POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Robot social navigation: a quantitative and qualitative benchmark of state-of-the-art algorithms in real-world

Supervisors

Prof. Marcello CHIABERGE

Eng. Mauro MARTINI

Eng. Andrea OSTUNI

Eng. Andrea EIRALE

Candidate

Stefano TREPELLA

October 2024

Abstract

Social navigation has seen a substantial number of developments in the last few years, considering that the possible tasks that a robot can perform while being aware of the presence of other people are numerous. There are many aspects of socially aware navigation that contribute to the achievement of the current objective, but this thesis focuses on the local planner, also known as the local controller. The controller handles the information around the robot to send the velocity command that controls the rotational and translational speed of the robot. The main objective of this thesis is to compare the performance of two different algorithms, namely the DWA and the MPPI, in a set of laboratory experiments. Six total variations correspond to the base version of the algorithms and two enhanced versions with, respectively, a social costmap plugin and the Social Force Model. Considering that the Social Force Model was implemented solely on the DWA algorithm, the first part of this thesis focused on developing a method to integrate said model into the MPPI. This led to the development of a plugin that operates alongside the other critics and computes the social work for the randomly sampled trajectories. The second half of this work consisted of performing the tests themselves. The main aspect that differentiates the work developed in this thesis from the other papers present in the literature is the acquisition of both subjective and objective metrics. The objective metrics are variables like the time of completion, the social work, and the minimum distance to the agents that were obtained through the acquisition of positions and velocities in the laboratory experiments. The subjective metrics, on the contrary, include unobtrusiveness, friendliness, smoothness, and avoidance foresight and they were decided by the agents themselves after the end of each experiment, allowing a complete evaluation of each algorithm. The experiments were conducted in the PIC4SeR laboratory using the Jackal robot from ClearPath and a set of VICON cameras which were essential in tracking the position and the velocities of both the robot and the human agents, who were equipped with a custom-made accessory each to make them recognizable for the cameras. Eight different scenarios were developed to test the navigation prowess of the robot in different conditions. Said scenarios were characterized by a different disposition of the robot, the human agents, and the static obstacles. Each algorithm was tested five times, totaling 240 tests.

Acknowledgements

I would like to express my gratitude to the PoliTO Interdepartemental Center for Service Robotics (PIC4SeR) for giving me the resources and support needed to complete this thesis. My experience in this center has allowed me to deepen my knowledge in some fields where I had a superficial understanding of the subjects. It also allowed me to learn some entirely novel notions while gaining some hands-on experience by interacting with the Jackal robot in the laboratory. I want to thank my supervisors, Prof. Marcello Chiaberge, Mauro Martini, Andrea Ostuni, and Andrea Eirale, for the help they have given me throughout this journey. I want to thank all the thesis students at PIC4SeR, who continuously helped me throughout my experiments and were pivotal in completing this project. I also thank everyone else who supported me along the way, including my friends and my family.

"The world ain't all sunshine and rainbows. It's a very mean and nasty place and I don't care how tough you are it will beat you to your knees and keep you there permanently if you let it. You, me, or nobody is gonna hit as hard as life. But it ain't about how hard you hit. It's about how hard you can get hit and keep moving forward. How much you can take and keep moving forward. That's how winning is done!"
Rocky Balboa

Table of Contents

List of Tables

List of Figures

Acronyms

eHMI

external Human Machine Interface

SFM

Social Force Model

DWA

Dynamic Window Approach

MPPI

Model Path Predictive Integral

GPU

Graphics Processing Unit

ROS2

Robot Operating System 2

NAV2

Navigation 2

DDS

Data Distribution Service

BT

Behavior tree

SLAM

Simultaneous Localization and Mapping

NavFn	Navigation Function
LiDaR	Light Detection and Ranging
PIC4SeR	PoliTo Interdepartimental Centre for Service Robotics
TEB	Timed Elastic Band
DRL	Deep Reinforcement Learning
APF	Artificial Potential Fields
MPC	Model Predictive Control
RL	Reinforcement Learning
PPO	Proximal Policy Optimization
PFZ	Potential Freezing Zone
CPU	Central Processing Unit
UNO	Unobtrusiveness
FL	Friendliness

SO
: Smoothness

AF
: Avoidance foresight

AMD
: Average minimum distance

TTC
: Time to complete

PL
: Path length

SW
: Social work

SWsec
: Social work per second

SR
: Success rate

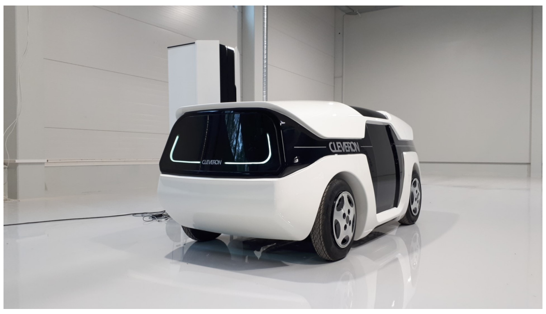
FOV
: Field of view

Introduction

Introduction to socially aware robot navigation

Autonomous navigation has been a topic of paramount importance in research in recent years.
When discussing autonomous navigation, it must be noted that, in most cases, the controlled machines may need to traverse populated environments, such as urban areas. For example, one of the most well-known applications globally is autonomous navigation in vehicles, particularly cars, for human transportation. These autonomous navigation tasks, which consider the possibility of people in the scene, fall under the category of social navigation. This approach aims to guide the robot through populated environments safely and efficiently and to make the robot's behavior "socially acceptable."

The objectives of a socially aware robot can be many, such as a wheeled robot used for parcel delivery or a humanoid robot designed for elderly assistance (1)


(a) Robot used for parcel delivery, reproduced from https://upload.wikimedia.org/wikipedia /commons/c/cc/Cleveron_self-driving_robot_courier.jpg

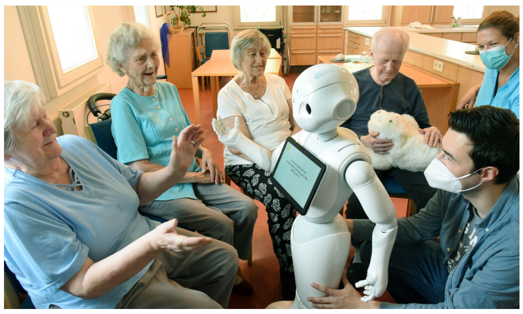

(b) Robot used for elderly assistance, reproduced from https://pursuit.unimelb.edu.au/articles/are-robots-the-answer-for-aged-care-during-pandemics

Figure 1: An example of a robot for elderly assistance and one for parcel delivery

The scope of this thesis focuses on socially aware robot navigation. It can be characterized by different aspects that were detailed in the work of Singamaneni [1]:

- Robot type: aerial, aquatic, or ground-based,
- Communication capabilities: the ability to interact and negotiate with humans in the environment,
- Task assignment: independent or assistive roles,
- Planning: how the global and local navigation strategies are handled,
- Social considerations: correct treatment of agents in the scene, obedience to social norms and avoidance of obstacles/restricted zones,
- Experimental setup: tools for conducting experiments and methods for evaluating robot performance.

A study on human-aware robot navigation can focus on various characteristics. Some elements, however, remain constant across different studies. For instance, the primary robots being tested are often ground-based wheeled robots.
Furthermore, evaluation parameters are frequently quantitative, measuring factors such as the distances between the robot and agents in the scene, which leads to the use of predominantly objective metrics.

A limitation of using only objective parameters is that they fail to account for social acceptability. Establishing qualitative parameters is also crucial in ensuring socially appropriate navigation for a robot. Qualitative parameters can be inferred by analyzing a robot's trajectories in different conditions. The paper presented by Teja et al. [2] compares two different algorithms based on the **TEB** by introducing both a short qualitative analysis and a quantitative one. One parameter introduced in this paper was called entanglement resolution. This resolution defines the robot's capability of getting unstuck, and it was deduced by visually comparing the algorithms' trajectories.

Even though such an analysis provides different insights into how an algorithm works, it fails to consider how the people involved in the scene may perceive a robot's actions. The psychological impact of seeing a robot autonomously navigate in an environment can't be ignored. Therefore, it is crucial to obtain the participants' opinions to gauge the robot's performance.

One study [3], for example, concentrated on two different communication modes for a delivery robot, based on either a display or a light-based **EHMI**.
Different designs were used for each of these interfaces, resulting in four designs for each type:

- Two text-based designs,
- Two graphical-based designs,
- Two single light designs,
- Two dual light designs.

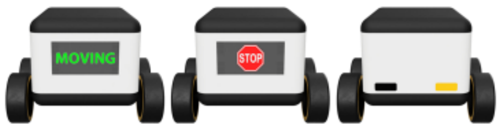


Figure 2: Three of the chosen HMI without perspective: text-based, graphical-based, dual light. (Reproduced from [3], 2021, p. 4)

There are two designs for every type due to the presence or absence of the robot perspective. For instance, "ROBOT" was appended to the text-based designs to explain that the task to be performed had to be carried out by the machine. Figure 2 shows some of the chosen configurations. The researchers aimed to identify the design that the experiment participants most liked. Their results showed that display-based HMIs were judged as the most understandable in most scenarios. Light-based designs, however, were considered suitable as complementary elements, suggesting that a redundancy in the information given by the robot could be appreciated.

Even though the previous example is based on the communication part of socially aware navigation, it shows that having a subjective input is essential to understanding what works.

Thesis overview

The aspect of social navigation that this thesis focuses on is the local controller, which gives the velocity command to the robot. The controller is crucial because a reliable local controller should not only follow the path given by the global planner; it should also avoid dynamic obstacles that may be present in the scene. In social navigation, another critical consideration is how the robot's trajectory is affected by people, as the algorithm should treat objects and human beings differently.

The main objective of this thesis was to construct a benchmark for a set of social scenarios that compares two different algorithms, each in three variants, totaling six algorithms.

The algorithms used were:

- DWA,
- MPPI.

The three variants for each algorithm were:

- Base version of the algorithm.
- Base with the addition of a social costmap.
- Base with the addition of the SFM.

The social costmap and the SFM will be explained in more detail in Chapter 2. The DWA algorithm already included a direct implementation of the SFM, while the social costmap could be easily added. In contrast, the MPPI could only access the social costmap, as the SFM had yet to be implemented.

For this reason, the objectives of this thesis became the following:

1. Adding a SFM implementation in the MPPI,
2. Comparing the six variations in a set of robot navigation scenarios, measuring quantitative and qualitative parameters.

Regarding the thesis' structure, it is divided into the following chapters:

- Chapter 1 discusses the possible solutions to make the local controller social present in the literature. It also acknowledges the methods used to create a benchmark that other works provide.
- Chapter 2 focuses on the theoretical aspects, beginning with an explanation of how the **DWA** and **MPPI** algorithms work. It then examines the **SFM** and introduces the basics on **NAV2**.
- Chapter 3 analyzes the software and hardware components utilized in the simulations and the lab experiments. It will also explain the setup used for the different lab experiments and the evaluation criteria,
- Chapter 4 presents the results obtained,
- Chapter 5 concludes and delves into possible future developments.

Chapter 1

State of the art

1.1 Local controller

Many algorithms can be employed to calculate the local motion generation. Most of these algorithms can be divided into three different approaches:

- planning-based approaches
- force-based approaches
- learning-based approaches

The planning-based approaches first generate a local trajectory for the robot to pursue, then use it to compute a velocity command for the robot. Many popular algorithms can be classified as planning-based, such as the **DWA**, **MPC**, and **TEB**. The force-based approaches, instead, use potential or force fields in the robot's environment and interaction forces given by the people's movement to generate the robot's control input. The **APF** algorithm is based on potential fields, while other implementations employ the **SFM**.

Learning-based approaches often do not generate a trajectory; instead, they compute the velocity command directly based on some form of machine learning algorithm. Utilizing deep learning, reinforcement learning, or imitation learning can achieve many possible solutions. These approaches, however, are sometimes not implemented independently. The literature contains many instances of two or more of these approaches employed in the same algorithm. The social force window planner used in this experiment combines the repulsive forces given by the SFM with the DWA, a planning-based algorithm. In a paper presented this year by Martini et al. [4], the objective was to improve this algorithm by introducing a learning phase aimed at dynamically changing the weights present in the cost function of the Social Force Window planner. The learning phase was introduced using a **DRL** agent whose objective was to learn an optimal policy. The policy learned by the agent was influenced by a reward function that was characterized by a set of parameters:

- Goal distance,
- Path alignment,
- Robot velocity,
- Obstacle avoidance,
- Social penalty.

The agent then induced the local planner to select the optimal velocity commands. This method greatly enhanced the algorithm's performance, resulting in more consistent behavior. Instead, a different implementation used the **MPC** and reinforcement learning. Romero's work [5] presents an algorithm called Action-Critic model predictive control, which uses a differentiable **MPC** as the final layer of the actor policy. This approach combines the robustness and replanning capabilities of an **MPC** with the flexibility of **RL**, creating an algorithm whose commands are always feasible concerning the dynamics. Tests conducted with this hybrid structure showed an uplift in performance compared to a standard **MPC** or a **PPO**, showing the potential of such a hybrid architecture. A different hybrid approach was presented by Sathyamoorthy et al. [6] to avoid a common issue in social navigation known as the freezing robot problem. This solution implements both **DRL**-based collision avoidance with a novel technique named Frozone. This technique first constructs a zone known as **PFZ** (which could be a zone similar to that in Fig 1.1), and then it computes a deviation velocity from this zone to prevent the robot from getting stuck. This approach allowed the robot to handle crowds of varying densities, leading to a higher success rate than other methodologies.

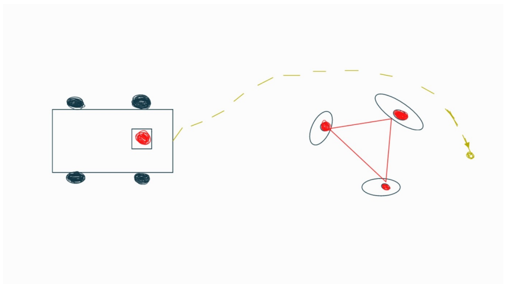


Figure 1.1: A representation of the **PFZ** (triangle with red edges) considered in the Frozone technique.

1.2 Evaluation of social robot navigation

This thesis also focuses on developing a benchmark that can compare different types of algorithms. Numerous evaluation criteria can be employed to compare various algorithms. The paper by Gao et al. [7] thoroughly analyzes the different evaluation methods employed throughout the years. They can be divided into four categories:

- case studies study standard robot interaction, like passing, crossing, and overtaking.
- simulations test the robot in a virtual environment before deploying it in the real world.
- laboratory studies analyze the robot's prowess in confined experiments.
- field studies utilize the robot in a larger-scale experiment.

Field studies are often the most difficult to implement among these categories, considering that a robot's deployment in a public space is not always allowed. Nonetheless, many experiments have been performed using this methodology, such as deploying a mobile robot in a shopping mall described by Shiomi et al. [8] in 2014. During this experiment, the robot interacted for several hours with the uninstructed pedestrians (as it can be seen of Fig. 1.2)

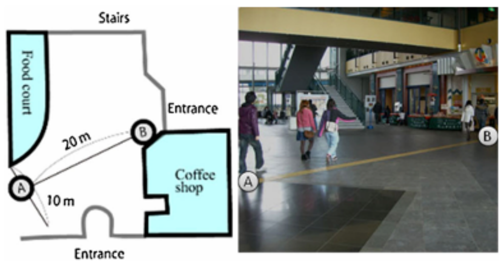


Figure 1.2: The field study performed in a mall using a mobile robot (reproduced from [8], 2014, p.449)

Simulations are used to compare different types of existing algorithms or test novel solutions, but they alone do not indicate a robot's social capabilities. Witnessing people's reactions to a robot's movement is often essential, and this cannot be achieved in a simulated framework. An array of simulation platforms have been developed throughout the years. Among these, one of the most recent is the HunavSim plugin described by Perez et al. [9] This platform works with different 3D simulators present in ROS2, including Gazebo.

It provides the possibility to spawn agents with customizable behavior in the environment, allowing agents to exhibit different reactions around the robots. An example can be seen in Fig. 1.3

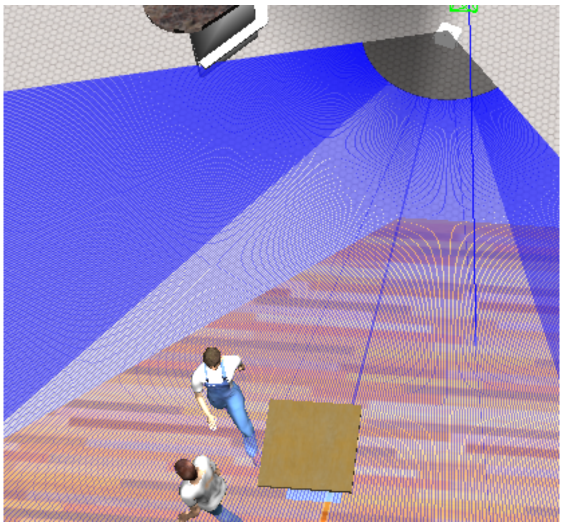


Figure 1.3: Two agents spawned by the HunavSim plugin in a cafeteria environment.

Laboratory studies often employ experimental tasks, allowing the human participants and the robot to move freely. They are easier to organize and allow for the acquisition of the participants' opinions, which can't be achieved in a field study where the people present aren't aware of the experiments' conditions. The study performed by Mavrogiannis et al. [10] in 2019 was based on a task that consisted of allowing three participants and a robot to move between six stations. This study involved more than 100 participants, and both subjective and objective metrics were collected.

The last category, case studies, is on a smaller scale compared to laboratory and field studies. It involves studying the robot's capabilities in a set of routine interactions that can be performed in a simulated or real-world setting. These commonly occurring scenarios include:

- passing.
- overtaking.
- crossing.
- approaching.
- following, leading, and accompanying.

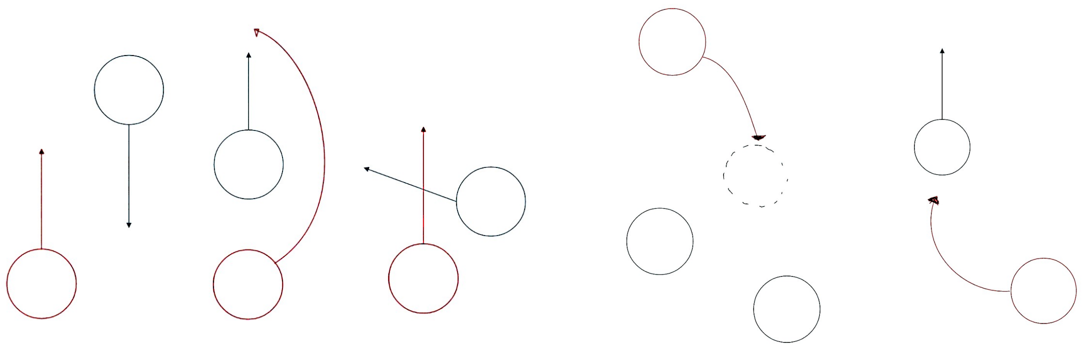

(a) Passing, overtaking, crossing (b) Approaching, following

Figure 1.4: Different standard scenarios, robot in red, agents in black

The scenarios in Fig. 1.4 depict a wide array of interactions the robot could face in a public environment. A robot should safely pass and overtake people when traveling on a sidewalk if it is used for parcel delivery; it should also be able to approach and follow people safely if its intended use is elders' assistance. Different researchers studied these fundamental interactions. Pacchierotti, in 2006 [11], focused on the passing scenario in a hallway environment where the human participant exhibited different behaviors, such as stopping in the middle of the room or walking at a constant speed. Kretzschmar, in 2016 [12], illustrated using reinforcement learning to pass two people in a hallway without cutting through the group. Among these scenarios and evaluation methods, choosing metrics to be used in the evaluation differentiates the papers throughout the years. As previously stated in the introduction, relying on quantitative metrics to assess the robot's performance is commonly preferred since navigation data allows for a wide array of metrics depending on the setting. The HunavSim evaluator introduced by Perez [9] contains a good variety of quantitative metrics, ranging from pure proxemics to metrics included in the SEAN2.0 evaluator [13] to SFM-based metrics. However, the qualitative metrics can significantly differ depending on the work proposed since they could be based on the participants' opinions or a subjective assessment of the robot's trajectory. This thesis relied on the quantitative metrics proposed by the Hunav evaluator and on a set of qualitative metrics scored by the agents.

Chapter 2

Methods and theory

2.1 DWA

The DWA (Dynamic-Window Approach) is a popular motion planning algorithm used in mobile robotics for local obstacle avoidance. It was first introduced in 1997 [14], and since then, it has been used in many papers as an effective way to handle autonomous robot motion control in the presence of obstacles [15].

The trajectories of the robot are approximated by a succession of circular arcs, with a linear bounded error due to a supposition made in the derivation, which assumes the velocity of the robot to remain constant within a time interval $[t_i, t_{i+1}]$.

In this algorithm, the search for commands moving the robot is performed in the space of velocities, which considers both translational and rotational velocities (ν, ω). The steps that are performed at each iteration are the following:

1. Restricting the set of possible velocity vectors to a subset called search space.
2. Choosing the velocity that maximizes the optimization function.

2.1.1 Search space

The controller's frequency determines how often the velocities pruning and optimization are performed. Assuming a frequency of 20 Hz would lead to a 50 ms time interval. As previously mentioned, the trajectories are approximated by a series of circular arcs, each uniquely determined by a velocity vector. Creating a path to a goal pose requires the robot to compute a set of velocities for each time interval from 0 to n needed to reach the target. This computation significantly increases the complexity of the problem. In this case, only the velocities considered for the first time interval are used to make the computations easier to handle. Using these velocities is not an issue, given that the procedure is repeated at every time interval.

All possible combinations of rotational and translational velocities are pruned following two rules:

1. Obstacle avoidance. A velocity is admissible if the robot can stop before reaching the obstacle. Let $\dot{\nu_b}$, and $\dot{\omega_b}$ be the braking decelerations. Then, the set V_a of admissible velocities is defined as

$$V_a = \{\nu, \omega | \nu \leq \sqrt{2 \cdot dist(\nu, \omega) \cdot \dot{\nu_b}} \wedge \omega \leq \sqrt{2 \cdot dist(\nu, \omega) \cdot \dot{\omega_b}}\} \tag{2.1}$$

   Where $dist(\nu, \omega)$ is the distance to the closest obstacle, granting the robot the ability to stop without colliding.

2. Dynamic window. To consider the limited accelerations the motors can produce, the overall search space is reduced to the dynamic window, which includes only the velocities that can be obtained within the following time interval. Let t be the time interval, let $\dot{\nu}$ and $\dot{\omega}$ be the accelerations applied for the duration of the t with an initial velocity vector ν_a, ω_a. The dynamic window V_d is defined as

$$V_d = \{\nu, \omega | \nu \in [\nu_a - \dot{\nu} \cdot t, \nu_a + \dot{\nu} \cdot t] \wedge \omega \in [\omega_a - \dot{\omega} \cdot t, \omega_a + \dot{\omega} \cdot t]\} \tag{2.2}$$

   The center of the dynamic window is fixed around the actual velocity. Furthermore, its size depends on the acceleration which the motor can provide.

The resulting search space is an area V_r, which considers both these limitations.

2.1.2 Optimization function

Once V_r is determined, the maximum of the objective function is computed over it. The maximizing velocity is then chosen as the one to be used. The objective function:

$$G(\nu, \omega) = \sigma(\alpha \cdot heading(\nu, \omega) + \beta \cdot dist(\nu, \omega) + \gamma \cdot velocity(\nu, \omega)) \tag{2.3}$$

aims at balancing the following criteria:

- Target heading: measures the angle between the robot and the target direction. Given by 180-θ, where θ is the angle of the target point relative to the robot's heading direction.
- Clearance: aimed at maximizing the distance to obstacles. The function dist(ν, ω) represents the distance to the closest obstacle intersecting with the circular arc.

- Velocity: the function velocity (ν, ω) is used to evaluate the robot's progress on the corresponding trajectory. It is simply a projection on the translational velocity ν.

The controller's behavior depends on how the weights of the optimization functions are chosen, meaning that a certain amount of tuning is required depending on the application where this algorithm needs to be employed.

2.2 MPPI

The **MPPI** controller is a local trajectory planner that employs the Model Predictive Path Integral algorithm to track a specified path while dynamically adjusting to prevent collisions. This algorithm is rooted in a stochastic trajectory approach, specifically within the path integral control framework [16].

2.2.1 Stochastic trajectory optimization

The path integral control method, which leverages random sampling of trajectories, provides a robust mathematical foundation for creating optimal control algorithms. Algorithms developed under this framework have a significant advantage in that they are independent of the need for derivatives of the dynamics or cost functions. This characteristic allows for more flexible and resilient system dynamics estimation and cost function design.

Path integral control is generally obtained using an exponential transformation of the value function within the optimal control problem. Additionally, it assumes that noise only affects the system's actuated states. This assumption enables the transformation of the stochastic Hamilton-Jacobi-Bellman equation into a linear partial differential equation, which can subsequently be converted into a path integral using the Feynman-Kac lemma. The resulting path integral represents an expectation over trajectories under the system's uncontrolled dynamics and is approximated using Monte Carlo sampling to determine the control for the current state.

However, an alternative formulation offers two advantages over the traditional path integral approach:

1. it allows for noise in both directly and indirectly actuated states by relaxing the relationship between noise and control authority
2. it provides optimal settings across the entire time horizon rather than just at the initial time

.In the context of stochastic dynamical systems, where the state and controls at time t are denoted as $x_t \in \mathbb{R}^n$ and $u_t \in \mathbb{R}^m$, respectively, these dynamics are influenced by Brownian motion $dw \in \mathbb{R}^p$. Let $u(\cdot) : [t_0, T] \to \mathbb{R}^m$ be the function mapping time to control inputs, and $\tau : [t_0, T] \to \mathbb{R}^n$ represent the system's trajectory. In classical stochastic optimal control, the goal is to find a control sequence $u(\cdot)$ that minimizes the objective function:

$$u^*(\cdot) = \arg\min_{u(\cdot)} \mathbb{E}\left[\varphi(x_T, T) + \int_{t_0}^{T} L(x_t, u_t, t)\, dt\right] \tag{2.4}$$

With the expectation being evaluated in relation to the dynamics:

$$dx = F(x_t, u_t, t)\, dt + B(x_t, t)\, dw. \tag{2.5}$$

Here, the cost function $L(x_t, u_t, t)$ is composed of an arbitrary state-dependent term and a quadratic control cost:

$$L(x_t, u_t, t) = q(x_t, t) + \frac{1}{2} u_t^T R(x_t, t) u_t \tag{2.6}$$

while the dynamics are affine in controls:

$$F(x_t, u_t, t) = f(x_t, t) + G(x_t, t) u_t \tag{2.7}$$

Path integral control can be interpreted using information-theoretic concepts such as free energy and relative entropy, like in [17]. The following equality captures this interpretation:

$$-\lambda F(S(\tau)) = \inf_Q \left[\mathbb{E}_Q[S(\tau)] + \lambda D_{KL}(Q \parallel P)\right] \tag{2.8}$$

Here, $\lambda \in \mathbb{R}^+$, and $S(\tau)$ represents the state-dependent cost-to-go term $\varphi(x_T, T) + \int_{t_0}^{T} q(x_t, t)\, dt$. The free energy $F(S(\tau))$ is described as $\log\left(\mathbb{E}_P\left[\exp\left(-\frac{1}{\lambda} S(\tau)\right)\right]\right)$. P denotes the probability measure over the space of trajectories generated by the uncontrolled stochastic dynamics: $f(x_t, t)\, dt + B(x_t, t)\, dw$. Q is any probability measure over the space of continuous trajectories with respect to P, meaning they agree on the sets that have measure zero. The relative entropy $D_{KL}(Q \parallel P)$ is defined as $\mathbb{E}_Q\left[\log\left(\frac{dQ}{dP}\right)\right]$.

Controlled dynamics introduce a different probability measure on the trajectory space, denoted as $Q(u)$. The relative entropy between the uncontrolled distribution P and the controlled distribution $Q(u)$ is calculated using Girsanov's theorem, leading to:

$$D_{KL}(Q(u) \parallel P) = \frac{1}{2} \int_{t_0}^{T} u_t^T G(x_t, t)^T \Sigma(x_t, t)^{-1} G(x_t, t)\, u_t\, dt \tag{2.9}$$

where $\Sigma(x_t, t) = B(x_t, t) B(x_t, t)^T$. Supposing that the control cost matrix is of the form:

$$R(x_t, t) = \lambda G(x_t, t)^T \Sigma(x_t, t)^{-1} G(x_t, t) \tag{2.10}$$

The following relationship can be established between the right-hand side of the earlier expression:

$$\mathbb{E}_{Q(u)}[S(\tau)] + \lambda D_{KL}(Q \parallel P) = \mathbb{E}_{Q(u)}\left[S(\tau) + \frac{1}{2}\int_{t_0}^{T} u_t^T R(x_t, t) u_t \, dt\right] \tag{2.11}$$

Additionally, [17] provides an explicit derivation of the optimal probability measure Q^*in relation to the Radon-Nikodym derivative with respect to the uncontrolled dynamics. This derivation takes the form:

$$\frac{dQ^*}{dP} = \frac{\exp\left(-\frac{1}{\lambda}S(\tau)\right)}{\mathbb{E}_P\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]} \tag{2.12}$$

To compute a control law independent of the Hamilton-Jacobi-Bellman (HJB) equation, it is possible to pursue the following optimization scheme: rather than directly solving the optimal control problem by computing the solution to the stochastic HJB equation, the minimization problem can be tackled by optimizing the probability distribution generated by the controller $Q(u)$ to match as closely as possible the optimal probability measure, Q^*, given by the Radon-Nikodym derivative $\frac{dQ^*}{dP}$. Employing the relative entropy between Q^* and $Q(u)$ as a distance metric leads to the formulation of the following minimization problem:

$$u^*(\cdot) = \arg\min_{u(\cdot)} D_{KL}(Q^* \parallel Q(u)) \tag{2.13}$$

This strategy is intuitively appealing since the optimal probability measure Q^* resembles a soft-max function with temperature λ. If a trajectory's cost is low, then $\frac{dQ^*}{dP}$ will have a high value. Thus, if a trajectory is sampled from Q^*, it will likely have a low cost. If $Q(u)$ can become roughly equal to Q^*, then applying the controls $u(\cdot)$ to the system is likely to produce a low-cost trajectory. One can aim to reduce the relative entropy between the optimal distribution Q^* and the distribution $Q(u)$ induced by the controller. By applying the definition of relative entropy:

$$D_{KL}(Q^* \parallel Q(u)) = \mathbb{E}_{Q^*}\left[\log\left(\frac{dQ^*}{dQ(u)}\right)\right] \tag{2.14}$$

To optimize this function, it is essential to determine an expression for the Radon-Nikodym derivative $\frac{dQ^*}{dQ(u)}$:

$$\frac{dQ^*}{dQ(u)} = \frac{dQ^*}{dP} \cdot \frac{dP}{dQ(u)} \tag{2.15}$$

At this moment, Girsanov's theorem can be applied to compute $\frac{dP}{dQ(u)}$:

$$\frac{dP}{dQ(u)} = \exp(D(\tau, u(\cdot))) \tag{2.16}$$

Where $D(\tau, u(\cdot))$ is defined as:

$$\begin{aligned} D(\tau, u(\cdot)) = &- \int_0^T u_t^T G(x_t,t)^T \Sigma(x_t,t)^{-1} B(x_t,t) dw(0) \\ &+ \frac{1}{2} \int_0^T u_t^T G(x_t,t)^T \Sigma(x_t,t)^{-1} G(x_t,t) u_t dt \end{aligned} \tag{2.17}$$

Here, $dw^{(0)}$ represents a Brownian motion with respect to P, satisfying $\mathbb{E}_P[\int_0^t dw^{(0)}] = 0$ for all t.

Applying the previously derived equation, $D_{KL}(Q^* \parallel Q(u))$ can be expressed as:

$$D_{KL}(Q^* \parallel Q(u)) = \mathbb{E}_{Q^*}\left[\log\left(\frac{\exp(-S(\tau)/\lambda) \cdot \exp(D(\tau, u(\cdot)))}{\mathbb{E}_P[\exp(-S(\tau)/\lambda)]}\right)\right] \tag{2.18}$$

This equation can be reformulated as:

$$D_{KL}(Q^* \parallel Q(u)) = \mathbb{E}_{Q^*}\left[-\frac{S(\tau)}{\lambda} + D(\tau, u(\cdot)) - \log(\mathbb{E}_P[\exp(-S(\tau)/\lambda)])\right] \tag{2.19}$$

Given that $S(\tau)$ is independent of the controls $u(\cdot)$, it is possible simplify the minimization problem:

$$\arg\min_{u(\cdot)} D_{KL}(Q^* \parallel Q(u)) = \arg\min_{u(\cdot)} \mathbb{E}_{Q^*}[D(\tau, u(\cdot))] \tag{2.20}$$

The objective is to determine the function $u^*(\cdot)$ that minimizes this equation. Since control is typically applied at discrete time intervals, it is viable to consider only step functions :

$$\mathbf{u}_t = \begin{cases} \vdots \\ \mathbf{u}_j & \text{if } j\Delta t \leq t \leq (j+1)\Delta t \\ \vdots \end{cases} \tag{17}$$

Applying this formulation to $D(\tau, u(\cdot))$, it becomes:

$$\begin{aligned} D(\tau, u(\cdot)) = &- \sum_{j=0}^{N} u_j^T \int_{t_j}^{t_{j+1}} G(x_t, t) dw^{(0)} \\ &+ \frac{1}{2} u_j^T \int_{t_j}^{t_{j+1}} H(x_t, t) dt u_j \end{aligned} \tag{2.21}$$

Where:

$$\begin{aligned} &\text{i.} \quad G(x,t) = G(x,t)^T \Sigma(x,t)^{-1} B(x,t) \\ &\text{ii.} \quad H(x,t) = G(x,t)^T \Sigma(x,t)^{-1} G(x,t) \\ &\text{iii.} \quad N = T/\Delta t \end{aligned}$$

Since u_j is trajectory-independent, applying the expectation operator yields:

$$\begin{aligned} \mathbb{E}_{Q^*}[D(\tau, u(\cdot))] = &- \sum_{j=0}^{N} u_j^T \mathbb{E}_{Q^*} \left[\int_{t_j}^{t_{j+1}} G(x_t, t) dw^{(0)} \right] \\ &+ \sum_{j=0}^{N} \frac{1}{2} u_j^T \mathbb{E}_{Q^*} \left[\int_{t_j}^{t_{j+1}} H(x_t, t) dt \right] u_j \end{aligned} \tag{2.22}$$

This formulation is convex with respect to each u_j. To find u_j^*, the gradient of the equation can be taken with respect to u_j, it can be set to zero, and can be solved:

$$u_j^* = \mathbb{E}_{Q^*} \left[\int_{t_j}^{t_{j+1}} H(x_t, t) dt \right]^{-1} \mathbb{E}_{Q^*} \left[\int_{t_j}^{t_{j+1}} G(x_t, t) dw^{(0)} \right] \tag{2.23}$$

For small Δt, the following approximations are reasonable:

$$\int_{t_j}^{t_{j+1}} H(x_t, t) dt \approx H(x_{t_j}, t_j) \Delta t \tag{2.24}$$

$$\int_{t_j}^{t_{j+1}} G(x_t, t) dw(0) \approx G(x_{t_j}, t_j) \int_{t_j}^{t_{j+1}} dw^{(0)} \tag{2.25}$$

This leads to:

$$u_j^* = \frac{1}{\Delta t} \mathbb{E}_{Q^*}[H(x_{t_j}, t_j)]^{-1} \mathbb{E}_{Q^*} \left[G(x_{t_j}, t_j) \int_{t_j}^{t_{j+1}} dw^{(0)} \right] \tag{2.26}$$

As sampling from Q^* is not feasible, it is necessary to modify the expectation to be with respect to the uncontrolled dynamics $\mathbb{P}$:

$$\mathbf{u}_j^* = \frac{1}{\Delta t}\mathrm{E}_{\mathbb{P}}\left[\frac{\exp(-\frac{1}{\lambda}S(\tau))\boldsymbol{\mathcal{H}}(\mathrm{x}_{t_j}, t_j)}{\mathbb{E}_{\mathbb{P}}[\exp(-\frac{1}{\lambda}S(\tau))]}\right]^{-1}\mathbb{E}_{\mathbb{P}}\left[\frac{\exp(-\frac{1}{\lambda}S(\tau))\boldsymbol{\mathcal{G}}(\mathrm{x}_{t_j}, t_j)\int_{t_j}^{t_{j+1}} dw^{(0)}}{\mathrm{E}_{\mathbb{P}}[\exp(-\frac{1}{\lambda}S(\tau))]}\right] \tag{2.27}$$

To approximate the controls, the trajectories can be sampled from $\mathbb{P}$

2.2.2 Matrix formulation

Assuming the diffusion matrix and the control matrix have the form

$$\begin{aligned} B(x_t) &= \begin{pmatrix} B_{a(x_t)} & 0 \\ 0 & B_c \end{pmatrix} \\ G &= \begin{pmatrix} 0 \\ G_c \end{pmatrix} \end{aligned} \tag{2.28}$$

In the scenario under examination, no correlations exist between the noise in directly and indirectly actuated states. Furthermore, the diffusion for the directly actuated states remains independent of the state. Consequently, the covariance matrix takes on the following form:

$$\Sigma(x) = \begin{bmatrix} B_a(x)B_a(x)^T & 0 \\ 0 & B_cB_c^T \end{bmatrix} \tag{2.29}$$

The expressions $\mathcal{H}(x_{t_j})$ and $\mathcal{G}(x_t)$ no longer exhibit state dependence, allowing for their simplification:

$$\mathcal{H} = G_c^T(B_cB_c^T)^{-1}G_c \tag{2.30}$$

$$\mathcal{G} = G_c^T(B_cB_c^T)^{-1}B_c \tag{2.31}$$

This state independence permits the extraction of these matrices from the expectation, resulting in:

$$\mathbf{u}_j^* = \frac{1}{\Delta t}\boldsymbol{\mathcal{H}}^{-1}\boldsymbol{\mathcal{G}}\left(\mathbb{E}_{\mathbb{P}}\left[\int_{t_j}^{t_{j+1}}\frac{\exp(-\frac{1}{\lambda}S(\tau))dw^{(0)}}{\mathbb{E}_{\mathbb{P}}[\exp(-\frac{1}{\lambda}S(\tau))]}\right]\right) \tag{2.32}$$

2.2.3 Numerical approximation

At this moment, numerical evaluation of equation 2.32 becomes essential. However, two distinct challenges must be addressed:

1. Reformulating the equation for discrete-time sampling

2. Devising a method to implement importance sampling in conjunction with equation 2.32, given that the expectation is relative to the uncontrolled dynamics

In the discrete-time domain, the system's dynamics can be expressed as:

$$\mathrm{dx}_{t_j} = (\mathbf{f}(\mathbf{x}_{t_j},\ t_j) + \mathbf{G}(\mathbf{x}_{t_j},\ t_j)\mathbf{u}_j)\Delta t + \mathbf{B}(\mathrm{x}_{t_j},\ t_j)\epsilon_j\sqrt{\Delta t} \tag{2.33}$$

In this context, ε_j represents a vector where each component is a standard normal random variable. Incorporating $\varepsilon_j\sqrt{\Delta t}$ into equation 2.32 results in:

$$\mathbf{u}_j^* = \frac{1}{\Delta t}\boldsymbol{\mathcal{H}}^{-1}\boldsymbol{\mathcal{G}}\left(\mathbb{E}_p\left[\frac{\exp(-\frac{1}{\lambda}S(\tau))\epsilon_j\sqrt{\Delta t}}{\mathbb{E}_p[\exp] - \frac{1}{\lambda}S(\tau))]}\right]\right) \tag{2.34}$$

Here, p denotes the probability distribution associated with the discrete-time uncontrolled dynamics (i.e., equation 2.33 with u_{t_j} set to zero). Rather than sampling from p, one might select a different probability distribution q_u^ν. Such a probability distribution is associated with the following dynamics:

$$\mathrm{dx}_{t_j} = \mathbf{f}(\mathbf{x}_{t_j},\ t_j)\Delta t + \mathbf{G}(\mathbf{x}_{t_j},\ t_j)\mathbf{u}_{t_j}\Delta t + \mathbf{B}_E(\mathbf{x}_{t_j},\ t_j)\epsilon_j\sqrt{\Delta t} \tag{2.35}$$

The new diffusion matrix B_E is represented as:

$$B_E(x_t) = \begin{bmatrix} B_a(x_t) & 0 \\ 0 & \nu B_c \end{bmatrix} \tag{2.36}$$

where $\nu \geq 1$. When sampling from the distribution mentioned above, the designer can select:

1. The initial controls around which sampling is centered.
2. The magnitude of the exploration variance, which is ν.

To sample from q_u^ν instead of p, it is necessary to compute the likelihood ratio between the two distributions [18]. Incorporating the likelihood ratio corresponds to modifying the running cost from $q(x_t, t)$ to:

$$\begin{aligned}\tilde{q}(\mathbf{x}_t,\ \mathbf{u}_t,\ \epsilon_t,\ t) =& q(\mathbf{x}_t,\ t) + \frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\mathbf{R}\mathbf{u}_t + \lambda\mathbf{u}^{\mathrm{T}}\boldsymbol{\mathcal{G}}\frac{\epsilon}{\sqrt{\Delta t}} + \\ & \frac{1}{2}\lambda(1-\nu^{-1})\frac{\epsilon^{\mathrm{T}}}{\sqrt{\Delta t}}\mathbf{B}_c^{\mathrm{T}}(\mathbf{B}_c\mathbf{B}_c^{\mathrm{T}})^{-1}\mathbf{B}_c\frac{\epsilon}{\sqrt{\Delta t}}\end{aligned} \tag{2.37}$$

The first two additional terms can be explained as penalties for shifting the mean of the exploration away from zero. In contrast, the last term penalizes sampling from an overly aggressive variance. $\mathbf{B}_c\frac{\epsilon}{\sqrt{\Delta t}}$ is the effective change in control input caused by noise. When sampling from a distribution with non-zero control input, the update law 2.32 changes. This change is due to the random variable shifting from the zero-mean term $B_c\varepsilon_j\sqrt{\Delta t}$ to the non-zero mean term: $G_c u\Delta t + B_E\varepsilon_j\sqrt{\Delta t}$. Substituting this term in 2.34 and simplifying, and denoting $\tilde{S}(\tau)$ as:

$$\tilde{S}(\tau) = \phi(\mathbf{x}_T,\ T) + \sum_{j=0}^{N}\tilde{q}(\mathbf{x}_t,\ \mathbf{u}_t,\ \epsilon_t,\ t)\Delta t \tag{2.38}$$

We obtain the iterative update rule:

$$\mathbf{u}_j^* = \mathbf{u}_j + \boldsymbol{\mathcal{H}}^{-1}\boldsymbol{\mathcal{G}}\left(\mathbb{E}_{q_u^\nu}\left[\frac{\exp(-\frac{1}{\lambda}\tilde{S}(\tau))\frac{\epsilon_j}{\sqrt{\Delta t}}}{\mathbb{E}_{q_u^\nu}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau)\right)\right]}\right]\right) \tag{2.39}$$

The component in the parentheses is estimated as:

$$\sum_{k=1}^{K}\left(\frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)\frac{\epsilon_{j,k}}{\sqrt{\Delta t}}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)}\right) \tag{2.40}$$

where each trajectory τ_k is drawn from the sampling dynamics 2.35. This iterative update law can be interpreted as computing the new control derived from a reward-weighted average over trajectories.

2.2.4 Control algorithm

The final update equation 2.39 can be applied in a model predictive control setting. In this context, optimization is performed dynamically: the trajectory is optimized, a single control input is executed, and then the optimization is repeated.

The path integral control derivation presented here offers a method for optimizing the entire control sequence rather than just the control input at the current time step. Therefore, the unused portion of the control sequence can initiate the subsequent optimization round.

This approach is essential for the algorithm's effectiveness, as a complex system functioning at an adequate control frequency typically allows only a limited number of iterations to be completed within each time step.

One of the main advantages of the presented implementation is that the bulk of the computation used in the update rule can be performed in parallel. The parallelization introduced in this case implies parallel trajectory sampling on a **GPU**, an efficient operation that can handle thousands of trajectories from complex dynamics. The description of the algorithm is given in 1. In order to run this algorithm, the simplest method is to parallelize the sampling for-loop by employing one thread per sample, which provides a massive uplift in the time required to perform the algorithm.

Algorithm 1 MPPI optimization sequence

```
**Given**:
$K$: Number of samples
$N$: Number of time-steps
$(u_0, u_1, \dots, u_{N-1})$: Initial control sequence
$\Delta t, x_{t_0}, f, G, B, B_E$: System/sampling dynamics
$\phi, q, R, \lambda$: Cost parameters
$u_{\text{init}}$: Value to initialize new controls to
while task not completed do
    for $k \leftarrow 0$ to $K-1$ do
        $x = x_{t_0}$
        for $i \leftarrow 1$ to $N-1$ do
            $x_{i+1} = x_i + (f + Gu_i)\Delta t + BE\epsilon_{i,k}\sqrt{\Delta t}$
            $\tilde{S}(\tau_k) = S(\tau_k) + \tilde{q}(x_i, u_i, \epsilon_{i,k}, t_i)$
        end for
    end for
    for $i \leftarrow 0$ to $N-1$ do
        $u_i = u_i + \mathcal{H}^{-1} g \left( \sum_{k=1}^{K} \frac{\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)\epsilon_{i,k}\sqrt{\Delta t}}{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)} \right)$
    end for
    send $(u_0)$ to actuators
    for $i \leftarrow 0$ to $N-2$ do
        $u_i = u_{i+1}$
    end for
    $u_{N-1} = u_{\text{init}}$
    Update the current state after receiving feedback
    check for task completion
end while
```

2.3 SFM

The **SFM** is a microscopic pedestrian behavior model introduced in 1995 [19]. It is widely used to model pedestrian movements and interactions in contexts such as:

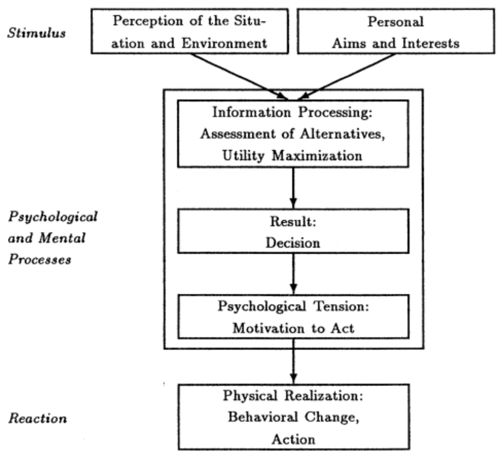
- Crowd simulation
- Evacuation dynamics
- Urban planning

The **SFM** handles pedestrians as particles subject to social forces that dictate their movement. This model operates continuously in space and time, modeling each pedestrian separately and allowing each agent to exhibit individual characteristics.

2.3.1 The social force concept

Human behavior can be difficult to predict in complex situations, but in simple instances, a stochastic behavioral model may describe behavioral probabilities that can be observed in a huge population of individuals [20]. Another approach for modeling behavioral changes is the usage of the so-called social fields or social forces, first suggested in 1951 [21]. According to figure 2.1, a sensory stimulus provokes a behavioral reaction influenced by personal aims and selected from a set of behavioral alternatives to maximize utility.

The classification of sensory stimuli can be organized into two types:



Stimulus
Perception of the Situation and Environment
Personal Aims and Interests
Information Processing: Assessment of Alternatives, Utility Maximization
Psychological and Mental Processes
Result: Decision
Psychological Tension: Motivation to Act
Reaction
Physical Realization: Behavioral Change, Action

Figure 2.1: Schematic illustration of processes causing behavioral changes (Reproduced from [19],1995, p.4282)

- Simple or standard situations are characterized by their predictability;
- Complex or new situations lend themselves to modeling through probabilistic approaches.

However, it should be noted that pedestrians often develop automatic responses through repeated exposure to various scenarios. These reactions are shaped by their accumulated experiences in determining the most efficacious course of action. Consequently, it becomes feasible to articulate the governing principles of pedestrian behavior through the formulation of an equation of motion.

According to this equation, the systematic temporal variations $\frac{d\vec{\omega_\alpha}}{dt}$ of the preferred velocity $\vec{\omega_\alpha}(t)$ of a pedestrian α are characterized by a vectorial quantity $\mathbf{F}_\alpha(t)$ that can be interpreted as a social force. This force must represent the influence of the environment (e.g., other pedestrians or boundaries) on the behavior of the described pedestrian.

Despite that, it is crucial to note that the environment does not physically exert a social force on a pedestrian's body. On the contrary, it is a quantity that describes the concrete motivation to act. In the context of pedestrian behavior, this incentive causes the physical production of an acceleration or deceleration force to respond to the perceived information that the individual obtains about their environment (see Fig. 2.1). In conclusion, one can affirm that a pedestrian behaves as if they were subject to external forces.

2.3.2 Formulation of the social force model

Multiple effects influence the motion of a pedestrian α. The pedestrian aims to reach a specific destination $\vec{\mathbf{r}_\alpha^0}$ with maximum comfort. Consequently, the individual typically selects a route without detours, choosing the shortest path. This path generally assumes the form of a polygon with edges $\vec{\mathbf{r}_\alpha^1}, \ldots, \vec{\mathbf{r}_\alpha^n} := \vec{\mathbf{r}_\alpha^0}$. If $\vec{\mathbf{r}_\alpha^k}$ denotes the next edge of this polygon to be approached, the pedestrian's desired direction of motion $\vec{\mathbf{e}_\alpha}(t)$ can be expressed as follows:

$$\vec{\mathbf{e}_\alpha}(t) := \frac{\vec{\mathbf{r}_\alpha^k} - \vec{\mathbf{r}_\alpha}(t)}{\|\vec{\mathbf{r}_\alpha^k} - \vec{\mathbf{r}_\alpha}(t)\|} \tag{2.41}$$

Where $\vec{\mathbf{r}_\alpha}(t)$ denotes the current position of the pedestrian at time t, and $\|\vec{\mathbf{r}_\alpha^k} - \vec{\mathbf{r}}(t)\|$ represents the Euclidean norm. The destinations are typically represented as gates or areas rather than specific points $\vec{\mathbf{r}_\alpha^k}$. In this scenario, the pedestrian continuously adjusts their trajectory towards the nearest point $\vec{\mathbf{r}_\alpha^k}(t)$ of the corresponding gate or area at each time t.

In the absence of disturbances, the pedestrian will proceed in the desired direction $\vec{\mathbf{e}_\alpha}(t)$ at a certain desired speed v_α^0.

A deviation of the actual velocity $\vec{v_\alpha}(t)$ from the desired velocity $\vec{v_\alpha^0}(t) := v_\alpha^0 \vec{\mathbf{e}_\alpha}(t)$, necessary to decelerate or to avoid obstacles, results in a tendency to revert to $\vec{v_\alpha^0}(t)$ within a specific relaxation time τ_α. An acceleration term of the form can characterize this behavior:

$$\vec{\mathbf{F}_\alpha^0}(\vec{v_\alpha}, v_\alpha^0 \vec{e_\alpha}) := \frac{1}{\tau_\alpha}(v_\alpha^0 \vec{e_\alpha} - \vec{v_\alpha}) \tag{2.42}$$

A pedestrian's movement is also affected by other pedestrians. Specifically, the individual maintains a certain distance from other pedestrians dependent on the pedestrian density and the desired speed v_α^0. Here, the private sphere of each pedestrian, which can be interpreted as territorial effect [22], plays an essential role. Pedestrians typically feel increasingly uneasy when approaching a stranger, who may respond aggressively. This discomfort leads to repulsive effects from other pedestrians β, which vector quantities can depict:

$$\vec{\mathbf{f}_{\alpha\beta}}(\vec{\mathbf{r}_{\alpha\beta}}) := -\nabla_{\vec{\mathbf{r}_{\alpha\beta}}} V_{\alpha\beta}[b(\vec{\mathbf{r}_{\alpha\beta}})]. \tag{2.43}$$

The repulsive potential $V_{\alpha\beta}(b)$ can be represented by a monotonic decreasing function of b with equipotential lines creating the shape of an ellipse directed into the direction of motion. This is because a pedestrian requires space for the next step, which is taken into consideration by other pedestrians. The semi-minor axis of the ellipse is denoted by b and is given by

$$2b := \sqrt{(\|\vec{\mathbf{r}_{\alpha\beta}}\| + \|\vec{\mathbf{r}_{\alpha\beta}} - v_\beta \Delta t \vec{e_\beta}\|)^2 - (v_\beta \Delta t)^2}, \tag{2.44}$$

where $\vec{\mathbf{r}_{\alpha\beta}} := \vec{\mathbf{r}_\alpha} - \vec{\mathbf{r}_\beta}$. $s_\beta := v_\beta \Delta t$ is of the order of the step width of pedestrian β. Despite the simplicity of this approach, it effectively describes avoidance maneuvers in pedestrian interactions.

It is important to note that pedestrians also tend to stay far from the borders of buildings, obstacles, streets, walls, etc. They become increasingly uneasy the closer to a border they get, considering they need to pay more attention to avoid getting hurt. A boundary B induces a repulsive effect on a pedestrian, which can be described as:

$$\vec{\mathbf{F}}_{\alpha B}(\vec{\mathbf{r}}_{\alpha B}) = -\nabla_{\vec{\mathbf{r}}_{\alpha B}} U_{\alpha B}(\|\vec{\mathbf{r}}_{\alpha B}\|) \tag{2.45}$$

with $U_{\alpha B}$ being a repulsive and monotonically decreasing potential. Here, the vector $\vec{r}_{\alpha B} := \vec{r}_\alpha - \vec{r}_B^\alpha$ has been introduced, where $\vec{r}_B^\alpha$ represents the location of the closest section of boundary B to pedestrian α.

Pedestrians are sometimes attracted to other persons or objects. These attractive influences $\vec{f_{\alpha i}}$ at the positions $\vec{r_i}$ can be modeled by attractive, monotonically increasing potentials $W_{\alpha i}(\|\vec{r}_{\alpha i}\|, t)$, similar to the repulsive effects:

$$\vec{f}_{\alpha i}(\|\vec{\mathbf{r}}_{\alpha i}\|, t) := -\nabla_{\vec{\mathbf{r}}_{\alpha i}} W_{\alpha i}(\|\vec{\mathbf{r}}_{\alpha i}\|, t) \tag{2.46}$$

where $\vec{r}_{\alpha i} := \vec{r}_\alpha - \vec{r}_i$. The main distinction, however, is that the attractiveness $\|\vec{f}_{\alpha i}\|$ typically decreases over time as interest diminishes. These attractive effects are, for instance, the main element that causes the formation of pedestrian groups (which can be associated with molecules).

However, these formulas for attractive and repulsive effects are only applicable to situations perceived in the intended direction $\vec{e}_\alpha(t)$ of motion. Pedestrians or objects placed behind the pedestrian have a lessened influence c with $0 < c < 1$. To consider this effect of perception (the effective angle 2φ of sight), direction-dependent weights are introduced:

$$w(\vec{e}, \vec{f}) := \begin{cases} 1 & \text{if } \vec{e} \cdot \vec{f} > \|\vec{f}\| \cos\varphi \\ c & \text{otherwise.} \end{cases} \tag{2.47}$$

In conclusion, the repulsive and attractive influences on a pedestrian's behavior can be expressed as:

$$\vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_\beta) := w(\vec{e}_\alpha, -\vec{f}_{\alpha\beta})\vec{f}_{\alpha\beta}(\vec{r}_\alpha - \vec{r}_\beta), \tag{2.48}$$

$$\vec{F}_{\alpha i}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_i, t) := w(\vec{e}_\alpha, \vec{f}_{\alpha i})\vec{f}_{\alpha i}(\vec{r}_\alpha - \vec{r}_i, t). \tag{2.49}$$

The equation computing a pedestrian's total motivation $\vec{F}_\alpha(t)$ can now be established. Given that all previously mentioned effects concurrently influence a pedestrian's decision, their total effect is considered the sum of all such effects, similar to the summation of forces. This results in:

$$\begin{aligned} \vec{F}_\alpha(t) :=& \vec{F}^0_\alpha(\vec{v}_\alpha, v^0_\alpha \vec{e}_\alpha) + \sum_\beta \vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_\beta) \\ &+ \sum_B \vec{F}_{\alpha B}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}^\alpha_B) \\ &+ \sum_i \vec{F}_{\alpha i}(\vec{e}_\alpha, \vec{r}_\alpha - \vec{r}_i, t). \end{aligned} \tag{2.50}$$

The social force model is then described by:

$$\frac{d\vec{w}_\alpha}{dt} = \vec{F}_\alpha(t) + \text{fluctuations}. \tag{2.51}$$

A fluctuation term that accounts for random variations in pedestrian behavior has been added. These fluctuations originate from ambiguous scenarios in which multiple behavioral alternatives are equivalent. Moreover, fluctuations occur from occasional deviations from the standard rules of motion.

Finally, it is necessary to introduce a correlation between the preferred velocity $\vec{\omega_\alpha}(t)$ and the actual velocity $\vec{v_\alpha}(t)$ to complete the pedestrian dynamics. Considering that the actual speed is bounded by a pedestrian's maximal acceptable speed v_α^{max}, the realized motion is considered to be given by

$$\frac{d\vec{r_\alpha}}{dt} = \vec{v_\alpha}(t) := \vec{\omega_\alpha}(t) g\left(\frac{v_\alpha^{max}}{\|\vec{\omega_\alpha}\|}\right) \tag{2.52}$$

with

$$g\left(\frac{v_\alpha^{max}}{\|\vec{\omega_\alpha}\|}\right) := \begin{cases} 1 & \text{if } \|\vec{\omega_\alpha}\| \leq v_\alpha^{max} \\ \frac{v_\alpha^{max}}{\|\vec{\omega_\alpha}\|} & otherwise \end{cases} \tag{2.53}$$

2.4 Nav2 introduction

The algorithms used in the experiment were implemented on both the simulations and the laboratory experiments using **ROS2**, which is a middleware that is often used in the field of robotics due to the tools it provides, which greatly help in connecting the hardware of the robot to its software. The package which allows the robot to navigate autonomously is called **NAV2**, introduced by Steve Macenski in 2020 [23]. The Navigation2 package employs a configurable behavior tree to manage the planning, control, and recovery tasks. Each behavior tree node calls upon a remote server to compute one of the tasks. Each server uses a standard plugin interface to implement new techniques or algorithms easily. The main characteristic of **NAV2** are:

- Reliability: **ROS2** leverages a **DDS** communication standard. Using the data distribution service security features, **ROS2** allows a user to safely convey information to the robot and cloud servers. Moreover, **ROS2** introduces the concept of managed nodes, also called lifecycle nodes. A managed node follows a structured server model with well-defined state transitions, from instantiation through destruction. This server is initialized upon the program's launch but remains idle until an external stimulus triggers its progression through a deterministic bring-up process. During shutdown or in the event of an error, the server transitions through its state machine, moving from an active to a finalized state. Each state transition carries specific responsibilities, including managing memory allocation, networking interfaces, and initiating task processing. In Navigation2, all servers utilize managed nodes to ensure deterministic program lifecycle management and efficient memory allocation.

- Modularity and reconfigurability: It is possible to create configurable behavior trees whose nodes and structure can be loaded at run-time. Each **BT** node controls the data transmission with a remote server containing an algorithm that can be written in various languages. The remote servers are modular, which means that many of them can run at the same time to compute actions.

The overall structure of **NAV2** can be seen in figure 2.2.

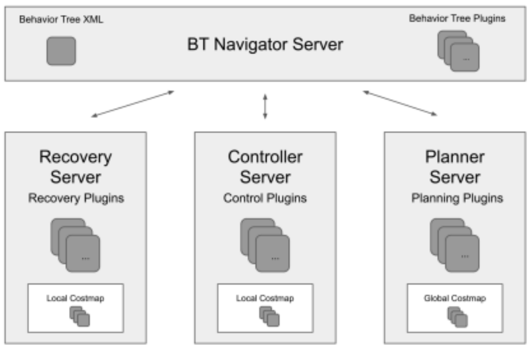


Figure 2.2: Representation of Nav2 design (Reproduced from [23], 2020, p.3)

The top-level node is where custom-made **BT** is loaded and run. Each node call uses a different server to finish a task.

2.4.1 Recovery server

Recovery behaviors are employed to prevent total navigation failures. A recovery server can initiate a recovery behavior, which can be conservative or aggressive. Some possible recovery behaviors are:

- Clear costmap: used to clear costmap layers whenever the perception system fails,
- Spin: used to rotate the robot to help it out of potential local failures,
- Wait: used to wait for dynamic obstacles to move out of the robot trajectory

2.4.2 Planner server and controller server

The objective of the global planner is to compute the shortest route to a goal while the controller uses local data to evaluate the best control signals. The global planner used in this thesis was the planner provided by **NAV2**, called **NavFn** planner, which implements a Dijkstra or A* expanded holonomic planner. The controller used was based on either the **DWA** or on the **MPPI**.

2.4.3 Perception

The recovery server, global planner, and local planner all need information about the robot's environment to function correctly. Nav2 provides information through a layered costmap, allowing the user to modify the used layers while also tuning the rate limits, resolution, etc. This approach also allows the introduction of custom-made layers, which is essential for this work considering that a social costmap layer needs to be introduced, which will be discussed in section 3.1. The costmap is a 2D grid representing the obstacles as black squares on the map (assuming that a scale of gray is used to visualize the map), which are usually surrounded by a halo of decreasing intensity to represent a high-cost zone. The costmap can represent the entire environment around the robot or only a small area surrounding the robot, called global and local costmap, respectively. The global planner uses the first one, while the recovery server and the controller use the second one.

The default Nav2 layers that were used include:

- Static layer: uses the static map of the environment, which is either loaded from disk or obtained by a **SLAM**, to initialize the grid occupancy data,
- Inflation layer: expands lethal obstacles on the costmap with exponential decay,
- Obstacle layer: represents dynamic or static obstacles present in the environment. It can employ a variety of sensors to represent the environment

Chapter 3

Implementation

3.1 Nav2 implementation

After introducing the main theoretical concepts related to the algorithms and the basics of **NAV2**, it is time to talk about the **NAV2** implementation of said algorithms, along with the social costmap plugin which is required for two of the six variants of the algorithms.

3.1.1 Social costmap plugin

The social costmap plugin [24] works as an additional layer that can be added to the costmap to prioritize avoiding human agents. The information about the agents is obtained through a custom topic **/people**. This plugin introduces a high-cost area around the people in the scene that takes the form of an ellipse whose major axis is parallel to the agent's heading. Consequently, the shape of the high-cost area is circular for static agents. This plugin is configurable with a series of parameters, which include:

Parameter	Type	Definition
enabled	bool (default: True)	Whether to apply this plugin or not.
cutoff	double (default: 5.0)	Smallest cost value to publish on the costmap.
amplitude	double (default: 255.0)	Amplitude of adjustments at peak. Maximum cost on the person center.
covariance_front_height	double (default: 0.25)	Covariance of the ordinate axis of the Gaussian at the person heading.

Parameter	**Type**	**Definition**
covariance_front_width	double (default: 0.25)	Covariance of the abscissa axis of the Gaussian at the person's heading.
covariance_rear_height	double (default: 0.25)	Covariance of the ordinate axis of the Gaussian at the person's rear.
covariance_rear_width	double (default: 0.25)	Covariance of the abscissa axis of the Gaussian at the person's rear.
covariance_when_still	double (default: 0.25)	Covariance employed to form a circular cost around the person when she is not moving.
use_vel_factor	bool (default: True)	Whether to use the current person velocity and the **speed_factor_multiplier** to modify the Gaussian at the person's front or not.
speed_factor_multiplier	double (default: 5.0)	Factor with which to scale the velocity.
use_passing	bool (default: True)	Whether to use another Gaussian on the right side of the person in order to ease the passing maneuvers, for instance, in corridors.
covariance_right_height	double (default: 0.25)	Covariance of the ordinate axis of the Gaussian at the person's right side.
covariance_right_width	double (default: 0.25)	Covariance of the abscissa axis of the Gaussian at the person's right side.
publish_occgrid	bool (default: False)	Whether to publish an OccupancyGrid with only the social costs.

Table 3.1: Social costmap plugin parameters

This thesis used the social costmap plugin only for the local costmap. This is because the main objective is to evaluate the difference in the performance of the local controller. Adding a social layer on the global costmap would also cause the computed global path to be different, giving an advantage to the algorithm variants using this plugin.

3.1.2 DWA implementation

There is a variety of algorithms that can be used in **NAV2** which provide different characteristics and can be used on different types of robots [25]. The problem is that most of them do not include a way to consider the influence of other human agents in the environment. Taking the DWA algorithm into account, a basic implementation would be to maximize the optimization function of the DWA (2.3) on the search space after adding a term that considers the presence of the agents. This was achieved [26] considering the effect of the **SFM** as a "social work", which effectively adds a penalty term to the optimization function. The total social work comprises the work performed by the robot W_r and the work caused by the robot on the nearby pedestrians W_p. Considering the robot as agent 0, and a total number of human agents n:

$$W_{social} = W_r + \sum_{i=1}^{n} W_{p_i} \tag{3.1}$$

Where:

- W_r is the summation of the modulus of the robot social force and the robot obstacle force along the trajectory.
- W_p is the summation of the modulus of the social forces originated by the robot for each close pedestrian.

The parameters involved in this algorithm consist of the following:

- **Robot Configuration Parameters**

Parameter	**Type**	**Definition**
max_trans_acc	double	Maximum acceleration in translation (m/s^2).
max_rot_acc	double	Maximum acceleration in rotation (rad/s^2).
max_trans_vel	double	Maximum linear velocity (m/s).
min_trans_vel	double	Minimum linear velocity (m/s).
max_rot_vel	double	Maximum angular velocity (rad/s).
min_rot_vel	double	Minimum angular velocity (rad/s).
min_in_place_rot_vel	double	Angular velocity of stationary rotations (rad/s).

Table 3.2: DWA robot configuration parameters

- **Goal Tolerance Parameters**

Parameter	Type	Definition
yaw__goal__tolerance	double	Tolerance in angular distance (rad) to determine that the goal has been achieved.
xy__goal__tolerance	double	Tolerance in Euclidean distance (m) to determine that the goal has been achieved.
wp__tolerance	double	Distance (m) from the robot to search for the next point of the global plan to pursue.

Table 3.3: DWA goal tolerance parameters

- **Forward Simulation Parameters**

Parameter	Type	Definition
sim__time	double	Time (seconds) to expand the robot movement and test for collisions. (default: 0.5).
sim__granularity	double	Resolution in meters to divide the expanded trajectory and inspect for collisions (default: 0.025).

Table 3.4: DWA forward simulation parameters

- **Sensor Interface Parameters**
  The sensor interface manages the acquisition of sensory input and subsequently updates the data regarding the social agents in the surrounding environment.

Parameter	Type	Definition
laser__topic	string	Topic in which the laser range finder is being published [sensor_msgs/LaserScan].

people_topic	string	Topic in which the people detected are being published [people_msgs/People].
odom_topic	string	Odometry topic [nav_msgs/Odometry].
max_obstacle_dist	double	Maximum distance (m) in which the obstacles are considered for the social force model.
naive_goal_time	double	Lookahead time (secs) to predict an approximate goal for the pedestrians.
people_velocity	double	Average velocity of the pedestrians (m/s).

Table 3.5: DWA sensor interface parameters

- **Social Force Model Parameters**
  The weights of the SFM forces for people trajectory computation.

Parameter	**Type**	**Definition**
sfm_goal_weight	double	Weight of the attraction force to the goal.
sfm_obstacle_weight	double	Weight of the repulsive force of the obstacles.
sfm_people_weight	double	Weight of the repulsive force of the pedestrians.

Table 3.6: DWA Social Force Model parameters

The function to optimize is slightly modified, and it assumes the following form:

$$C_{traj} = C_s\omega_s + C_{cm}\omega_{cm} + C_a\omega_a + C_\nu\omega_\nu + C_d\omega_d \tag{3.2}$$

Where the C terms represent the cost, and their respective weights are:

- **social_weight** ω_s is the weight attributed to W_{social}
- **costmap_weight** ω_{cm} is the weight attributed to the costmap value.
- **angle_weight** ω_a is the weight attributed to the angle difference between robot heading and path heading.
- **vel_weight** ω_ν is the weight attributed to the difference between the linear maximum velocity allowed and the linear velocity evaluated.

- **distance_weight** ω_d is the weight attributed to the distance between the final point of the projected robot trajectory and the current local goal.

It has to be noted that, by setting the social_weight to zero, this function corresponds to a **DWA**, which means that this algorithm provides both the basic **DWA** and the version which implements the **SFM**.

3.1.3 MPPI implementation

In the **NAV2** implementation of the **MPPI**, the controller leverages tensor representations to group process trajectories. The trajectory planner produces randomly noised controls utilizing Gaussian distributions for each control axis. The generated controls are added to the previous optimal trajectory's velocities to develop a series of trajectories with random disturbances. Constraints are imposed on these trajectories' velocity limits and turning radii to render them physically achievable for omnidirectional, Ackermann, and differential-drive robots. Using time samples in the trajectory, the velocities are then projected to build the trajectory poses. Next, these trajectories are scored employing several plugin-based critic functions to identify the best trajectory in the batch. The output scores are utilized to set the best control with a softmax function. This control is then considered the basis of the starting control of the following time step. This implementation of the MPPI also includes a trajectory visualizer to represent the sampled trajectories used in the algorithm in real-time (in the case the controller is used in a simulation environment). The main parameters to be customized are:

- **Controller parameters**

Parameter	**Type**	**Definition**
motion_model	string	Default: DiffDrive. Type of model which represents the used robot.
critics	string	Default: None. Critics (plugins) names to be added to score the trajectories.
iteration_count	int	Default 1. Iteration count in MPPI algorithm.
batch_size	int	Default 1000. Count of randomly generated possible trajectories.
time_steps	int	Default 56. Amount of time steps (points) in each sampled trajectory.
model_dt	double	Default 0.05. Time interval (s) between two sampled points.

vx_std	double	Default 0.2. Sampling standard deviation for vx.
vy_std	double	Default 0.2. Sampling standard deviation for vy.
wz_std	double	Default 0.4. Sampling standard deviation for Wz.
vx_max	double	Default 0.5. Max vx (m/s).
vy_max	double	Default 0.5. Max vy in either direction, if holonomic (m/s).
vx_min	double	Default -0.35. Min vx (m/s).
wz_max	double	Default 1.9. Max wz (rad/s).
temperature	double	Default 0.3. Selectiveness of trajectories by their costs. 0 represents control with the best cost, while a large value leads in mean of all trajectories.
gamma	double	Default 0.015. A trade-off between smoothness (high) and low energy (low).
visualize	bool	Default false. Publish visualization of trajectories.
retry_attempt_limit	int	Default 1. Number of attempts to find viable trajectories on failure.
regenerate_noises	bool	Default false. Whether to regenerate noises each iteration.

Table 3.7: MPPI controller parameters

- **Trajectory visualizer parameters**

Parameter	**Type**	**Definition**
trajectory_step	int	Default: 5. The step between trajectories to visualize to downsample candidate trajectory pool.
time_step	int	Default: 3. The step between points on trajectories to visualize to downsample trajectory density.

Table 3.8: MPPI trajectory visualizer parameters

- **Path handler parameters**

Parameter	**Type**	**Definition**
max_robot_pose_search_dist	double	Default: Costmap half-size. Max integrated distance ahead of robot pose to search for nearest path point in case of path looping.
prune_distance	double	Default: 1.5.Distance ahead of the nearest point on the path to the robot to prune path to (m).
transform_tolerance	double	Default: 0.1. Time tolerance for data transformations with TF.
enforce_path_inversion	double	Default: False. If true, it will prune paths containing cusping points for segments changing directions, forcing the controller to change direction at the planner requested point
inversion_xy_tolerance	double	Default: 0.2. Cartesian proximity (m) to path inversion point to be considered "achieved" to pass on the rest of the path after path inversion.
inversion_yaw_tolerance	double	Default: 0.4. Angular proximity (radians) to path inversion point to be considered "achieved" to pass on the rest of the path after path inversion. 0.4 rad = 23 deg.

Table 3.9: MPPI path handler parameters

This controller, by default, includes a wide array of critics, which can greatly impact the behavior of the **MPPI**:

- **Constraint Critic**: when active, the controller is more inclined to keep the velocity value within the defined limits.
- **Goal angle Critic**: when active, the controller is more likely to reach the target with an angle θ_r that closely approximates the goal pose angle θ.
- **Goal Critic**: when active, the controller aims to reach the given goal pose.
- **Obstacles Critic**: when active, the controller tries to avoid obstacles and keep a large distance from them.

- **Path Align Critic**: when active, the controller tries to align its optimal trajectory to the provided global path.
- **Path Angle Critic**: when active, the controller keeps its optimal trajectory parallel to the global path.
- **Path Follow Critic**: when active, the controller is more inclined to keep moving along its currently defined trajectory. Keeping this high enough value is important to prevent the robot from stalling in low-cost areas.

In its basic form, the **MPPI** controller performs well in obstacle avoidance and is highly customizable, making it an excellent choice for differential robots. The only problem is that it is not social on its own, which would make it undesirable considering the thesis' scope is on performing experiments in social scenarios. The plugin structure, however, allows the user to create and implement custom plugins to add extra functionalities. This customizability led to developing a new plugin that implements the **SFM** to score the random trajectories: the social critic.

3.2 Social critic

The social critic acts alongside the other critic functions already provided by the MPPI controller, so its structure shares some similarities with those plugins. It starts with an Initialize function and gets all the necessary parameters defined in the YAML file; the main difference is that three different subscribers and two publishers are also created. The three subscribers used are:

- *people_sub*, which gets information about agents in the scene;
- *odom_sub*, which obtains the odometry of the robot, agent 0;
- *laser_sub*, which acquires information from the LiDar.

The subscribers mentioned above are associated with three callback functions, respectively with the *people_callback*, *odom_callback*, and *laser_callback*, which provide the plugin with the necessary data. The two publishers are instead employed in creating markers to distinguish the used agents at a particular time from the points obtained by the **LiDaR**, respectively. The other aspect consistent with all the other critics is the presence of the scoring function, which assigns a cost term to the considered trajectories. One central aspect of this plugin is how the input parameters are handled. The *odom_callback* creates the first agent, while another function called *get_Odom* gets the current information about odometry and "freezes" it to be used in the *laser_callback*. This function is introduced to use the robot's current position to transform the points obtained by the **LiDaR** from the robot frame to the global frame, using a roto-translation defined by x and y coordinates plus the yaw. Another concept employed in the *laser_callback* function is the presence of a filter. Before transforming the points, a parameter *laser_filter* defines the number of points to be used as obstacles for the agents: the total amount of points given by the **LiDaR** is divided by said parameter, leading to a lower computational burden for the algorithm. Points are chosen by dividing the observed points into groups with as many elements as defined by the *laser_filter* parameter. In each group, the nearest point to the robot is chosen and assigned in the middle of its interval. The laser points, which are filtered and transformed, are then passed to a function that calls upon the first publisher. Another critical component of this algorithm is the *people_callback*, which transforms the people in the scene into agents coherent with their definition in the **SFM** library and adds the robot as the first agent. The *getAgents*() part of this code locks the agents' data for the entirety of the scoring process. This information gathering is then used in the final part to assign a cost to the chosen trajectories. Another filter, however, is applied before assigning a penalizing term: this filter is the agent filter. To lessen the amount of computation needing to be performed, the only agents considered in the scene are the ones that meet two separate requirements:

- They are within a certain distance from the robot;
- They are within a certain angle with respect to the direction the robot's **LiDaR** is pointing, allowing consideration of only the agents the robot is looking at directly, angle called **FOV**

The filter is represented in Fig. 3.1. This filter allows the robot to evaluate the presence of an agent only when it could significantly impact the trajectory scoring. It also avoids strange behavior when agents are nearby but far from the possible chosen trajectories. After the filtering process, the algorithm's final section is executed, which contains two for loops:

- an outer loop that iterates through all the different trajectories the controller selects.
- An inner loop that segments the i-th trajectory into several time steps defined by the controller.

The outer loop defines the robot agent's local goal as the trajectory's endpoint. It also creates an agent vector to be updated in the inner loop. The inner loop performs the following operations at every time step:

- computation of the forces defined in the SFM plugin;
- update of the agents' position and velocity as a result of said forces;

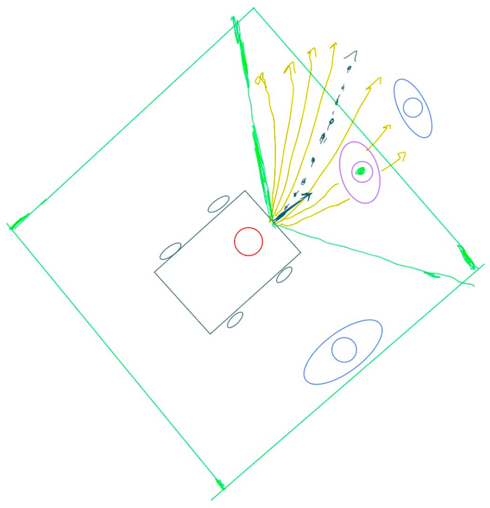


Figure 3.1: The agents in blue are not considered in the computation since they do not satisfy both requirements; the agent in purple is considered for the opposite reason.

- update of the robot's position and velocity, which are given by the trajectory;
- computation of the social work, which is provided by the sum of the norm of the social forces;

After the inner loop is completed, the social cost for the i-th trajectory is the sum of the social work at every time step. Once the outer loop is also completed, the vector of social costs is used to compute the actual cost used by the MPPI controller. Every cost is multiplied by a weight and divided by the number of time steps, allowing this to be considered along with all the other critics. The complete list of parameters which can be set include:

Parameter	**Type**	**Definition**
enabled	bool	whether the plugin is enabled or not
cost_power	int	power of the computed cost, one by default
social_weight	float	weight of the social critic
step_grouping	int	time steps to skip to reduce computational burden
field_of_view	float	angle used to reduce the number of agents considered
max_distance_robo_agent_x	float	cut-off distance for an agent to be considered on global x-axis

max_distance_robo_agent_y	float	cut-off distance for an agent to be considered on global y-axis
laser_filter	int	reduces the number of LiDAR points for computation
laser_distance_cut_off	float	distance inside which laser points are considered
global_frame	string	name of the global frame to be considered

Table 3.10: SocialCritic Configuration

3.3 Simulation

After completing the first part of the thesis, the next step was to experiment with the different variations of the algorithm. Before employing the algorithms on a real robot, simulations were used to observe the algorithms' behavior and tune their parameters accordingly. Different software was used to perform these simulations.

3.3.1 Gazebo

Gazebo was chosen as the simulator for the robot and the agents mainly because it was compatible with the other software employed in this thesis. Gazebo allows for the configuration of worlds relatively quickly and allows for the easy implementation of the robot. In Fig. 3.2, the Gazebo world used to tune the algorithms' parameters can be seen.

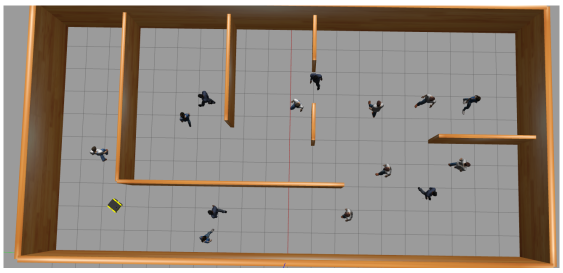


Figure 3.2: The Gazebo world used in simulation to test the algorithms.

3.3.2 RViz

Alongside Gazebo, RViz is a handy software for visualizing **ROS2** topics in a 3D environment.

RViz allows the visualization of the trajectory being computed by the robot along the map loaded by the specific algorithm being tested. It is also the tool used to set the robot's objective, making it a crucial part of the simulation.

In Fig. 3.3, there is the RViz panel associated with the Gazebo world.

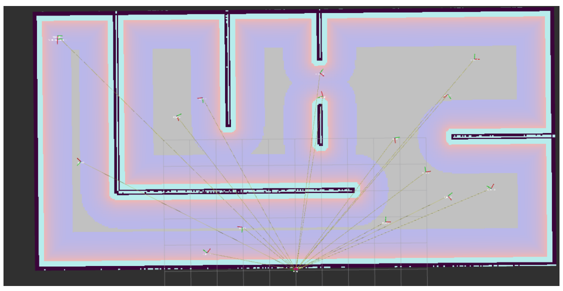


Figure 3.3: The RViz panel sets the goals and checks that the robot and agents are correctly recognized.

3.3.3 HunavSim

The HunavSim package, mentioned in Chapter 1, provides many tools that aided the simulation. The tools included are:

- *hunav_agent_manager*. This tool handles the number and the behavior of the agents using a set of behavior trees and a configuration file that can specify the behavior of each agent.
- *hunav_evaluator*. This tool evaluates the robot's performance based on metrics by gathering information on the /agents and/odom topics.
- *hunav_msgs*. This tool adds the "Agent" and "Agents" message types used by the evaluator. This package requires the *people_msgs* topic [27].
- *hunav_rviz_panel*. This tool implements an RViz panel, which allows the agents to easily be configured in the simulation by setting their goals, starting positions, and behaviors.

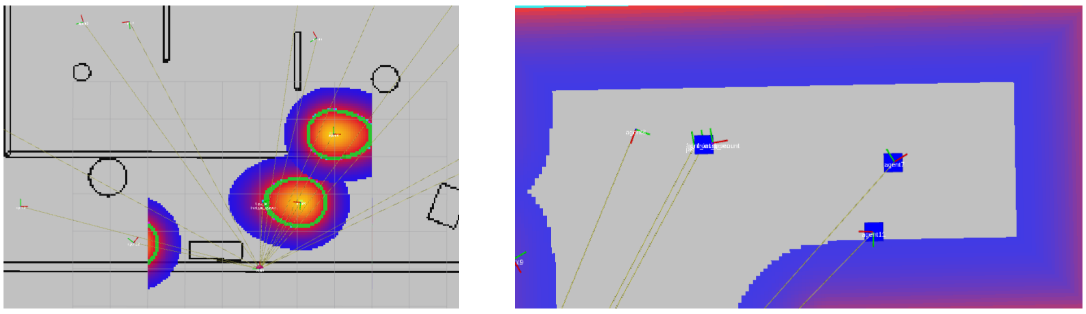
3.3.4 HuNav gazebo wrapper

The *hunav_gazebo_wrapper* is a **ROS2** wrapper that allows HuNavSim to be used inside Gazebo. This wrapper includes a set of different default scenarios, including other worlds and agent configurations. This tool allows agents to be spawned in the Gazebo world.

3.3.5 Observations made in the simulations

The six algorithms' variations were tested in the Gazebo world shown in Fig. 3.2 to tune the weights and test whether every part of the different algorithms works. In the simulated environment, which represents a series of connected rooms with eleven agents, the **DWA**-derived variations had more difficulty moving, especially in narrow corners and corridors near human agents. On the other hand, the **MPPI**-derived variations were responsive and managed to traverse the scenario in almost every test. The social costmap plugin worked as intended, adding an elliptical cost zone around the people, as shown in Fig. 3.4.

The **SFM** in the DWA algorithm worked as intended, but it was very conservative with the weights chosen, leading to navigation failures in tight spaces. The social critic of the **MPPI** algorithm was a noticeably smoother implementation of the **SFM**, leading to a more careful approach near agents while keeping the dexterity of the **MPPI**. To check whether the agents were correctly filtered, a publisher was added to the social critic to implement markers, which could be observed in RViz. The blue markers belong to the agents and the robot and can be seen in Fig. 3.4. After the simulation phase was completed, it was time to test the algorithms on the physical version of the Jackal.



(a) The cost zone attributed to agents by the social costmap plugin.

(b) Blue markers are the agent which are going to be considered by the critic.

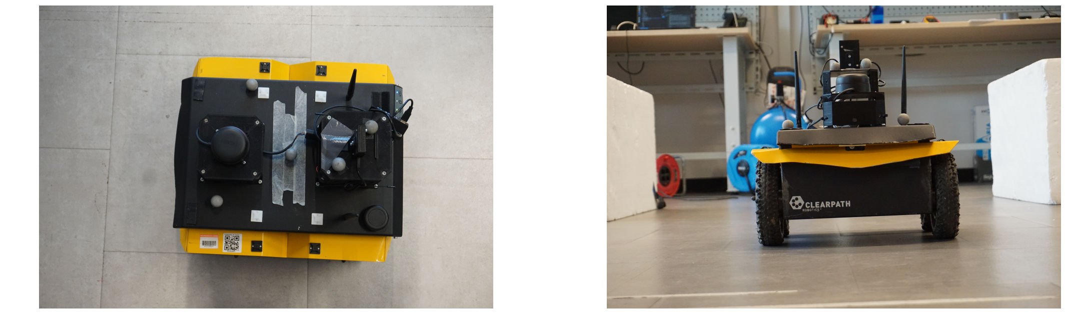
Figure 3.4: The agent markers and the social grid

3.4 Laboratory experiments

The laboratory experiments aimed to reproduce various scenarios where a robot could engage in a typical social navigation activity. The experiments were carried out in the PIC4SeR laboratory, and eight scenarios were tested. The first four were based on case studies, while the last four reproduced more complex scenarios. Each variation of the algorithms was tested five times, reaching 240 experiments.

3.4.1 Tools and hardware required

Information about the robot and the agents was necessary to make the algorithms work in a real-life test. As previously mentioned, the robot is the Jackal from ClearPath Robotics, which is relatively light-weight, weighing about seventeen kilograms. It is also a solid choice in testing because of its small size, good battery life, and the possibility to install and configure easily **ROS2**. The internal **CPU** of the Jackal was upgraded to an I7 10-th generation chip from Intel to handle some of the more complex variations of the algorithms. The Jackal was also equipped with a **LiDaR**, which allowed it to recognize the surrounding obstacles. The Jackal with the added **LiDaR** can be seen in Fig.3.5


(a) This view shows the spheres surrounded with reflective tape used by the Vicon cameras.

(b) The LiDaR can be seem form this view.

Figure 3.5: The Jackal seen from the front and seen from above

The information about the position of the agents and the odometry was captured using ten Vicon cameras placed around the laboratory's perimeter, which allowed for the tracking of different objects. The different objects were associated with the Vicon camera system using a unique disposition of spheres surrounded by reflective tape. In order to recognize the people as agents, each person wore a cardboard collar that featured many of the spheres mentioned above.

Both the cameras and the "agent identifier" can be seen in Fig. 3.6.

The complete setup allows one to track both the robot's and agents' positions. The information is then stored in two **ROS2** topics, which in this case are called "/vicon/odom" and "/vicon/people."

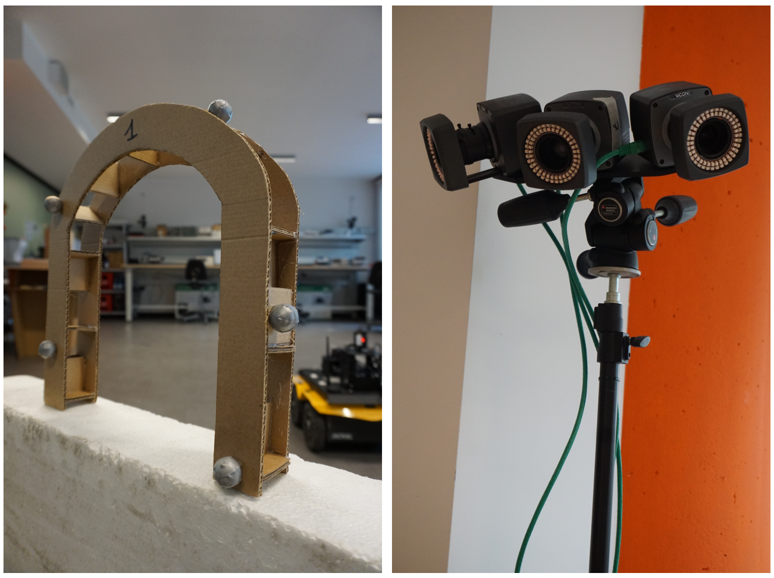

(a) Cardboard collar, with some spheres surrounded by reflective tape.
(b) Three Vicon cameras.

Figure 3.6: Three of the Vicon cameras and one of the agent collars.

3.4.2 Controller parameters

To keep the comparison between the controllers as fair as possible, the controllers had their maximum velocity set at 0.6 m/s. In contrast, the minimum velocity was set at 0.08 m/s. The minimum velocity was set at this value because even though the **MPPI**-derived algorithms can go in reverse, the **DWA**-derived algorithms can only go forward. The target frequency for all the controllers was set at 20 Hz. The social costmap plugin was kept the same throughout the two variations that included it, and its value can be seen in table 3.1. The **DWA**-derived algorithms had the following parameters:

DWA parameter	Value	Cost weight	DWA variations
max linear vel	0.6 [m/s]	social weight	0.0 / 0.0 / 2.0
min linear vel	0.08 [m/s]	costmap weight	2.0 / 2.0 /2.0
max angular vel	1.5 [rad/s]	velocity weight	0.8

Continued on next page

Table 3.11 – continued from previous page

DWA parameter	Value	Cost weight	DWA variations
waypoint tol	2.0 [m]	angle weight	0.6
sim time	2.5 [s]	distance weight	1.0

Table 3.11: DWA controller main parameters

The difference in the SFM variation is due to increased social weight. The **MPPI**-derived algorithms had the following parameters:

MPPI parameter	Value	Critic	MPPI variations
max linear vel	0.6 [m/s]	constraint critic	5.0
min linear vel	0.08 [m/s]	goal critic	15.0
max angular vel	1.5 [rad/s]	obstacles critic	0.45(normal)-20.0(critical)
time per step	0.05 [s]	path follow critic	8.0
number of time steps	60	path align critic	2.0
random trajectories	750	social critic	0.0/0.0/22.0

Table 3.12: MPPI controller main parameters

The difference between the standard variation and the SFM implementation is the enabling of the custom social critic. The social critic had the following parameters:

Social critic parameter	Value
step_grouping	12
field_of_view	90.0 [degrees]
max_distance_robo_agent_x	3.5 [m]
max_distance_robo_agent_y	3.5 [m]
laser_filter	6
laser_distance_cut_off	3.5 [m]
global_frame	"map"

Table 3.13: Social critic parameters

Having set both the hardware and software parts of the experiments, the experiments will now be presented.

3.4.3 Scenarios

The scenarios aim to present a wide array of situations in which the robot could be involved. The first scenario is the passing one, one of the standard interactions between the robot and the agent. The map presents a central corridor that the robot and one agent traverse in opposite directions. The map for the first scenario can be seen in Fig. 3.7 a. The second scenario is the overtaking case. The map is identical to the one presented in the first scenario, but the difference is that the robot and the agent traverse the corridor in the same direction. The goals of both the agent and the robot are different and can be seen in Fig. 3.7 b. In every map, the agents are colored in blue, the robot in red, the robot's objective in yellow, and the agents' objective in green.

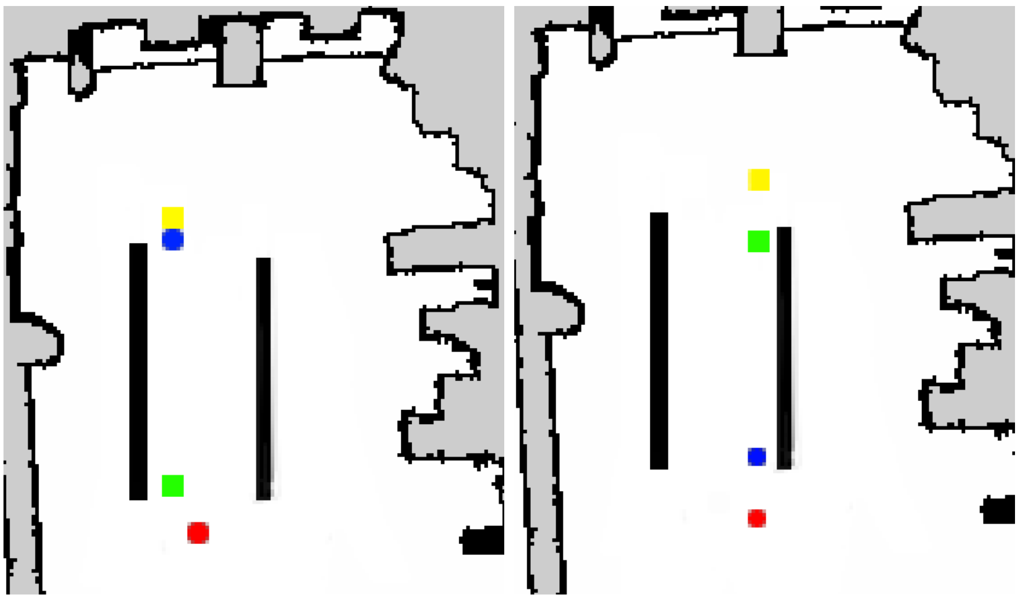

(a) Passing map **(b)** Overtaking map

Figure 3.7: Passing and overtaking maps

The third and fourth scenarios represent two variations of the crossing case. After traversing a small corridor, the robot has to deal with two agents crossing the room with a slight delay. The difference is that in the third scenario, the two agents cross in the same direction. In comparison, in the fourth scenario, the two agents cross in opposite directions, starting from opposite sides of the room. In both scenarios, the nearest agent to the robot starts crossing when it has traversed the corridor, while the second agent crosses when the first agent has traversed half its path. Both the maps can be seen in 3.8.

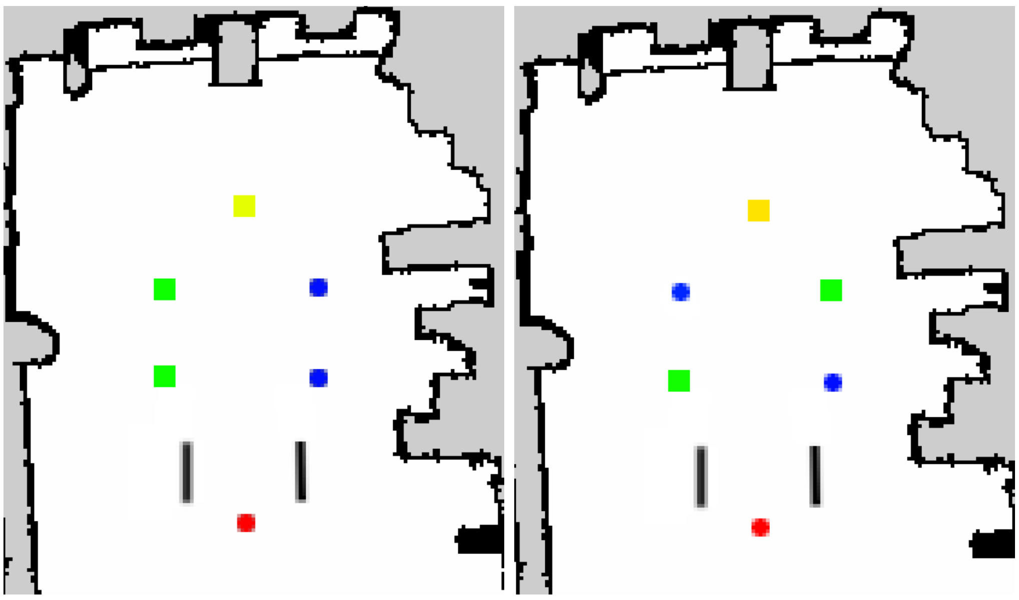
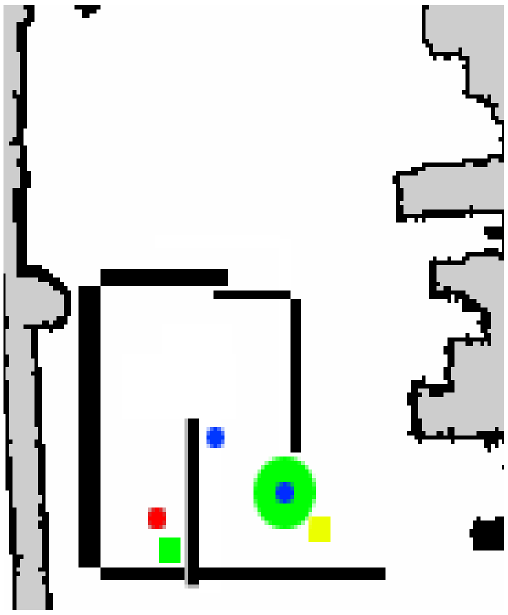

(a) Crossing1 map **(b)** Crossing2 map

Figure 3.8: The two crossing maps

The fifth scenario is the first non-standard case, which presents a more complex navigation path: the robot has to go into a hallway while a person is going around a corner, and the navigation goal is set near a person who is walking back and forth, making the experiment more unpredictable. The map for the fifth experiment is represented in Fig. 3.9 a, where the green area around the second agent indicates that it does not have a fixed objective. The sixth scenario is the first involving three agents. The first agent walks diagonally to test the robot's reaction; the second agent stands still near the middle of the room, and the third walks horizontally as soon as the robot approaches. The complete configuration is shown in Fig. 3.9 b.

The seventh scenario is an alteration of the crossing scenario, where two agents walk diagonally side-by-side while the other agent walks in the opposite diagonal direction. The map is shown in Fig. 3.10 a. The eighth scenario is different because the single agent present does not have a precise objective; it just aims to disturb the robot's trajectory, which has the same starting position and static obstacles as the seventh scenario. The map is shown in Fig. 3.10 b.


(a) Fifth scenario map

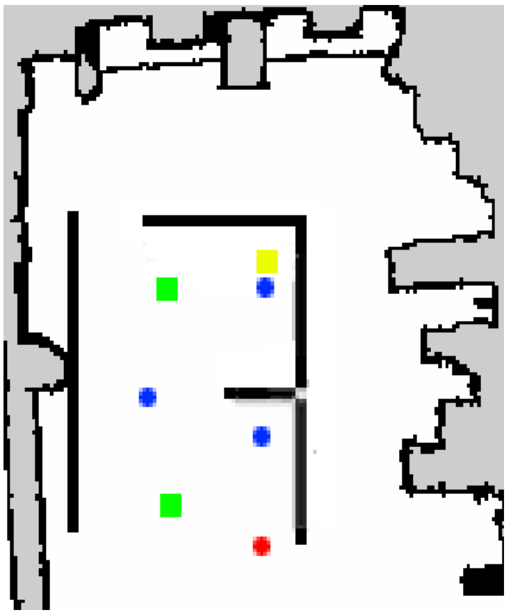
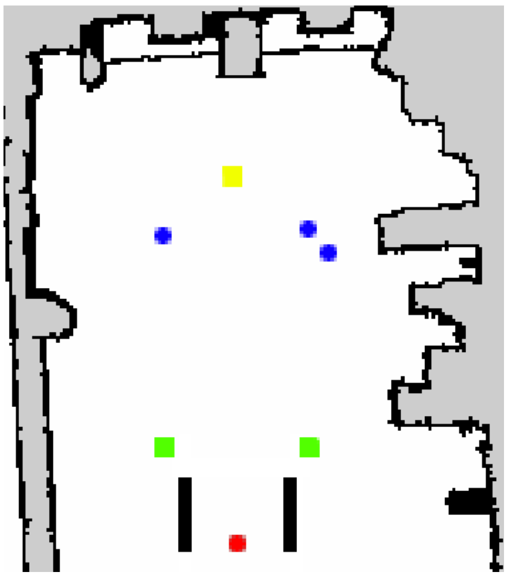

(b) Sixth scenario map

Figure 3.9: The fifth and sixth scenario maps


(a) Seventh scenario map

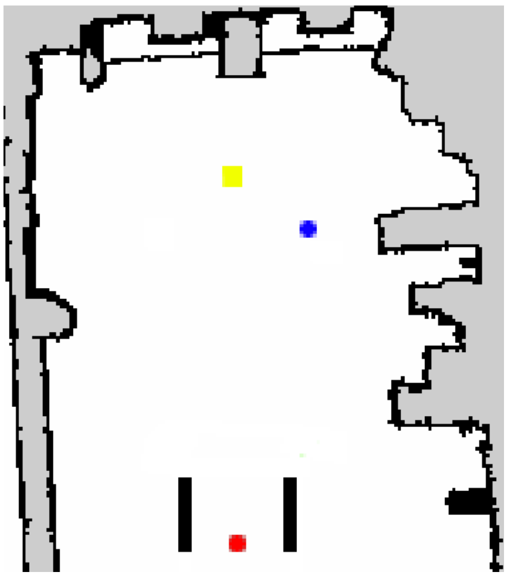

(b) Eighth scenario map

Figure 3.10: The seventh and eighth scenario maps

3.5 Test methodology

Having prepared all the scenarios and the necessary tools, it was now possible to conduct the tests. The tests were conducted between the second half of July and September. The only thing to note is that each person who participated in the experiment was asked to evaluate the robot's performance based on four parameters:

- Unobtrusiveness is the robot's ability to go unnoticed, which was expressed to the agent as the level of hindrance provided by the robot.
- Smoothness is the robot's ability to traverse smoothly in the environment, penalized by sudden stopping movements.
- Avoidance foresight is the robot's ability to react to environmental obstacles and change its trajectory.
- Friendliness, a parameter to express the overall friendliness of the robot, a slightly more subjective parameter and dependant on the other parameters.

It was possible to visualize the position of the agents and the robot in real-time using the Vicon software before and during the experiment to ensure correct tracking of the relevant data to be recorded. Fig. 3.11 shows an example from said software.

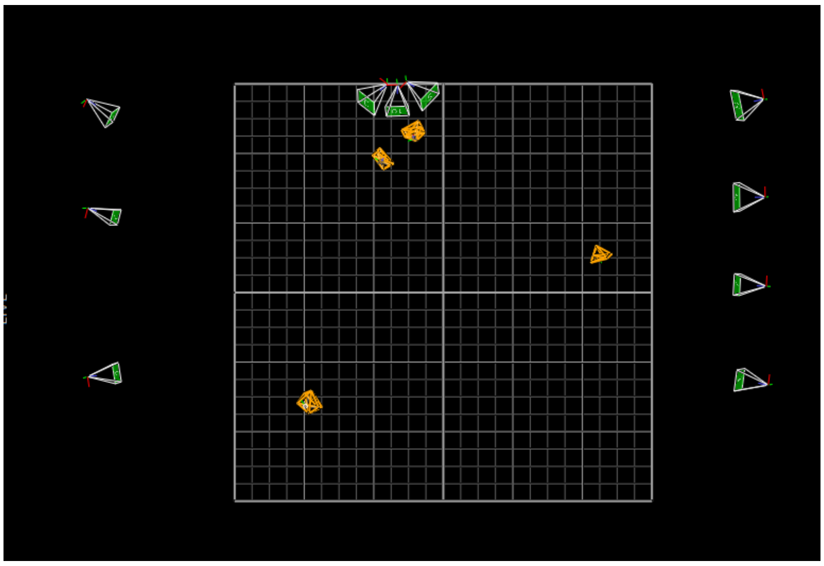


Figure 3.11: The robot (on the right side) and three agents (on the left side) as they appear on the Vicon software, represented by orange objects.

RViz could also verify that the map server loaded the correct map and that every frame was being tracked.

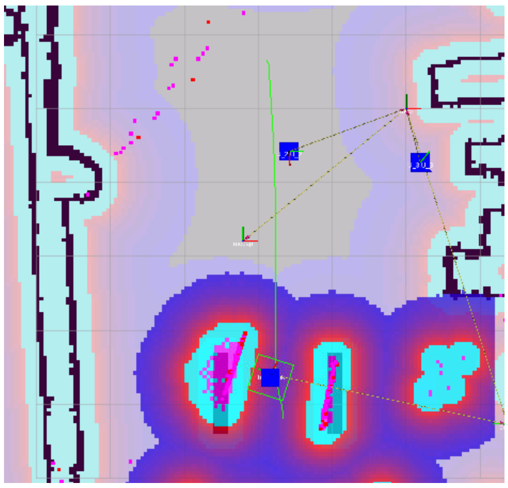
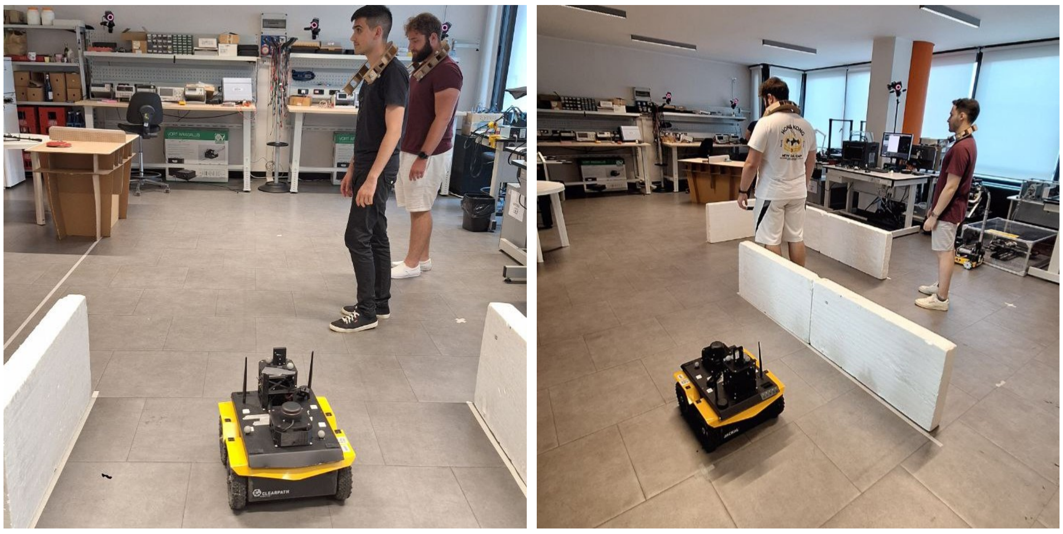


Figure 3.12: The RViz interface of the robot using MPPI with social critic

To ensure consistency, a simple Python script was written to start the recording of a **ROS2** bag and then send the **NAV2** objective to the robot after two seconds to start navigation. Moreover, the robot's objective was continuously published as a topic for evaluation. Fig. 3.13 shows two runs from the third and fifth scenarios.



(a) First crossing scenario run **(b)** First advanced scenario run

Figure 3.13: Two examples from the laboratory experiments

Chapter 4

Results

After completing the laboratory experiments, it was time to analyze the data acquired. The experiments were evaluated with the following criteria:

- A run was considered unsuccessful if the robot failed to reach the objective or collided with an agent. This influenced the robot's overall success rate.
- A run was considered for evaluation if it was successful or if it resulted in a collision. Runs that failed due to timeout often raised the standard deviations of the different metrics by a considerable amount, so they were not included in the computation of the average and the standard deviations since they had already reduced the overall success rate of the robot.

The same criteria were applied to both quantitative and qualitative metrics.

4.1 Quantitative metrics

The quantitative metrics considered are some of the ones present in the HuNavSim evaluator; they consist of the following:

- **Success rate** (SR) is the ratio between the successful experiments and the total number of experiments.
- **Social work** (SW) is obtained by normalizing the values of the social force between the robot and the different agents throughout the experiment.
- **Path length** (PL) reflects the total distance the robot travels to reach its objective.
- **Time to complete** (TTC) indicates the algorithm's time to finish the navigation.

- **Average minimum distance to agents** (AMD) shows the average of the robot's distance to its nearest agent throughout the experiment.
- **Social work per second** (SWsec) is the social work over the time to complete; it expresses the trade-off between completion velocity and social forces applied to the agents.
- **Space intrusion metrics** represent the percentage of time the robot spent in one of four spaces: intimate, personal, social, and public. These spaces are concentric circles drawn around the agents, with the closest being the intimate space and the furthest being the public space.

To represent the following metrics, both the mean values and the standard deviations have been computed. The scenarios have been numbered in the order they were presented, starting from passing(1) going all the way to the fourth advanced case(8).

	Algorithm	**SR** (%)	**SWsec**	**SW**	**PL**[m]	**TTC**[s]	**AMD**[m]
1	DWA	100.00	15.32	478.90	5.18	8.86	1.83
	DWA_costmap	100.00	11.13	374.53	5.36	12.71	1.93
	SFM_DWA	100.00	9.05	270.06	5.63	10.92	1.77
	MPPI	100.00	12.53	492.65	4.78	8.55	2.09
	MPPI_costmap	100.00	9.07	551.67	5.24	9.63	1.85
	SFM_MPPI	100.00	14.95	479.86	5.49	9.84	2.06
2	DWA	100.00	17.86	600.24	7.34	12.71	0.46
	DWA_costmap	100.00	10.01	529.25	6.19	11.50	0.34
	SFM_DWA	60.00	14.49	1164.44	8.62	49.59	0.70
	MPPI	60.00	15.47	1426.99	5.82	9.00	0.21
	MPPI_costmap	100.00	6.78	353.63	6.81	12.62	0.53
	SFM_MPPI	80.00	6.62	476.29	6.73	13.51	0.48
3	DWA	100.00	22.11	616.23	4.95	10.46	0.79
	DWA_costmap	100.00	24.40	645.97	4.87	10.15	0.72
	SFM_DWA	100.00	31.81	745.87	5.84	14.99	1.07
	MPPI	100.00	17.41	888.11	4.87	9.51	0.61
	MPPI_costmap	80.00	21.83	1132.82	5.63	13.43	0.81
	SFM_MPPI	100.00	17.64	542.42	5.38	10.92	0.92
4	DWA	80.00	17.86	459.14	5.05	11.33	0.87
	DWA_costmap	100.00	23.57	577.35	5.17	10.30	0.72
	SFM_DWA	80.00	33.06	1323.62	8.28	27.38	0.81
	MPPI	100.00	17.24	856.16	6.17	14.44	0.79
	MPPI_costmap	80.00	15.11	674.98	6.46	14.61	0.70
	SFM_MPPI	100.00	12.66	544.54	5.84	11.84	0.77

Table 4.1: Quantitative metrics for the first four scenarios, representing the mean values

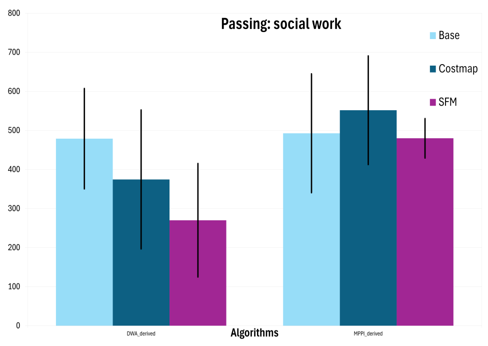
	Algorithm	**SR** (%)	**SWsec**	**SW**	**PL**[m]	**TTC**[s]	**AMD**[m]
5	DWA	100.00	146.31	6020.52	9.07	17.52	0.34
	DWA_costmap	100.00	154.74	6341.91	7.91	19.28	0.36
	SFM_DWA	20.00	-	-	-	-	-
	MPPI	60.00	210.57	9532.01	8.26	14.53	0.51
	MPPI_costmap	60.00	75.76	4593.62	8.92	18.38	0.45
	SFM_MPPI	100.00	118.42	4687.75	8.02	15.77	0.49
6	DWA	100.00	32.02	1549.96	7.41	27.62	0.22
	DWA_lcostmap	80.00	28.12	972.01	7.15	19.89	0.20
	SFM_DWA	100.00	37.76	1398.22	10.08	21.28	0.50
	MPPI	40.00	20.09	2144.23	7.61	16.66	0.32
	MPPI_costmap	80.00	12.09	862.94	12.57	31.96	0.30
	SFM_MPPI	80.00	16.61	826.22	8.16	21.82	0.31
7	DWA	100.00	46.87	1181.09	6.66	13.43	1.61
	DWA_costmap	100.00	44.50	1281.31	7.50	15.65	1.75
	SFM_DWA	100.00	39.23	1167.53	6.92	15.18	1.66
	MPPI	100.00	37.60	1504.81	6.46	15.65	1.15
	MPPI_costmap	40.00	49.94	2129.33	5.37	23.03	0.88
	SFM_MPPI	80.00	44.61	1429.80	6.69	15.55	1.32
8	DWA	100.00	40.46	1624.54	8.41	21.87	0.46
	DWA_costmap	100.00	50.95	1834.27	8.39	20.29	0.56
	SFM_DWA	80.00	25.52	1663.95	10.61	31.41	0.60
	MPPI	100.00	43.39	1819.88	10.24	18.31	1.15
	MPPI_costmap	100.00	45.68	1958.47	12.88	17.37	0.89
	SFM_MPPI	100.00	45.34	1893.09	8.71	18.06	0.76

Table 4.2: Quantitative metrics for the last four scenarios, representing the mean values.

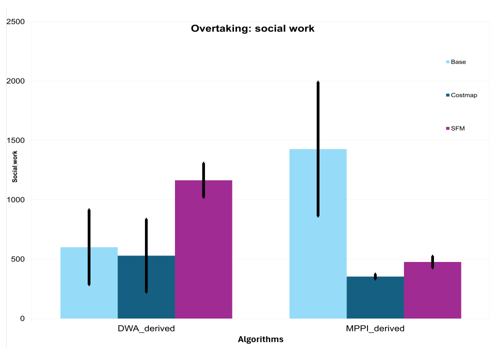
In table 4.1 and 4.2, just the mean values were shown, but it has to be noted that even among the successful experiments, some of them showed a different behavior compared to overall standard behavior, which can be attributed to:

- The starting position of the robot, which could have slight variations between one run and the next, even though it was kept fairly consistent
- The different agents across the experiments, since they have slightly different walking speeds and body types.
- In the case of the MPPI derived algorithms, the randomly generated trajectories could result in slightly different paths taken even in starting equal conditions.

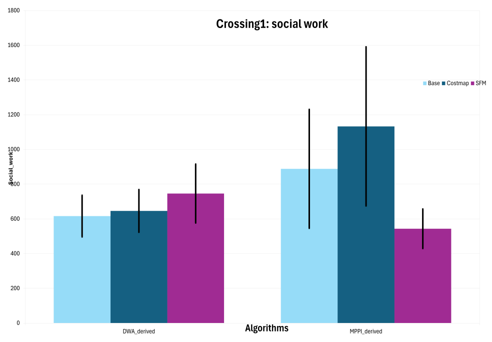
The social work of the different experiments is represented in Fig. 4.1 and 4.2.



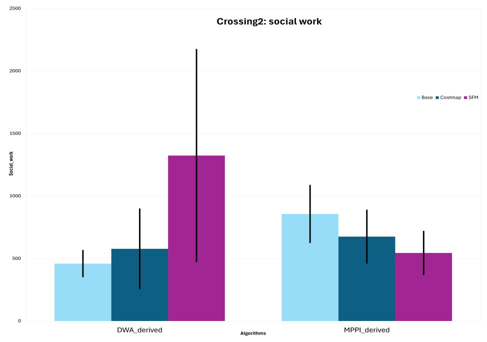
(a) passing social work



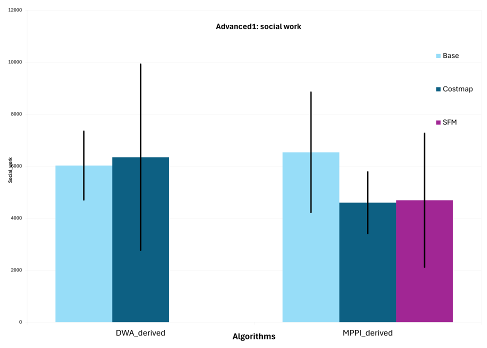
(b) overtaking social work



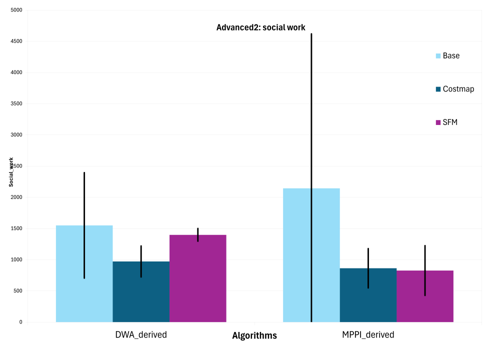
(c) crossing first social work



(d) crossing second social work

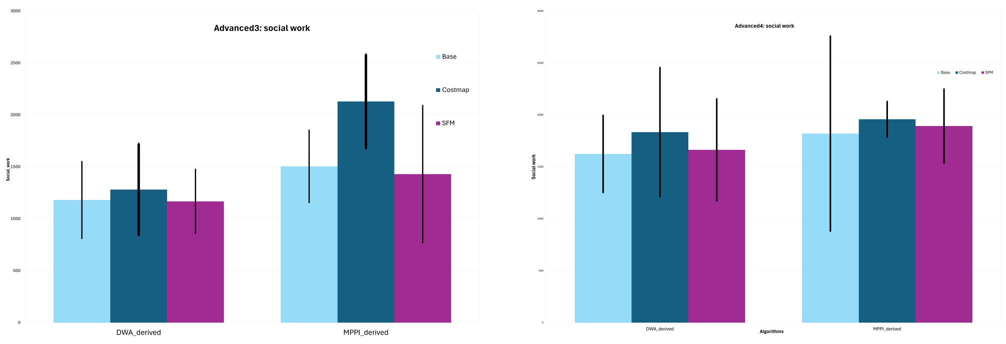


(e) advanced first social work



(f) advanced second social work

Figure 4.1: The average and standard deviation for the social work of the first six scenarios. On the x-axis, the different algorithms: DWA derived algorithms and MPPI derived algorithms. On the y-axis, the social work. The black lines represent the standard deviation. Light blue represents the base versions of the algorithms, dark blue represent the social costmap version, purple represents the SFM version.



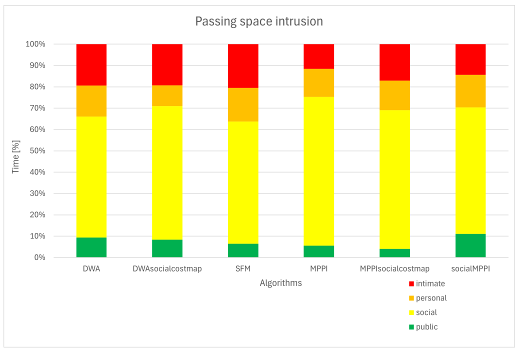
(a) advanced third social work **(b)** advanced fourth social work

Figure 4.2: The average and standard deviation for the social work of the last two scenarios. On the x-axis, the different algorithms: DWA derived algorithms and MPPI derived algorithms. On the y-axis, the social work. The black lines represent the standard deviation. Light blue represents the base versions of the algorithms, dark blue represent the social costmap version, purple represents the SFM version.

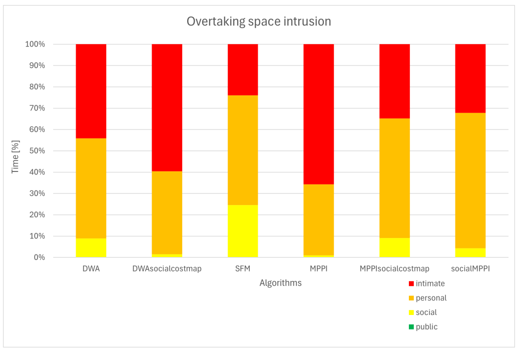
As can be seen, the standard deviation is high in certain situations. As an example, the **MPPI** base version in the sixth experiment, which is the (f) plot in Fig. 4.1, shows a high standard deviation given by the fact that one of the runs that was evaluated resulted in a collision, providing a way higher value of the social work compared to the standard runs and significantly increasing both the average and the standard deviation related to the social work.

The (e) plot in Fig. 4.1 also shows that the **SFM** variation of the **DWA** wasn't evaluated; this is because this is the only case where one of the algorithms failed to reach the goal in four different runs, making it impossible to calculate both the average and the standard deviations. There is also one run for the second crossing experiment of the **SFM** variation of the **DWA**, which was way smoother than the other runs, providing way better results and lowering the average value of the social work while raising the standard deviation. Apart from the occasional navigation failure, the **DWA**-derived algorithms show a more consistent behavior, with an overall lower standard deviation across the different experiments. As previously mentioned, the **MPPI**-derived algorithms show a higher standard deviation across the board due to their stochastic approach.

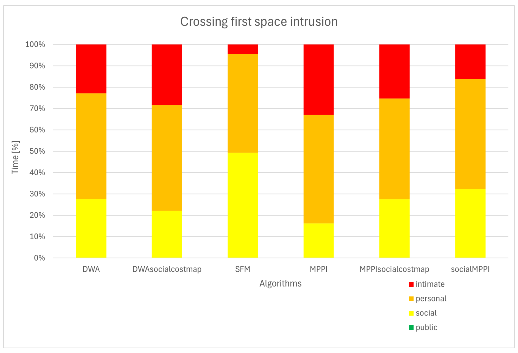
The intrusion metrics are going to be represented in Fig. 4.3 and 4.4.



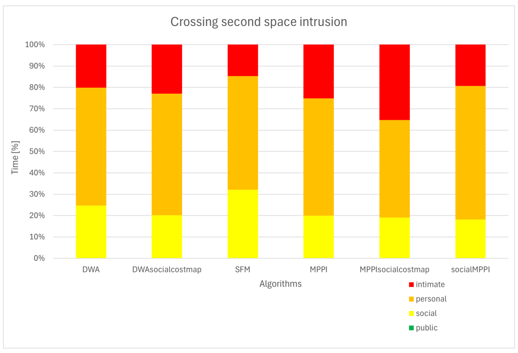
(a) passing intrusion



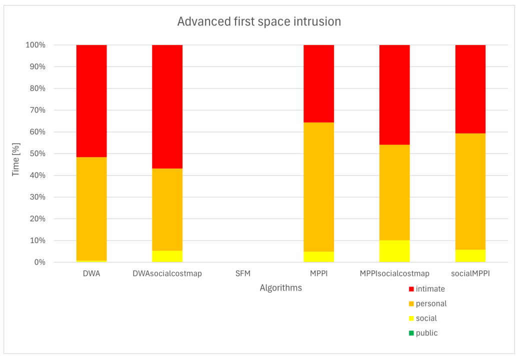
(b) overtaking intrusion



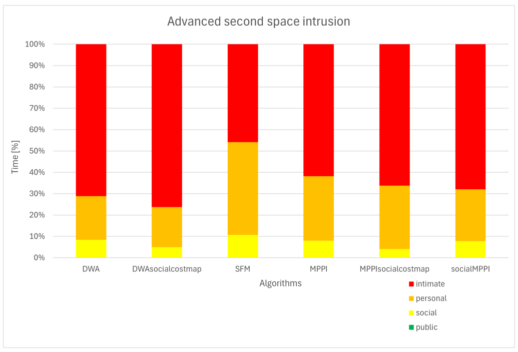
(c) crossing first intrusion



(d) crossing second intrusion

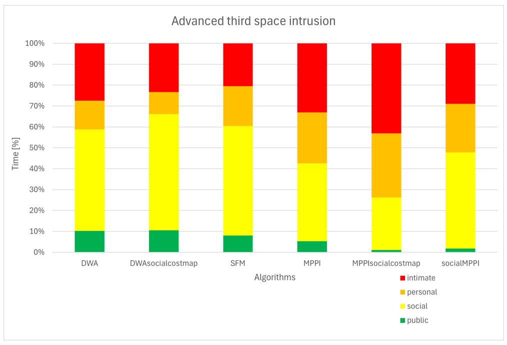


(e) advanced first intrusion

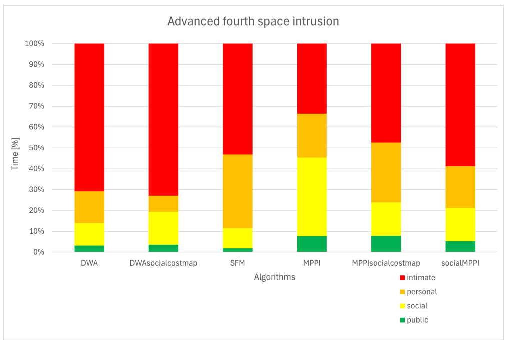


(f) advanced second intrusion

Figure 4.3: The average intrusion values for the first six scenarios.
On the x-axis, the different algorithms: DWA,DWA_costmap,SFM_DWA, MPPI,MPPI_costmap,SFM_MPPI. On the y-axis, the percentage values of total time. Green represents public space, yellow represents social space, orange represents personal space, red represents intimate space.



(a) advanced third intrusion



(b) advanced fourth intrusion

Figure 4.4: The average intrusion values for last two scenarios.
On the x-axis, the different algorithms: DWA,DWA_costmap,SFM_DWA, MPPI,MPPI_costmap,SFM_MPPI. On the y-axis, the percentage values of total time. Green represents public space, yellow represents social space, orange represents personal space, red represents intimate space.

From these metrics, the intimate space intrusion in the **SFM** variations has a lower value in most of the experiments compared to the other variations of the same algorithm. The **SFM** variation of the **DWA**, in particular, shows the lowest value for space intrusion in most of the experiments, demonstrating that it is the algorithm that consistently tries to keep the safest distance to the agents. The problem that can also be observed when analyzing the rest of these metrics is that this variation also has the highest time to complete in five out of the eight experiments. This algorithm suffers significantly in situations with a narrow space to traverse with an agent nearby, which makes the algorithm unable to move unless the agent moves. This behavior causes this algorithm to fail in 80% of the cases in the fifth experiment, where the agent is near the goal. The **SFM** variation of the **MPPI** shows, for most of the experiments, a lower social work compared to the other **MPPI** variations. It suffers mostly in scenarios where the other **MPPI** variations also have problems, in particular the sixth and the seventh scenario, where the success rate for the **MPPI** variations is much lower than the **DWA** variations. Overall, the success rate for the **SFM** variation of the **MPPI** is higher than the other **MPPI** controllers, showing that, with the others parameters being equal, the Social Force Model helps this type of algorithm. On the contrary, the Social Force Model helps the **DWA** algorithm only in the more straightforward passing scenario, but it makes navigation more difficult in more complicated scenarios.

To get an overall evaluation, some of the quantitative metrics (SW,SWsec,PL,TTC, and AMD) were given a score from zero to one to compare all the algorithms across different scenarios. This process was performed to decide the best-performing algorithm compared to the rest. In particular, the score was assigned following these two criteria:

$$\text{value}_{\text{norm}} = \begin{cases} \frac{\min(\text{values})}{\text{value}} & \text{for TTC}, \text{SW}_{\text{sec}}, \text{SW}, \text{PL} \\ \frac{\text{value}}{\max(\text{values})} & \text{for AMD} \end{cases}$$

given by the fact that TTC,SW,SWsec,PL should be as low as possible while AMD should be as high as possible. This normalization was done because different scenarios with different durations and agent numbers wouldn't be easy to compare otherwise; this method allows drawing an average between the quantitative metrics of all the algorithms across every scenario, leading to the result in the tab. 4.3.

Algorithms	**Success Rate**	**SWsec**	**SW**	**PL**	**TTC**	**AMD**
DWA	98%	0.60	0.82	0.92	0.84	0.68
DWA_costmap	98%	0.62	0.83	0.94	0.84	0.66
SFM_DWA	80%	0.60	0.65	0.80	0.55	0.91
MPPI	80%	0.68	0.59	0.92	0.94	0.72
MPPI_costmap	80%	0.87	0.79	0.81	0.74	0.71
SFM_MPPI	93%	0.80	0.88	0.89	0.85	0.76

Table 4.3: Normalized Quantitative Metrics

As it can be seen, the **SFM** variation of the **MPPI** shows good results across the board, being the best performer in SW. The **SFM** variation of the **DWA** shows the best score in AMD, but it suffers in TTC, also leading to an increase in SW, worsening its performance. The standard MPPI has the fastest average TTC but also the lowest performance in SW. The costmap variation of the DWA performs the best in PL and also has good SW performance. The costmap variations in both cases provide an uplift in performance in both SW and SWsec. After analyzing the quantitative metrics, the qualitative metrics are going to be presented.

4.2 Qualitative metrics

The qualitative metrics used for evaluation are mentioned in section 3.5, voted by the agents at the end of each run: unobtrusiveness(UNO), friendliness (FL), smoothness (SO), avoidance foresight (AF). The results are going to be presented in a table using the following criteria:

- The average values across the five runs will be presented, including results from failed runs, since agent feedback could still be obtained and be valid in a failed run.

- For experiments containing multiple agents, the average values between all the agents will be presented to give an overview of the global performance.
- For the eighth scenario, considering that the agent is actively trying to get in the robot's way, the unobtrusiveness parameter will not be considered.
- The average across all the experiments will also be considerd to evaluate the performance.

The results will be shown in table 4.4 and 4.5.

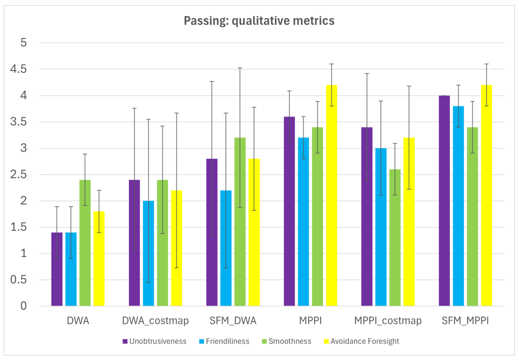
	Algorithms	**UNO**	**FL**	**SO**	**AF**
1	DWA	1.4	1.4	2.4	1.8
	DWA_costmap	2.4	2.0	2.4	2.2
	SFM_DWA	2.8	2.2	3.2	2.8
	MPPI	3.6	3.2	3.4	4.2
	MPPI_costmap	3.4	3.0	2.6	3.2
	SFM_MPPI	4.0	3.8	3.4	4.2
2	DWA	3.8	3.6	3.8	3.6
	DWA_costmap	3.4	3.4	3.4	3.2
	SFM_DWA	4.0	3.2	1.4	4.2
	MPPI	1.8	1.8	3.6	1.2
	MPPI_costmap	3.8	3.4	3.0	3.4
	SFM_MPPI	3.4	3.4	2.6	3.2
3	DWA	3.1	3.0	2.9	3.1
	DWA_costmap	3.4	2.9	2.5	3.3
	SFM_DWA	4.1	3.0	1.3	4.2
	MPPI	2.5	2.5	2.9	3.1
	MPPI_costmap	3.0	2.8	2.6	2.7
	SFM_MPPI	3.7	3.7	3.7	3.5
4	DWA	3.0	2.6	3.0	3.0
	DWA_costmap	3.1	2.8	2.8	3.0
	SFM_DWA	2.5	2.0	1.5	2.5
	MPPI	2.9	2.3	2.3	2.6
	MPPI_costmap	1.9	1.9	1.8	1.9
	SFM_MPPI	3.9	3.4	3.2	3.7

Table 4.4: Qualitative metrics for the first four scenarios

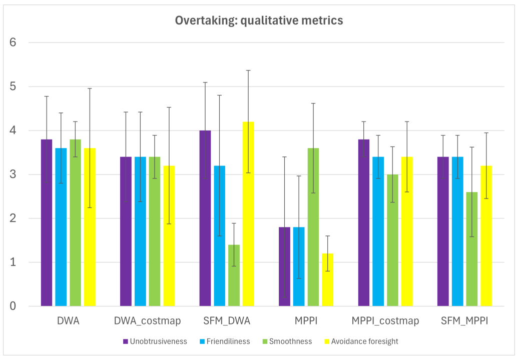
	Algorithms	**UNO**	**FL**	**SO**	**AF**
5	DWA	2.6	2.3	3.1	2.8
	DWA_costmap	3.2	2.0	2.4	3.4
	SFM_DWA	3.6	2.3	1.0	2.6
	MPPI	4.0	3.6	3.9	3.9
	MPPI_costmap	3.2	3.3	3.3	3.5
	SFM_MPPI	3.7	3.7	3.9	3.9
6	DWA	2.7	2.2	1.9	2.5
	DWA_costmap	2.5	2.1	2.3	2.3
	SFM_DWA	4.2	3.8	3.1	3.7
	MPPI	2.8	2.7	3.0	2.9
	MPPI_costmap	3.7	3.4	3.4	3.4
	SFM_MPPI	2.8	2.7	2.9	2.9
7	DWA	3.6	3.5	3.1	3.5
	DWA_costmap	3.8	3.7	2.7	3.3
	SFM_DWA	3.5	3.3	3.0	3.4
	MPPI	2.9	3.1	3.1	2.9
	MPPI_costmap	3.4	3.7	3.1	3.6
	SFM_MPPI	2.8	2.9	3.2	2.7
8	DWA	-	3.4	2.8	3.0
	DWA_costmap	-	2.8	2.8	2.8
	SFM_DWA	-	2.2	2.4	2.4
	MPPI	-	3.6	3.8	3.4
	MPPI_costmap	-	3.6	3.8	3.4
	SFM_MPPI	-	4.4	4.8	4.6
Average	DWA	2.89	2.75	2.88	2.91
	DWA_costmap	3.11	2.84	2.66	2.94
	SFM_DWA	3.53	2.75	2.11	3.23
	MPPI	2.93	2.85	3.25	3.03
	MPPI_costmap	3.20	3.16	2.95	3.14
	SFM_MPPI	3.47	3.53	3.41	3.61

Table 4.5: Qualitative metrics for the last four scenarios and the average across all the experiments

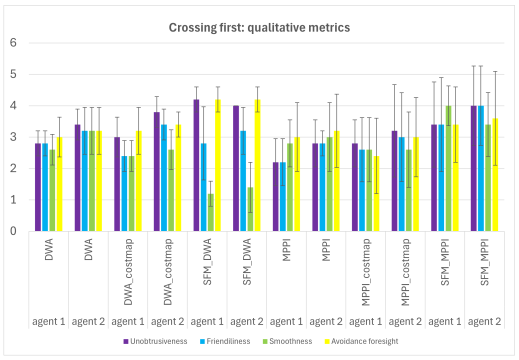
These average values show how the **SFM** variation of the **MPPI** performs the best in five out of the eight experiments, with the main exceptions being the sixth and the seventh scenarios where the **SR** for the **MPPI** algorithms was quite low. Also, in the overtaking scenario, this variation didn't score as highly, mainly due to some uncertainties shown when trying to surpass the agent, which resulted in an oscillating behavior, penalizing mainly its smoothness in the agent's evaluations. Fig. 4.5 and 4.6 will represent the distinct values for each agent and the standard deviations.



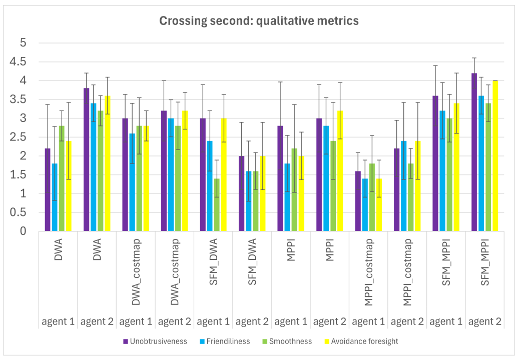
(a) passing qualitative parameters



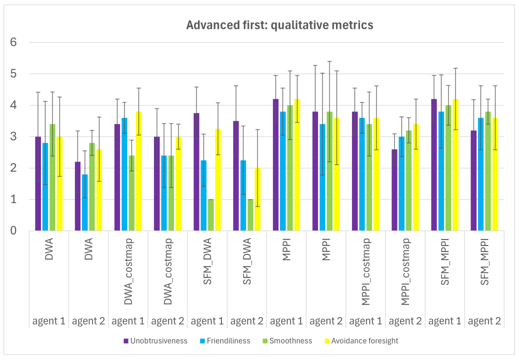
(b) overtaking qualitative parameters



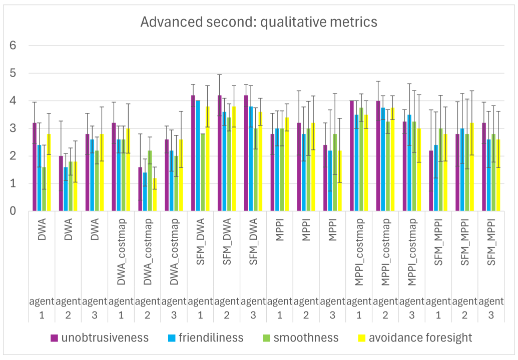
(c) crossing first qualitative parameters



(d) crossing second qualitative parameters

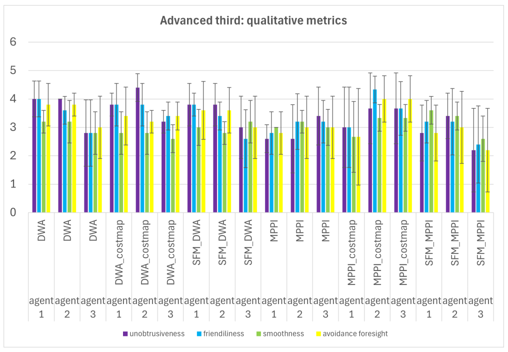


(e) advanced first qualitative parameters

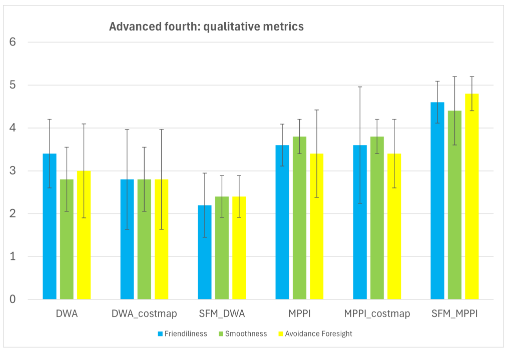


(f) advanced second qualitative parameters

Figure 4.5: The average qualitative metrics for the first six scenarios.
On the x-axis, the different algorithms: DWA,DWA_costmap,SFM_DWA, MPPI,MPPI_costmap,SFM_MPPI, showing the values for each agent if there are multiple. On the y-axis, numerical values from one to five. The black lines represent the standard deviation, purple is for **UNO**, light blue is for **FL**, green is for **SO**, yellow is for **AF**.



(a) advanced third qualitative parameters



(b) advanced fourth qualitative parameters

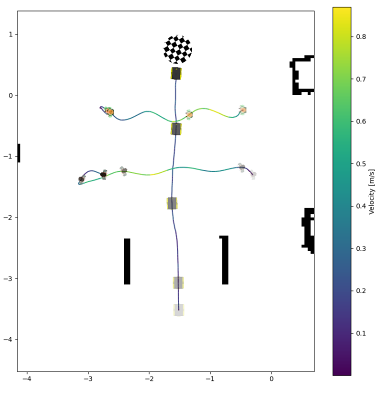
Figure 4.6: The average qualitative metrics for the first six scenarios.
On the x-axis, the different algorithms: DWA,DWA_costmap,SFM_DWA, MPPI,MPPI_costmap,SFM_MPPI, showing the values for each agent if there are multiple. On the y-axis, numerical values from one to five. The black lines represent the standard deviation, purple is for **UNO**, light blue is for **FL**, green is for **SO**, yellow is for **AF**.

The singular agent behavior reflects the algorithms' overall performance. The **SFM** variation of the **DWA** has, in most experiments, the worst smoothness score, mainly due to its uncertain behavior near the participants. At the same time, its avoidance foresight and unobtrusiveness were usually deemed better. The **MPPI** algorithms have a typically higher value for their smoothness compared to the **DWA** controllers. The costmap variations for both algorithms perform slightly better, but the difference is not as noticeable as the inclusion of the **SFM**. Overall, the **MPPI** algorithms are the ones that perform better considering the qualitative metrics. Considering the average values across all experiments, the algorithm with the best overall UNO is the **SFM** variation of the **DWA**, while the **SFM** variation of the **MPPI** performs the best in all the other metrics.

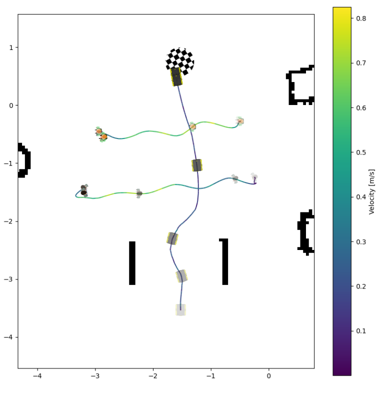
4.3 Quantitative and qualitative data comparison

By confronting both the qualitative and the quantitative metrics, some inferences can be made:

- In five out of the eight experiments, the unobtrusiveness value is higher for the algorithm which had the higher overall **AMD**, proving that the feedback of the agents and the results obtained from the data are coherent in this case
- The smoothness behavior does not seem to be directly correlated to any of the metrics shown since the trajectory taken by the robot seems to be much more important than the total length of its path or the time taken. Figures 4.7 show the differences between a trajectory considered the smoothest and the least smooth.
- Friendliness is a more difficult value to attribute to a specific quantitative metric. However, in half of the experiments, the higher friendliness value corresponds to the algorithm with the lower or second lowest social work per second. Considering the average parameters, the two algoritms with the highest FL are the ones with the best SW and SWsec
- Avoidance foresight is scored the best when the **AMD** is the highest in half the scenarios. However, once again, this value depends more on the trajectory the agent sees rather than on a specific quantitative metric.



(a) The trajectory of the **SFM** variation of the **DWA**.



(b) The trajectory of the **SFM** variation of the **MPPI**.

Figure 4.7: Two trajectories for the first crossing experiment

Chapter 5

Conclusions

The analysis conducted on the algorithms provided a considerable amount of information on their navigational capabilities, by testing them in complex scenarios. The result is a benchmark of their performances that considers both quantitative and qualitative metrics, providing a complete overview of an algorithm's capabilities and also considering the agent's opinion, that is of paramount importance in social navigation and allows distinguishing which is the best algorithm in cases when the quantitative metrics do not provide a clear-cut answer. The implementation of the **SFM** works well with the **MPPI** algorithm, but it often caused problems in the **DWA** variation, especially in some of the more complex scenarios. Future tests could try to implement a **FOV** to filter the agents and make fewer trajectories unfeasible, making this implementation more nimble. Furthermore, the paper by Martini et al. [4] proposes a method to dynamically change the weights during navigation, which could help overcome some of the most critical cases. The **SFM** variation of the **MPPI** could be improved by implementing a scalable radius for the agent detected by the robot. This adjustable radius would be based on the distance to the robot to provide further security to an agent the closer it gets to the robot. This change could help in computing more trajectories near the agent with a higher cost, making it more socially capable. In future applications, these algorithms could be tested on a robot equipped with a camera to recognize the agents without relying on the VICON camera system. The merits of testing a robot in a real-life setting must be considered, and procedures that include quantitative and qualitative metrics should become the standard for future evaluation of different algorithms.

Bibliography

[1] Phani Teja Singamaneni, Pilar Bachiller-Burgos, Luis J. Manso, Anaís Garrell, Alberto Sanfeliu, Anne Spalanzani, and Rachid Alami. «A survey on socially aware robot navigation: Taxonomy and future challenges». In: *The International Journal of Robotics Research* (Feb. 2024). ISSN: 1741-3176. DOI: 10.1177/02783649241230562. URL: http://dx.doi.org/10.1177/02783649241230562 (cit. on p. 2).

[2] Phani Teja S. and Rachid Alami. «HATEB-2: Reactive Planning and Decision making in Human-Robot Co-navigation». In: *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2020, pp. 179–186. DOI: 10.1109/RO-MAN47096.2020.9223463 (cit. on p. 2).

[3] Shyam Sundar Kannan, Ahreum Lee, and Byung-Cheol Min. «External Human-Machine Interface on Delivery Robots: Expression of Navigation Intent of the Robot». In: *CoRR* abs/2108.03045 (2021). arXiv: 2108.03045. URL: https://arxiv.org/abs/2108.03045 (cit. on pp. 2, 3).

[4] Mauro Martini, Noé Pérez-Higueras, Andrea Ostuni, Marcello Chiaberge, Fernando Caballero, and Luis Merino. *Adaptive Social Force Window Planner with Reinforcement Learning*. 2024. arXiv: 2404.13678 [cs.RO]. URL: https://arxiv.org/abs/2404.13678 (cit. on pp. 5, 63).

[5] Angel Romero, Yunlong Song, and Davide Scaramuzza. *Actor-Critic Model Predictive Control*. 2024. arXiv: 2306.09852 [cs.RO]. URL: https://arxiv.org/abs/2306.09852 (cit. on p. 6).

[6] Adarsh Jagan Sathyamoorthy, Utsav Patel, Tianrui Guan, and Dinesh Manocha. *Frozone: Freezing-Free, Pedestrian-Friendly Navigation in Human Crowds*. 2020. arXiv: 2003.05395 [cs.RO]. URL: https://arxiv.org/abs/2003.05395 (cit. on p. 6).

[7] Yuxiang Gao and Chien-Ming Huang. «Evaluation of Socially-Aware Robot Navigation». In: *Frontiers in Robotics and AI* 8 (2022), pp. 1–21. DOI: 10.3389/frobt.2021.721317. URL: https://www.frontiersin.org/articles/10.3389/frobt.2021.721317/full (cit. on p. 7).

[8] Masahiro Shiomi, Francesco Zanlungo, Kotaro Hayashi, and Takayuki Kanda. «Towards a Socially Acceptable Collision Avoidance for a Mobile Robot Navigating Among Pedestrians Using a Pedestrian Model». In: *International Journal of Social Robotics* 6 (2014), pp. 443–455. URL: `https://api.semanticscholar.org/CorpusID:45161498` (cit. on p. 7).

[9] Noé Pérez-Higueras, Roberto Otero, Fernando Caballero, and Luis Merino. «HuNavSim: A ROS 2 Human Navigation Simulator for Benchmarking Human-Aware Robot Navigation». In: *IEEE Robotics and Automation Letters* (Sept. 2023). Preprint Version, pp. 1–8. URL: `https://github.com/robotics-upo/hunav_sim` (cit. on pp. 7, 9).

[10] Christoforos Mavrogiannis, Alena M. Hutchinson, John Macdonald, Patrícia Alves-Oliveira, and Ross A. Knepper. «Effects of distinct robot navigation strategies on human behavior in a crowded environment». In: *Proceedings of the 14th ACM/IEEE International Conference on Human-Robot Interaction*. HRI '19. Daegu, Republic of Korea: IEEE Press, 2020, pp. 421–430. ISBN: 9781538685556 (cit. on p. 8).

[11] Elena Pacchierotti, Henrik I. Christensen, and Patric Jensfelt. «Embodied Social Interaction for Service Robots in Hallway Environments». In: *International Symposium on Field and Service Robotics*. 2005. URL: `https://api.semanticscholar.org/CorpusID:6993014` (cit. on p. 9).

[12] 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 (2016), pp. 1289–1307. URL: `https://api.semanticscholar.org/CorpusID:14275694` (cit. on p. 9).

[13] Nathan Tsoi et al. «SEAN 2.0: Formalizing and Generating Social Situations for Robot Navigation». English (US). In: *IEEE Robotics and Automation Letters* 7.4 (Oct. 2022). Publisher Copyright: © 2022 IEEE., pp. 11047–11054. ISSN: 2377-3766. DOI: `10.1109/LRA.2022.3196783` (cit. on p. 9).

[14] D. Fox, W. Burgard, and S. Thrun. «The dynamic window approach to collision avoidance». In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33. DOI: `10.1109/100.580977` (cit. on p. 10).

[15] Marija Seder and Ivan Petrovic. «Dynamic window based approach to mobile robot motion control in the presence of moving obstacles». In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 1986–1991. DOI: `10.1109/ROBOT.2007.363613` (cit. on p. 10).

[16] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. «Aggressive driving with model predictive path integral control». In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440. DOI: `10.1109/ICRA.2016.7487277` (cit. on p. 12).

[17] Evangelos A Theodorou and Emanuel Todorov. «Relative entropy and free energy dualities: Connections to path integral and kl control». In: *2012 ieee 51st ieee conference on decision and control (cdc)*. IEEE. 2012, pp. 1466–1473 (cit. on pp. 13, 14).

[18] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. *Model Predictive Path Integral Control using Covariance Variable Importance Sampling*. 2015. arXiv: `1509.01149 [cs.SY]`. URL: `https://arxiv.org/abs/1509.01149` (cit. on p. 18).

[19] Dirk Helbing and Péter Molnár. «Social force model for pedestrian dynamics». In: *Phys. Rev. E* 51 (5 May 1995), pp. 4282–4286. DOI: `10.1103/PhysRevE.51.4282`. URL: `https://link.aps.org/doi/10.1103/PhysRevE.51.4282` (cit. on p. 21).

[20] Rainer Feistel. «Weidlich, W., und G. Haag: Concepts and Models of a Quantitative Sociology. The Dynamics of Interacting Populations. Springer Series in Synergetics Vol. 14, Berlin, Heidelberg, New York: Springer-Verlag 1983.» In: *Zeitschrift für Physikalische Chemie* (Sept. 1983) (cit. on p. 21).

[21] Kurt Lewin. *Field Theory in Social Science: Selected Theoretical Papers*. Ed. by D. Cartwright. New York: Harper & Row, 1951 (cit. on p. 21).

[22] Albert E. Scheflen and Norman Ashcraft. «Human territories : how we behave in space-time». In: 1976. URL: `https://api.semanticscholar.org/CorpusID:154079003` (cit. on p. 23).

[23] Steve Macenski, Francisco Martin, Ruffin White, and Jonatan Gines Clavero. «The Marathon 2: A Navigation System». In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2020. DOI: `10.1109/iros45743.2020.9341207`. URL: `http://dx.doi.org/10.1109/IROS45743.2020.9341207` (cit. on pp. 25, 26).

[24] Noé Perez-Higueras. *Implementation of the social costmap*. URL: `https://github.com/robotics-upo/nav2_social_costmap_plugin` (cit. on p. 28).

[25] S. Macenski, T. Moore, DV Lu, A. Merzlyakov, and M. Ferguson. «From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2». In: *Robotics and Autonomous Systems* (2023). URL: `https://arxiv.org/pdf/2307.15236` (cit. on p. 30).

[26] Noé Perez-Higueras. *Implementation of the DWA algorithm with SFM.* URL: `https://github.com/robotics-upo/social_force_window_planner` (cit. on p. 30).

[27] David Lu. *Implementation of the people as a* ***ROS2*** *message.* URL: `https://github.com/wg-perception/people/tree/ros2` (cit. on p. 40).