity in the possible behaviours. This consequently leads to a limitation of the model's description ability.

Another issue related to the parameters is their calibration. Their values may vary considerably depending on the context in which the model is applied, leading to

Symbol	Meaning
$n \in \mathbb{N}$	number of pedestrians
$X \in \mathbb{R}^2$	planar space
$\mathbf{p}_i(t) \in \mathbb{R}^2$	position of pedestrian i at time t
$\mathbf{v}_i(t) \in \mathbb{R}^2$	velocity of pedestrian i at time t
$\mathbf{p}_j(t) \in \mathbb{R}^2$	position of pedestrian j at time t
$\mathbf{p}_i^{goal} \in \mathbb{R}^2$	position of the goal of pedestrian i
$\mathbf{F}_i^{goal}(t) \in \mathbb{R}^2$	attractive force for pedestrian i at time t
$\mathbf{F}_{i,j}^{rep}(t) \in \mathbb{R}^2$	repulsive force of pedestrian j on i at time t
$\mathbf{F}_i^{obs}(t) \in \mathbb{R}^2$	repulsive force of the closest obstacle on pedestrian i at time t
$\mathbf{F}_i^{res}(t) \in \mathbb{R}^2$	resultant of the forces at time t
$r_i \in \mathbb{R}_{>0}$	radius of the personal space of pedestrian i
$r_j \in \mathbb{R}_{>0}$	radius of the personal space of pedestrian j
$v_i^d \in \mathbb{R}_{>0}$	desired velocity (in module) of pedestrian i
$\alpha_i \in \mathbb{R}_{\leq 0}$	relaxation time of pedestrian i
$A_i \in \mathbb{R}_{>0}$	strength of interaction force for pedestrian i
$B_i \in \mathbb{R}_{>0}$	range of interaction force for pedestrian i
$R_0 \in \mathbb{R}_{>0}$	minimum admissible distance between pedestrian and obstacle
$\gamma_{i,j}(t) \in \mathbb{R}$	bearing of pedestrian j measured by pedestrian i
$F_{i,j}^{fov} \in [0,1]$	anisotropic factor for pedestrian-pedestrian interaction
$\gamma_{i,obs}(t) \in \mathbb{R}$	bearing of the nearest obstacle measured by pedestrian i
$F_{i,obs}^{fov} \in [0,1]$	anisotropic factor for pedestrian-object interaction
$\lambda \in [0,1]$	strength of anisotropic behavior

Table 3.1: SFM variables and parameters

more or less realistic behavior of the simulated pedestrians. For this reason, it becomes essential to determine the optimal set of parameters that best describe the scenario to be analysed. Consequently, this characteristic restricts the generality of SFM and its ability to adapt to various navigation situations. An example of such a behaviour can be seen in fig. 3.1.

Isotropic motion

In their original model, Helbling et al. implicitly assumes a holonomic motion. In this context, pedestrians are characterised by isotropic motion determining the ability to move in any direction at any time, regardless of their orientation. However, in reality, the biomechanics of the human body makes pedestrians move forward most of the time, aligning their direction of movement with the orientation

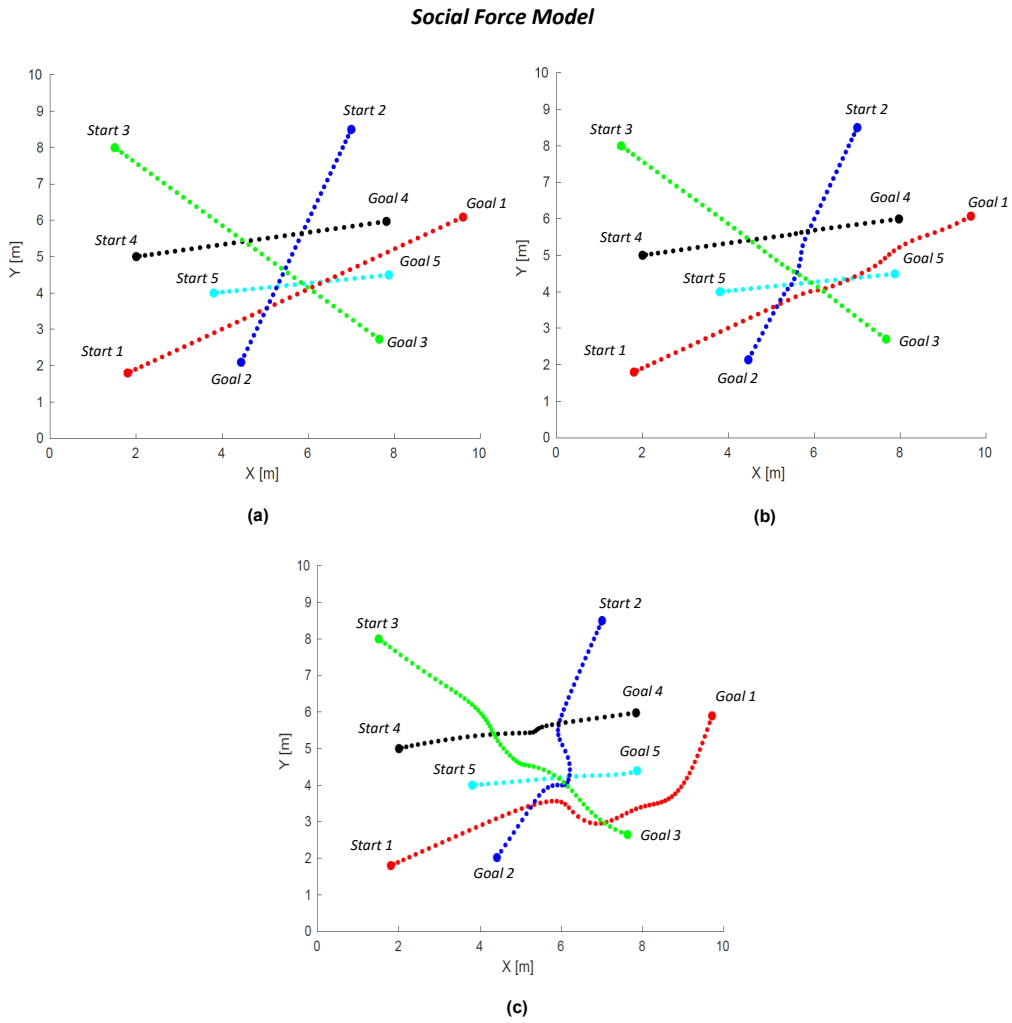


Figure 3.1: Example of pedestrians navigation behaviour with different sets of parameters: (a) $A_i = 0.2, B_i = 0.1, r_i = r_j = 0.1$; (b) $A_i = 0.45, B_i = 0.3, r_i = r_j = 0.4$; (c) $A_i = 0.8, B_i = 0.7, r_i = r_j = 0.7$. For simplicity, the variability was modelled by varying only the pedestrian interaction parameters and assuming that pedestrians have the same parameters in terms of desired speed v_i^d, relaxation time α_i and anisotropic strength λ. However, this assumption is not necessarily true in reality

of their head [17]. This behaviour has been analysed in various state-of-the-art studies such as [63], where the authors demonstrate how the movement of humans can be approximated by means of a nonholonomic motion model. Overall, this factor limits the real-world applicability of the original SFM, since it is not capable of representing all possible characteristics of human motion.

3.2 Game Theory

3.2.1 Overview

Game theory is a discipline that studies mathematical models capable of describing strategic interactions among rational agents. It is a fundamental tool for analysing problems involving multiple decision-makers, where the quality of the decision depends strictly on both the individual's choices and those of the other participants in the game. Although the first discussions on the mathematics of games date back to the second half of the XVI century (1564) with Girolamo Cardano, it was only in 1928 that modern game theory was born, with John von Neumann's contribution. He published an article proving the general theory for solving cooperative zero-sum games in the case of only 2 players, and later extended it to more players with the help of Morgenstern, with whom he published the results in their book in 1944 [64]. In 1950, however, John Nash succeeded in extending the criteria proposed by von Neumann by developing and proving the concept of the *"Nash equilibrium"* [65], which applied to a wider range of games than those proposed by von Neumann and demonstrated the existence of a solution even for non-cooperative games that were not necessarily zero-sum.

Nowadays, thanks to its ability to describe very complex phenomena, game theory is used in many different fields, such as economics and finance [66][67], politics [68], biology and natural selection [69], control theory [70], computer science and artificial intelligence [71][72].

3.2.2 Terminology related to game theory

A clear explanation of the main terminology used is necessary to better understand how game theory works and its application in the context of social navigation, which will be presented in the following chapters.

A description of the most commonly used words related to game theory is presented below:

- ***Players:*** they are the participants in the game;
- ***Action set:*** each player has a set of possible actions (or moves) that uses to make decisions during the different phases of the game;
- ***Stages:*** they are the phases of the game. In each phase, one or more players must perform an action;

- ***Game state:*** it corresponds to the set of information about the various players at a given stage of the game;
- ***Strategy:*** it represents a plan, a set of planned actions that the player intends to follow in order to achieve his goal. It is important to emphasize the difference with the concept of *move*, which corresponds to a specific action performed by a player during the game;
  There are several types of strategy in game theory. The most common ones used to describe the human motion in social navigation [8] are:
  - *Pure strategy:* is a type of strategy where the action the player will take based on the game state is deterministic and free of uncertainty. Therefore, the player will be completely sure of his decision;
  - *Mixed strategy:* the action/decision to be taken is determined by a probability distribution. Therefore, based on the state of the game, each action in the player's action set has a certain probability of being performed;
- ***Cost function:*** It corresponds to the way in which the individual player evaluates which decision to make during the game. Therefore, it is a representation of what drives the player to perform a particular action and will depend on the combination of strategies applied by all players;
- ***Solution:*** It is a type of rule that describes which strategies the players will adopt at a given stage, thus allowing the predict how the game will be played [73].

3.2.3 Game types

Over time, several types of games have been categorized. Each of them describes particular features that distinguish it from others. Below, an overview of the most common game types [74] is presented. This will be helpful in understanding how the game, implemented in the developed robot's navigation algorithm, models interactions with other pedestrians in the environment.

Cooperative games or non-cooperative games

A *cooperative game* models situations in which players can collaborate to achieve common goals. This situation can be represented as players minimizing a common cost function.
In contrast, a *non-cooperative game* describes situations where players cannot cooperate. Each player acts independently, aiming to reduce their individual cost.

Zero-sum games or non-zero sum games

In a *zero-sum game*, the gain or loss for one participant, generally referred to as the payoff, is exactly equal to the loss or gain for another player. Thus, the total sum of gains and losses of all participants in the game always remains constant and equal to zero.
In the case of *non-zero-sum games*, the total payoff of the players is not constant. This implies that one player's win does not necessarily determine another player's loss and thus cooperation between players to obtain common benefits is possible.

Static or dynamic games

In a *static game*, players make decisions simultaneously without the knowledge about other players' choices. Some examples of static game could be rock-paper-scissors or the well-known prisoner's dilemma in its general formulation.
In contrast, a *dynamic game* is characterized by players that make decisions sequentially. This implies that the choices of previous players influence the current player's decision. An instance of such a game is chess.

Perfect information games or imperfect information games

In *perfect information games*, all players have complete knowledge about the game's state and the moves made by others. Also in this case, the most famous example is chess.
In *imperfect information games*, participants possess limited and incomplete knowledge regarding the game's state. Examples of such games include Poker or Blackjack.

Symmetric or Non-symmetric games

Symmetric games are characterized by all players having the same number and types of strategies and moves that they can perform. A classic example of a symmetric game is the game 'rock-paper-scissors', in which all players have the same set of possible actions.
On the other hand, in *asymmetric games*, a player may have a different number and types of strategies and actions than other participants.

Finite or Non-finite games

A *finite game* is a game in which there are a limited number of participants and each of them has a limited number of actions they can perform.
In *non-finite games*, the number of players and the number of moves for each player are not fixed.

3.2.4 Nash equilibrium for Non-cooperative games

As proven in [8], it's possible to model human navigation as a non-cooperative non-zero sum game. In fact, during navigation each pedestrian will try to achieve its goal by reducing its individual cost. In addition, if an agent "win" (reaches its goal), it does not determine the *defeat* of another player (other pedestrian will still have the possibility to achieve their goal).
Therefore, the game's resolution will enable the individual agent to make predictions concerning other pedestrians' movements. Based on these predictions, each agent will choose the optimal action to perform.

One of the most well-known solutions for this type of game is the previously mentioned Nash equilibrium concept [65]. It can be defined as a combination of strategies where no agent can reduce its own cost by changing its action if the other agents stick to their actions [7]. Thus, the Nash equilibrium represent the optimal response for all the agent.

Expressing a generic combination of agents' strategies at a given game stage j as $s^j = \left(s_1^j, ..., s_i^j, ..., s_n^j\right)$, we can denote the Nash equilibrium at the same stage j with an asterisk:

$$s^{j*} = \left(s_1^{j*}, ..., s_i^{j*}, ..., s_n^{j*}\right) \tag{3.10}$$

where s_i^{j*} represent the best strategy of agent i.

The Nash equilibrium is established when each *i-th* player can no longer reduce his individual cost given the choices of the other players. In mathematical terms, when the following equation is satisfied:

$$J_i\left(s_1^{j*}, s_2^{j*}, ..., s_i^{j*}, ..., s_n^{j*}\right) \leq J_i\left(s_1^{j*}, s_2^{j*}, ..., s_i^{j}, ..., s_n^{j*}\right) \tag{3.11}$$

where J_i corresponds to the cost function of the *i-th* agent.

Applying the same concept to all agents results in a system of $n \in \mathbb{N}$ inequalities that must be satisfied simultaneously [75]:

$$\begin{aligned} J_1\left(s_1^{j*}, s_2^{j*}, ..., s_n^{j*}\right) \leq & J_1\left(s_1^{j}, s_2^{j*}, s_3^{j*}, ..., s_n^{j*}\right) \\ J_2\left(s_1^{j*}, s_2^{j*}, ..., s_n^{j*}\right) \leq & J_2\left(s_1^{j*}, s_2^{j}, s_3^{j*}, ..., s_n^{j*}\right) \\ & \vdots \\ J_n\left(s_1^{j*}, s_2^{j*}, ..., s_n^{j*}\right) \leq & J_n\left(s_1^{j*}, s_2^{j*}, ..., s_{n-1}^{j*}, s_n^{j}\right) \end{aligned} \tag{3.12}$$

Chapter 4

Game-Theoretic Social Force Model

4.1 Navigation algorithm

Despite its many advantages, the SFM is a purely reactive algorithm [12]. As consequence, it lacks the incorporation of a human motion model that could enable robot to make predictions. This characteristic makes the SFM less suitable for applications in a social navigation context, where anticipating mutual collision is a key aspect to generate human-like trajectories and enhance social acceptability [7]. In this regard, recent studies have shown how navigation can be effectively modeled as a non-cooperative game [8]. Through this assumption and using Game Theory, it is possible to incorporate reasoning about the actions that other agents will take within the navigation algorithms. This results in both the ability to handle possible conflict situations and an improvement in overall navigation performance, as the algorithm will try to figure out what are the best actions that each pedestrian might take.
Thus, the idea behind the algorithm proposed in this thesis is to model navigation as a non-cooperative game between the robot and the pedestrians in the environment and exploit the SFM to determine the possible actions that the pedestrians can choose. Subsequently, the game is solved through the application of the concept of Nash equilibrium.

The next section provides a detailed explanation of the game's features and the working principles of the algorithm.

Below, we will refer to the robot and pedestrians as "*agents*".

4.1.1 Overview

The game used for navigation is defined as a non-cooperative, static, perfect information and finite game with a finite number of rational $n \in \mathbb{N}$ agents $\mathcal{N} = \{1, ..., n\}$ who want to minimize their individual cost. Each agent is associated with a limited set of p actions denoted as $\mathcal{A}_i = \{a_i^1, ..., a_i^p\}$. In this case, we assumed $p = 4$. Therefore, the *i-th* agent will be characterized by an action set $\mathcal{A}_i = \{a_i^1, a_i^2, a_i^3, a_i^4\}$. The generic action $a_i^p \in \mathcal{A}_i$ represents a specific set of parameters of the SFM $a_i^p = \left[A_i, B_i, r_i, r_j, \lambda, \alpha, v_i^d\right]^p$ already described in table 3.1.

The main idea behind the algorithm is to associate each set of parameters of the *i-th* agent with the corresponding trajectories, and then use game theory to identify the optimal action to implement.
Since the model assumes a "*perfect information game*", all the agents have the knowledge about actions performed by other players and their positions at a given time instant t. Consequently, each agent i can use the SFM to compute the trajectory τ_i^p associated with the corresponding sets of parameters $a_i^p \in \mathcal{A}_i$. In particular, each trajectory will be computed by exploiting the system of discretized differential equations (3.9) over a fixed-time horizon of T_{prev} time-steps, where each of them has the duration of Δ_t.

Next, a cost will be assigned to each trajectory of *i-th* agent by means of a cost function $J(\tau_i^p)$, which will be described in detail later. Regardless of the considered cost function, by assuming that pedestrians have rational behavior and modelling navigation as a "*non-cooperative game*", agents will aim to minimize their individual cost choosing the optimal action for themselves, i.e. their optimal trajectory.

Thus, the application of the game theory can be seen as the definition of a system of $n \in \mathbb{N}$ interdependent optimization problems through which it is possible to rationally determine which parameters are most suitable for the effective movement of agents through the SFM.
The solution to such a game can be seen as the convergence to a Nash equilibrium, a particular situation where each agent has no incentive to unilaterally change its action unless others changing theirs [8].
When each agent computes its own best trajectories, it will apply only the motion related to the first time step Δ_t of the considered trajectory. At this point, the scenario is changed and the process restarts.

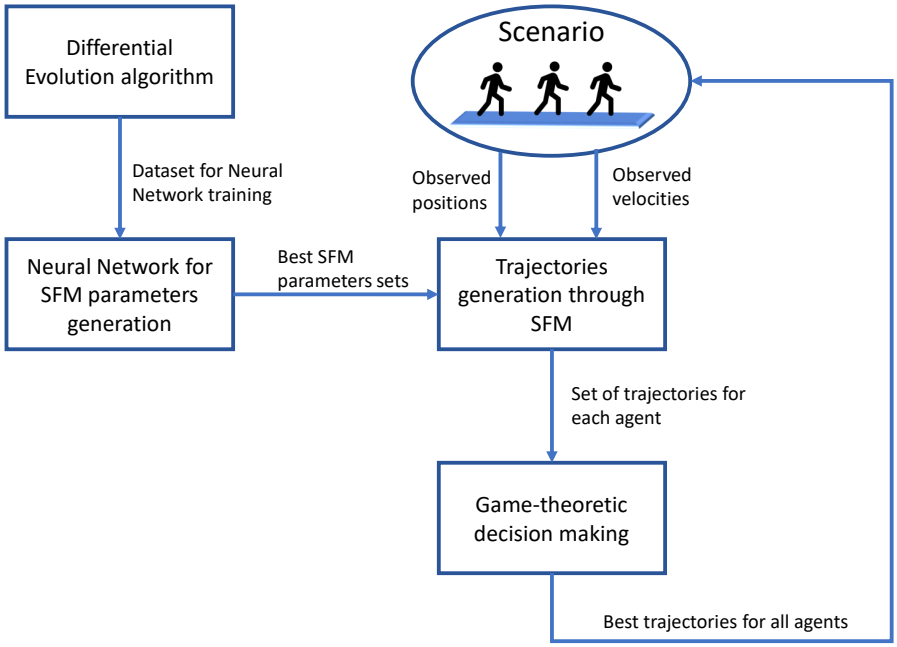


Figure 4.1: Conceptual structure of the Game-Theoretic Social Force Model (GTSFM).

One of the main advantages of the developed algorithm is the use of the SFM which guarantees a very low computational complexity, extremely necessary to compute multiple trajectories over a fixed-time horizon. In this way, decision-making can be performed at every time step, mimicking the sequential nature of the human decision-making process and making the Game-Theroretic Social Force Model more suitable for social navigation than the original SFM. In addition, at each time step, the SFM parameters of all agents are generated by a neural network previously trained using a labeled dataset obtained by applying a Differential Evolution algorithm (more details later). The conceptual structure of the overall algorithm is depicted in fig. 4.1.
However, the robot is able to generate its trajectories by means of the SFM parameters since it knows the final position it must reach. In the case of the other pedestrians' trajectories, this condition does not occur. In fact, in a real navigation context, the robot does not have the possibility to know a priori the goal position that a specific pedestrian will try to achieve.
For these reasons, in this thesis we assume the presence of a fictitious goal for each pedestrian, whose position is recomputed at each time instant. Therefore, each pedestrian will be attracted to a goal located along its direction of motion at time instant t. Specifically, the fictitious goal position will be place at a certain distance from the pedestrian and computed as the product of the corresponding

pedestrian's velocity $\mathbf{v}_j(t)$ and a time interval equal to the time horizon ($\Delta_t T_{prev}$) used for trajectory generation. In other words, the position of the fictitious goal at time t will be given by:

$$\mathbf{p}_j^{goal}(t) = \mathbf{p}_j(t) + (\Delta_t T_{prev})\mathbf{v}_j(t) \tag{4.1}$$

4.1.2 Cost function

As expressed above, each *i-th* pedestrian exploits a cost function $J(\tau_i^p)$ to associate a cost with the τ_i^p trajectory generated by the corresponding a_i^p parameters. Accordingly, the choice of such a cost function affects the overall performance of the algorithm.
In general, humans tend to move according to a minimization principle [8]. Following this idea, multiple types of cost functions have been developed over time, which consider the minimization of different motion aspects and characteristics. Some examples widely used at the state-of-the-art are cost functions that minimize the path length [76] or the energy consumption [77].
Although minimizing the length of the generated path seems to be the main goal during movement, this single feature cannot assign realistic costs for human navigation [7]. Therefore, it is necessary to incorporate preferences and physical properties of pedestrians that are actually relevant during navigation without causing an excessive increase in the complexity of the resulting cost function. In fact, higher complexity does not necessarily result in improved performance [8].
In this context, we chose to exploit a cost function previously presented in [43] and enriching it by introducing an additional term that explicitly governs the intensity of pedestrian interactions. Accordingly, the final cost function $J(\tau_i^p)$, used for assigning costs to each trajectory τ_i^p, can be formulated as a sum of three distinct components:

$$J(\tau_i^p) = \Phi_{goal}(\tau_i^p) + \Phi_{smooth}(\tau_i^p) + \Phi_{int}(\tau_i^p) \tag{4.2}$$

The details of each term are described below.

- ***Path length***
  The first term of the cost function is defined as:

$$\Phi_{goal}(\tau_i^p) = \sum_{k=1}^{T_{prev}} \|\mathbf{p}_i(t + k\Delta_t) - \mathbf{p}_i{}^{goal}\| \tag{4.3}$$

  This term is intended to take into account the overall length of the path generated to arrive toward the goal position p_i^{goal}. The reduction of this term necessarily implies the choice of a more goal-oriented motion.

- ***Path regularity***
  The second term of the cost function is expressed as:

$$\Phi_{smooth}(\tau_i^p) = \sum_{k=1}^{T_{prev}} |\theta_i(t + k\Delta_t) - \theta_i(t + (k-1)\Delta_t)| \tag{4.4}$$

  where $\theta_i(t + k\Delta_t)$ and $\theta_i(t + (k-1)\Delta_t)$ are the orientation angle of the agent i at the time $(t + k\Delta_t)$ and $(t + (k-1)\Delta_t)$, respectively. The aim of this term is the penalization of excessive rotations in order to promote the choice of smooth trajectories. As we can seen, both the first and second terms are closely related to the concepts of path and energy consumption minimization described previously, which describes the human tendency to choose shorter paths [7] and avoid too many changes in orientation [77].

- ***Interaction with other pedestrians***
  This term is the additional term mentioned above and is defined as:

$$\Phi_{int}(\tau_i^p) = \sum_{j \in \mathcal{N} \setminus \{i\}} \sum_{k=1}^{T_{prev}} \frac{\rho}{\|\mathbf{p}_i(t + k\Delta_t) - \mathbf{p}_j(t + k\Delta_t)\|} \tag{4.5}$$

  where $\mathbf{p}_i(t+k\Delta_t)$ and $\mathbf{p}_j(t+k\Delta_t)$ are the position of pedestrian i and pedestrian j at the time $(t + k\Delta_t)$, respectively, and $\|\mathbf{p}_i(t + k\Delta_t) - \mathbf{p}_j(t + k\Delta_t)\|$ is their reciprocal distance. Such a term aims to encourage selecting trajectories that maximize the distance between the two pedestrians participating in the interaction. Minimizing such a term implies increasing the distance between the two pedestrians, reducing the possibility of collision and maintaining interpersonal distance. Finally, the constant weighting factor $\rho \in \mathbb{R}_{>0}$ is used in order to adjust the relative influence with respect to the first and second term in the cost function.

4.1.3 Sequential best response for Nash equilibria

As mentioned above, it is possible to model navigation as a non-cooperative game. In this context, it is possible to consider the choice of the optimal trajectory by the various pedestrians as the achievement of the Nash equilibrium [8].
However, for the existence of the Nash equilibrium, two fundamental assumptions are necessary:

- *common knowledge of all players:* a quite realistic assumption if we consider that human beings learn the possible alternatives to reach their goal and how other people may behave when navigating from the everyday life experience;

- *strictly rational behaviour of all agents:* so that players try to effectively minimise their cost [8];

In general, both conditions are incorporated in our model assumptions. Whether these two conditions are met, the existence of such an equilibrium point is dependent on the type of strategy that characterizes the game.

In case of:

- *mixed strategies:* the existence of the Nash equilibrium is always guaranteed;
- *pure strategies:* the existence of the Nash equilibrium is ensured only for specific forms of cost function [8];

In our navigation model, we consider "pure strategies". However, the use of the previously described cost function (Eq. 4.2) ensure the existence of the Nash equilibrium. To compute it, we use the "*sequential best response*" method [78]. To give an idea of how this numerical method works, an example that involves two agents is given below.

Example of "sequential best response":

Consider two agents, named 1 and 2. Agent 1 observes agent 2's navigation at time t and chooses the best trajectory to reach its goal, given the last observed movement of agent 2. Subsequently, in each iteration, a control action is performed to check whether the strategies of both agents remain unchanged from the previous iteration; if so, the game has reached a Nash equilibrium. If not, agent 2 calculates its optimal strategy based on the last observed strategy of agent 1. This iterative process continues until the equilibrium is met. The logic presented can be extended to n agents.

4.2 Parameters estimation from real trajectories dataset

The navigation algorithm proposed above exploits an action set consisting of $p = 4$ different trajectories for each pedestrian. At each time instant t, these trajectories are generated directly by the various agents using the SFM. Therefore, it is necessary to estimate four different sets of parameters, one for each trajectory. As expressed in Chapter 3, one of the main drawbacks of the SFM is the high sensitivity of the model to the choice of these parameters, which represent the intensities of the agents interactions during navigation. Therefore, it is extremely necessary to carry out a careful evaluation to identify which are the most suitable

parameters according to the considered scenario.

Basically, in the state of the art, it is possible to divide the approaches for establishing SFM parameter values into two main methodologies, which vary according to the simulated context.
In particular, for:

- *Crowd simulation:* the number of pedestrians is very high. Therefore, the most common approach is to use optimization algorithms or learning strategies to derive the best parameters directly from real public datasets [59][79][80];

- *Social navigation:* the limited number of pedestrians allows to derive the parameters through a trial-and-error process [17][81][82]. In fact, analyzing the behaviour of individual agents becomes more challenging in situations where there are large crowds;

To obtain real parameter sets, it is also necessary to apply optimization algorithms in the context of social navigation. In this thesis, we used Differential Evolution, a well-known state-of-the-art optimization algorithm, that is highly used to estimate the best SFM parameters in crowd simulation contexts from real datasets [83][59].
However, in order to represent the real navigation more accurately, two further fundamental aspects must be taken into account. During navigation, the type of behaviour of pedestrians may vary over time according to how the environment evolves. Furthermore, as already expressed in Chapter 3, each pedestrian is characterized by different movement features, resulting in a certain heterogeneity of behaviours. Therefore, in order to achieve realistic trajectory generation from the SFM, it is necessary to ensure that parameters are different for each pedestrian and can vary according to the environment conditions perceived by the agent.

Although DE allows to estimate best parameters that approximate real human trajectories, this type of optimization algorithm is not suitable for real-time application due to the high estimation time. To solve this problem, it is possible to exploit a neural network that approximates DE operations and generates the parameter sets required by the navigation algorithm in real-time. However, training a neural network needs an immense amount of labelled data, which is difficult to obtain in the state-of-the-art, especially in the case of SFM.
To address this lack of data, DE is applied to the Thör dataset in order to estimate the four best parameter sets that approximate same portions of trajectories needed to the navigation algorithm (from generic instant t to instant $(t + \Delta_t T_{prev})$). Subsequently, these sets are used to generate a new dataset for training the neural network. In this way, it is possible to obtain more realistic SFM parameters values

than those achievable through trial-and-error process.

Below, a general description of how the DE works and the characteristics of the Thör dataset, used for parameters estimation, are presented. Then, the chapter provides an in-depth explanation of the DE algorithm used to create the dataset used for the neural network training and a description of the implemented neural network.

4.2.1 Differential Evolution algorithm (DE)

The DE operates on a population of *NP D-dimensional real-valued parameter vectors*, where NP represents the number of parameter vectors in the population and D corresponds to the number of parameters contained in each vector. These parameters vectors are commonly known as "*chromosomes*" and the corresponding parameters are called "*genes*". Each chromosome represents a candidate solution of the considered optimization problem. Through the various steps, the algorithm will establish the most suitable chromosomes to pass on to the next generation by applying typical EAs operators such as *mutation*, *crossover* and *selection*, which will be described in more detail later. This process is repeated for a defined number of generations, denoted as NG.

Therefore, given a generic generation j, it is possible to indicate the *i-th* chromosome with the following notation:

$$\mathcal{X}_{i,j} = \left\{x_{i,j}^1, ..., x_{i,j}^D\right\}, \text{ for } i = 1, ..., NP \text{ and } j = 1, ..., NG \tag{4.6}$$

To determine which chromosome will survive into the next generation, it is essential to evaluate its performance by defining a "*fitness function*" $f(\mathcal{X})$, which must be designed on the basis of the system to be analysed. This function will depend on the parameters contained in each chromosome. The purpose of the DE is then to search for the optimal chromosome $\mathcal{X}^*$ that will ensure the minimization of the fitness function among all the possible chromosomes obtained in the various NG generations.

At the state-of-the-art, there are multiple variations of the Differential Evolution. These variants differ in the strategies applied to perform the individual steps of the algorithm. For this thesis, the classical version of DE developed by Storm and Prince [44] was used. The main phases will be described below.

1. *Initialization of Parameter Vectors:*
   The first stage of the DE algorithm involves creating an initial population

of parameter vectors which will represent the starting generation *j=0*. As previously mentioned, each chromosome comprises D real values that represent the physical parameters of the system to be optimised. Since they are physical parameters, their values will be restricted within specific ranges determined by the nature of the physical quantity they represent. For instance, if the parameter under consideration represents a mass or a time instant, negative values are impossible.

By defining the extremes of the range intervals for all D parameters, the minimum and maximum limits can be expressed as:

$$\begin{aligned} \mathcal{X}_{min} &= \left\{x^1_{min}, ..., x^D_{min}\right\} \\ \mathcal{X}_{max} &= \left\{x^1_{max}, ..., x^D_{max}\right\} \end{aligned} \tag{4.7}$$

As stated in [45], the aim of the initial population is to cover these intervals as much as possible by uniformly randomizing the chromosomes within the search space constrained by the specified boundaries.

For this reason, the initial value of the *k-th* parameter of chromosome i at generation *j=0* will be given by:

$$x^k_{i,0} = x^k_{min} + rand\,[0,1]^k_i \left(x^k_{max} - x^k_{min}\right), \text{ for } k = 1, ..., D \tag{4.8}$$

where $rand\,[0,1]^k_i$ is a uniformly distributed random number in the range [0,1] generated independently for each *k-th* parameters of the *i-th* chromosome.

2. *Mutation with Difference Vectors:*
In the context of DE algorithms, mutation represents the process whereby each chromosome in the current population, called *Target vector* $\mathcal{X}_{i,j}$, undergoes sudden perturbations of one or more genomes through the introduction of a random element.
Therefore, in each generation j, the DE-mutation is responsible for associating with each *Target Vector* $\mathcal{X}_{i,j}$, a chromosome derived from it called *Donor Vector* $\mathcal{V}_{i,j}$:
$$\mathcal{V}_{i,j} = \left\{v^1_{i,j}, ..., v^D_{i,j}\right\}, \text{ for } i = 1, ..., NP \tag{4.9}$$
The creation of such a *Donor Vector* $\mathcal{V}_{i,j}$ requires the sampling of three other parameter vectors from the current population, denoted as $\mathcal{X}_{r^i_1,j}$, $\mathcal{X}_{r^i_2,j}$ and $\mathcal{X}_{r^i_3,j}$. These vectors will be randomly chosen by selecting indices within the range [1,NP], which must be mutually exclusive and also different from the the current chromosome index i.

In the simplest form of DE-mutation, the *Donor Vector* is obtained by adding to the first random vector the difference between the other two, which will be scaled by a factor F. It is known as "*Mutation Factor*" and is one of the parameters that influences the overall performance of the DE algorithm. Through this factor, it is possible to regulate the effect that the difference vector will have on the mutation of the considered chromosome. Thus, the *Donor Vector* will be given by the following expression:

$$\mathcal{V}_{i,j} = \mathcal{X}_{r_1^i,j} + F\left(\mathcal{X}_{r_2^i,j} - \mathcal{X}_{r_3^i,j}\right) \tag{4.10}$$

The process of mutation described above is graphically represented in fig. 4.2.

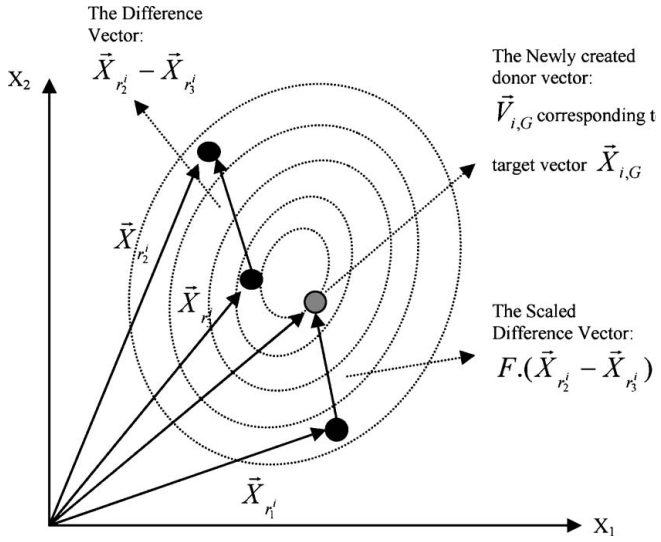


Figure 4.2: Standard DE-mutation in a 2-D parametric space

3. *Crossover:* Performed immediately after mutation, this phase introduces actual diversification within the chromosomes population. The purpose is to mix parameters contained in the generic *Target Vector* $\mathcal{X}_{i,j}$ with those in the associated *Donor Vector* $\mathcal{V}_{i,j}$ in order to generate a new chromosome denoted as *Trial Vector*: $\mathcal{U}_{i,j}$:

$$\mathcal{U}_{i,j} = \left\{u_{i,j}^1, ..., u_{i,j}^D\right\}, \text{ for } i = 1, ..., NP \tag{4.11}$$

The number of genes in the *Target Vector* that will actually be replaced is regulated through another crucial parameter of the algorithm, i.e. the *Crossover rate Cr*.
Although there are different types of strategies to perform crossover at the

state-of-the-art, in this thesis we decided to perform the binomial crossover. Its name derives from the fact that the number of inherited parameters from the *Donor Vector* will have a (nearly) binomial distribution [45] controlled by the value of *Cr*. For this reason, the *k-th* element of the *Trial Vector* $\mathcal{U}_{i,j}$ can be derived as follows:

$$u_{i,j}^{k} = \begin{cases} v_{i,j}^{k}, \text{ if } rand_{i}^{k}\,[0,1] \leq CR \text{ or } k = k_{rand} \\ x_{i,j}^{k}, \text{otherwise} \end{cases} \tag{4.12}$$

where $rand_{i}^{k}$ is a uniformly distributed random number within the range [0,1]. In addition, $k_{rand} \in [1,2,...,D]$ represents a randomly chosen index which guarantees that the trial contains at least one gene inherited from the donor. This method ensures that the resulting *Trial Vector* $\mathcal{U}_{i,j}$ can never be identical to the *Target Vector* $\mathcal{X}_{i,j}$.

4. *Selection:*
   It is the process of selecting which chromosome between the *i-th Target Vector* $\mathcal{X}_{i,j}$ and corresponding *Trial Vector* $\mathcal{U}_{i,j}$ will be passed on to the next generation. As previously mentioned, the choice is based on the value of the fitness function determined by the individual chromosome:

$$\mathcal{X}_{i,j+1} = \begin{cases} \mathcal{U}_{i,j}, \text{ if } f(\mathcal{U}_{i,j}) \leq f(\mathcal{X}_{i,j}) \\ \mathcal{X}_{i,j}, \text{otherwise} \end{cases} \tag{4.13}$$

Once this last step is completed, a new population of chromosomes is obtained, which will undergo mutation, crossover and selection phases again. This process continues recursively until the final NG generation is reached.

Control Parameters of the algorithm

This section aims to provide a detailed explanation of the effects of varying the control parameters on the performance of the Differential Evolution algorithm.

As mentioned above, the DE algorithm is characterised by three main control parameters:

- ***Population Size NP***: it corresponds to the number of chromosomes within the generic population. Typically, the NP value is kept within the range [5D; 10D], where D is the dimension of the problem [45] (i.e. the number of parameters in each chromosome);

- ***Mutation factor F***: it corresponds to the factor used to scale the difference vector $\left(\mathcal{X}_{r_2^i,j} - \mathcal{X}_{r_3^i,j}\right)$ (Eq. 4.10). In [45], authors suggest an initial value of F equal to 0.5, indicating the range of values [0.4; 1] as the operating range. A higher value of F consequently leads to a greater perturbing effect of the difference vector.
  The main aspect of the mutation process in the DE algorithm is that the chromosome perturbation is generated by a difference vector obtained directly from the chromosomes in the population and not by means of a probability density function (as in other EAs). This feature determines one of the main advantages of this type of mutation, namely "counter matching" [44]. It corresponds to the process by which, using difference vectors, the algorithm is able to identify and then automatically explore the most promising points in the solution space of the considered objective function.
  This feature enables the algorithm to autonomously transfer the search point of the solution, allowing it to jump out of possible local minima. Therefore, through the choice of the value of F, it is possible to adjust this transfer capacity of the search point. A higher value of F results in a more distant transfer of the search point within the solution space;

- ***Crossover rate Cr***: it controls the number of parameters altered within members of the population and it can takes value between 0 and 1 [45].
  Considering the Eq. 4.12 of binomial crossover we can say that for:

  - *Low values of Cr:* a lower number of parameters change between one generation and the other. In fact, the trial vector $\mathcal{U}_{i,j}$ will have a greater number of parameters derived from the target vector $\mathcal{X}_{i,j}$.
    Therefore, the chromosomes of the new generation will be very similar to those of the previous generation. This results in an exploration of the solution space by stepwise movements orthogonal to the coordinate axes (i.e. parameters axis);

  - *High values of Cr (near 1):* a larger number of parameters are inherited from the donor vector $\mathcal{V}_{i,j}$. As a result, the chromosomes of the new generation will have a greater probability of being significantly different from those of the previous generation (the final population will be obtained by selection anyway).
    Therefore, this feature prohibits the exploration of the solution space by steps orthogonal to the coordinate axes;

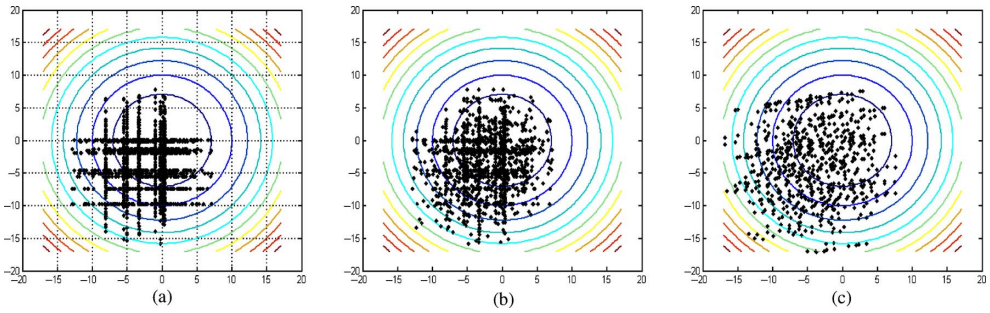


Figure 4.3: Effect of different values of Cr on a distribution of candidate trial vectors obtained by running DE on a single starting population of ten vectors for 200 generations with selection disabled [45]: (a) $Cr = 0$; (b) $Cr = 0.5$; (c) $Cr = 1$.

In the state of the art, there are various studies suggesting possible operating ranges and initial values of F and Cr. However, many of these studies lack sufficient experimental justification [45].
For these reasons, the original ranges and reference values suggested by Storm and Price [44] were chosen in this thesis work.

The table below provides a summary of parameters values used in the implemented DE algorithm:

Symbol	Value	Meaning
NP	42	Number of chromosomes in the population
NG	90	Number of generations
F	0.5	Mutation factor
Cr	0.6	Crossover rate

Table 4.1: Control parameters of the implemented DE algorithm

4.2.2 Thör dataset

Thör [9] is a public dataset of human motion trajectories, recorded in a controlled indoor experiment. To enhance the comprehension of the dataset's content, the following section will provide a description of the main features of the experiment and the instrumentation used to collect these data.

Environment

The dataset was recorded in a laboratory room measuring 8.4 x 18.8 m.

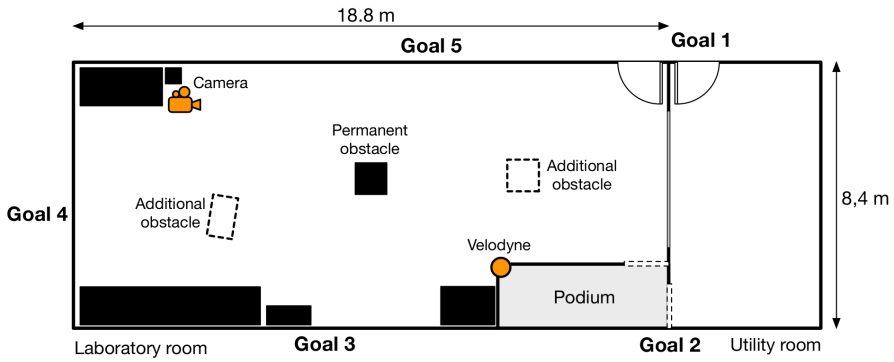


Figure 4.4: Schematic presentation of the laboratory room where the trajectories were recorded.

As shown in fig. 4.4, five goals are placed in the room to force the pedestrians to navigate and to create as many interactions as possible during navigation. There are also a variable number of obstacles to prevent too straight a movement between the goals.
The experiment also involves positioning a fixed camera and a LiDAR system. The first device was employed to record the movements of pedestrians during the experiment. The second instrument was used to acquire data on the environment, including the location of the walls in the room.

Motion capture system

To track people, the authors used 10 Qualisys Oqus 7+ infrared cameras (fig. 4.5 a) mounted around the perimeter of the room to identify specific markers capable of reflecting infrared light.

(a) **(b)**

Figure 4.5: (a) Qualisys Oqus 7+ infrared cameras; (b) helmets with reflective markers, used to track pedestrians.

These reflective markers are then attached to the helmets worn by the pedestrians during the experiment (fig. 4.5 b). This allows the infrared cameras to identify the movement of each reflective marker in real time. Nine helmets were used in the experiment, one for each participant, numbered from 2 to 10.

Experiment description

To ensure different pedestrian interactions and acquire a heterogeneous dataset, Rudenko et al. assigned social roles and tasks to each participant in the experiment. The goal was to replicate typical activities found in densely populated indoor environments, like offices and shopping malls, as closely as possible. Through the use of goals and tasks, participants are motivated to engage in natural and purposeful movements and create a rich variety of unplanned interactions.

The experiment involves three different types of pedestrians, each with its own objective:

- ***Visitors:*** they move either individually or in groups of up to five people, between four possible goal positions. At each reached target position, the person selects a random card that indicates the next goal;
- ***Workers:*** they are pedestrians that transports large boxes between different goals;
- ***Inspectors:*** type of pedestrian whose task is to navigate alone within the room among multiple additional targets, represented by a QR code, without any specific order. Upon reaching each QR code, the person must stop and scan the code before moving on to the next target.

In total, the experiment involves 6 visitors, 2 workers and 1 inspector.

Data format e Data management

The dataset comprises 13 distinct records, which are classified into three types of scenarios:

1. *One obstacle:* pedestrians move into the environment without any robots present. There is only a stationary obstacle located in the center of the room;
2. *Moving robot:* pedestrians move through the environment with the robot. A fixed obstacle is placed at the center of the room. Fig.4.6 shows the ideal trajectory of each participant in the experiment, including the robot;

3. *Three obstacles:* as in the first scenario, pedestrians move into the environment where there isn't any robot. In this case, however, there will be 3 stationary obstacles in the room;

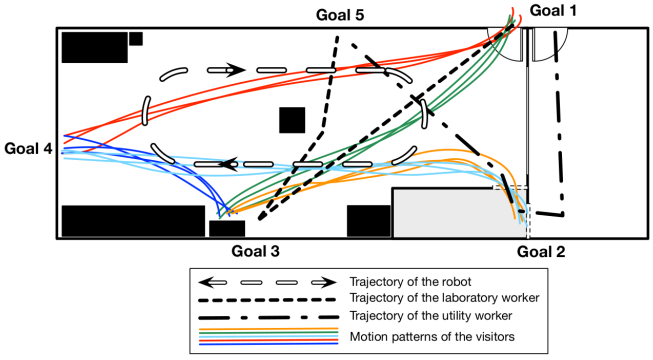


Figure 4.6: Ideal trajectories of the robot and the various types of pedestrians

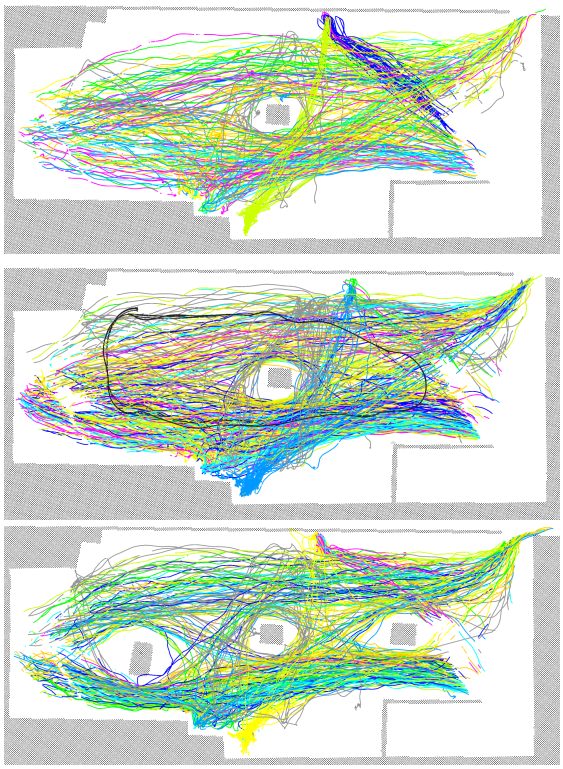


Figure 4.7: Real trajectories of experiment's partecipants in the three different scenarios: (top) *One obstacle* - (centre) *Moving robot* - (bottom) *Three obstacles.*

In order to ensure greater consistency, we chose to use only the recordings from the first scenario in the dataset. This choice is related to a significant similarity with the type of experiment we are going to propose for this thesis work, which will be described in detail below.
This approach enabled training the previously mentioned neural network by utilizing navigation data gathered from situations resembling those encountered during the experiment, minimizing the risk of underfitting the data.

4.2.3 Algorithm description and simulation results

The process of parameters estimation by means of DE algorithm proposed in this thesis work takes inspiration from [84]. In their study, Johansson et al. exploit an evolutionary algorithm to estimate SFM parameters from video tracking data. Based on a similar idea, in our DE each pedestrian in the dataset is replaced (one at a time) with a simulated robot, which moves using the SFM. Such a robot will start from the same position as the real pedestrian and move for successive T_{prev} time steps Δ_t.
This makes it possible to compare the positions reached by the simulated robot driven by SFM with the actual positions reached by the real pedestrian. This comparison will be executed during the selection phase using the following fitness function:

$$f(\mathcal{X}) = \frac{\sum_{k=t}^{(t+\Delta_t T_{prev})} \|\mathbf{p}_{robot}(k) - \mathbf{p}_{pedToSub}(k)\|}{N} \tag{4.14}$$

where N represents the total number of points in the trajectory generated by SFM.

The proposed algorithm is used to estimate the $D = 7$ main parameters of the social force model summarized in table 4.2.

Symbol	Meaning
$A_i \in \mathbb{R}_{>0}$	strength of interaction force for pedestrian i
$B_i \in \mathbb{R}_{>0}$	range of interaction force for pedestrian i
$r_i \in \mathbb{R}_{>0}$	radius of the personal space of pedestrian i
$R_0 \in \mathbb{R}_{>0}$	minimum admissible distance between robot and obstacle
$\lambda \in [0,1]$	strength of anisotropic behavior
$v_i^d \in \mathbb{R}_{>0}$	desired velocity (in module) of pedestrian i
$\alpha_i \in \mathbb{R}_{\leq 0}$	relaxation time of pedestrian i

Table 4.2: SFM parameters estimated by the implemented Differential Evolution algorithm

The most important steps executed by the DE are described in Algorithm 1. Each pedestrian within the Thör dataset is characterized by a certain number of trajectories. Each point on the trajectory is represented by a tuple *(x,y,t)* which will contain the coordinates in the plane and the corresponding time instant t.
Algorithm 1 takes as input the time step Δ_t, the number T_{prev} of time step for generating the simulated trajectory and the Thör dataset. It returns as output the final dataset used for the training of the neural network.
For each pedestrian to substitute (pedToSub) contained in the dataset, the algorithm extracts information about all associated trajectories using the "*getTrajectoriesInfo()*" function (line 1-2). This makes it possible to know the starting position and goal position information of each trajectory.
Since the DE is applied to each time instant t of the real trajectory, it is necessary to verify that the simulated one (subsequently used for the selection phase) does not exceed the time instant in which the real pedestrian actually reaches his goal position.
For this reason, a check is performed at each time instant t to control whether the maximum time instant associated with the simulated trajectory $(t + \Delta_t T_{prev})$ is greater than the time instant in which the real pedestrian reaches its goal (line 4-5). If this condition is met, then the algorithm moves on to the next trajectory, since the remaining points of the considered one are not sufficient to perform an effective selection (line 6). Otherwise, the best parameters of the SFM that approximate the actual trajectory in the time interval $[t; (t + \Delta_t T_{prev})]$ are estimated (line 7).
At time t, the position of the robot is initialized to the same position of the real pedestrian (line 8). Next, 4 types of distances are stored (line 9-16). Specifically, the minimum distance between the replaced pedestrian and the nearest obstacle and the minimum distances between the replaced pedestrian and the 3 other nearest pedestrians are considered. The 4 best parameter sets are estimated by means of the "*bestParamsEstim()*" function (line 17), which is described in more detail in Algorithm 2. Finally, the 4 distances are labelled by the 4 sets of estimated parameters that best approximate that portion of trajectory (line 18).

Algorithm 1 Main algorithm

Input: Δ_t, T_{prev} ThorDataset
Output: paramDataset

foreach pedToSub *in* ThorDataset
  pedTrajectoriesInfo $\leftarrow$ $getTragectoriesInfo$(pedToSub)
  foreach trajectory *in* pedTrajectoriesInfo
    foreach t *in* durationOfTrajectory
      if $(t + \Delta_t T_{prev}) >$ reachingGoalTime **then**
        break
      else
        $\mathbf{p}_{robot}(t) = \mathbf{p}_{pedToSub}(t)$
        $d_{robot,obs}(t) = \|\mathbf{p}_{robot}(t) - \mathbf{p}^{obs}\|$

        $d_{robot,ped_1}(t) = \|\mathbf{p}_{robot}(t) - \mathbf{p}_{ped_1}(t)\|$
        $d_{robot,ped_2}(t) = \|\mathbf{p}_{robot}(t) - \mathbf{p}_{ped_2}(t)\|$
        $d_{robot,ped_3}(t) = \|\mathbf{p}_{robot}(t) - \mathbf{p}_{ped_3}(t)\|$

        $\mathbf{d}_{3closestPed}(t) = [d_{robot,ped_1}(t), d_{robot,ped_2}(t), d_{robot,ped_3}(t)]$

        bestSets $\leftarrow$ $bestParamsEstim(NP, F, Cr, paramsConstr)$
        paramDataset $\leftarrow$ $[d_{robot,obs}(t), \mathbf{d}_{3closestPed}(t),$ bestSets]
      end if
    end
  end
end
return

Overall, Algorithm 2 represents the operation of the DE algorithm described above.
After initialising the empty vector "bestSets" (which will contain the various parameters)(line 1), the DE is applied four consecutive times on the same portion of the trajectory (line 2). For each set, the algorithm initializes three variables: "bestChrom", "bestFitnessVal" and $\mathcal{P}_{curr}$ (line 3-5). The first one will contain the final best chromosome, i.e. the best set of parameters that approximate the considered trajectory. The second one is used to compare the best chromosome obtained from each generation with the best chromosome of all generations. For this reason, its initial value is set very high. The last one is initialized as empty vector, which will contain the initial population.
Each chromosome of the initial population is generated using the "*generateChromosome()*" function. It will take as input the constraints associated with the various

SFM parameters, i.e. genes of each chromosome (line 7-10).
Subsequently, at each generation j, the algorithm first initializes an empty vector of chromosomes that will represent the new population $\mathcal{P}_{new}$ and then applies the mutation, crossover and selection steps as described above (line 12-24) in order to generate the next generation of chromosome.
Specifically, the selection function (described in detail in Algorithm 3) will return both the chromosome selected between the trial vector $\mathcal{U}$ and the target vector $\mathcal{X}$ (which will be part of the next generation) and the corresponding fitness function value "minFitnessVal" (line 22).
To ensure the extraction of the best performing chromosome among all generations, at the end of each selection step the resulting fitness function value "minFitnessVal" is compared with the "bestFitnessVal", initially set to 1000 (line 25-27). If the fitness value associated with the chromosome has a lower value than the "bestFitnessVal", the chromosome is stored in "bestChrom" and the "bestFitnessVal" is updated. When the last generation is reached, the "bestChrom" will be the actual chromosome with the lowest fitness function value so it will be stored inside the "bestSets" vector (line 31).

Considering in more detail Algorithm 3, the "*selection()*" function will take as input the pair consisting of target vector $\mathcal{X}_{i,j}$ and trial vector $\mathcal{U}_{i,j}$.
First the algorithm initializes the variable "fitnessValues" as an empty vector (line 1). Then, for each of the two chromosomes, it sets the corresponding parameters within the SFM integrated in the robot (line 3).
A vector of coordinates $\mathbf{d}_{robot,ped}$ is initialized as empty vector in order to keep track of the distances between the simulated robot and the real pedestrian positions at each time step (line 4). For each time step k the algorithm updates the distance vector $\mathbf{d}_{robot,ped}$ and subsequently moves the robot to the next position by applying the SFM equations (line 6-9).
After completing the generation of the trajectory, the algorithm evaluates the fitness function value corresponding to the considered chromosome by using the "*evaluateFitness()*" function (line 11). Such a function will take as input the vector of coordinates $\mathbf{d}_{robot,ped}$ and computes the fitness value associated with the chromosome using Eq. 4.14. These values are stored in the "fitnessValues" vector. Finally, the algorithm determines which chromosome ensures the minimum fitness value contained in "fitnessValues" (line 14-15).

Algorithm 2 The bestParametersEstimation function

Input: F, Cr, NP, paramsConstr
Output: bestSets

bestSets = {}
for $n_{set} = 1 : 4$ **do**
    bestChrom = [0,0, ...,0]
    bestFitnessVal = 1000
    $\mathcal{P}_{curr} = \{\}$

    for $i = 1 : NP$ **do**
        $\mathcal{X}_{i,0} \leftarrow generateChromosome$(paramsConstr)
        $\mathcal{P}_{curr} \leftarrow \mathcal{X}_{i,0}$
    end for

    for $j = 1 : NG$ **do**
        $\mathcal{P}_{new} = \{\}$
        $\mathcal{V}_{i,j}$ = [0,0, ...,0]
        $\mathcal{U}_{i,j}$ = [0,0, ...,0]

        foreach $\mathcal{X}_{i,j} \in \mathcal{P}_{curr}$
            $idx_{rand} \leftarrow rand([1, NP],1,3)$
            $\mathcal{X}_{r_1^i,j}, \mathcal{X}_{r_2^i,j}, \mathcal{X}_{r_3^i,j} = \mathcal{P}_{curr}(idx_{rand})$
            $\mathcal{V}_{i,j} \leftarrow differenceMutation(\mathcal{X}_{r_1^i,j}, \mathcal{X}_{r_2^i,j}, \mathcal{X}_{r_3^i,j}, F$,paramsConstr)
            $\mathcal{U}_{i,j} \leftarrow binomialCrossover(\mathcal{X}_{i,j}, \mathcal{V}_{i,j}, Cr)$
            $\mathcal{X}_{i,j+1}$, minFitnessVal $\leftarrow selection(\mathcal{X}_{i,j}, \mathcal{U}_{i,j})$

            $\mathcal{P}_{new} \leftarrow \mathcal{X}_{i,j+1}$
            if minFitnessVal $\leq$ bestFitnessVal **then**
                bestFitnessVal = minFitnessVal
                bestChrom = $\mathcal{X}_{i,j+1}$
            end if
        end
    end for
    bestSets $\leftarrow$ bestChrom
end for
return

Algorithm 3 The selection function

Input: $\mathcal{X}_{i,j}, \mathcal{U}_{i,j}$
Output: $\mathcal{X}_{i,j+1}$, minFitnessVal

fitnessValues = [ ]
foreach chromosome *in* $[\mathcal{X}_{i,j}, \mathcal{U}_{i,j}]$
    $robot \leftarrow setSFMparams$(chromosome)
    $\mathbf{d}_{robot,ped} = [\,]$

    for $k = t : (t + \Delta_t T_{prev})$ **do**
        $\mathbf{d}_{robot,ped} \leftarrow \|\mathbf{p}_{robot}(k) - \mathbf{p}_{pedToSub}(k)\|$
        $\mathbf{p}_{robot}(k+1) \leftarrow moveOneStepSFM()$
    end for

    fitnessValues $\leftarrow evaluateFitness(\mathbf{d}_{robot,ped})$

end
minFitnessVal $\leftarrow$ $min(fitnessValues)$
$\mathcal{X}_{i,j+1}$ = chromosome *in* $[\mathcal{X}_{i,j}, \mathcal{U}_{i,j}]$ associated with minFitnessVal
return

In fig. 4.8 are shown some trajectories generated by the SFM parameters obtained from the implemented Differential Evolution algorithm.

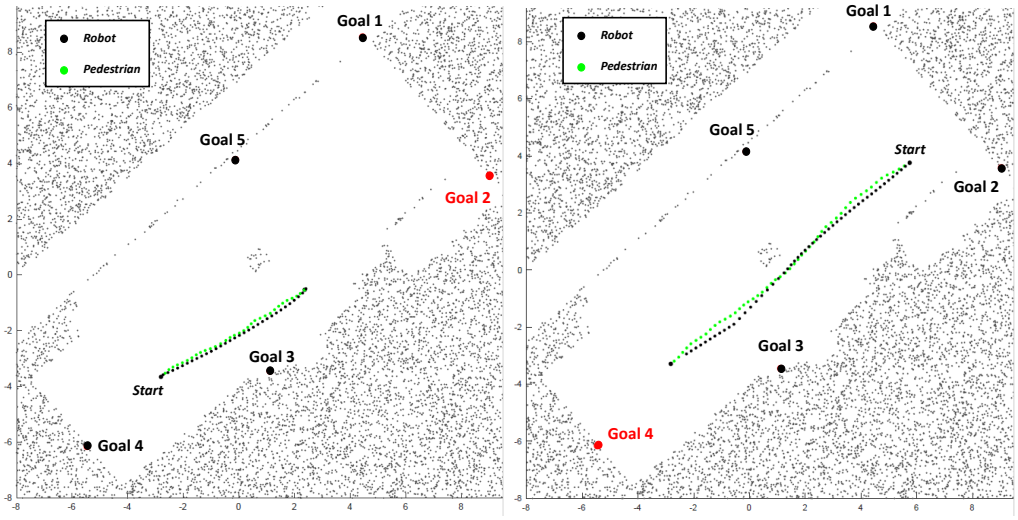


Figure 4.8: Comparison between real human trajectories contained in Thör (green) with the corresponding robot trajectories (black) obtained by using SFM parameters from DE estimation.

For completeness, such portions of trajectories are compared with the corresponding human trajectories from which parameters have been estimated.

4.2.4 Real-time parameters estimation through Neural Network

The dataset obtained by DE algorithm is used to train a fully connected neural network. The navigation algorithm will exploit the neural network for predicting and providing the SFM parameters that will be used for generating both the robot and pedestrians trajectories needed to Game Theory.
The neural network configuration involves the use of four input quantities, which include the distance between the robot and the nearest obstacle as well as distances between the robot and the three nearest pedestrians. This configuration results in an input layer consisting of four neurons.
Once the neural network has been trained, it will output a total of 28 parameters, corresponding to the 7 parameters of the four best sets. As a result, the output layer consists of 28 neurons. The neural network's internal structure comprises three hidden layers that contain 200 neurons each. These neurons utilize the ReLU (Rectified Linear Unit) function as activation function to process the combination of inputs from the previous layer.
The training phase was performed by exploiting the "*adam*" optimizer, an optimization algorithm implementing a stochastic gradient descent method. The loss function employed to perform the training phase is Mean Absolute Error (MAE). The same metrics has been used to evaluate performances of the neural network in both the training and testing phase. The training phase has been performed with a dataset of 420 labelled data. Overall, the neural network has been trained for 100 epochs with a final MAE of 0.2665. The test phase has been performed exploiting a testing dataset of 100 labelled data, with a final MAE of 0.2676.

Chapter 5

Hardware description

Before conducting real-world experiments, it is necessary to evaluate the developed navigation algorithm through a simulation campaign. In this way, it is possible to derive preliminary results to understand the effectiveness of the algorithm. To perform such simulations and obtain concrete data, it is necessary to test the algorithm on a simulated mobile robot model, possibly the same one that will be used in subsequent real-world experiments. In our laboratory we have a Locobot WX250s. For this reason, we used its simulated model in Gazebo for testing the three considered navigation algorithms (more details in chapter 7). Therefore, the purpose of this chapter is to describe the various hardware components of the Locobot WX250s used for simulations.

Figure 5.1: Mobile robot Locobot WX250s-6DOF used in the simulation campaign

5.1 Mobile robot hardware

The simulation campaign employed the Locobot WX250s (fig.5.1), which was developed and marketed by Trossen Robotics.
The system comprises numerous components that are essential for its operation. In the following, the description of each component is presented.

5.1.1 Mobile base

The mobile base is the hardware responsible for enabling the movement of the robot in the environment. The platform used as a mobile base is the Create3, developed by iRobot. It is compatible with Wi-Fi, Bluetooth, Ethernet, or USB connections.

Figure 5.2: Create3 mobile base [85].

The entire operating software is implemented on ROS2. This implies that sensors data or actuator commands are sent or received using this framework. To ensure the possibility of ROS1 development, the mobile base uses a *'bridge'* that enables real-time communication between the different ROS versions. This approach enables the development of navigation algorithms in ROS1 that are fully compatible with the integrated software [85].

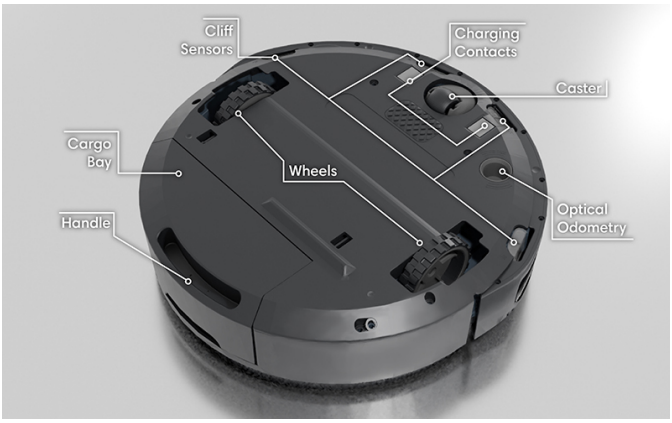


Figure 5.3: Bottom view of the Create3 mobile base.

As it can be seen from the fig. 5.3, Create3 has two independent drive wheels. To ensure greater stability during movement, the mobile base also has a 'front caster', as the centre of gravity of the base is moved forward in relation to its vertical axis. Each wheel is equipped with current sensors, encoders and optical odometry sensors. Data from these sensors are integrated with data from the IMU (Inertia Measurement Unit) to generate a fused odometry estimate. Lastly, the entire platform is powered by a separate and independent battery which does not power any other components of the robot.

5.1.2 Intel NUC NUC8i3BEH Mini PC

The robot's processing system, which enables its operations and manages the sensors, is a NUC computer with the following specifications:

- 8th Gen Intel Dual-Core i3;
- 8GB DDR4 Ram;
- 240 GB Solid State Drive (SSD);
- Intel Iris Plus Graphics 655;
- Wi-fi;
- Bluetooth 5.0;
- Gigabit Ethernet;
- USB;
- Thunderbolt 3;

- Ubuntu 20.04;

Figure 5.4: Intel NUC NUC8i3BEH Mini PC

In contrast to the mobile base, the NUC is powered by a separate 50000 mAh secondary battery, responsible for powering other robot components that will be discussed below.

5.1.3 WidowX-250 Robot arm

WidowX-250 Robot arm is a manipulator with six degrees of freedom. It is controlled with 9 smart servos from DYNAMIXEL-X Series Actuators developed by Robotis. Thanks to these features, it guarantees superior maneuverability, reaching a maximum distance of 650 mm and full 360° rotation. Furthermore, it ensures a working payload of 250 g with a recommended extension of no more than 50%.

Figure 5.5: WidowX-250 6DOF Robot arm.

Since our main objective is to study social navigation without necessarily including objects manipulation, the robotic arm was not used for the purposes of this thesis.

5.1.4 RPLIDAR A2M8

RPLIDAR A2M8 is a 360 degree 2D LiDAR designed for indoor environments. It is mounted on top of the robot, at a height of approximately 63 cm from the floor. Its high rotation speed enables it to take up 8000 laser ranging samples per second, with a maximum identifiable distance of 12 meters. The resulting 2D point cloud data can be used for both SLAM and navigation. The lidar was a critical tool for conducting the simulation campaign. It enabled us to accurately execute the SLAM and nearest obstacle detection phases, both of which are crucial for evaluating the performance of the navigation algorithms.

Figure 5.6: RPLIDAR A2M8.

The main drawback of this type of LiDAR is its inability to provide vertical resolution. As a 2D LiDAR, it can only scan in the plane perpendicular to its rotational axis. This implies that it is unable to detect obstacles with a lower height than the laser's mounting position on the robot, which limits its ability to explore and identify the environment. However, during the simulation phase, we have considered an environment with walls high enough to overcome this limitation.

5.1.5 Intel RealSense Depth Camera D435

The RealSense Depth Camera D435 is a 3D camera developed by Intel. To reconstruct a three-dimensional environment and provide depth data in real time, It uses two synchronized stereoscopic cameras to capture the image from different angles. It finds its primary application in robotics, virtual reality, and augmented reality applications [86]. Additionally, it can also be utilized in both indoor and outdoor environments [87].

Figure 5.7: Intel RealSense Depth Camera D435

- ***Depth sensor:*** The D435 is equipped with an infrared depth sensor that measures the distance between the camera and objects present in the given scene. This enables obtaining an accurate depth map of the objects captured;
- ***RGB camera:*** The D435 has a built-in colour (RGB) camera that offers a resolution of 1280x720 and a maximum frame rate of 90 fps;

The device is mounted on the Locobot, approximately 53 cm above the floor and is primarily used during the SLAM phase. Subsequently, the data obtained from the camera is integrated with the one acquired from the LiDAR to create a static map that will be used for actual navigation.

Chapter 6

ROS

To simulate and test new navigation algorithms, it is necessary to use a framework that allows communication between the different simulated components of the robot. In this context, **ROS** is a very useful framework due to its open source nature and the numerous tools available. These features provide the flexibility to work with both simulated and real robots. In particular, one of the most important tools integrated in ROS is the **navigation stack**, a set of software components that allow the movement of the mobile robot and the development of new navigation algorithms.
In addition, ROS allows direct communication with Gazebo, the simulator used to test the three navigation algorithms (more details in chapter 7).

Since the Game-theoretic Social Force Model requires a neural network to approximate the operation of the Differential Evolution algorithm, **Tensorflow** was chosen. It is a framework that allows the development of different machine learning models. Once the neural network was developed, ROS was used to ensure communication between it and the navigation stack.

Therefore, this chapter aims to provide a description of the underlying **concepts of ROS**. Then, the different components of the **navigation stack** will be briefly described, focusing mainly on the "*move base*" and its plugins. Both are necessary for the implementation of the proposed navigation algorithm. Finally, the concepts behind **Tensorflow** and the methodology used for the integration of the neural network into ROS will be given.

6.1 Overview

The Robot Operating System (ROS) [88] is an open source meta-operating system used for the development of various robotic applications. It was developed by Willow Garage in 2007 and is currently maintained by the Open Source Robotics Foundation. It is slowly becoming the standard technology for developing and programming robots in both industry and research. Despite the name, ROS is actually a framework (sometimes called middleware) that provides a set of tools and functionalities needed to properly develop and operate a robot. Some of them are:

- *Hardware abstraction:* through ROS, each hardware component of the robot can be thought as a software component that can be programmed;

- *Multiprocess communication and management:* to easily enable and manage communication among the different components and devices of the robot.

Before the introduction of ROS, robot design was extremely complex challenge because each new application required the creation of a new and specific software infrastructure and drivers, which were essential to ensure communication between the various devices (sensors, actuators, programs). This situation led to a constant *"reinventing the wheel"* situation, where the entire process had to be started from scratch each time, resulting in a significant reduction in innovation capacity.

ROS has been developed to mitigate this problem. In fact, its main objective is to promote 'code reuse' by supporting a modular infrastructure based on open-source code [89]. This distributed infrastructure consists of several processes (also called Nodes, described in the next paragraph) that can be designed independently and loosely coupled at runtime. These processes can be grouped into *Packages* (sets of nodes) and *Stacks* (sets of packages), which can be easily shared and distributed through *Public Repositories* over the Internet. This strategy allows developers to focus mainly on the innovative phase of the development process rather than the integration of heterogeneous robotic components, ensuring a significant increase in efficiency.

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			June 27, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)

Figure 6.1: Different ROS distribution over the years, from 2016 until today.

For this reason, in this thesis we have chosen to use ROS Noetic, the latest version and currently the only one that is still continuously supported and updated. To better understand its working principles, some key concepts and types of communication implemented in ROS will be explained below.

6.2 Basic concepts and Communication paradigms

Conceptually, ROS tries to abstract the robot as a "network" where each device, sensor, actuator can communicate with the others through standard and efficient communication protocols. This network will basically consist of two basic ROS components:

- ***Nodes:*** a ROS node is a software process that usually performs a single task (such as reading data from a sensor or performing localization). In general, a robotic system consists of several nodes that provide the hardware abstraction mentioned above. Therefore, the use of ROS nodes provides multiple benefits to the overall system, allowing a reduction in the complexity of writing code and avoiding monolithic software architectures. In addition, the independent development of each node guarantees the ability to write the associated code in both C++ and Python, ensuring greater flexibility in the development process;

- ***Master:*** it is the entity that manages communication between the different components of the robot, allowing decentralized communication also known as peer-to-peer communication. Its main function is to assign unique names to

each active node in the ROS system and to record all the relevant information about them. Examples of such information could be the name of a second node with which it needs to exchange information or the type of information and how it is exchanged from one node to another. By recording this information, the master can then enable the ROS nodes to "locate" each other, ensuring a direct mutual communication.

In a ROS system, communication among nodes can basically take place via two possible paradigms:

1. ***Publish/subscribe:*** it is a type of paradigm that guarantees asynchronous communication among nodes. Thanks to its logic, it ensures a decoupling between who produces the information and who consumes it. Indeed, the receiver of the information does not need to know the identity of who produced it. This guarantees both extreme flexibility in the management of communication (especially in many-to-many communication) and near real-time behaviour of the system.

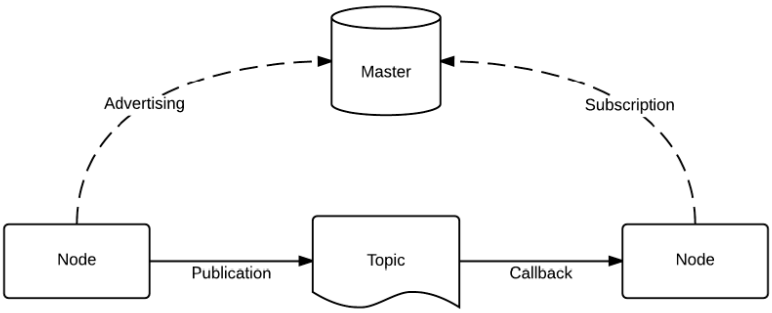


Figure 6.2: Conceptual representation of the Publish/Subscribe communication paradigm in ROS.

This type of communication is implemented in ROS through the use of topics and messages:

- ***Topic:*** a ROS topic corresponds to a named communication channel through which nodes can exchange information in the form of messages. Therefore, we call *Publisher* the node that publishes the information on the topic, while *Subscriber* the node that subscribes to the same topic to receive the information. Fig. 6.2 shows a simplified scheme of how this communication process works. Both nodes must then inform the ROS master of the name of the topic they wish to publish or subscribe. However, each node can subscribe and publish to more than one topic at the same time, ensuring a multi-communication process;

- ***Message:*** a ROS message is a data structure used by a node to exchange information with another node by means of a topic. Each message is characterised by a "type fields", which can be standard (generic data types such as integer, boolean, double, etc.) or structured, i.e. made up of several data types nested together to form more complex data structures;

2. ***Request/response:*** it is a paradigm whose operation is based on two elements: the *client* (which executes the request) and the *server* (which receives the request, processes it and returns the response). It guarantees a synchronous communication since the client, after sending the request, waits until it receives a response from the server. Unlike the approach described above, there is no decoupling between the nodes that produces the information and the nodes that consumes it. because the two actors must know each other in order to communicate. This type of paradigm is implemented using services.

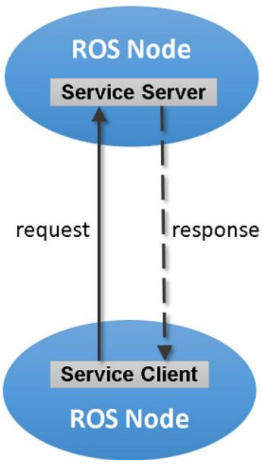


Figure 6.3: A schematic representation of the request/response communication paradigm implemented in ROS.

- ***Services:*** A ROS service corresponds to a specific type of computation or, more generally, action that is exposed/executed by the server. Each type of service is defined by a pair of messages (characterised by their specific data type), one for the request and the other for the response. Therefore, following the logic described above, we will call *ROS server* a ROS node that provides a service, while *ROS client* a node that requests/calls this service and waits for the response. In fig. 6.3, the working principles of how a ROS client and a ROS server communicate are described.

6.3 Navigation stack

6.3.1 Overview

The navigation stack is a set of ROS packages and nodes that allow the robot to perform the various stages of autonomous navigation described in Chapter 2. It exploits information from the odometry and sensors to generate velocity commands. These commands are subsequently processed and converted by the mobile base controller into commands that can be sent to the actuators, which will enable the actual movement of the robot.

The navigation stack:

- is intended for both differential drive and holonomic wheeled robots, typically with circula or square shape;
- requires at least one LiDAR. The sensor will be used for both mapping and localization phases (SLAM);

As a stack, it consists of different packages with specific purposes:

- ***Gmapping:*** it is the package that enables the use of sensors on the robot (LiDAR and stereo camera) to perform the mapping phase. This generates a static map of the environment [90];
- ***Map server:*** once the mapping phase is complete, this package is responsible for storing and later providing the static map obtained earlier, which will be used for both localization and path planning [91];
- ***Amcl:*** using LiDAR and the static map, this package performs the Monte Carlo localization algorithm to identify the robot's position in the environment [92];
- ***Move base:*** it performs the path and trajectory planning phase by using information from the other packages in the stack [93].

The primary goal of this thesis is to develop navigation algorithms that are socially acceptable. Therefore, in the following section, the discussion will focus exclusively on the *move base* package composed by Nodes with different functionalities.

6.3.2 Move base package

The main purpose of the *move base* is receiving a Goal pose as input (which will contain the final position and orientation that the robot will have to reach) and returning as output the necessary linear and angular velocity commands to attain that specific final pose [93].

As shown in fig. 6.4, the package consists of several interconnected nodes that communicate with each other and with the other packages mentioned above through specific ROS topics and messages.

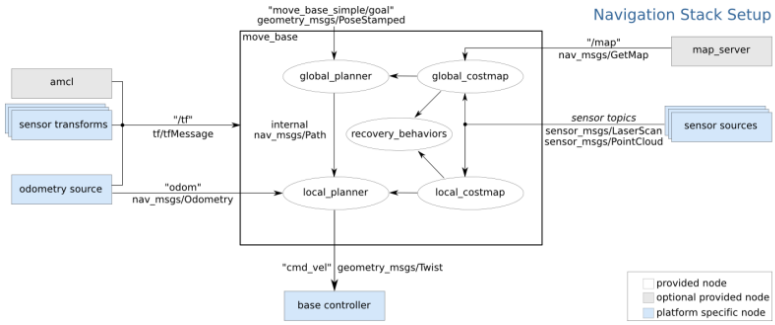


Figure 6.4: Navigation stack packages and nodes with their related ROS topics and messages [93].

Below, a high-level overview of the operation of each nodes is provided.

Global and local costmap

A costmap [94] is a spatial representation of the surrounding environment, created using an occupancy grid. It divides the perceived environment into cells, each with an assigned value that represents the cost or accessibility of that region of the environment for the robot. This information allows the robot to determine whether a given cell is occupied, free or unknown.

When it is initialized, the cost map will automatically subscribe to the sensor topics and will start to update itself accordingly. The acquired data will then be used to perform two fundamental operations:

- ***Marking***: to insert obstacle information into the costmap, i.e. the cost of each individual cell;
- ***Clearing***: to remove obstacle information from the costmap;

To provide more accurate information, the *move base* employs the sensor data to execute the inflation phase. This refers to the process of extending cost values from the occupied cells, which decrease as distance increases. Indeed, each cell has a cost ranging from 0 to 254. Based on the individual value assumed by each cell, it will be possible to identify areas around obstacles that, although free, still represent a danger to the robot's movement.

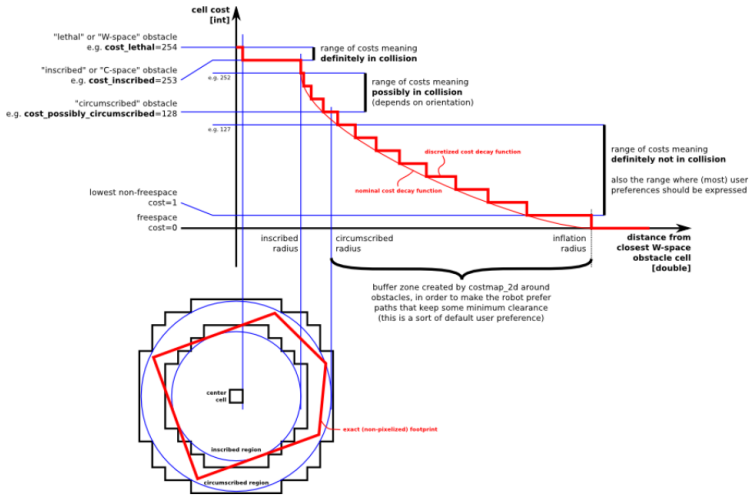


Figure 6.5: Costmap inflation process [94]. The value associated with the occupied cell is the maximum cost (254). This value decreases with increasing distance from the center of the obstacle. Lower cost represents a lower probability of collision hazard.

As shown in fig. 6.4, the navigation stack uses two types of costmap. The method of assigning and updating costs is the same for both types. However, they differ in the way they are initialized and the types of information they can provide. Consequently, each costmap will have a specific purpose:

1. ***Global costmap:*** Its aim is to identify the elements in the environment that represent actual static obstacles. To build this costmap, the data from sensors is combined with a static map created during the mapping phase. As a result, the size of the costmap will match the size of the static map used as reference. In general, it is normally used in conjunction with a localization system, such as the "amcl package" mentioned above. By performing the localization process, it will be possible to use sensors data both to identify the position of the robot and to update the costmap as it moves through the environment. When performing SLAM directly, the procedure will be the

same. In this scenario, the costmap will be acquired by incorporating data from the sensors and the map being constructed;

2. ***Local costmap:*** The purpose of the local costmap is to recognize dynamic objects that might be close to the robot. Contrary to the global one, the local costmap will be smaller, directly definable by the user, and will always be centred with respect to the robot, moving in solidarity with it as it moves through the environment. To create the local costmap, only the sensor data within the defined costmap area is used.

After their creation, these costmaps will be given directly to the corresponding planners in order to proceed with the next planning phases.
Although the local and global costmaps are extremely useful tools for recognizing static and dynamic objects, the navigation algorithms developed in this thesis do not strictly rely on their use. Indeed, the position of the closest obstacle is obtained directly from LiDAR, as well as the pedestrians positions and robot velocity are obtained from Gazebo related topics (more details in chapter 7).

Global planner

As can be seen from fig. 6.4, the node responsible for implementing the global planner receives as an external input the target pose that the robot has to reach. The global planner [95] uses the information available from the global cost map and applies a path planning algorithm with the aim of creating a safe path that allows the robot to move from the initial position to the goal, avoiding both fixed obstacles and neighbouring areas that may be potentially dangerous.

By default, ROS implements two types of global planner algorithms:

- *A**;
- *Dijkstra*;

The explanation of how such algorithms work is beyond the scope of this thesis.

Local planner

The local planner receives as input the path generated by the global planner. It then applies a local planning algorithm, using both the odometry data and the information provided by the local costmap, with the aim of generating velocity

commands to follow the previously generated global path [96]. If dynamic obstacles are nearby the robot, the local planner must modify velocity commands to change the path in order to safely avoid these obstacles.
By default, ROS implements two local planning algorithms:

- *Dynamic Window Approach (DWA)*;
- *Trajectory Rollout*;

Also in this case, the description of such algorithms is beyond the scope of this thesis.

Recovery behaviours

The *move base* package also implements default recovery behaviours [93]. They are a series of actions that the robot can perform when it perceives itself as stuck, i.e. unable to find a free path and produce valid velocity commands. This scenario arises when the robot is surrounded by costmap cells whose values do not allow it to move safely. These values could represent actual obstacles as well as some possible errors that may occur during the costmap updating. By employing these recovery behaviours, the *move base* tries to clean up the cost of the cells in order to verify whether obstacles are actually present.
Basically, the *move base* package performs two types of recovery behaviours:

- ***Clear costmap:*** it uses the static map and sensor data to regenerate the values of the individual cells of the global and local costmap. In this way, it is possible to remove any wrongly detected obstacles and restore a correct representation of the environment in order to resume navigation [97];
- ***Rotate recovery:*** the robot performs a 360 degree rotation in place to clean the space and try to find a free direction for resuming navigation [98];

In general, the behaviour of the robot can be described as a state machine:

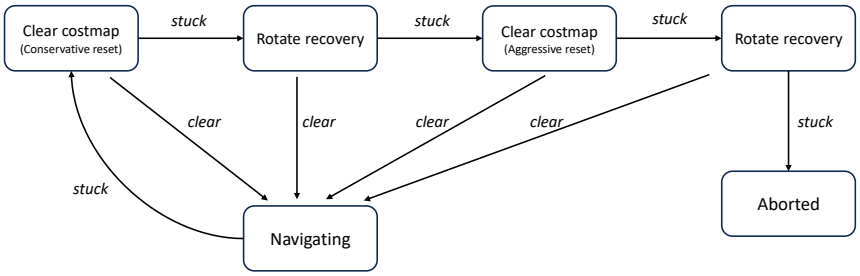


Figure 6.6: Schematic representation of the move base working conditions as a state machine.

Therefore, if the robot perceives itself as stuck, it will perform one recovery behaviour at a time and in a sequential manner as shown in fig. 6.6.

It is important to note that the “clear costmap” recovery behaviour can be performed in two different ways [93]:

- *Conservative reset:* the robot cleans the costmap by removing obstacles outside a user-specified region;
- *Aggressive reset:* the robot cleans the costmap by removing all obstacles outside the rectangular region in which can rotate in place;

After each recovery behaviour, the *move base* will attempt to compute a new path integrating the information from the new costmap. If the planning is successful, the *move base* returns to the navigation phase and resumes generating velocity commands via the local planner. Otherwise, the next recovery behaviour will be executed. If all recovery behaviours fail, the robot will conclude that the goal is not feasible and will abort the navigation.

Global and local planner plugins

The move-base package is entirely implemented in C++. This feature ensures better efficiency and speed, enabling the robot to achieve excellent real-time performance. To provide greater flexibility, ROS allows new planners, both local and global, to be implemented through the use of plugins, which are portions of code used as "extensions" to an existing program. The use of plugins makes the implementation and testing phases of different planners easier. It also allows the user to customize the behavior of each single planner according to the needs. To properly interact with the move base package, such plugins must possess standard interfaces (functions), which differ depending on the type of planner to be developed (fig. 6.7). Through these interfaces, different planners and navigation algorithms can be developed independently while still preserving the key functionality provided by the move base.

The following is a brief description of the interfaces needed to develop plugins for global and local planners [99]:

- ***Global Planner:***
  - *initialize:* is the function that allows to initialize the global planner and set the various parameters necessary for its operation. It returns true if the initialization is successful or false otherwise;

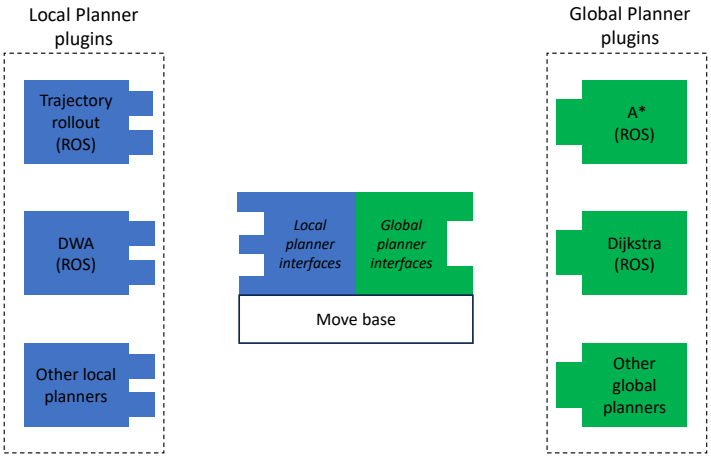


Figure 6.7: Schematic representation of move base local and global planner interfaces and plugins.

  - *makePlan:* it encapsulates the operation of the global planning algorithm. This function is called every time the robot receives a new goal. Knowing the current position of the robot, this function computes a feasible free path to reach the final position. It returns a boolean flag as output, which can be true if the path is created correctly or false otherwise. In case the plan has been computed correctly, the function will send the plan to the local planner via ros topic;

- ***Local Planner:***

  - *initialize:* as in the global planner, this function allows the local planner to be initialized. Even in this case, it returns a boolean value according to whether the initialization is successful or not;
  - *setPlan:* this function is called immediately after the "*makePlan*" function described above. It is responsible for receiving and setting the path received from the global planner. It returns true or false according to whether the obtained global plan can be set properly or not;
  - *computeVelocityCommands:* this function encapsulates the operation of the robot's local navigation algorithm. It is called periodically until the robot reaches the specified goal pose. Its purpose is to compute a new velocity command at each time instant. Then, such a command will be sent to the move base controller and subsequently transformed into a direct command to robot's actuators. The function returns true if a feasible speed command is computed successfully or false otherwise;

– *isGoalReached:* whenever a valid velocity command is produced, this function is executed in order to check if the goal has been reached. It returns the value true or false according to whether or not the final pose has been achieved;

The local planning algorithm described in Chapter 4 has been developed so that its operation conforms to the described interfaces required to interact with the base move.

6.4 TensorFlow

As stated in Chapter 4, the developed navigation algorithm employs a neural network to generate the parameter sets required for the application of Game Theory. TensorFlow was required to construct, train and test this neural network, which was later integrated into ROS.

6.4.1 Overview

TensorFlow [100] is an open-source machine learning framework developed by Google Brain. It enables the creation, training, and deployment of diverse machine learning models, such as deep neural networks, across multiple platforms and devices.

This framework is based on two basic concepts:

- ***Tensor:*** It represents a general multidimensional array. Tensors can be used to represent scalars, vectors and matrices, including those of order greater than 2. In TensorFlow, all data is represented as a tensor. Depending on the data represented, the tensor will be characterized by a "*shape*", that summarizes its organization, and a "*rank*" that indicates the dimension of the tensor. Some examples of tensors are shown in fig. 6.8 ;

- ***Graphs:*** These are patterns that TensorFlow builds to organize the mathematical operations necessary for training and testing the developed model without actually performing the computation. A graph can be represented as a collection of nodes and arcs. Nodes are the actual mathematical operations, like multiplying matrices, applying activation functions, and optimizations. Arcs represent the data (tensors) that the nodes take as input and return as output after processing. Therefore, through the graph, it is possible to describe the computational flow that will be performed on the tensors. An example is shown in fig. 6.9;

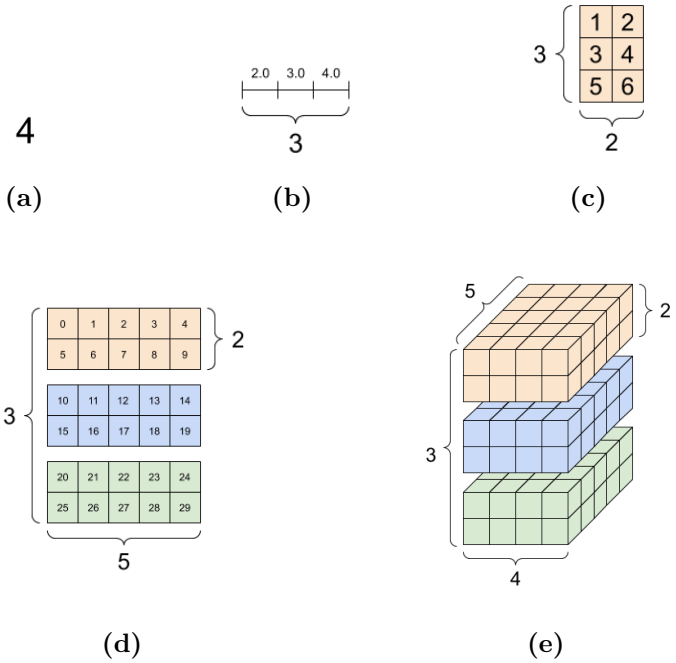


Figure 6.8: Some examples of tensors [101] with different shapes and ranks: (a) rank=0, shape=[ ] ; (b) rank=1, shape=[3]; (c) rank=2, shape=[3,2]; (d) rank=3, shape=[3,2,5]; (e) rank=4, shape=[3,2,4,5].

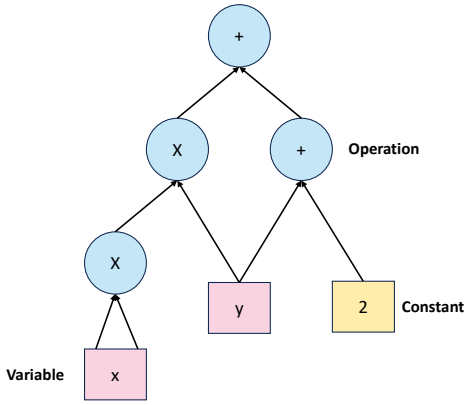


Figure 6.9: Example of the TensorFlow computational graph of the function $x^2y + y + 2$ [102]. Nodes are represented in blue while arcs are represented in pink (variables) and yellow (constant).

6.4.2 ROS Integration

TensorFlow offers various APIs for several programming languages such as C++ and Python. This characteristic makes TensorFlow easy to integrate with the ROS environment. This enables the implementation of more complex robot functionality through the application of machine learning.
To ensure the proper functioning of the developed algorithm, communication between the neural network and the local planner was necessary. This allowed the latter to send the correct inputs and receive the corresponding outputs, i.e. the parameter sets for generating the trajectories.

It's possible to summarize this integration process in the following three steps:

1. ***Building the Neural Network:*** Using TensorFlow, it's possible to develop, train and test a neural network that is completely independent from ROS interfaces. At this stage, it will be necessary to specify the different characteristics of the model, such as:

   - *number of hidden-layers;*
   - *number of neurons for each layer;*
   - *activation functions;*
   - *cost function and optimization method for the training phase;*

   The model must be able to handle both input and output tensors, which will have different ranks and shapes. Once the model has been tested, it is possible to save it, i.e. to store the weights and biases of the different layers obtained as a result of the training phase. This makes possible to delete and reload the exact same model, even in different code scripts.

2. ***Creating a ROS node:*** In order to integrate the neural network into the ROS environment, a simple standard node must be created. When initialized, the node will have the task of loading the neural network saved in the previous step. In this way, the model will be able to use all possible standard ROS functionalities and interfaces to communicate with the other components of the robot.

3. ***Design of ROS interfaces:*** After loading the neural network into the node, it will be necessary to develop the actual communication interfaces with the local planner, such as topics and services. In this way, the neural network will be able to receive inputs, make predictions and send outputs to the various

other nodes in the system.
Given the method in which the neural network is deployed, the most efficient choice is to allow communication between the node and the local planner by means of ROS services.
Therefore, the node implementing the neural network will act as servers and expose a single service. This service ensures the standard operation of the neural network. It is responsible for receiving input from the local planner, executing the corresponding prediction, and returning as output the parameter sets necessary for the operation of the navigation algorithm.

The fig. 6.10 shows the final system architecture with the communication interfaces between the neural network and the move base package:

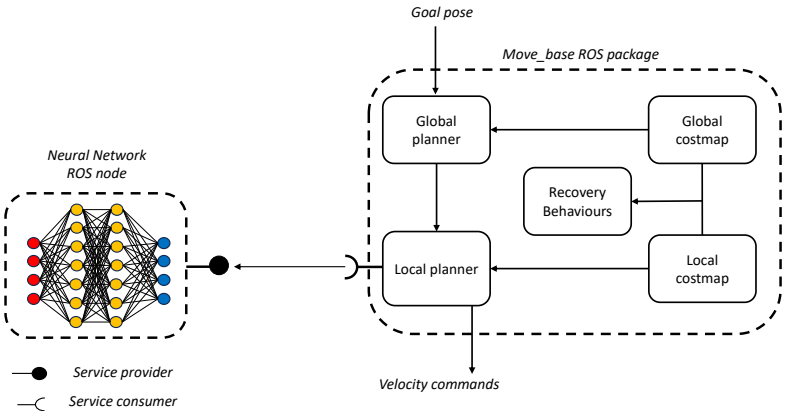


Figure 6.10: Representation of the final architecture implemented in ROS.

Chapter 7

Experimental setup

In this chapter we expose in detail the tools and methods used to develop the experimental simulation campaign. In the first section, we present a brief overview of the two main simulation software (Gazebo and Rviz). Then, we provide a complete description of the design of the simulations used to ensure the interaction between robots and pedestrians. Finally, we briefly describe the metrics used to quantitatively assess the social acceptability ensured by the algorithm.

7.1 Simulation tools

7.1.1 Gazebo

Gazebo [103] is an open-source simulator that is commonly used in robotics in conjunction with the framework ROS. By utilizing various physics engines, it offers a remarkably accurate 3D simulation environment which is crucial in the development, testing, and validation of complex robotic systems.
With Gazebo, it is possible to model the physical behaviour of various kinds of robots (such as mobile robots, drones or anthropomorphic robotic arms) and objects in the environment. Furthermore, Gazebo allows virtual devices such as cameras, lidar and ultrasound sensors to be integrated, providing realistic data for robotics applications.
Thanks to its integration with ROS, robots in Gazebo can be controlled using nodes and standard ROS interfaces such as topics and services, exactly as with real robots. This greatly simplifies the transition from simulation to the real world. In this way, it ensures the possibility to design and test the proper functioning of the robot system in a virtual environment before moving to real implementation, leading to benefits such as reduction in development costs and time.

SFM plugin for pedestrians

To test and evaluate navigation algorithms through simulations, it is necessary to create a virtual environment whose elements of interest, in this case pedestrians, are as similar as possible to the real ones.
In its standard version, Gazebo offers the possibility to include pedestrian models, simply called "*actors*", in the simulation. They will be able to move from a starting point to an end point in the virtual environment along more or less complex trajectories defined by a series of waypoints that the individual actor must reach in a given time interval.
However, this procedure does not guarantee a "rational" behaviour of simulated pedestrian. They will follow the defined trajectory despite the presence of potential obstacles such as walls or other pedestrians. This results in a type of navigation that differs significantly from the previous assumptions, where navigation was modelled as a game in which agents (pedestrians and robots) make rational decisions based on the situation. For this reason, we used a different approach to ensure robot-pedestrians interactions during simulations.
In fact, thanks to its open-source nature, Gazebo enables the use of plugins, i.e. software components that allow users to add specific features and robot models to the simulator. In this project, an open-source SFM plugin [104] was used to simulate real pedestrian motion in Gazebo. It is based on the original model developed by Helbling et al. [15] and its later extensions [105][106]. This plugin allows agents to move around the virtual environment according to the logic of the Social Force Model described in Chapter 3. In the simulation, each agent has its own individual parameters. In this way, besides ensuring a certain rationality in choosing movement directions, it is possible to define different navigation logic for each pedestrian, resulting in a more realistic simulation.

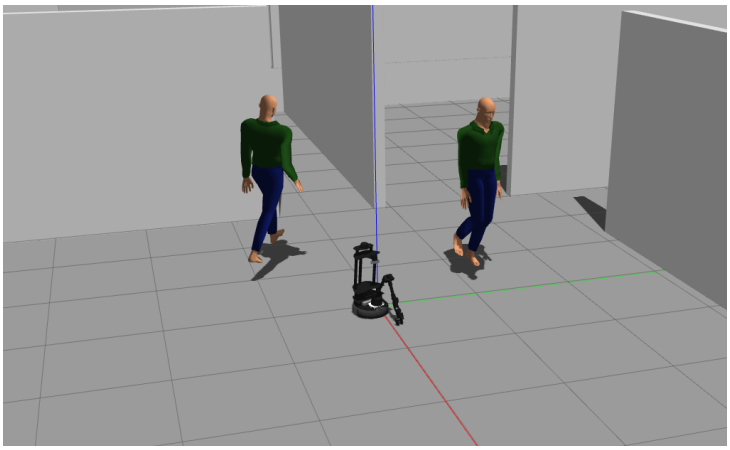


Figure 7.1: An example of two pedestrians moved by SFM plugins in Gazebo

7.1.2 RViz

RViz (Robot Visualization) [107] is a 3D visualization tool included in the ROS ecosystem. It is fundamental for the design of a robot, since it provides specific functionalities that are particularly useful, such as:

- ***Real-time visualization:*** through RViz it is possible to graphically visualize a wide range of data from sensors such as LiDAR and cameras, as well as maps and planned trajectories. These data are read directly from the various topics through which the active nodes of the system exchange messages containing the necessary information for the correct behaviour of the robot. It is also possible to design and visualize the model of the robot itself or any objects that may be present in the environment;

- ***Planning and simulation:*** the tool provides the possibility to specify at runtime the *Goal pose* that the robot has to reach and to display it directly on the map. By reading specific topics, RViz can be used to visualize the path generated by the global planner, possible changes made by the local planner, and the real-time movement of the robot in the environment;

- ***Debugging:*** the ability to visualize the data and the resulting real-time behaviour of the robot makes RViz a powerful resource for debugging and testing, both in a simulated environment, such as Gazebo, and with the real robot;

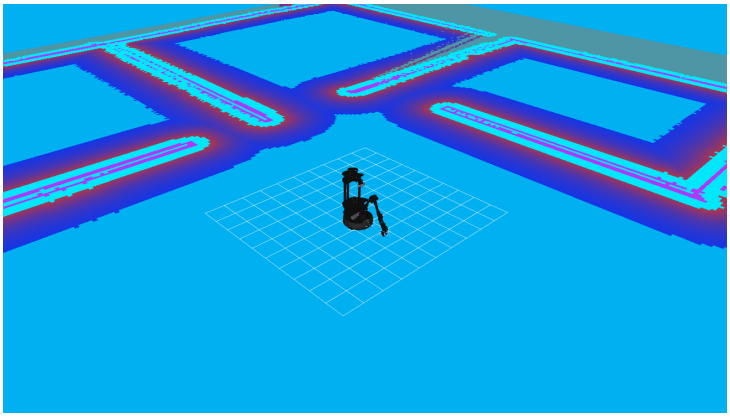


Figure 7.2: Example of locobot visualization in Rviz with the global costmap related to the obstacles (walls) in the environment.

7.2 Experiments and methods

The experimental phase aims to test the performance of the developed navigation algorithm in terms of naturalness and comfort in a simulated environment. The final goal is to evaluate social acceptability. For an accurate evaluation, the game-theoretic Social Force Model has been compared with two other widely used state-of-the-art approaches: the Social Force Model and Optimal Reciprocal Collision Avoidance (ORCA) [20].
In total, the experimental campaign involved 180 simulations (trials) for each of the considered algorithms.
The 180 simulations has been divided into two sets of 90 simulations each. Each set of simulations has been performed considering two different navigation scenarios. The first one includes the presence of three pedestrians while the second one involves four pedestrians.
Both scenarios are designed to represent realistic navigation situations under low crowd density conditions, while still providing some variability in possible interactions between robots and pedestrians.
In these contexts, all pedestrians move from an initial position to a final position by means of the SFM Gazebo's plugin described above. The use of this plugin is essential to make simulated pedestrians capable to avoid the robot enhancing realistic evaluations of socially aware navigation. As a matter of fact, there are a variety of benchmarks in literature for evaluating the performance of social navigation algorithms. One of the most important ones is SocNavBench [108]. It is a simulator-based benchmark that uses real-world pedestrian trajectories to simulate the movement of pedestrians. The interaction among robot and pedestrian is made by replacing one pedestrian in the crowd with the robot. However, this approach doesn't take into account how the robot motion affect the pedestrian's path [109]. All pedestrians will apply a passive avoidance. They will be able to avoid each other but not the robot. This makes it difficult to obtain realistic evaluations of socially aware navigation.
On the other hand, modeling the movement of pedestrians through SFM makes it possible to ensure an "active" pedestrian avoidance. Therefore, they will be able to recognize the robot as an obstacle and avoid it.

7.2.1 Simulated environment description

All simulations have been performed in a room modeled in Gazebo having dimensions 8.5m x 5.5m. The total area is approximately 47 m^2, comparable to environments used in literature to perform real-world experiments [7][81].
Therefore, the room has been divided into six fictitious zones, as shown in fig. 7.3. For each simulation, zone F is used to allow the correct spawning of the robot in

the environment. Zones A, B, C, and D are used to spawn pedestrians. Before the start of each simulation, every pedestrian in the scenario chooses its own spawn zone from the available ones. This choice is characterized by a uniform distribution. Each zone $\mathcal{Z}$ is defined by coordinates ranges:

$$\begin{aligned} \mathcal{Z}_x &= \{x^z_{min}; x^z_{max}\} \\ \mathcal{Z}_y &= \{y^z_{min}; y^z_{max}\} \end{aligned} \tag{7.1}$$

By knowing these ranges, the *j-th* pedestrian can randomly generate its initial position within the established area z. The generation of the initial position is performed discretizing the coordinate ranges as:

$$\begin{aligned} x^z_j &= x^z_{min} + rand(0{,}10)\frac{x^z_{max} - x^z_{min}}{10} \\ y^z_j &= y^z_{min} + rand(0{,}10)\frac{y^z_{max} - y^z_{min}}{10} \end{aligned} \tag{7.2}$$

where $rand(0{,}10)$ represents a uniform random number between 0 and 10. Overall, such zones are mutually exclusive. This means that a zone chosen by one pedestrian cannot be chosen as a spawn zone by another individual.

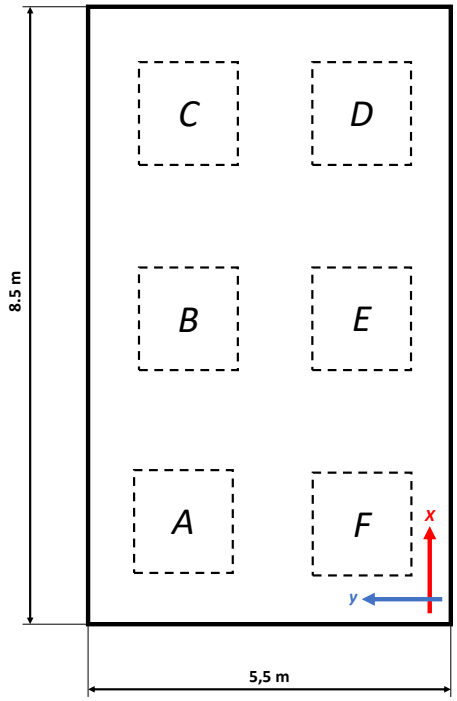


Figure 7.3: Simulated environment and related spawn zones

To ensure that each simulation is characterized by at least one pedestrian-robot

interaction, each spawn zone is associated with specific goal zones. The relationship between the spawn zones and the corresponding goal zones is described in table 7.1. As can be seen from fig. 7.3, the target zones are placed in opposite positions to the related spawn zone, ensuring a greater number of encounters while navigating.

Spawn zone	Goal zone
A	D, E
B	D, F
C	E, F
D	A, B
F	C

Table 7.1: Goal zones associated to each spawn zone.

Once its initial position is established, each pedestrian randomly chooses the goal zone following the order described in tab. 7.1 and generates the final position to be reached. This process is performed by means of the same methodologies applied for the spawn zone and the initial position. Even in this case, the goal zones are mutually exclusive in order to avoid that multiple pedestrians can have a common target area.

Since the robot can spawn only in zone F, the goal position is always generated in zone C. Therefore, in each simulation, the robot aims to reach the target position located at the opposite corner of the room, while trying to avoid pedestrians as naturally and comfortably as possible.

7.2.2 Evaluation metrics

In Chapter 1, we emphasized how developing a navigation algorithm that can guarantee a certain level of naturalness and comfort is crucial to enhance social acceptability of the robot. To achieve this goal, it is essential to evaluate the navigation algorithm using specific metrics to quantify the level of naturalness and comfort that it is able to guarantee.
A variety of metrics designed for this purpose exist in the state of the art. Each of these metrics attempts to measure different aspects associated with these two characteristics.
To ensure a comprehensive evaluation of the algorithm proposed in this thesis, we performed a frequency analysis to select the most widely used metrics in the state of the art. The results of this analysis are shown in fig. 7.4 and fig. 7.5.

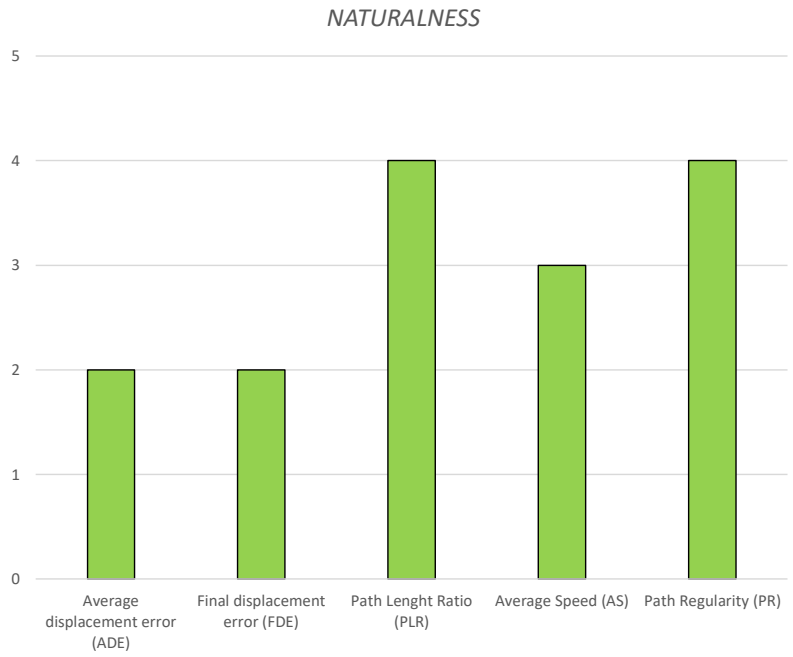


Figure 7.4: Frequencies of the most commonly used naturalness metrics in literature.

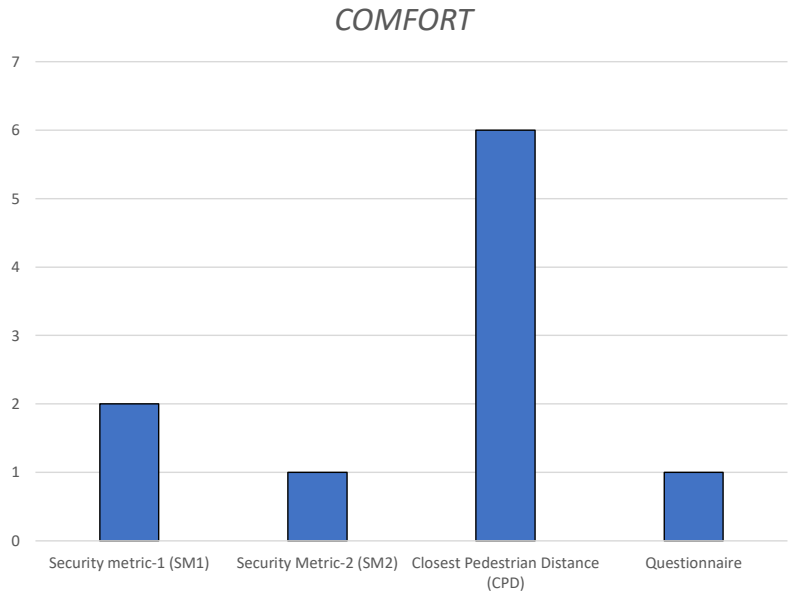


Figure 7.5: Frequencies of the most commonly used comfort metrics in literature.

NATURALNESS	
Metric	Articles
Average displacement error (ADE)	[110] [111]
Final displacement error (FDE)	[110] [111]
Path Length Ratio (PLR)	[108] [110] [43] [112]
Average Speed (AS)	[108] [110] [113]
Path Regularity (PR)	[108] [110] [43] [114]

Table 7.2: References of the analyzed naturalness metrics

COMFORT	
Metric	Articles
Security metric-1 (SM1)	[115] [112]
Security metric-2 (SM2)	[115]
Closest Pedestrian Distance (CPD)	[115] [13] [21] [43] [109] [112]
Questionnaire	[110]

Table 7.3: References of the analyzed comfort metrics

The final metrics that have been employed to evaluate the algorithm are:

- ***Path Length Ratio (PLR)***
  it is the ratio between the straight line between the initial and final position of the robot and the actual path length of the robot trajectory:

$$PLR = \frac{\|\mathbf{p}_{robot}(T_{goal}) - \mathbf{p}_{robot}(0)|}{\sum_{t=1}^{T_{goal}} \|\mathbf{p}_{robot}(t) - \mathbf{p}_{robot}(t-1)\|} \tag{7.3}$$

  where T_{goal} is the number of time-steps needed for the robot to reach its goal. Hence, this metric takes values in the range [0,1]. In general, a higher PLR is preferred since it indicates that the robot tends to reach its destination minimizing the length of the path;

- ***Closest Pedestrian Distance (CPD)***
  it is the distance between the robot and the closest pedestrian j over the entire considered trajectory:

$$CPD = \min_{t,\ j} \|\mathbf{p}_{robot}(t) - \mathbf{p}_j(t)\| \tag{7.4}$$

  A higher CPD value indicates that the robot tends to stay further away from pedestrians. This ensures a higher level of comfort for humans;

- ***Average Speed (AS)***
  it is the average speed of the robot over the entire trajectory;

$$AS = \frac{\sum_{t=1}^{T_{goal}} \mathbf{v}_{robot}(t)}{T_{goal}} \tag{7.5}$$

- ***Path Regularity (PR)***
  it measures the (normalized) robot rotations during the navigation. It is computed as:

$$PR = 1 - \frac{\sum_{t=1}^{T_{goal}} |\theta(t) - \theta(t-1)|}{PI_{max}} \tag{7.6}$$

  where $\theta(t)$ represents the robot's orientation at a generic time instant t and the denominator PI_{max} is the normalizing factor computed as:

$$PI_{max} = \max_{\substack{SFM,\\ ORCA,\\ GTSFM}} \sum_{t=1}^{T_{goal}} |\theta(t) - \theta(t-1)| \tag{7.7}$$

  Such normalized index takes values in the range [0,1]. A value equal to 1 indicates a straight path from the initial position of the robot to the goal. A value closer to 0 denotes a great number of rotations during the navigation. Therefore, it is preferred to have values of PR closer to 1, since they represent a smoother robot trajectory.

Together, these metrics give us a complete view to evaluate the effectiveness of the navigation algorithm in a simulated environment, such as Gazebo.
All information about the positions of pedestrians and the positions and orientations of the robot has been recorded and saved as ROS bag files and then analyzed using Matlab.

Chapter 8

Test Results and Discussions

The results of our Monte Carlo numerical simulation are shown in fig. 8.1. Tables 8.1, 8.2, 8.3, and 8.4 report the mean values and corresponding standard deviations of each algorithm for each considered metric: PLR, CPD, AS, and PR, respectively. Our numerical validation shows that the GTSFM (Game-theoretic Social Force Model) has significant improvements in almost all metrics, especially compared to the SFM.

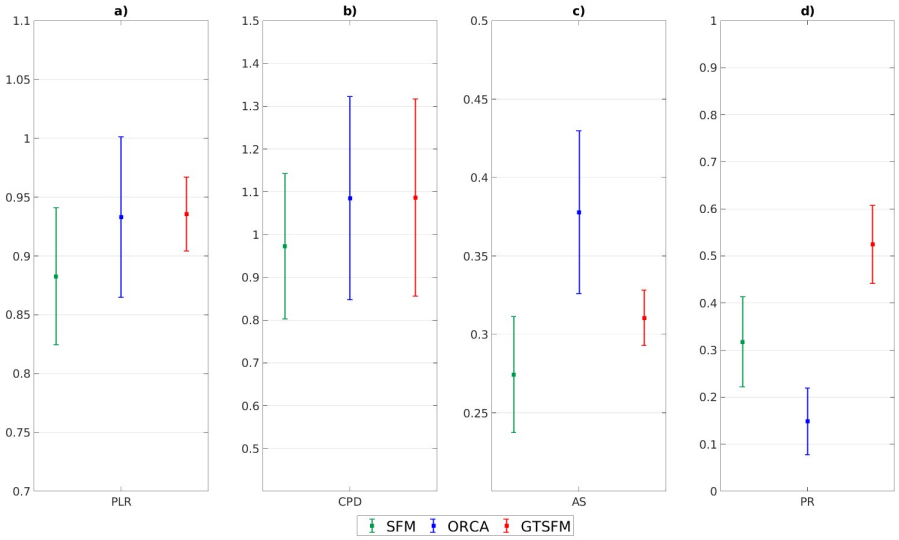


Figure 8.1: Mean value and standard deviation of the considered metrics of each tested algorithm: SFM (Social Force Model), ORCA (Optimal Reciprocal Collision Avoidance), GTSFM (Game-theoretic Social Force Model). The performance metrics are: a) PLR (Path Length Ratio), b) CPD (Closest Pedestrian Distance), c) AS (Average Speed), d) PR (Path Regularity).

Specifically, the GTSFM has a higher mean value of Path Length Ratio than the other two algorithms. This results in a more direct movement of the robot towards the goal, allowing a minimization of the total traversed path. Although ORCA has a very similar mean value to the GTSFM, it has a much larger standard deviation. This characteristic accounts for a large variability in the performance metric between simulations.
In addition, as can be seen from the fig. 8.1, this standard deviation can result in Path Length Ratio values greater than 1, which is the upper limit of the metric. They represent situations where the robot was not able to definitely reach the goal. However, this phenomenon can be explained by the distribution of the data after the 180 simulations. The majority of the ORCA-related PLR data is distributed above the mean value shown in fig. 8.1 and reported in tab. 8.1. However, within this data there are numerous outliers, i.e. individual simulations characterized by a very low PLR value. Therefore, the presence of such outliers results in a decrease of the mean value and a significant increase in the standard deviation.

PLR		
Algorithm	Mean	Standard deviation
SFM	0.8825	0.0582
ORCA	0.9331	0.0683
GTSFM	0.9356	0.0314

Table 8.1: Mean value and standard deviation of the PLR (Path Length Ratio) for each algorithm

In terms of Closest Pedestrian Distance, tab. 8.2 shows that both the GTSFM and ORCA have almost similar mean values and standard deviations between them, resulting in superior performance if compared to SFM. This suggests that both are able to maintain greater distances from pedestrians providing greater comfort than SFM to humans during navigation.
Although they are characterized by very similar values, it should be noted that the GTSFM has a slightly higher mean value than the ORCA and a slightly lower standard deviation.

Considering the average speed, ORCA provides higher mean value than the other two algorithms, but results in a very high standard deviation. Even in this case, the phenomenon can be explained by the presence of outliers that increase the overall standard deviation. This implies the inability of ORCA to effectively manage its speed in scenarios characterized by different numbers of robot-human interactions.

CPD

Algorithm	Mean	Standard deviation
SFM	0.9727	0.1697
ORCA	1.0847	0.2377
GTSFM	1.0861	0.2304

Table 8.2: Mean value and standard deviation of the CPD (Closest Pedestrian Distance) for each algorithm

On the other hand, it can be seen from fig. 8.1 that the GTSFM still exhibits a higher mean value than the SFM but more importantly a lower standard deviation than the ORCA (see tab. 8.3). This implies that the average speed guaranteed by the GTSFM remains about the same even in different scenarios. However, the lower standard deviation indicates a greater adaptive capacity of the GTSFM than the ORCA, even in cases with different numbers of robot-human interactions.
Although the results are promising, the obtained average speed values are still not comparable to human speed values. This is related to the physical constraints of the robot used for simulations, which has a maximum speed limit of 0.5 m/s.

AS

Algorithm	Mean	Standard deviation
SFM	0.2743	0.0370
ORCA	0.3778	0.0518
GTSFM	0.3104	0.0177

Table 8.3: Mean value and standard deviation of the AS (Average Speed) for each algorithm

The most impressive metric that highlights the greatest benefit of using game theory is Path Regularity. As can be seen from fig. 8.1, the GTSFM provides a significant improvement over both the SFM and the ORCA. The ORCA's Path Regularity mean value is consistent with the mean values of the previous described metrics, specifically the Path Length Ratio and Average Speed.
In general, ORCA determines paths characterized by very irregular and segmented movements.
In fact, ORCA computes the velocity to be applied to the robot by solving an optimization problem that takes as input the velocities of dynamic obstacles (in this case pedestrians). In the case of a large number of interactions, the algorithm may find it difficult to determine safe velocities that allow it to avoid pedestrians and still move toward the goal.
In such a deadlock situation, it tends to rotate and move in the opposite direction

from the goal in order to try to find a new speed value that does not cause collisions with pedestrians. This behavior results in a significant reduction in the path regularity of the algorithm.

PR		
Algorithm	Mean	Standard deviation
SFM	0.3171	0.0956
ORCA	0.1487	0.0709
GTSFM	0.5247	0.0829

Table 8.4: Mean value and standard deviation of the PR (Path Regularity) for each algorithm

On the other hand, the SFM is able to slow down and avoid pedestrians even in situations characterized by a high number of interactions. Despite low speeds, it determines the generation of paths that, allow to exit from the situation without performing an excessive number of rotations and reducing the overall steering angle. This results in a much smoother path than ORCA.
In this context, the implementation of game theory in the GTSFM makes it possible to determine which parameters of the SFM are best suited to ensure such an evasive maneuver as smoothly as possible. This leads to a significant increase in PR compared to the basic SFM.

Chapter 9

Future Works and Conclusions

In this thesis we focused on the study of human-robot interaction during the navigation process. The main goal was to develop a local planning algorithm based on the combination of Game Theory and the Social Force Model, in order to enhance the social acceptability of the robot. Specifically, the actions of each player correspond to individual sets of SFM parameters. Different trajectories can be generated from each of these sets. Game theory was then used to determine which of the trajectories generated by the individual agent was the optimal trajectory to follow, taking into account the possible choices of other players.
To ensure social acceptance, it was necessary to focus and increase the naturalness and comfort provided by the algorithm. To achieve this goal, a Differential Evolution algorithm was adopted. This algorithm was used to estimate the optimal parameters of the SFM that best approximate the trajectories contained in Thör, a dataset of real human trajectories. A neural network was employed to approximate the functioning of the DE and ensure real-time operation. In this way, at each time step, every single agent in the game is able to derive a set of possible actions (trajectories). These actions vary according to the surrounding environment conditions.
Through the experimental session, we tried to validate the effectiveness of our approach and quantitatively evaluate performances of the algorithm. Such quantitative validation provided promising results, although it is based only on simulations. Overall, the developed algorithm provides a very high level of naturalness and comfort compared to the other two state of the art algorithms, as demonstrated by the Path Length Ratio and Closest Pedestrian Distance indices. These results remain consistent even in different navigation situations, especially in cases with a high number of human-robot interactions.

In addition, it is important to note that the use of Game Theory allows for a significant increase in Path Regularity than SFM and ORCA. This leads to much smoother movements and paths than those generated by other state-of-the-art algorithms, resulting in a significant increase in naturalness.
Our work could find promising applications in a variety of fields other than just the autonomous movement of mobile robots. A concrete example may be Virtual Reality, where the model can be used to simulate realistic human motions [116]. The proposed algorithm could also be useful in gaming applications to develop Non-Player Characters (NPCs) that respond realistically to player's movements within the game [117] or autonomous guidance, where the algorithm could result in increased safety of people through the prediction of the future human movements [118].
Our work can be extended in various directions. An example could be increasing the number of actions (set of SFM parameters) available to each agent. This would allow for greater variability in the behaviors of the various agents involved in the game. Another approach might be to increase the number of agents so as to test performances even in situations with higher crowd density. However, increasing the number of agents and actions would decrease the overall performance of the algorithm and its ability to find real-time solutions of the game. In that case, a possible solution might be to apply a neural network that approximates how game theory works and derive the Nash equilibrium directly.
In addition, it would be useful to investigate other possible types of solutions of the game, different from the Nash equilibrium, and compare them with the solutions established by our algorithm. Some examples can be Pareto optimal [119] or Stackelberg equilibrium [120].
An alternative approach could be to improve the quality of the optimal parameters estimation by applying possible variants of the Differential Evolution algorithm. A promising example is Self Adaptive Differential Evolution (SaDE) [121]. As a matter of fact, it can automatically adapt the values of DE control parameters (mutation factor F and Crossover rate Cr) by learning directly from previous experiences of generating promising solutions [45].
Nevertheless, future works will focus on the creation of a real-world experimental session that involves both the robot and humans in order to obtain more consistent results. In that case, it might also be interesting to perform an evaluation from a qualitative point of view. Such qualitative evaluation could include the use of questionnaires that are already widely used in the state of the art, such as Goodspeed [122], RoSAS [123] or HRIES [124]. This makes it possible to obtain feedback on the social acceptability of the robot directly from the pedestrians involved in the navigation.

Bibliography

[1] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. «Experiences with an interactive museum tour-guide robot». In: *Artificial intelligence* 114.1-2 (1999), pp. 3–55 (cit. on p. 1).

[2] Hideki Asoh, Satoru Hayamizu, Isao Hara, Yoichi Motomura, Shotaro Akaho, and Toshihiro Matsui. «Socially embedded learning of the office-conversant mobile robot jijo-2». In: *IJCAI (2)*. 1997, pp. 880–887 (cit. on p. 1).

[3] Masaki Takahashi, Takafumi Suzuki, Hideo Shitamoto, Toshiki Moriguchi, and Kazuo Yoshida. «Developing a mobile robot for transport applications in the hospital domain». In: *Robotics and Autonomous Systems* 58.7 (2010), pp. 889–899 (cit. on p. 1).

[4] Eduardo Zalama, Jaime Gómez García-Bermejo, Samuel Marcos, Salvador Domínguez, Raúl Feliz, Roberto Pinillos, and Joaquín López. «Sacarino, a service robot in a hotel environment». In: *ROBOT2013: First Iberian Robotics Conference: Advances in Robotics, Vol. 2*. Springer. 2014, pp. 3–14 (cit. on p. 1).

[5] Jorge Rios-Martinez, Anne Spalanzani, and Christian Laugier. «From proxemics theory to socially-aware navigation: A survey». In: *International Journal of Social Robotics* 7 (2015), pp. 137–153 (cit. on pp. 1, 2).

[6] Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. «Human-aware robot navigation: A survey». In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743 (cit. on pp. 2–4, 16).

[7] Annemarie Turnwald and Dirk Wollherr. «Human-like motion planning based on game theoretic decision making». In: *International Journal of Social Robotics* 11 (2019), pp. 151–170 (cit. on pp. 2, 4, 17, 31, 32, 35, 36, 80).

[8] Annemarie Turnwald, Daniel Althoff, Dirk Wollherr, and Martin Buss. «Understanding human avoidance behavior: interaction-aware decision making based on game theory». In: *International journal of social robotics* 8 (2016), pp. 331–351 (cit. on pp. 4, 16, 29, 31–33, 35–37).

[9] Andrey Rudenko, Tomasz P Kucner, Chittaranjan S Swaminathan, Ravi T Chadalavada, Kai O Arras, and Achim J Lilienthal. «Thör: Human-robot navigation data collection and accurate motion trajectories dataset». In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 676–682 (cit. on pp. 5, 44).

[10] Jose Ricardo Sanchez-Ibanez, Carlos J Perez-del-Pulgar, and Alfonso García-Cerezo. «Path planning for autonomous mobile robots: A review». In: *Sensors* 21.23 (2021), p. 7898 (cit. on p. 8).

[11] BK Patle, Anish Pandey, DRK Parhi, AJDT Jagadeesh, et al. «A review: On path planning strategies for navigation of mobile robot». In: *Defence Technology* 15.4 (2019), pp. 582–606 (cit. on p. 8).

[12] Jiyu Cheng, Hu Cheng, Max Q-H Meng, and Hong Zhang. «Autonomous navigation by mobile robots in human environments: A survey». In: *2018 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE. 2018, pp. 1981–1986 (cit. on pp. 9, 11, 13–17, 32).

[13] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P How. «Socially aware motion planning with deep reinforcement learning». In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1343–1350 (cit. on pp. 9–12, 15, 84).

[14] Truong Xuan Tung and Trung Dung Ngo. «Socially aware robot navigation using deep reinforcement learning». In: *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. IEEE. 2018, pp. 1–5 (cit. on pp. 9–11).

[15] Dirk Helbing and Peter Molnar. «Social force model for pedestrian dynamics». In: *Physical review E* 51.5 (1995), p. 4282 (cit. on pp. 10, 21, 78).

[16] Francesco Zanlungo, Zeynep Yücel, Florent Ferreri, Jani Even, Luis Yoichi Morales Saiki, and Takayuki Kanda. «Social group motion in robots». In: *Social Robotics: 9th International Conference, ICSR 2017, Tsukuba, Japan, November 22-24, 2017, Proceedings 9*. Springer. 2017, pp. 474–484 (cit. on p. 10).

[17] Francesco Farina, Daniele Fontanelli, Andrea Garulli, Antonio Giannitrapani, and Domenico Prattichizzo. «Walking ahead: The headed social force model». In: *PloS one* 12.1 (2017), e0169734 (cit. on pp. 10, 21, 27, 38).

[18] Paolo Fiorini and Zvi Shiller. «Motion planning in dynamic environments using velocity obstacles». In: *The international journal of robotics research* 17.7 (1998), pp. 760–772 (cit. on p. 10).

[19] Jur Van den Berg, Ming Lin, and Dinesh Manocha. «Reciprocal velocity obstacles for real-time multi-agent navigation». In: *2008 IEEE international conference on robotics and automation*. Ieee. 2008, pp. 1928–1935 (cit. on p. 10).

[20] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. «Reciprocal n-body collision avoidance». In: *Robotics Research: The 14th International Symposium ISRR*. Springer. 2011, pp. 3–19 (cit. on pp. 11, 80).

[21] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. «Socially compliant navigation through raw depth inputs with generative adversarial imitation learning». In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 1111–1117 (cit. on pp. 11, 13, 84).

[22] Zhanteng Xie, Pujie Xin, and Philip Dames. «Towards safe navigation through crowded dynamic environments». In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 4934–4940 (cit. on p. 11).

[23] NH Singh and K Thongam. *Neural network-based approaches for mobile robot navigation in static and moving obstacles environments. Intel. Serv. Robot. 12 (1), 55–67 (2018)*. 2018 (cit. on p. 12).

[24] Santosh Balajee Banisetty, Vineeth Rajamohan, Fausto Vega, and David Feil-Seifer. «A deep learning approach to multi-context socially-aware navigation». In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 23–30 (cit. on p. 12).

[25] Huihui Sun, Weijie Zhang, Runxiang Yu, and Yujie Zhang. «Motion planning for mobile robots—Focusing on deep reinforcement learning: A systematic review». In: *IEEE Access* 9 (2021), pp. 69061–69081 (cit. on p. 12).

[26] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. «Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning». In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 285–292 (cit. on p. 12).

[27] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. «Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning». In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 6015–6022 (cit. on p. 12).

[28] Keyu Li, Yangxin Xu, Jiankun Wang, and Max Q-H Meng. «SARL*: Deep reinforcement learning based human-aware navigation for mobile robot in indoor environments». In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2019, pp. 688–694 (cit. on p. 13).

[29] Yigit Yildirim and Emre Ugur. «Learning Social Navigation from Demonstrations with Deep Neural Networks». In: () (cit. on p. 13).

[30] Omar A Islas Ramírez, Harmish Khambhaita, Raja Chatila, Mohamed Chetouani, and Rachid Alami. «Robots learning how and where to approach people». In: *2016 25th IEEE international symposium on robot and human interactive communication (RO-MAN)*. IEEE. 2016, pp. 347–353 (cit. on p. 13).

[31] Shiying Sun, Xiaoguang Zhao, Qianzhong Li, and Min Tan. «Inverse reinforcement learning-based time-dependent A* planner for human-aware robot navigation with local vision». In: *Advanced Robotics* 34.13 (2020), pp. 888–901 (cit. on p. 13).

[32] Skanda Vaidyanath. *Inverse Reinforcement Learning*. `https://skandavaidyanath.github.io/post/inverse-rl-paper/`. Accessed: 2023-08-27 (cit. on p. 14).

[33] Alessandro Antonucci, Gastone Pietro Rosati Papini, Paolo Bevilacqua, Luigi Palopoli, and Daniele Fontanelli. «Efficient prediction of human motion for real-time robotics applications with physics-inspired neural networks». In: *IEEE Access* 10 (2021), pp. 144–157 (cit. on p. 14).

[34] Sheng Fei Chik, Che Fai Yeong, Eileen Lee Ming Su, Thol Yong Lim, Feng Duan, Jeffrey Too Chuan Tan, Ping Hua Tan, and Patrick Jun Hua Chin. «Gaussian pedestrian proxemics model with social force for service robot navigation in dynamic environment». In: *Modeling, Design and Simulation of Systems: 17th Asia Simulation Conference, AsiaSim 2017, Melaka, Malaysia, August 27–29, 2017, Proceedings, Part I 17*. Springer. 2017, pp. 61–73 (cit. on p. 15).

[35] Yuying Chen, Ming Liu, and Lujia Wang. «Rrt* combined with gvo for real-time nonholonomic robot navigation in dynamic environment». In: *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. IEEE. 2018, pp. 479–484 (cit. on p. 15).

[36] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33 (cit. on p. 15).

[37] Noé Pérez-Higueras, Fernando Caballero, and Luis Merino. «Learning human-aware path planning with fully convolutional networks». In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 5897–5902 (cit. on p. 15).

[38] Óscar Gil, Anaís Garrell, and Alberto Sanfeliu. «Social robot navigation tasks: Combining machine learning techniques and social force model». In: *Sensors* 21.21 (2021), p. 7087 (cit. on p. 15).

[39] Pete Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. «Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation». In: *The International Journal of Robotics Research* 34.3 (2015), pp. 335–356 (cit. on p. 16).

[40] SF Chik, CF Yeong, ELM Su, TY Lim, Y Subramaniam, and PJH Chin. «A review of social-aware navigation frameworks for service robot in dynamic human environments». In: *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)* 8.11 (2016), pp. 41–50 (cit. on p. 16).

[41] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. «Social lstm: Human trajectory prediction in crowded spaces». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971 (cit. on p. 16).

[42] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. «Socially compliant mobile robot navigation via inverse reinforcement learning». In: *The International Journal of Robotics Research* 35.11 (2016), pp. 1289–1307 (cit. on p. 16).

[43] Giada Galati, Stefano Primatesta, Sergio Grammatico, Simone Macrì, and Alessandro Rizzo. «Game theoretical trajectory planning enhances social acceptability of robots by humans». In: *Scientific Reports* 12.1 (2022), p. 21976 (cit. on pp. 17, 35, 84).

[44] Rainer Storn. «Differrential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces». In: *Technical report, International Computer Science Institute* 11 (1995) (cit. on pp. 18, 39, 43, 44).

[45] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. «Differential evolution: A survey of the state-of-the-art». In: *IEEE transactions on evolutionary computation* 15.1 (2010), pp. 4–31 (cit. on pp. 18, 40, 42–44, 91).

[46] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006 (cit. on p. 19).

[47] Hui-Yuan Fan and Jouni Lampinen. «A trigonometric mutation operation to differential evolution». In: *Journal of global optimization* 27 (2003), pp. 105–129 (cit. on p. 19).

[48] Hamid R Tizhoosh. «Opposition-based learning: a new scheme for machine intelligence». In: *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*. Vol. 1. IEEE. 2005, pp. 695–701 (cit. on p. 19).

[49] Shih-Lian Cheng and Chyi Hwang. «Optimal approximation of linear systems by a differential evolution algorithm». In: *IEEE Transactions on Systems, man, and cybernetics-part a: systems and humans* 31.6 (2001), pp. 698–707 (cit. on p. 19).

[50] Hassan Yousefi, Heikki Handroos, and Azita Soleymani. «Application of differential evolution in system identification of a servo-hydraulic system with a flexible load». In: *Mechatronics* 18.9 (2008), pp. 513–528 (cit. on p. 19).

[51] Hesheng Tang, Songtao Xue, and Cunxin Fan. «Differential evolution strategy for structural system identification». In: *Computers & Structures* 86.21-22 (2008), pp. 2004–2012 (cit. on p. 19).

[52] Prathyush P Menon, Jongrae Kim, Declan G Bates, and Ian Postlethwaite. «Clearance of nonlinear flight control laws using hybrid evolutionary optimization». In: *IEEE transactions on evolutionary computation* 10.6 (2006), pp. 689–699 (cit. on p. 19).

[53] Luis Moreno, Santiago Garrido, Dolores Blanco, and M Luisa Munoz. «Differential evolution solution to the SLAM problem». In: *Robotics and Autonomous Systems* 57.4 (2009), pp. 441–450 (cit. on p. 20).

[54] Serkan Aydin and Hakan Temeltas. «Fuzzy-differential evolution algorithm for planning time-optimal trajectories of a unicycle mobile robot on a predefined path». In: *Advanced Robotics* 18.7 (2004), pp. 725–748 (cit. on p. 20).

[55] Jayasree Chakraborty, Amit Konar, Lakhmi C Jain, and Uday K Chakraborty. «Cooperative multi-robot path planning using differential evolution». In: *Journal of Intelligent & Fuzzy Systems* 20.1-2 (2009), pp. 13–27 (cit. on p. 20).

[56] Bidyadhar Subudhi and Debashisha Jena. «Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification». In: *Neural Processing Letters* 27 (2008), pp. 285–296 (cit. on p. 20).

[57] George D Magoulas, Vassilis P Plagianakos, and Michael N Vrahatis. «Neural network-based colonoscopic diagnosis using on-line learning and differential evolution». In: *Applied Soft Computing* 4.4 (2004), pp. 369–379 (cit. on p. 20).

[58] Gonzalo Ferrer, Anais Garrell, and Alberto Sanfeliu. «Social-aware robot navigation in urban environments». In: *2013 European Conference on Mobile Robots*. IEEE. 2013, pp. 331–336 (cit. on p. 21).

[59] Meifang Li, Yongxiang Zhao, Lerong He, Wenxiao Chen, and Xianfeng Xu. «The parameter calibration and optimization of social force model for the real-life 2013 Ya'an earthquake evacuation in China». In: *Safety science* 79 (2015), pp. 243–253 (cit. on pp. 21, 38).

[60] Qian Liu. «A social force model for the crowd evacuation in a terrorist attack». In: *Physica A: Statistical Mechanics and its Applications* 502 (2018), pp. 315–330 (cit. on p. 21).

[61] Francesco Zanlungo, Tetsushi Ikeda, and Takayuki Kanda. «Social force model with explicit collision prediction». In: *Europhysics Letters* 93.6 (2011), p. 68005 (cit. on p. 23).

[62] Xu Chen, Martin Treiber, Venkatesan Kanagaraj, and Haiying Li. «Social force models for pedestrian traffic–state of the art». In: *Transport reviews* 38.5 (2018), pp. 625–653 (cit. on p. 25).

[63] Gustavo Arechavaleta, Jean-Paul Laumond, Halim Hicheur, and Alain Berthoz. «On the nonholonomic nature of human locomotion». In: *Autonomous Robots* 25 (2008), pp. 25–35 (cit. on p. 27).

[64] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior (60th Anniversary Commemorative Edition)*. Princeton university press, 2007 (cit. on p. 28).

[65] John Nash. «Non-cooperative games». In: *Annals of mathematics* (1951), pp. 286–295 (cit. on pp. 28, 31).

[66] W Brian Arthur. «Complexity in economic and financial markets». In: *Complexity* 1.1 (1995), pp. 20–25 (cit. on p. 28).

[67] Kalyan Chatterjee and William Samuelson. *Game theory and business applications*. Springer, 2001 (cit. on p. 28).

[68] Steven J Brams. *Game theory and politics*. Courier Corporation, 2011 (cit. on p. 28).

[69] Peter Hammerstein and Reinhard Selten. «Game theory and evolutionary biology». In: *Handbook of game theory with economic applications* 2 (1994), pp. 929–993 (cit. on p. 28).

[70] Jason R Marden and Jeff S Shamma. «Game theory and control». In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 105–134 (cit. on p. 28).

[71] Tim Roughgarden. «Algorithmic game theory». In: *Communications of the ACM* 53.7 (2010), pp. 78–86 (cit. on p. 28).

[72] Yoav Shoham. «Computer science and game theory». In: *Communications of the ACM* 51.8 (2008), pp. 74–79 (cit. on p. 28).

[73] Wikipedia contributors. *Solution concept — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-August-2023]. 2023. URL: `https://en.wikipedia.org/w/index.php?title=Solution_concept&oldid=1149748360` (cit. on p. 29).

[74] Kevin Leyton-Brown and Yoav Shoham. *Essentials of game theory: A concise multidisciplinary introduction*. Springer Nature, 2022 (cit. on p. 29).

[75] Annemarie Turnwald, Wiktor Olszowy, Dirk Wollherr, and Martin Buss. «Interactive navigation of humans from a game theoretic perspective». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 703–708 (cit. on p. 31).

[76] Stephen Bitgood and Stephany Dukes. «Not another step! Economy of movement and pedestrian choice point behavior in shopping malls». In: *Environment and behavior* 38.3 (2006), pp. 394–405 (cit. on p. 35).

[77] R McNeill Alexander. «Energetics and optimization of human walking and running: the 2000 Raymond Pearl memorial lecture». In: *American journal of human biology* 14.5 (2002), pp. 641–648 (cit. on pp. 35, 36).

[78] Simone Sagratella. «Algorithms for generalized potential games with mixed-integer variables». In: *Computational Optimization and Applications* 68.3 (2017), pp. 689–717 (cit. on p. 37).

[79] Zhiqiang Wan, Xuemin Hu, Haibo He, and Yi Guo. «A learning based approach for social force model parameter estimation». In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 4058–4064 (cit. on p. 38).

[80] Ramachandra Raghavendra, Alessio Del Bue, Marco Cristani, and Vittorio Murino. «Abnormal crowd behavior detection by social force optimization». In: *Human Behavior Understanding: Second International Workshop, HBU 2011, Amsterdam, The Netherlands, November 16, 2011. Proceedings 2*. Springer. 2011, pp. 134–145 (cit. on p. 38).

[81] Xuan-Tung Truong and Trung Dung Ngo. «Toward socially aware robot navigation in dynamic and crowded environments: A proactive social motion model». In: *IEEE Transactions on Automation Science and Engineering* 14.4 (2017), pp. 1743–1760 (cit. on pp. 38, 80).

[82] Hasan Kivrak, Furkan Cakmak, Hatice Kose, and Sirma Yavuz. «Social navigation framework for assistive robots in human inhabited unknown environments». In: *Engineering Science and Technology, an International Journal* 24.2 (2021), pp. 284–298 (cit. on p. 38).

[83] Jinghui Zhong and Wentong Cai. «Differential evolution with sensitivity analysis and the Powell's method for crowd model calibration». In: *Journal of computational science* 9 (2015), pp. 26–32 (cit. on p. 38).

[84] Anders Johansson, Dirk Helbing, and Pradyumn K Shukla. «Specification of the social force pedestrian model by evolutionary adjustment to video tracking data». In: *Advances in complex systems* 10.supp02 (2007), pp. 271–288 (cit. on p. 48).

[85] *Create3 mobile base*. `https://iroboteducation.github.io/create3_docs/`. Accessed: 2023-08-23 (cit. on p. 56).

[86] *RealSense | Use cases*. `https://www.intelrealsense.com/use-cases/`. Accessed: 2023-10-05 (cit. on p. 59).

[87] *RealSenseD435*. `https://www.intelrealsense.com/depth-camera-d435/`. Accessed: 2023-10-05 (cit. on p. 59).

[88] *ROS*. `https://www.ros.org/`. Accessed: 2023-08-23 (cit. on p. 62).

[89] *ROS/Introduction*. `https://wiki.ros.org/ROS/Introduction`. Accessed: 2023-08-23 (cit. on p. 62).

[90] *ROS gmapping package*. `http://wiki.ros.org/gmapping`. Accessed: 2023-08-23 (cit. on p. 66).

[91] *ROS map_server package*. `http://wiki.ros.org/map_server?distro=noetic`. Accessed: 2023-08-23 (cit. on p. 66).

[92] *ROS amcl package*. `http://wiki.ros.org/amcl`. Accessed: 2023-08-23 (cit. on p. 66).

[93] *ROS move_base package*. `http://wiki.ros.org/move_base`. Accessed: 2023-08-23 (cit. on pp. 66, 67, 70, 71).

[94] *ROS costmap_2d package*. `http://wiki.ros.org/costmap_2d`. Accessed: 2023-08-23 (cit. on pp. 67, 68).

[95] *ROS global_planner package*. `http://wiki.ros.org/global_planner?distro=noetic`. Accessed: 2023-08-23 (cit. on p. 69).

[96] *ROS base_locale_planner package*. http://wiki.ros.org/base_local_planner?distro=noetic. Accessed: 2023-08-23 (cit. on p. 70).

[97] *ROS clear_costmap_recovery package*. http://wiki.ros.org/clear_costmap_recovery?distro=noetic. Accessed: 2023-08-23 (cit. on p. 70).

[98] *ROS rotate_recovery package*. http://wiki.ros.org/rotate_recovery?distro=noetic. Accessed: 2023-08-23 (cit. on p. 70).

[99] Sergi Hernández Juan and Fernando Herrero Cotarelo. «Autonomous navigation framework for a car-like robot». In: (2015) (cit. on p. 71).

[100] *TensorFlow*. https://www.tensorflow.org/overview?hl=it. Accessed: 2023-08-23 (cit. on p. 73).

[101] *Introduzione ai tensori | TensorFlow*. https://www.tensorflow.org/guide/tensor?hl=it. Accessed: 2023-08-23 (cit. on p. 74).

[102] *Easy TensorFlow*. https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session. Accessed: 2023-08-23 (cit. on p. 74).

[103] *Gazebo simulator*. https://gazebosim.org/home. Accessed: 2023-08-23 (cit. on p. 77).

[104] *gazebo_sfm_plugin*. https://github.com/robotics-upo/gazebo_sfm_plugin. Accessed: 2023-08-23 (cit. on p. 78).

[105] Mehdi Moussaïd, Dirk Helbing, Simon Garnier, Anders Johansson, Maud Combe, and Guy Theraulaz. «Experimental study of the behavioural mechanisms underlying self-organization in human crowds». In: *Proceedings of the Royal Society B: Biological Sciences* 276 (2009), pp. 2755–2762 (cit. on p. 78).

[106] Mehdi Moussaïd, Niriaska Perozo, Simon Garnier, Dirk Helbing, and Guy Theraulaz. «The walking behaviour of pedestrian social groups and its impact on crowd dynamics». In: *PloS one* 5.4 (2010), e10047 (cit. on p. 78).

[107] *Rviz docs*. http://wiki.ros.org/rviz. Accessed: 2023-08-23 (cit. on p. 79).

[108] Abhijat Biswas, Allan Wang, Gustavo Silvera, Aaron Steinfeld, and Henny Admoni. «Socnavbench: A grounded simulation testing framework for evaluating social navigation». In: *ACM Transactions on Human-Robot Interaction (THRI)* 11.3 (2022), pp. 1–24 (cit. on pp. 80, 84).

[109] Noé Pérez-Higueras, Roberto Otero, Fernando Caballero, and Luis Merino. «HuNavSim: A ROS 2 Human Navigation Simulator for Benchmarking Human-Aware Robot Navigation». In: *arXiv preprint arXiv:2305.01303* (2023) (cit. on pp. 80, 84).

[110] Yuxiang Gao and Chien-Ming Huang. «Evaluation of socially-aware robot navigation». In: *Frontiers in Robotics and AI* 8 (2022), p. 721317 (cit. on p. 84).

[111] A Alessandro. «Socially aware robot navigation». In: (2022) (cit. on p. 84).

[112] Fagner Pimentel and Plinio Aquino. «Performance evaluation of ROS local trajectory planning algorithms to social navigation». In: *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*. IEEE. 2019, pp. 156–161 (cit. on p. 84).

[113] Junxian Wang, Wesley P Chan, Pamela Carreno-Medrano, Akansel Cosgun, and Elizabeth Croft. «Metrics for Evaluating Social Conformity of Crowd Navigation Algorithms». In: *2022 IEEE International Conference on Advanced Robotics and Its Social Impacts (ARSO)*. IEEE. 2022, pp. 1–6 (cit. on p. 84).

[114] Dizan Vasquez, Billy Okal, and Kai O Arras. «Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1341–1346 (cit. on p. 84).

[115] Nelson David Munoz Ceballos, Jaime Alejandro Valencia, and Nelson Londono Ospina. «Quantitative Performance Metrics for Mobile Robots Navigation, Mobile Robots Navigation». In: *Alejandra Barrera (Ed.)* (2010) (cit. on p. 84).

[116] Sahil Narang, Andrew Best, and Dinesh Manocha. «Simulating movement interactions between avatars & agents in virtual worlds using human motion constraints». In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. 2018, pp. 9–16 (cit. on p. 91).

[117] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. «Deep reinforcement learning for navigation in aaa video games». In: *arXiv preprint arXiv:2011.04764* (2020) (cit. on p. 91).

[118] Kunming Li, Mao Shan, Karan Narula, Stewart Worrall, and Eduardo Nebot. «Socially aware crowd navigation with multimodal pedestrian trajectory prediction for autonomous vehicles». In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–8 (cit. on p. 91).

[119] SS Rao. «Game theory approach for multiobjective structural optimization». In: *Computers & Structures* 25.1 (1987), pp. 119–127 (cit. on p. 91).

[120] Valeriu Ungureanu. *Pareto-Nash-Stackelberg game and control theory*. Vol. 80. Springer, 2018 (cit. on p. 91).

[121] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. «Differential evolution algorithm with strategy adaptation for global numerical optimization». In: *IEEE transactions on Evolutionary Computation* 13.2 (2008), pp. 398–417 (cit. on p. 91).

[122] Christoph Bartneck, Dana Kulić, Elizabeth Croft, and Susana Zoghbi. «Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots». In: *International journal of social robotics* 1 (2009), pp. 71–81 (cit. on p. 91).

[123] Colleen M Carpinella, Alisa B Wyman, Michael A Perez, and Steven J Stroessner. «The robotic social attributes scale (RoSAS) development and validation». In: *Proceedings of the 2017 ACM/IEEE International Conference on human-robot interaction*. 2017, pp. 254–262 (cit. on p. 91).

[124] Nicolas Spatola, Barbara Kühnlenz, and Gordon 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* 13.7 (2021), pp. 1517–1539 (cit. on p. 91).