POLITECNICO DI TORINO

Department of Electronics and Telecommunications
Master Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

Path Planners Benchmark and Design of a Comfort-Oriented Critic Algorithm

Supervisor:
Prof. Marcello Chiaberge
Eng. Andrea Eirale
Eng. Andrea Ostuni
Eng. Mauro Martini
Eng. Stefano Trepella

Candidate:
David Luigi Perotti

Anno Accademico 2025/2026

Abstract

In recent decades, demographic trends have highlighted a progressive shift towards an ageing society, driven by increased life expectancy. As a consequence, the number of people with reduced mobility is steadily increasing, leading to a growing demand for assistive mobile mobility technologies such as electric powered wheelchairs (EPWs). Autonomous navigation has been widely studied for mobile robots and EPWs, with safety traditionally considered the primary objective and the passenger comfort treated as a secondary aspect typically limited to vibration analysis or mechanical design considerations. However, abrupt acceleration, excessive over-acceleration and unnatural motion patterns significantly affect user comfort and trust in assistive mobile robots. The main objective of this thesis is to evaluate the impact of different global planners and local planners combination on navigation performance and passenger comfort with the ROS2 Navigation Stack (Nav2). The global planners used are the *SmacPlannerHybrid*, *SmacPlannerLattice* and *NavFn* and the local planners are *MPPI controller* and *DWB controller* and in addition to the MPPI controller, a comfort-oriented critic is designed and implemented to improve the comfort during navigation. The proposed approach is evaluated in realistic simulated environments built in Gazebo, inspired by the layout of an airport terminal and enriched with both static and dynamic obstacles. Six different scenarios are tested for each of the nine possible controller-planner combinations and each run is simulated fifteen times, to guarantee data convergence, resulting in a total of 810 simulation runs. The results provide a comparative analysis of navigation strategies in terms of both classical performance metrics and comfort-oriented indicators, highlighting how comfort-aware control can improve the experience of passengers. This work contributes to the design of safer and more comfortable autonomous EPWs and provides insights for future human-centered navigation systems.

Contents

Acronyms **11**

Introduction **13**

1 Fundamentals **15**
1.1 Autonomous Navigation 15
1.1.1 Mobile robot localization 15
1.1.2 Mapping and the SLAM problem 16
1.1.3 Path Planning 16
1.1.3.1 Dijkstra's algorithm 17
1.1.3.2 A* algorithm 18
1.1.3.3 Comparison between Dijkstra and A* 19
1.1.4 Control 19
1.1.4.1 Dynamic Window Approach 20
1.1.4.2 MPC 21
1.2 Social Robot Navigation 23
1.3 The Robot Operating Systems 28
1.3.1 Introduction 28
1.3.2 Nodes and Topics 28
1.3.3 Messages, Services and Actions 30
1.4 The Navigation2 Framework 31
1.4.1 Introduction 31
1.4.2 Global Planners 33
1.4.2.1 Smac Planners 33
1.4.2.2 NavFn planner 38
1.5 Controllers 42
1.5.1 MPPI 42
1.5.1.1 Stochastic trajectory optimization 42
1.5.1.2 State independent control matrix 46
1.5.1.3 Numerical approximation 47
1.5.1.4 Model predictive control algorithm 48
1.5.1.5 Comparison Between Classical MPC and MPPI 50
1.5.1.6 Suitability of MPPI for Autonomous Navigation 50
1.5.2 DWB 51
1.5.2.1 Difference between DWA and DWB 51

2 Comfort-oriented critic and Benchmark **52**
2.1 Critic implementation 52
2.2 Benchmark creation 54

2.2.1 Gazebo environment 54
2.2.2 HunavSim 55
2.2.3 Gazebo test 56
2.2.4 Social evaluation graphs 56
2.2.5 Scenarios 57
2.2.6 Metrics evaluated 58

3 Results of simulations **61**
3.1 Parameters tuning 61
3.2 Metrics evaluation 62
3.2.1 Straight navigation in narrow corridor with crossing agents 62
3.2.2 Straight navigation in narrow corridor with passing agents 70
3.2.3 Curve trajectory in narrow corridor 78
3.2.4 Curve trajectory with crossing agents 85
3.2.5 Straight navigation with crossing and passing agents . . . 92
3.2.6 Curve in narrow corridor with crossing agents 100
3.2.7 Composite Score Evaluation 107
3.2.7.1 Average Comfort Score 108
3.2.7.2 Average Safety Score 108
3.2.7.3 Average Performance Score 109
3.2.8 Temporal metrics evaluation (MPPI+NavFn) 110

4 Conclusion **114**

Bibliography **116**

List of Figures

1 [2]Global life expectancy at birth by gender from 1950 to 2023, with projections until 2100 13
2 [1] Population pyramids of the EU in 2009 and 2024(% of the total population) 13

1.1 [6] Basic structure of MPC 22
1.2 [9] Comfort model in an indoor corridor environment (normalized). 24
1.3 [10] Wheelchair turning right in a corridor corner. 25
1.4 [11] Definition of invasion ratio into the passenger's personal space. 26
1.5 [11] Overview of the three-layer path planning. 27
1.6 [13]ROS2 topic communication pattern 29
1.7 ROS2 service communication pattern 30
1.8 [13]ROS2 action comunication pattern 30
1.9 **doi:10.1126/scirobotics.abm6074**ROS2 node interfaces 31
1.10 [14]Overview of Navigation2 design 32

1.11 [14]Graphical comparison of search algorithms. Left: A*. Center: D* and Theta*. Right: Hybrid A* 35

2.1 Gazebo world used in the simulation for the benchmark 55
2.2 [25] HunavSim architecture 56
2.3 Agents configuration in the scenario east-west 57
2.4 Agents configuration in the scenario south-north 57

3.1 Time to reach goal metric for scenario 1 62
3.2 Path length metric for scenario 1 63
3.3 Maximum linear acceleration metric for scenario 1 63
3.4 Count of linear acceleration threshold exceedances for scenario 1 64
3.5 Maximum linear over-acceleration metric for scenario 1 64
3.6 Count of linear over-acceleration threshold exceedances for scenario 1 65
3.7 Maximum angular acceleration metric for scenario 1 65
3.8 Count of angular acceleration threshold exceedances for scenario 1 66
3.9 Maximum angular over-acceleration metric for scenario 1 66
3.10 Count of angular over-acceleration threshold exceedances for scenario 1 67
3.11 Minimum distance to people for scenario 1 67
3.12 Minimum distance to obstacles for scenario 1 67
3.13 Time to reach goal metric for scenario 2 71
3.14 Path length metric for scenario 2 71
3.15 Maximum linear acceleration metric for scenario 2 71
3.16 Count of linear acceleration threshold exceedances for scenario 2 72
3.17 Maximum linear over-acceleration metric for scenario 2 72
3.18 Count of linear over-acceleration threshold exceedances for scenario 2 73
3.19 Maximum angular acceleration metric for scenario 2 73
3.20 Count of angular acceleration threshold exceedances for scenario 2 74
3.21 Maximum angular over-acceleration metric for scenario 2 74
3.22 Count of angular over-acceleration threshold exceedances for scenario 2 74
3.23 Minimum distance to people for scenario 2 75
3.24 Minimum distance to obstacles for scenario 2 75
3.25 Time to reach goal metric for scenario 3 78
3.26 Path length metric for scenario 3 79
3.27 Maximum linear acceleration metric for scenario 3 79
3.28 Count of linear acceleration threshold exceedances for scenario 3 79
3.29 Maximum linear over-acceleration metric for scenario 3 80
3.30 Count of linear over-acceleration threshold exceedances for scenario 3 80
3.31 Maximum angular acceleration metric for scenario 3 81
3.32 Count of angular acceleration threshold exceedances for scenario 3 81
3.33 Maximum angular over-acceleration metric for scenario 3 81
3.34 Count of angular over-acceleration threshold exceedances for scenario 3 82
3.35 Minimum distance to people for scenario 3 82
3.36 Minimum distance to obstacles for scenario 3 82

3.37 Time to reach goal metric for scenario 4 86
3.38 Path length metric for scenario 4 86
3.39 Maximum linear acceleration metric for scenario 4 86
3.40 Count of linear acceleration threshold exceedances for scenario 4 87
3.41 Maximum linear over-acceleration metric for scenario 4 87
3.42 Count of linear over-acceleration threshold exceedances for scenario 4 87
3.43 Maximum angular acceleration metric for scenario 4 88
3.44 Count of angular acceleration threshold exceedances for scenario 4 88
3.45 Maximum angular over-acceleration metric for scenario 4 89
3.46 Count of angular over-acceleration threshold exceedances for scenario 4 89
3.47 Minimum distance to people for scenario 4 89
3.48 Minimum distance to obstacles for scenario 4 90
3.49 Time to reach goal metric for scenario 5 93
3.50 Path length metric for scenario 5 93
3.51 Maximum linear acceleration metric for scenario 5 93
3.52 Count of linear acceleration threshold exceedances for scenario 5 94
3.53 Maximum linear over-acceleration metric for scenario 5 94
3.54 Count of linear over-acceleration threshold exceedances for scenario 5 95
3.55 Maximum angular acceleration metric for scenario 5 95
3.56 Count of angular acceleration threshold exceedances for scenario 5 95
3.57 Maximum angular over-acceleration metric for scenario 5 96
3.58 Count of angular over-acceleration threshold exceedances for scenario 5 96
3.59 Minimum distance to people for scenario 5 97
3.60 Minimum distance to obstacles for scenario 5 97
3.61 Time to reach goal metric for scenario 6 100
3.62 Path length metric for scenario 6 100
3.63 Maximum linear acceleration metric for scenario 6 101
3.64 Count of linear acceleration threshold exceedances for scenario 6 101
3.65 Maximum linear over-acceleration metric for scenario 6 101
3.66 Count of linear over-acceleration threshold exceedances for scenario 6 102
3.67 Maximum angular acceleration metric for scenario 6 102
3.68 Count of angular acceleration threshold exceedances for scenario 6 102
3.69 Maximum angular over-acceleration metric for scenario 6 103
3.70 Count of angular over-acceleration threshold exceedances for scenario 6 103
3.71 Minimum distance to people for scenario 6 104
3.72 Minimum distance to obstacles for scenario 6 104
3.73 Average comfort score across the six navigation scenarios 108
3.74 Average safety score across the six navigation scenarios 109
3.75 Average performance score across the six navigation scenarios . . 110
3.76 Linear acceleration temporal evaluation for episode 1 111
3.77 Linear over-acceleration temporal evaluation for episode 1 111
3.78 Angular acceleration temporal evaluation for episode 1 112
3.79 Angular over-acceleration temporal evaluation for episode 1 . . . 112

List of Tables

1.1 Comfortable values for the kinematic parameters reported in [8] . 24
1.2 Comparison between Topics, Services, and Actions in ROS 2. . . 30

2.1 Thresholds used in ComfortCritic 53

3.1 Success rate — Episode 1 68
3.2 Time to reach goal (s) — Episode 1 68
3.3 Path length (m) — Episode 1 68
3.4 Cumulative heading changes (rad) — Episode 1 68
3.5 Maximum linear acceleration (m/s^2) — Episode 1 69
3.6 Linear acceleration threshold exceedances — Episode 1 69
3.7 Maximum linear overacceleration (m/s^3) — Episode 1 69
3.8 Linear overacceleration threshold exceedances — Episode 1 . . . 69
3.9 Maximum angular acceleration (rad/s^2) — Episode 1 69
3.10 Angular acceleration threshold exceedances — Episode 1 69
3.11 Maximum angular overacceleration (rad/s^3) — Episode 1 70
3.12 Angular overacceleration threshold exceedances — Episode 1 . . 70
3.13 Minimum distance to people (m) — Episode 1 70
3.14 Minimum distance to obstacles (m) — Episode 1 70
3.15 Success rate — Episode 2 76
3.16 Time to reach goal (s) — Episode 2 76
3.17 Path length (m) — Episode 2 76
3.18 Cumulative heading changes (rad) — Episode 2 76
3.19 Maximum linear acceleration (m/s^2) — Episode 2 76
3.20 Linear acceleration threshold exceedances — Episode 2 77
3.21 Maximum linear overacceleration (m/s^3) — Episode 2 77
3.22 Linear overacceleration threshold exceedances — Episode 2 . . . 77
3.23 Maximum angular acceleration (rad/s^2) — Episode 2 77
3.24 Angular acceleration threshold exceedances — Episode 2 77
3.25 Maximum angular overacceleration (rad/s^3) — Episode 2 77
3.26 Angular overacceleration threshold exceedances — Episode 2 . . 78
3.27 Minimum distance to people (m) — Episode 2 78
3.28 Minimum distance to obstacles (m) — Episode 2 78
3.29 Success rate — Episode 3 83
3.30 Time to reach goal (s) — Episode 3 83
3.31 Path length (m) — Episode 3 83
3.32 Cumulative heading changes (rad) — Episode 3 83
3.33 Maximum linear acceleration (m/s^2) — Episode 3 84
3.34 Linear acceleration threshold exceedances — Episode 3 84
3.35 Maximum linear overacceleration (m/s^3) — Episode 3 84

3.36 Linear overacceleration threshold exceedances — Episode 3 . . . 84
3.37 Maximum angular acceleration (rad/s^2) — Episode 3 84
3.38 Angular acceleration threshold exceedances — Episode 3 84
3.39 Maximum angular overacceleration (rad/s^3) — Episode 3 85
3.40 Angular overacceleration threshold exceedances — Episode 3 . . 85
3.41 Minimum distance to people (m) — Episode 3 85
3.42 Minimum distance to obstacles (m) — Episode 3 85
3.43 Success rate — Episode 4 90
3.44 Time to reach goal (s) — Episode 4 90
3.45 Path length (m) — Episode 4 90
3.46 Cumulative heading changes (rad) — Episode 4 91
3.47 Maximum linear acceleration (m/s^2) — Episode 4 91
3.48 Linear acceleration threshold exceedances — Episode 4 91
3.49 Maximum linear overacceleration (m/s^3) — Episode 4 91
3.50 Linear overacceleration threshold exceedances — Episode 4 . . . 91
3.51 Maximum angular acceleration (rad/s^2) — Episode 4 91
3.52 Angular acceleration threshold exceedances — Episode 4 92
3.53 Maximum angular overacceleration (rad/s^3) — Episode 4 92
3.54 Angular overacceleration threshold exceedances — Episode 4 . . 92
3.55 Minimum distance to people (m) — Episode 4 92
3.56 Minimum distance to obstacles (m) — Episode 4 92
3.57 Success rate — Episode 5 97
3.58 Time to reach goal (s) — Episode 5 98
3.59 Path length (m) — Episode 5 98
3.60 Cumulative heading changes (rad) — Episode 5 98
3.61 Maximum linear acceleration (m/s^2) — Episode 5 98
3.62 Linear acceleration threshold exceedances — Episode 5 98
3.63 Maximum linear overacceleration (m/s^3) — Episode 5 98
3.64 Linear overacceleration threshold exceedances — Episode 5 . . . 99
3.65 Maximum angular acceleration (rad/s^2) — Episode 5 99
3.66 Angular acceleration threshold exceedances — Episode 5 99
3.67 Maximum angular overacceleration (rad/s^3) — Episode 5 99
3.68 Angular overacceleration threshold exceedances — Episode 5 . . 99
3.69 Minimum distance to people (m) — Episode 5 99
3.70 Minimum distance to obstacles (m) — Episode 5 100
3.71 Success rate — Episode 6 104
3.72 Time to reach goal (s) — Episode 6 105
3.73 Path length (m) — Episode 6 105
3.74 Cumulative heading changes (rad) — Episode 6 105
3.75 Maximum linear acceleration (m/s^2) — Episode 6 105
3.76 Linear acceleration threshold exceedances — Episode 6 105
3.77 Maximum linear overacceleration (m/s^3) — Episode 6 105
3.78 Linear overacceleration threshold exceedances — Episode 6 . . . 106
3.79 Maximum angular acceleration (rad/s^2) — Episode 6 106
3.80 Angular acceleration threshold exceedances — Episode 6 106
3.81 Maximum angular overacceleration (rad/s^3) — Episode 6 106
3.82 Angular overacceleration threshold exceedances — Episode 6 . . 106
3.83 Minimum distance to people (m) — Episode 6 106
3.84 Minimum distance to obstacles (m) — Episode 6 107
3.85 Weights used in the comfort score 108

3.86 Weights used in the safety score 109
3.87 Weights used in the performance score 110

List of Algorithms

1 NavFn Algorithm (Potential Calculation) 40
2 NavFn Algorithm (Path Extraction) 42
3 MPPI optimization sequence 49
4 ComfortMPPICritic Scoring Logic 54

List of Equations

1.1 17
1.2 17
1.3 17
1.4 18
1.5 18
1.6 18
1.7 20
1.8 21
1.9 21
1.10. 22
1.11. 22
1.12. 22
1.13. 26
1.14. 26
1.15. 26
1.16. 26
1.17. 27
1.18. 27
1.19. 27
1.20. 27
1.21. 34
1.22. 34
1.23. 36
1.24. 37
1.25. 43
1.26. 43
1.27. 43
1.28. 43
1.29. 44
1.30. 44
1.31. 44
1.32. 44
1.33. 44
1.34. 44
1.35. 45

1.36. 45
1.37. 45
1.38. 45
1.39. 45
1.40. 45
1.41. 45
1.42. 45
1.43. 46
1.44. 46
1.45. 46
1.46. 46
1.47. 46
1.48. 46
1.49. 47
1.50. 47
1.51. 47
1.52. 47
1.53. 47
1.54. 47
1.55. 47
1.56. 47
1.57. 48
1.58. 48
1.59. 48
1.60. 48
2.1 52
2.2 52
2.3 53
2.4 53
2.5 53
2.6 53
2.7 53
2.8 53
3.1 107
3.2 107
3.3 107
3.4 107
3.5 107

Acronyms

A*
A-star

API
Application Programming Interface

BT
Behavior Tree

CPU
Central Processing Unit

DDS
Data Distribution Service

D*
Dynamic A-star

DWA
Dynamic Window Approach

EKF
Extended Kalman Filter

EPW
Electric Powered Wheelchair

GPU
Graphical Processing Unit

GVD
Generalized Voronoi Diagram

HJB
Hamilton-Jacobi-Bellman

MPC
Model Predictive Control

MPPI
Model Path Predictive Integral

Nav2
Navigation 2

ROS2
Robot Operating System 2

SLAM
Simultaneous Localization and Mapping

VCS
Version Control System

Introduction

Demographic Trends and Assistive Mobility

Autonomous navigation is a well-established research topic in robotics and has been extensively studied in the literature, particularly for mobile robots in indoor environments. In recent years, autonomous navigation systems have been increasingly deployed in real-world applications, especially in warehouses, where the coexistence between people and robots requires reliable navigation strategies capable of operating safely in shared and dynamic environments. Beyond industrial settings, autonomous navigation has also gained significant relevance in human transportation systems, including service robots and electric powered wheelchairs (EPWs).
The relevance of studies on EPWs is strongly connected to demographic trends, which indicate a long-term shift toward an ageing society, both globally and within the European Union, as shown by Figure [1]. The increment of life expectancy has led to a growing proportion of older individuals, particularly those aged 65 and over. In 2024 these cohorts occupy a significantly larger share of the total population compared to 2009, suggesting an increase in people with mobility difficulties. As a consequence, the demand for assistive mobile technologies, such as EPWs, is expected to rise steadily in the coming decades. These systems aim to support people with reduced mobility in maintaining independence and quality of life. Safety has traditionally been the primary objective in autonomous navigation for EPWs and mobile robots. Collision avoidance, robustness and reliability have been widely addressed in the literature and modern navigation frameworks provide mature solutions to ensure safe operation in dynamic environments. However, safety is not sufficient to guarantee user acceptance due to the importance of comfort during navigation. Abrupt acceleration, excessive over-acceleration and frequent changes in direction lead to discomfort, stress and reduce trust in autonomous systems, even when safety constraints are respected.

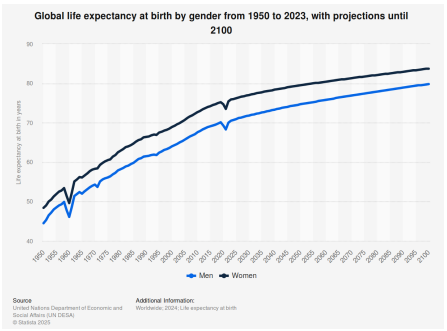

Figure 1: [2]Global life expectancy at birth by gender from 1950 to 2023, with projections until 2100

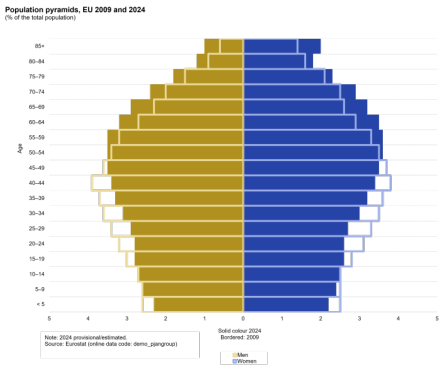

Figure 2: [1] Population pyramids of the EU in 2009 and 2024(% of the total population)

Thesis overview

This thesis investigates autonomous navigation for electric powered wheelchairs with a strong focus on passenger comfort. The work aims to evaluate how different combinations of global planners and local planners influence both navigation performance and comfort-related aspects, within the ROS2 Navigation Stack (NAV2). In addition to the local planner proposed by NAV2, a comfort-oriented critic is added to the MPPI controller to improve comfort. The global planners used are the *SmacPlannerHybrid*, *SmacPlannerLattice* and *NavFn* and the local planners are *MPPI controller* and *DWB controller*.

- **Chapter 1** reviews the state of the art in autonomous navigation and human-centered mobility, with specific attention to comfort considerations in robot wheelchairs and service robots. Then introduce the methodological background of the thesis, presenting an overview of the ROS2 ecosystems and the Nav2 framework, describing the architecture, the main components and the global and local planners considered in this thesis.
- **Chapter 2** describes the design and implementation of a comfort-oriented critic aimed at improving passenger experience during navigation. The chapter details the simulation environment, the adopted scenarios and the evaluation pipeline. The integration of dynamic human agents using Gazebo and HuNavSim is also presented.
- **Chapter 3** reports the results of the experimental evaluation. A systematic benchmark is conducted by testing multi controller-planner combinations across six different scenarios. The tables containing the numerical results are reported after the evaluation. In addition to the metrics, a temporal evolution of four kinematic quantities is described.
- **Chapter 4** summarizes the main findings of the thesis and discusses possible directions for future work.

Fundamentals

1.1 Autonomous Navigation

A mobile robot is a structure capable of moving and acting (autonomously or remotely operated) in terrestrial, underwater, or aerial environments, which can be totally or partially known or unknown. The fundamental problems in mobile robotics are the following:

- Locomotion: the motion of the robot in the environment.
- Perception: the robot perceives data from the environment through sensors.
- Representation: the organization of the knowledge about the environment.
- Mapping: building the map of the environment.
- Localization: determining the position of the robot in the environment.
- Path planning: action to be performed by the robot to complete a specified task.

1.1.1 Mobile robot localization

Mobile robot localization refers to the problem of estimating the pose of a robot within a **known** environment [3]. The robot pose is generally defined by its position and orientation in the space and it represents the transformation between the robot's local coordinate frame and the global coordinate system of the map. Accurate localization is a fundamental requirement for most robotic tasks, as it enables the robot to reason about its own position and to interact correctly with surrounding objects. Accurate localization allows the planners to compute the path, to start the navigation and determine whether the target location is reached. From a probabilistic perspective, localization can be formulated as an inference problem, where the goal is to establish correspondence between the map coordinate system and the robot's local coordinate system. This coordinate transformation enables the robot to express the location of objects of interest within its own coordinate frame, which is essential for navigation. Knowledge of the pose $x_t = (x\ y\ \theta)^T$ is sufficient to determine this coordinate transformation, but unfortunately, the pose can not usually be sensed directly. The key difficulty arises from the fact that a single sensor measurement is usually insufficient to determine the pose and the robot has to integrate data over time to determine it. To determine the pose of the robots, the literature proposes different algorithms:

- Markov localization
- EKF localization: extended Kalman filter (special case of the Markov localization)
- Grid localization

- Monte Carlo localization

These algorithms are described in detail in the book [3]

1.1.2 Mapping and the SLAM problem

In the previous section, mobile robot localization was addressed under the assumption of a known map, which had been previously created during a mapping phase. **Mapping** refers to the problem of building a representation of the environment using sensor data collected by a mobile robot, knowing the pose; unlike localization, mapping addresses scenarios in which no prior map exists and it is a common situation that arises in real-world environments, where architectural plans may be incomplete or outdated. The challenge of acquiring maps with mobile robots depends in particular on the hypothesis space [3], which is the space of all possible maps and since maps are described over a continuous space, the space of all maps has infinitely many dimensions and the Bayesian filtering used for the localization is inapplicable to the problem of learning maps. To overcome the problem of the robot not having access to a map of the environment, SLAM is introduced. Simultaneous localization and mapping (SLAM) describe the action of the robot to acquire a map of its environment while simultaneously localizing itself relative to this map [3]. To solve this dual problem, there exist different approaches, mainly related to different representations of the map such as the **EKF-SLAM** and the **particle filters**, respectively, related to the landmark based maps and occupancy grids.

1.1.3 Path Planning

Path planning aims at determining a collision-free path that allows a robot to move from an initial position to a target one and to perform this task, knowledge of the environment is required. In its basic formulation, path planning focuses exclusively on the geometric aspect of the problem, neglecting the temporal evolution of motion, which is considered by the controller. Path planning represents only one component of the broader trajectory planning problem, generating a geometric path $q(t)$, while trajectory planning determines the control inputs $u(t)$ required to actuate the robot so that the computed path can be followed while respecting the robot's dynamic constraints. In the literature several algorithms have been developed to solve the motion planning problem and over time, these methods have been extended with optimization techniques aimed at improving the navigation, for instance by minimizing the traveled distance or the energy consumption. Among the most widely used approaches are heuristic search-based planners, such as **A***, which explores the search space by evaluating paths according to a cost function and selects the solution with the minimum cost. **Greedy search** makes locally optimal choices at each step in order to rapidly approximate a solution. **Dijkstra's algorithm** computes the shortest path between nodes in a graph representation of the environment and the **D*** extends Dijkstra's algorithm by planning backwards, from the goal to the initial configuration.

1.1.3.1 Dijkstra's algorithm

Many global planning tasks in autonomous navigation can be formulated as a shortest-path problem, such as route planning on a road map, where intersections are modeled as nodes and roads as edges, each associated with a cost depending on distance or travel time. Since it is computationally infeasible to enumerate all possible routes, polynomial-time shortest-path algorithms are typically employed. Formally, let $G = (V, E)$ be a directed weighted graph, where the weight function assigns a real-valued cost to each edge:

$$w : E \to \mathbb{R} \tag{1.1}$$

A path p has total cost:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{1.2}$$

The shortest-path distance between two vertices u and v is defined as:

$$\delta(u, v) = \begin{cases} \min\{w(p) \mid u \rightsquigarrow v\} & \text{if a path exists,} \\ \infty & \text{otherwise.} \end{cases} \tag{1.3}$$

In mobile robot navigation, edge weights do not necessarily represent metric distances; instead, they may encode travel time, risk, terrain difficulty, or more generally any additive cost accumulated along a path.
Dijkstra's algorithm is one of the most widely used methods for solving the single-source shortest-path problem on weighted graphs. It relies on the assumption that all edge weights are non-negative. Under this condition, the algorithm is guaranteed to compute optimal shortest paths.
Dijkstra's algorithm incrementally builds the set of vertices whose shortest-path distance from the source is known with certainty and maintains the following data structures:

- a distance estimate $d[v]$, representing the current best known cost from the source s to vertex v;
- a predecessor $\pi[v]$, used to reconstruct the shortest path;
- a set $S \subseteq V$, containing the vertices for which the shortest-path distance has been finalized;
- a min-priority queue containing the remaining vertices, ordered according to their distance estimates.

Initially, the distance of the source vertex is set to zero, $d[s] = 0$, while all other vertices are assigned an infinite distance. The set S is initialized as empty, and all vertices are inserted into the priority queue. At each iteration, the algorithm selects the vertex $u \in V \setminus S$ with the smallest distance estimate $d[u]$, removes it from the priority queue, and adds it to the set S. Then, all outgoing edges $(u, v) \in E$ are relaxed, updating the distance estimate $d[v]$ whenever a shorter path through u is found.
This process is repeated until all vertices have been extracted from the priority queue, at which point the shortest-path distances from the source to all reachable vertices have been computed.

Dijkstra's algorithm follows a greedy strategy: at each step, it permanently selects the vertex with the smallest tentative distance. While greedy strategies do not always guarantee optimal solutions in general optimization problems, Dijkstra's algorithm is provably correct under the assumption that all edge weights are non-negative. The key invariant maintained by the algorithm is that, at the moment a vertex u is added to the set S, its distance estimate satisfies $d[u] = \delta(s, u)$, where $\delta(s, u)$ denotes the true shortest-path distance from the source s to the vertex u. This property ensures that once a vertex is added to S, its shortest-path distance will not change in subsequent iterations. At termination, the predecessor relationships $\pi[v]$ define a shortest-path tree rooted at the source vertex s.

1.1.3.2 A* algorithm

While Dijkstra's algorithm guarantees optimal shortest paths, it explores the graph uniformly in all directions, without any explicit guidance toward the goal. In large or complex environments, this may result in the expansion of a large number of vertices that are not relevant to the final solution. To address this limitation, informed search strategies incorporate additional knowledge about the problem domain in the form of heuristic functions.
A* (A-star) is an informed graph-search algorithm that extends Dijkstra's approach by introducing a heuristic estimate of the remaining cost to reach the goal. At each iteration, vertices are selected for expansion according to an evaluation function defined as

$$f(n) = g(n) + h(n) \tag{1.4}$$

where $g(n)$ denotes the cost of the path from the source to vertex n, and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal.
The evaluation function $f(n)$ can therefore be interpreted as an estimate of the total cost of a solution path that passes through vertex n. By prioritizing vertices with lower values of $f(n)$, A* biases the search toward regions of the graph that are both inexpensive to reach and appear promising with respect to the goal.
The behavior and optimality of A* depend critically on the properties of the heuristic function $h(n)$. A heuristic is said to be *admissible* if it never overestimates the true cost to reach the goal,

$$h(n) \leq \delta(n, \text{goal}) \tag{1.5}$$

for all vertices n. When an admissible heuristic is used, A* is guaranteed to return a cost-optimal solution.
A stronger condition is *consistency* (also known as monotonicity), which requires the heuristic to satisfy a triangle inequality,

$$h(n) \leq w(n, n') + h(n') \tag{1.6}$$

for every edge $(n, n') \in E$. Consistency implies admissibility and ensures that once a vertex is expanded, the shortest path to that vertex has been found, thus preventing the need for re-expansion.

When the heuristic function is identically zero, that is $h(n) = 0$ for all vertices, the evaluation function reduces to $f(n) = g(n)$ and A* becomes equivalent to Dijkstra's algorithm. In this sense, A* can be seen as a strict generalization of Dijkstra's method, where the heuristic term improves efficiency without sacrificing optimality under appropriate conditions.
In the context of autonomous navigation, A* is widely adopted as a global path planning algorithm due to its favorable trade-off between computational efficiency and optimality. By incorporating domain-specific heuristics, such as Euclidean or Manhattan distance on grid-based maps, A* typically expands significantly fewer vertices than uninformed search methods while preserving optimal path guarantees.

1.1.3.3 Comparison between Dijkstra and A*

Dijkstra's algorithm and A* share a common foundation, as both are graph-search methods designed to compute shortest paths from a source vertex to a goal or to all reachable vertices. Both algorithms guarantee optimal solutions under appropriate conditions and rely on similar mechanisms such as distance estimates, edge relaxation, and priority-based expansion of vertices.
The key difference between the two approaches lies in the use of heuristic information. Dijkstra's algorithm is an uninformed search method: vertex expansion is driven solely by the accumulated path cost from the source, without considering the direction or proximity of the goal. As a result, the algorithm explores the graph uniformly in all directions, potentially expanding a large number of vertices that do not contribute to the final solution, especially in large-scale environments.
A*, on the other hand, augments the path cost with a heuristic estimate of the remaining cost to the goal. By combining the accumulated cost $g(n)$ with the heuristic term $h(n)$, A* prioritizes the exploration of vertices that are both cheap to reach and likely to lead toward the goal. When the heuristic is admissible, A* preserves optimality while significantly reducing the number of expanded vertices compared to Dijkstra's algorithm.
From a theoretical perspective, Dijkstra's algorithm can be seen as a special case of A* in which the heuristic function is identically zero. This relationship highlights how A* generalizes Dijkstra's method by introducing informed guidance without altering the underlying optimality guarantees.
In the context of autonomous navigation, this distinction has important practical implications. While Dijkstra's algorithm is often used as a baseline planner due to its simplicity and robustness, A* is generally preferred for global path planning in structured environments, as it achieves a better trade-off between computational efficiency and optimality by exploiting domain-specific heuristics.

1.1.4 Control

In autonomous mobile robot navigation, control algorithms are responsible for transforming planned paths into executable motion commands. While path planning algorithms operate at a higher level by computing collision-free paths in the environment, controllers act at a lower level and generate control inputs that directly drive the robot actuators.

The control problem is inherently dynamic, as the robot must continuously react to changes in the environment, disturbances, and modeling uncertainties. For this reason, controllers are typically designed to operate in a closed-loop fashion, continuously updating the control commands based on the current state of the robot and its surroundings.
In the context of navigation, local controllers aim to ensure safe, smooth, and dynamically feasible motion while tracking the path provided by the planner. They must explicitly account for the robot's kinematic and dynamic constraints, such as velocity and acceleration limits, and guarantee real-time performance.
Several control strategies have been proposed in the literature to address this problem. Among them, velocity-based methods such as the Dynamic Window Approach and optimization-based techniques such as Model Predictive Control represent two fundamental and widely adopted paradigms. The following sections present their theoretical foundations and operational principles.

1.1.4.1 Dynamic Window Approach

The Dynamic-Window Approach is a seminal local motion planning algorithm in mobile robotics, primarily designed for real-time obstacle avoidance. Firstly introduced in 1997 [4] and then it has been mainly used as an effective way to handle autonomous robot motion control in the presence of obstacle [5]. The trajectories of the robot are approximated by a sequence of circular and straight line arcs with a linear bounded error due to a supposition made in the derivation, to assume the velocity to be constant within a time interval. Within this framework, the planner searches directly in the velocity space, defined by the translation and rotational components (ν, ω), to find the command pair that best satisfies both dynamic feasibility and collision avoidance. The steps performed at each iteration are the following:

1. **Restriction of the search space**: the set of possible velocity pairs is reduced to be a dynamically feasible subset.

2. **Optimization**: the velocity vector that maximizes the objective function is selected and then applied to the robot.

The controller frequency determines how often the pruning and optimization processes are executed. For instance, a control frequency of 20 Hz correspond to a 50 ms time interval. Since trajectories are represented as circular arcs uniquely defined by a velocity pair, the robot would ideally compute an entire sequence of such velocities from t_0 to t_N to reach the goal. This computation leads to a computationally expensive problem. Instead, DWA evaluates only the velocities corresponding to the next time interval, exploiting the fact that the procedure is repeated at every control step. All admissible velocity pairs are selected according to two main constraints:

1. **Obstacle avoidance constraints:** a velocity vector is considered admissible if the robot can decelerate in time to full stop before colliding with any obstacle. The computation of the admissible velocitis is defined as:

$$v_a = \nu, \omega | \nu \leq \sqrt{2 \cdot dist(\nu, \omega) \cdot \dot{\nu_b}} \wedge \omega \leq \sqrt{2 \cdot dist(\nu, \omega) \cdot \dot{\omega_b}} \tag{1.7}$$

   - ν_b and ω_b are the linear and angular braking deceleration.

- V_a is the set of admissible velocities.
- $dist(\nu, \omega)$ is the distance to the closest obstacle, granting the robot the ability to stop without collision.

2. **Dynamic window:** To account for the physical acceleration limits of the robot's acutators, the overall velocity space is futher restricted to the *dynamic window*, which includes only the velocities that can be obtained within the following time interval. Let t be the interval duration, $\dot{\nu}$ and $\dot{\omega}$ the maximum accelerations and (ν_a, ω_a) the current velocity vector. The dynamic window V_d is defined as:

$$V_d = \nu, \omega | \nu \in [\nu_a - \dot{\nu}t, \nu_a + \dot{\nu_a}t], \omega \in [\omega_a - \dot{\omega}t, \omega_a + \dot{\omega_a}t] \tag{1.8}$$

The center of this window corresponds to the current velocity, while its width depend on the available acceleration capabilities. The final search space, V_r is the intersection of V_a and V_d, ensuring both safety and dynamic feasibility.
Once the feasible search space V_r is defined, the algorithm evaluates an objective function for each admissible velocity pair and selects the one that maximizes it:

$$G(\nu, \omega) = \sigma(\alpha \cdot heading(\nu, \omega) + \beta \cdot vel(\nu, \omega) + \gamma \cdot dist(\nu, \omega)) \tag{1.9}$$

- σ is a normalization factor
- $\alpha\beta\gamma$ are user-defined weighting coefficients

The objective function (1.9) balances three main criteria:

- **Target heading:** measures the alignment of the robot's heading with the goal direction. It is computed as $180 - \theta$, where θ is the angular deviation of the target relative to the robot's orientation
- **Clearance:** promotes safer trajectories by maximizing the distance to nearby obstacles. The function $dist(\nu, \omega)$ gives the minimum distance to obstacles along the corresponding arc.
- **Velocity:** encourages forward motion by favoring higher translation speeds. The term velocity (ν, ω) represent the projection of the translational velocity along the current trajectory.

The overall behavior of the controller depends heavily on the chosen weights ($\alpha, \beta,$ and γ), which must be tuned according to the robot's dynamics and the operating environment.

1.1.4.2 MPC

Model Predictive Control (MPC) is an optimization-based control framework widely used in robotics and autonomous systems to generate control actions, while explicitly accounting for system dynamics and constraints. MPC controller differs from the classical feedback controllers, by computing control inputs by repeatedly solving a finite-horizon optimal control problem, using a predictive model of the system.

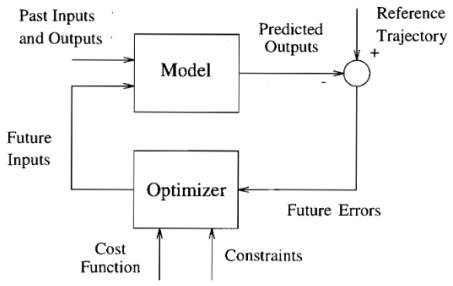

Figure 1.1: [6] Basic structure of MPC

The system to be controlled is generally described by a discrete-time state-space model:

$$x_{k+1} = f(x_k, u_k) \tag{1.10}$$

where $x_k \in \mathbb{R}^n$ represents the system state at time step k and $u_k \in \mathbb{R}^m$ denotes the control input. Given the current state, MPC optimizes a sequence of control inputs over a finite prediction horizon of length N.
The control objective is encoded in a cost function that typically penalizes both deviations from a desired reference trajectory and excessive control effort. A common formulation is

$$J = \sum_{k=0}^{N-1} \left(\|x_k - x_k^{\text{ref}}\|_Q^2 + \|u_k\|_R^2 \right) + \|x_N - x_N^{\text{ref}}\|_{Q_f}^2 \tag{1.11}$$

where x_k^{ref} denotes a reference state, and Q, R, and Q_f are positive semi-definite weighting matrices. This formulation allows balancing tracking accuracy, control smoothness, and terminal performance.
The optimization is subject to system constraints, which may include bounds on states and control inputs:

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}. \tag{1.12}$$

Such constraints allow MPC to explicitly enforce physical, safety, and comfort requirements, making it particularly suitable for autonomous navigation applications.
At each control cycle, only the first control input of the optimized sequence is applied to the system. The optimization is then repeated at the next time step using updated state measurements, following the *receding horizon* principle. This closed-loop strategy enables MPC to react to disturbances and model inaccuracies while maintaining predictive capabilities.
From a conceptual standpoint, MPC unifies trajectory planning and control within a single optimization framework. Its ability to handle multivariable systems, constraints, and nonlinear dynamics has led to its widespread adoption in mobile robot navigation, autonomous driving, and aerial robotics. However, the need to solve an optimization problem online introduces computational challenges, which have motivated the development of efficient solvers and sampling-based variants.

1.2 Social Robot Navigation

Social Robot Navigation addresses the problem of autonomous navigation in environments shared with humans, where a robot must reach a destination while ensuring safety, efficiency, and compliance with social norms. Unlike classical navigation problems, which primarily focus on geometric feasibility and obstacle avoidance, social navigation explicitly accounts for human presence, motion, and expectations, making it a multi-objective and human-centered problem.
According to the survey by Mavrogiannis et al. [7], the complexity of social robot navigation arises from the need to jointly address challenges related to motion planning, behavior generation, and evaluation in dynamic, human-populated environments. These challenges span multiple levels of the navigation stack and cannot be effectively solved by purely geometric or reactive approaches.
From a planning perspective, navigation is commonly formulated as a hierarchical process composed of global planning and local planning. Global planning typically operates on a static or slowly changing representation of the environment and aims to compute a collision-free path toward a goal. Local planning, instead, is executed online and must react to dynamic obstacles, system constraints, and short-horizon objectives. In social robot navigation, the local planning layer plays a particularly critical role, as it must continuously adapt robot behavior in response to human motion and interaction [7].
A fundamental distinction highlighted in the literature concerns the relationship between prediction and planning. Early approaches adopt a decoupled architecture, where human motion is predicted independently of the robot's actions and humans are treated as dynamic but non-reactive obstacles. While such approaches are computationally efficient, the survey reports that they often lead to conservative behaviors, oscillatory avoidance maneuvers, or the so-called freezing robot problem, where the robot stops due to excessive uncertainty [7].
To overcome these limitations, more recent approaches adopt coupled prediction and planning strategies, where human motion is explicitly modeled as responsive to robot behavior. These interaction-aware methods reason about cooperation, intent, and mutual adaptation between humans and robots. Although coupled approaches generally improve navigation performance and social acceptance in dense environments, they introduce significant computational complexity and challenging non-convex optimization problems, limiting their applicability in real-time systems.
Beyond collision avoidance and efficiency, social robot navigation places strong emphasis on human-centered criteria such as comfort, legibility, predictability, and perceived safety. The survey emphasizes that these criteria are often difficult to formalize and evaluate, yet they play a central role in determining the acceptability of robot behavior in public and assistive environments .
The majority of articles and books presented in the literature related to passenger comfort in a vehicle or an Electric Powered Wheelchair (EPW) focus on the response of different parts of the human body to vibrations and on the conditions under which such vibrations are perceived as uncomfortable, often suggesting solutions that optimize suspension systems to suppress these effects. However, as highlighted in recent works, comfort is not solely determined by mechanical vibrations, but also by the characteristics of the executed path and motion [7].
Enhancing comfort in autonomous navigation therefore requires understanding

comfort as a holistic concept that includes motion smoothness, predictability, and alignment with human driving behavior. Generating trajectories that resemble human motion patterns has been shown to improve user acceptance while preserving feasibility and safety.
A first class of methods to improve comfort during navigation is directly related to the regulation of kinematic quantities, as abrupt changes in motion are commonly associated with discomfort. The most relevant quantities include:

- linear velocity ($\dot{x}$), linear acceleration ($\ddot{x}$), and linear jerk (j);
- angular velocity ($\dot{\theta}$), angular acceleration ($\ddot{\theta}$), and angular jerk (j_θ).

In [8], an experimental study investigates passenger comfort during straight and curved navigation, reporting comfortable thresholds for both linear and angular kinematic parameters. These values provide quantitative guidelines that can be exploited to design comfort-aware navigation strategies, particularly in assistive mobility applications.

Parameter	**Threshold**
Linear velocity	$v_{\max} = 0.80\,\mathrm{m/s}$
Angular velocity	$\omega_{\max} = 0.60\,\mathrm{rad/s}$
Linear acceleration	$a_{\max} = 0.10\,\mathrm{m/s^2}$
Angular acceleration	$\alpha_{\max} = 0.40\,\mathrm{rad/s^2}$

Table 1.1: Comfortable values for the kinematic parameters reported in [8]

A second approach to improving comfort is not directly related to kinematic constraints, but instead concerns human perception during navigation. An experiment conducted in Japan [9] shows a preferred spatial location of an EPW during navigation in a straight corridor, suggesting that passengers feel more comfortable when the wheelchair moves closer to the center of one side of the corridor rather than along the centerline.

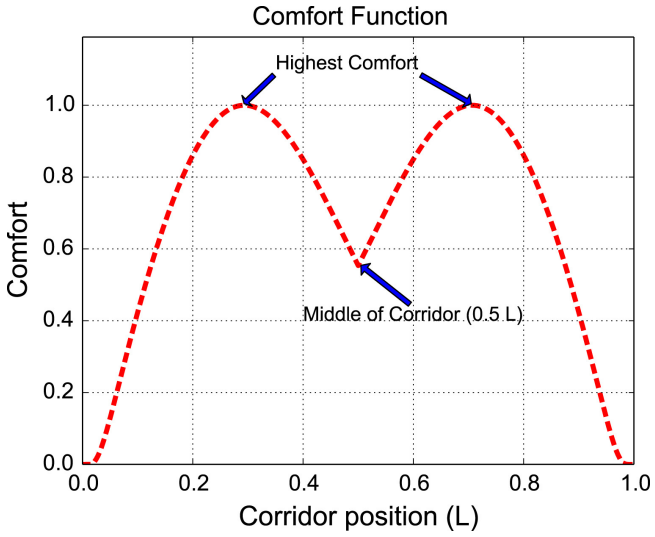

Figure 1.2: [9] Comfort model in an indoor corridor environment (normalized).

Another factor that significantly affects comfort is visibility, particularly in the presence of blind corners in corridors. In such scenarios, humans naturally approach corners more slowly and execute smoother turns due to uncertainty and the risk of potential collisions. Conversely, when visibility is good, higher velocities and sharper turns are generally perceived as acceptable.
To account for visibility and preferred positioning during navigation, a possible solution proposed in [10] consists in augmenting the planner objective with additional weighted cost terms related to comfort and visibility. This approach enables the generation of trajectories that better align with human expectations while preserving safety and feasibility.

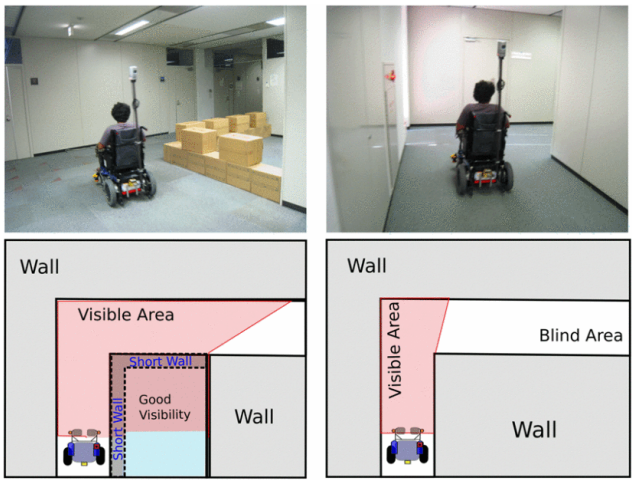

Figure 1.3: [10] Wheelchair turning right in a corridor corner.

Overall, as emphasized in [7], social robot navigation can be formulated as a multi-objective optimization problem, where efficiency, safety, comfort, and social compliance must be balanced rather than optimized independently. These considerations motivate the adoption of planning and control frameworks capable of explicitly integrating human-centered criteria into their objective functions and constraints.
Comfort can be interpreted through the lens of situation awareness, where perception of surrounding elements (e.g., pedestrians and obstacles) and comprehension of their spatial relationships play a central role in shaping the passenger's experience. In crowded indoor environments, discomfort does not arise solely from abrupt motion, but also from how the passenger perceives interactions with nearby agents.
Experimental investigations conducted in simulated crowded scenarios [11] show that passenger discomfort is strongly associated with violations of personal space (PS) by surrounding pedestrians. The PS is modeled as an anisotropic region centered on the passenger, elongated in the forward direction and reduced laterally and rearward. To quantify the extent of spatial intrusion, an *invasion ratio* is introduced, defined as the proportion between the invaded portion of the personal space and its total area:

$$I = \frac{I_b}{I_a} \tag{1.13}$$

where I_b represents the invaded area and I_a denotes the total personal space area. The geometric definition of the personal space and the invasion ratio is illustrated in Fig. 1.10.

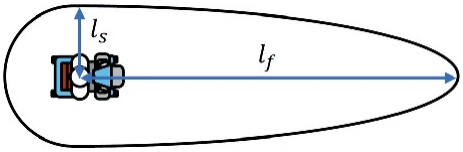

Figure 1.4: [11] Definition of invasion ratio into the passenger's personal space.

During the experiment reported in [11],passenger discomfort was collected continuously using an analog input scale. Since participants exhibited heterogeneous reporting behaviors, raw discomfort values were normalized using a median-based discretization strategy. For each participant, the input signal q was transformed into a corrected value q' as follows:

$$q' = \begin{cases} 0 & \text{if } q = 0 \\ 0.5 & \text{if } 0 < q \leq q_{\text{median}} \\ 1 & \text{if } q > q_{\text{median}} \end{cases} \tag{1.14}$$

where q_{median} denotes the median discomfort value for the individual participant. The averaged corrected signal $q'(t)$ across subjects was then used as a collective discomfort indicator.
To analyze the relationship between spatial intrusion and perceived discomfort, cumulative measures were introduced. The time-integrated invasion ratio over a given interval was computed as:

$$I_{ir} = \sum_{j \in S} \int I_j(t)\, dt \tag{1.15}$$

where S is the set of pedestrians entering the personal space and $I_j(t)$ is the instantaneous invasion ratio associated with pedestrian j. Similarly, cumulative discomfort was evaluated as:

$$I_{pd} = \int_{t_1}^{t_2} q'(t)\, dt \tag{1.16}$$

where t_1 and t_2 delimit the considered time interval.
The results reveal a clear positive correlation between cumulative invasion (I_{ir}) and cumulative discomfort (I_{pd}), with a coefficient of determination close to 0.6. This finding suggests that preserving personal space is a major determinant of perceived comfort during autonomous navigation in dense environments.
Nevertheless, the relationship is not purely geometric. Additional contextual factors were shown to modulate discomfort perception. In particular, proximity to static obstacles such as walls increased discomfort even in the absence of significant pedestrian intrusion. Moreover, invasions occurring immediately after

motion onset were associated with lower-than-expected discomfort levels, likely due to transient dynamics during the acceleration phase. These observations indicate that passenger comfort depends not only on spatial intrusion and kinematic smoothness, but also on environmental context and temporal evolution of the interaction.
Building upon the identified characteristics of passenger comfort recognition, [11] further proposes an autonomous navigation framework explicitly designed to reduce discomfort in crowded pedestrian environments. Two main requirements are derived from the experimental findings: (i) minimizing the invasion ratio into the passenger's personal space, and (ii) reducing the time intervals in which the vehicle could avoid an approaching pedestrian but does not actively perform avoidance maneuvers.
To satisfy these requirements, the authors extend the classical Dynamic Window Approach (DWA) by introducing a three-layer planning architecture composed of a Global Planner, a Middle Path Planner, and a Local Planner. The key idea is to incorporate both immediate spatial intrusion and anticipated future interactions into the planning objective.

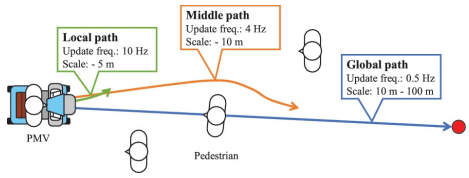

Figure 1.5: [11] Overview of the three-layer path planning.

In the Middle Path Planner, candidate trajectories are generated over a finite prediction horizon of $3T_{\text{middle}}$, each composed of three consecutive motion segments with constant velocity and steering angle. Among the candidate paths, the one maximizing the following evaluation function is selected:

$$G_{\text{middle}} = \beta_{\text{goal}} L_{\text{goal}} + \beta_{\text{inv}} L_{\text{inv}} \tag{1.17}$$

where L_{goal} represents the distance to the global target point and L_{inv} encodes cumulative spatial intrusion into the personal space. The latter is defined as a discounted time integral of pedestrian and wall invasion ratios:

$$L_{\text{inv}} = -\int_0^{3T_{\text{middle}}} \gamma^t \big(I_{\text{ped}}(t) + I_{\text{wall}}(t)\big)\, dt \tag{1.18}$$

where γ is a discount factor that regulates the importance of future intrusions. This formulation encourages early avoidance behaviors by penalizing predicted invasions before they occur.
The Local Planner operates over a shorter horizon T_{local} and computes the control commands (linear velocity and steering angle) by maximizing:

$$G_{\text{local}} = \beta'_{\text{goal}} L'_{\text{goal}} + \beta'_{\text{inv}} L'_{\text{inv}} \tag{1.19}$$

where L'_{inv} is defined as:

$$L'_{\text{inv}} = -\int_0^{T_{\text{local}}} \big(I_{\text{ped}}(t) + I_{\text{wall}}(t)\big)\, dt \tag{1.20}$$

By embedding invasion-based costs at both mid-term and short-term planning levels, the proposed framework integrates both the *Current Situation* (instantaneous spatial intrusion) and the *Future Status* (predicted interactions) within the situation awareness paradigm.
Simulation-based validation demonstrates that incorporating personal space preservation into the evaluation function significantly reduces perceived discomfort compared to baseline navigation strategies. In particular, discomfort increases when the vehicle delays avoidance actions despite having the possibility to react, highlighting the importance of anticipatory behavior in dense environments.
Overall, this work shows that comfort-aware navigation can be formulated as an optimization problem where invasion ratio minimization and proactive avoidance are explicitly embedded into the planner objectives. However, the study also reports limitations related to the VR-based experimental setup, limited participant diversity, and restricted scenario variability, suggesting the need for further real-world validation and broader experimental sampling.

1.3 The Robot Operating Systems

1.3.1 Introduction

The Robot Operating System is a meta-operating system, a middleware that provides functionalities for robot applications. The project started at Stanford University by a team of researchers in 2007 and it was taken over by Willow Garage, a robotics incubator which named the project ROS. The first release of ROS1 contains some problems for the development of robust commercial robots, but also in terms of security and reliability and for these reasons, in 2017 the second version ROS2 has been released. ROS2 is based on the Data Distribution Service (DDS), which is an open standard for communications and enables the operating systems to obtain the best-in-class security, embedded and real-time support and multi-robot communications. One of the main differences between the two versions of ROS, is that the system is purely decentralized and distributed in ROS2. While the first version used a Master node that drives client/server architecture and allows the nodes inside the network to retrieve each other, ROS2 uses DDS, that drives a distributed architecture; in this way each node is independent, while in ROS1 each node depends on the master node. Unless otherwise specified, all the implementations and discussion in this thesis refer to ROS2.

1.3.2 Nodes and Topics

ROS has three levels of concepts: the Filesystem level, the Computation Graph level and the Community level [12]. The Filesystem level concepts cover ROS resources encountered on disk:

- **Packages:** The organization of software in ROS. The packages are the most atomic build item and release item.
- **Metapackages:** Specialized packages which only serve to represent a group of related other packages.

- **Package Manifest:** (package.xml) provides metadata about a package.
- **Repositories:** A collection of packages which share a common VCS system.
- **Message (msg) types:** Message descriptions that define the data structure for messages sent in ROS.
- **Service (srv) types:** Service descriptions that define the request and response data structure for services in ROS.

The next level of concepts is the Computation Graph and it is the peer-to-peer network of ROS processes that are processing data together. ROS implements the "divide et impera" paradigm, for which each complex task can be divided into simpler tasks. The process can be visualized as a graph formed by vertices and edges connecting them: in ROS each running program is called node and the inter-communication channels are topics.

- **Nodes:** Processes that execute computations. ROS adopts a modular architecture where a robotic system is typically composed of multiple nodes, each handling a specific task. For instance, one node may manage the wheel motors, another the laser range finder, others may perform localization, path planning, or visualization. Each node is implemented using a ROS client library, such as `rclcpp` or `rclpy`.
- **Topics:** Named communication channels that enable message exchange following a publish/subscribe model. A node publishes data to a topic, while other nodes subscribe to receive it. This decouples data producers from consumers and supports multiple concurrent publishers and subscribers for the same topic.

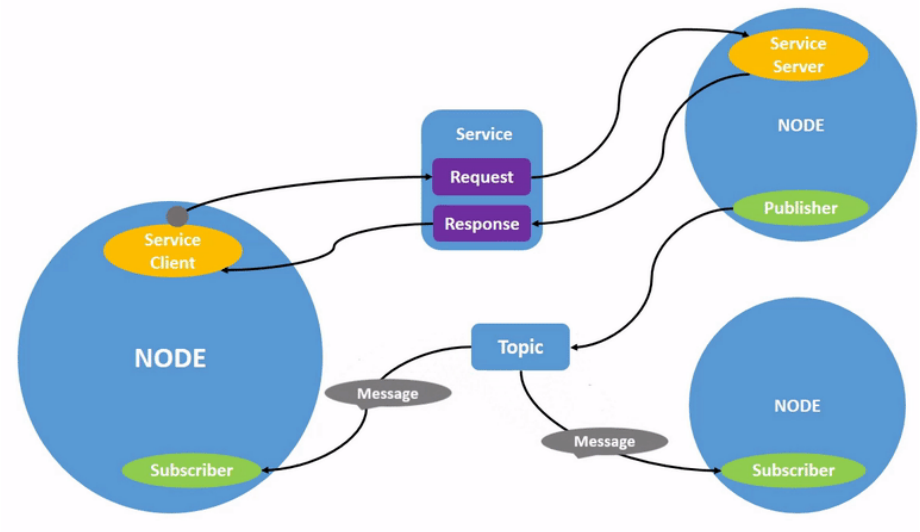

Figure 1.6: [13]ROS2 topic communication pattern

1.3.3 Messages, Services and Actions

To share information or data, the nodes use publishers to send them to a topic and to receive that information or data, a node needs a subscriber to the same topic. Besides its unique name, each topic has a message type.
Topics should be used for continuous data streams while for quick requests, the services are used. The services are a communication channel among nodes, characterized by a service type. Furthermore, there exists another type of communication and it is given by Actions, which are used for asynchronous communication such as to cancel the request during execution or to get periodic feedback. They are topics to send goal messages from a client to the server.

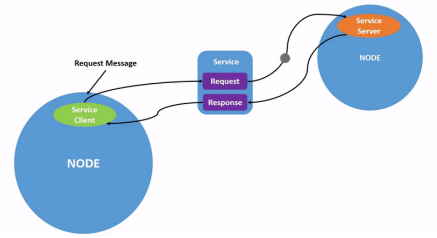

Figure 1.7: ROS2 service communication pattern

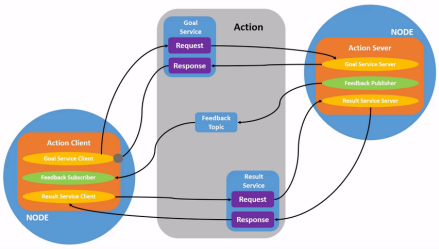

Figure 1.8: [13]ROS2 action comunication pattern

Topics	Used for continuous data streams (e.g., sensor data, robot state). Designed for ongoing data flow where messages may be published or received at any time, independent of specific senders or receivers.
Services	Used for remote procedure calls that complete quickly. They represent simple blocking interactions, typically for fast tasks such as requesting or setting specific information.
Actions	Used for long-running or discrete behaviors. They can maintain state during goal execution and may be preempted or canceled dynamically by the client.

Table 1.2: Comparison between Topics, Services, and Actions in ROS 2.

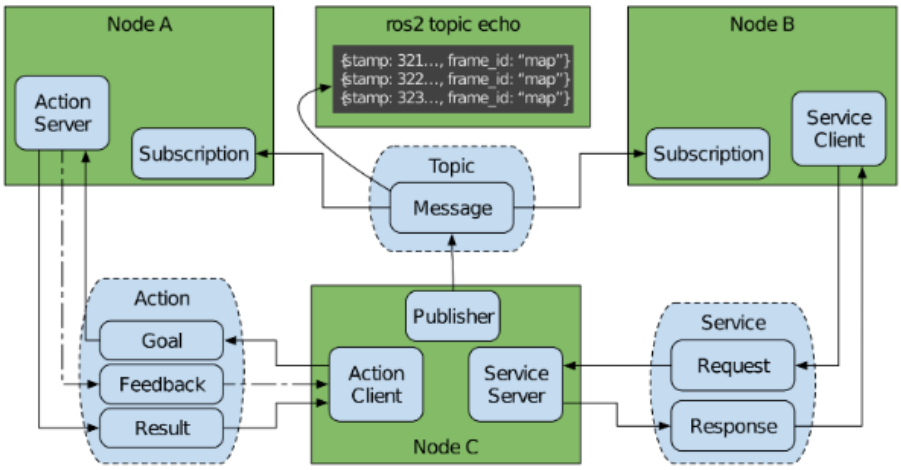

Figure 1.9: **doi:10.1126/scirobotics.abm6074**ROS2 node interfaces

The ROS Community level concepts are ROS resources that enable separate communities to exchange software and knowledge.

1.4 The Navigation2 Framework

1.4.1 Introduction

Over the past decade, several mobile robot navigation frameworks have been proposed, including the ROS navigation Stack from ROS1, which had a significant impact on industrial, academic and research applications. This framework provided a modular and configurable navigation system, integrating the most common planning and control algorithms such as *A** and *Dynamic Window Approach (DWA)*. However, it exhibited limitations in dynamic environments and did not provide sufficient safety for the modern standard. To overcome this limitation, the Navigation2 framework was introduced as a complete redesign built on ROS2. Nav2 is designed for a high degree of configurability and is in continuing expansion; it is built on a real-time capable meta-operating system to address functional safety standards and determinism for ROS2[14].
Navigation2 offers high modularity and reconfigurability in order to allow the system to work with a large variety of robots in many environments. To accomplish this, two design patterns are created:

- A behavior tree navigator
- A task-specific asynchronous server

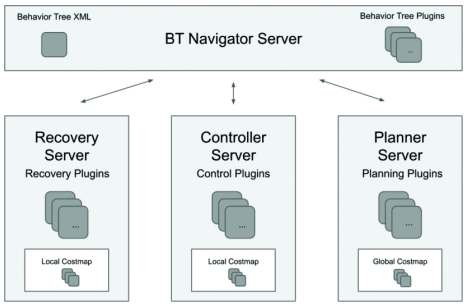

Figure 1.10: [14]Overview of Navigation2 design

The Behavior Tree Navigator uses a behavior tree to coordinate the navigation tasks, activating and tracking the progress of planner, controller and recovery server to navigate and it is possible through the ROS2 action interface. The implementation of ROS2 allows the BT nodes in the navigator to call long running asynchronous server in other processor cores. Navigation2 BT employs dynamic libraries to load BT node plugins. This approach enables the creation of reusable primitive nodes that can be dynamically loaded into a BT XML file at run time, without requiring static linkage to the navigator itself. Beyond leveraging multiple processor cores, these nodes are capable of invoking remote server operating on different CPUs and implemented in any language supported by a ROS2 client library. Each task-specific server hosts a ROS2 service interface, an environmental model and a run-time-selectable algorithm plugin. The architecture is modular, allowing any numbers of such servers to run concurrently or independently to compute actions. The ROS2 service serves as the entry point for BT Navigator nodes, handling requests such as cancellations, preemption and updates. These requests are then passed to the algorithm plugin, which performs the required task using the provided environmental model. The action servers implemented in Nav2 are the following:

- **Recovery**
- **Planner**
- **Controller**

Recovery behaviors are used to handle situations in which the navigation fails to complete its task. These behaviors are triggered by lead nodes in the BT and executed by the recovery server. By design, they are typically arranged from the least to the most intrusive corrective actions. They may operate at different levels: some are local to a specific subtree, while other belong to a system-level recovery subtree, handling general navigation failures. The most used recoveries in Nav2 are:

- **Clear Costmap:** a recovery to clear costmap layers in case of perception system failure.
- **Spin:** a recovery to clear out free space and nudge the robot out of potential local failures.
- **Wait:** a recovery to wait in case of time-based obstacles like human traffic or collecting more sensor data.

The global planner is responsible for computing the shortest path to a target, while the controller (local planner) generates the local trajectory and control commands using nearby environmental information, such as distance to obstacles or presence of dynamic obstacles such as humans. Both planner and controller are implemented as plugins executed by dedicated asynchronous servers and to optimize the performance, the global and local costmaps are maintained directly within their respective servers.

1.4.2 Global Planners

Nav2 frameworks implements several different global planners:

- Navfn Planner
- SmacPlannerHybrid
- SmacPlanner2D
- SmacPlannerLattice
- ThetaStarPlanner

In this thesis work, three planners are tested in the benchmark: **SmacPlannerHybrid**, **SmacPlannerLattice** and **NavFN**.
The default planner of Nav2 framework is the NavFn planner, an efficient planner but it lacks the necessary features for Ackermann, legged or non-circular robot systems. For this reason the *Smac Planner* is implemented in Nav2.
The *Smac Planner* is designed to maximize performance, modularity and extensibility. At its core is a templated A* algorithm, which decouples the search strategy from planner-specific logic. This templated structure allows different planning approaches to be implemented as node types that define custom behavior, including state transitions, cost evaluations and neighbor generation. This design enables code reuse while supporting different motion models and heuristics. An important design choice is the independence of the *Smac Planner* to the middleware, with ROS2 integration handled at a high abstraction layer to facilitate portability and integration into various robotic systems [15].

1.4.2.1 Smac Planners

A key innovation of the *Smac Planner* is its **Cost-Aware Obstacle Heuristic**, which integrates environmental costmap information directly into the search process. It differs from the traditional feasible planners that rely solely on binary obstacle occupancy due to the heuristic which takes into account graded cost values from the environment, such as inflation layers or dynamic risk zones. By embedding these costs field into the planning logic, the system can avoid regions with soft constraints (areas that are technically traversable but undesirable due to risk). The heuristic operates by executing a dynamic programming-based differential A* search from the goal, storing accumulated path costs which are then used to guide forward A* search during planning. This approach significantly reduces planning time by steering the search away from dead ends, inefficient paths, and high-cost regions. The traversal cost C is defined as a

function of both traversal distance d and normalized costmap intensity, given by the equation:

$$C = d \cdot \left(1 + \alpha \cdot \frac{c_{i,j}}{c_{max}}\right) \tag{1.21}$$

where $c_{i,j}$ is the cost of traversing from cell i to cell j, c_{max} is the maximum cost in the map and α is a tunable weight parameter. A higher value of α emphasizes obstacle avoidance at the expense of path length, guiding the planner through lower-cost regions. This cost function is used consistently for both heuristic estimation and traversal scoring, ensuring admissibility and consistency throughout the search. To further improve path quality, the planner includes penalty functions that modify traversal cost based on the characteristics of each motion primitive. The non-straight penalty β introduces a cost for turning, encouraging straight-line motion when possible, reducing unnecessary curvature. The change penalty γ penalizes changes in turning direction between consecutive primitives, promoting smoother and more deliberate motion. These adjustments are incorporated into the final cost $C*$ as follows:

$$C^* = \begin{cases} C & \text{if } \delta_i = 0, \\ (1+\beta) \cdot C & \text{if } \operatorname{sgn}(\delta_i) = \operatorname{sgn}(\delta_{i-1}), \\ (1+\beta+\gamma) \cdot C & \text{if } \operatorname{sgn}(\delta_i) \neq \operatorname{sgn}(\delta_{i-1}). \end{cases} \tag{1.22}$$

Where δ_i denotes the angular change associated with the i-th motion primitive. Additionally, a reverse penalty can be applied to discourage backward motion unless necessary for maneuvers such as turnarounds, which is particularly relevant for Ackermann-type vehicles. The *Smac Planner* framework supports three planner types:

- 2D-A*
- Hybrid-A*
- State Lattice

All planners supported are built on the shared A* template and differentiated primarily by their motion models and state representations.
The **2D-A*** planner is the simplest variant, operating in the two-dimensional space (x, y) and intended for circular robots. It leverages the Cost-Aware Obstacle Heuristic and uses an 8-connected Moore neighborhood for state expansion. Although it does not enforce kinematic feasibility, it provides efficient baseline planning suitable for many differential or holonomic platforms.
The **Hybrid-A*** planner extends planning into the continuous (x, y, θ) space, using Dubins or Reeds-Shepp motion models with a specific minimum turning radius. It stores continuous poses of each node, resulting in dynamically feasible trajectories without the need for discrete angular constraints. Two heuristics are combined for admissibility. The planner dynamically generates motion primitives according to grid resolution and supports flexible goal orientation modes, including bidirectional and orientation-agnostic arrivals. The planner optionally uses an analytic expansion module, which computes optimal paths between the current node and goal pose using circular arcs and applies coarse-to-fine search to ensure feasibility near the goal. The SmacPlannerHybrid of Nav2

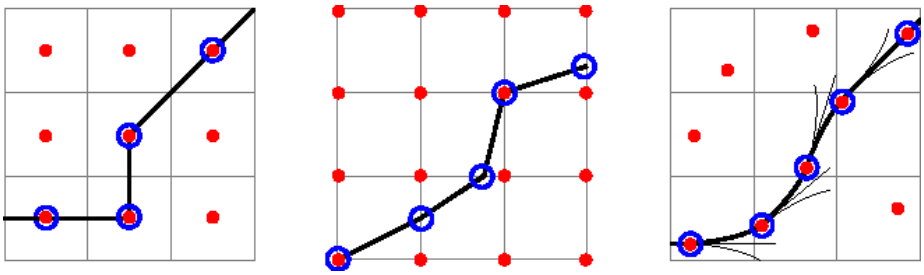

Figure 1.11: [14]Graphical comparison of search algorithms. Left: A*. Center: D* and Theta*. Right: Hybrid A*

implements the algorithm as proposed in [16], with modifications to the heuristic, traversal function and to increase path quality without needing expensive optimization-based smoothing. The variation introduced by the Hybrid-State A* Search modifies the state-transition update to embed continuous state information within the discrete search nodes of A*. Contrary to the traditional A*, which restricts state to cell centers, the hybrid-state formulation associates each grid cell with a full continuous 3D vehicle state, as shown in Figure 1.11. However, the search space (x, y, θ) is always discretized.
Due to the fact that multiple continuous states may map to the same discrete cell, hybrid-state A* does not guarantee a globally minimal-cost path, but the resulting path is guaranteed to be dynamically feasible, a property that A* does not guarantee. Even if the globally minimal-cost path is not followed, the algorithm in practice lies close to the neighborhood of the global optimum, aallowing the algorithm to arrive at the globally optimal solution via the second phase of the algorithm. A key benefit of hybrid-state A* emerges in constrained environments, where discretization artifacts would otherwise compromise path feasibility; both forward and reverse motion are supported, applying penalties for driving backward and for switching motion direction. The work [16] uses two heuristics for the search algorithm, both compatible with hybrid-state A* as well as other search-based planners. **The first heuristic**, termed non-holonomic-without-obstacles, ignores obstacle information but incorporates the non-holonomic constraints of the vehicle. It is computed by assuming a goal state $(x_g, y_g, \theta_g) = (0, 0, 0)$ and computing the shortest path to the goal from every point (x, y, θ) in some discretized neighborhood of the goal, assuming complete absence of obstacles. It is combined with the 2D Euclidean distance by taking their maximum, resulting in the avoidance of search branches that approach the goal with infeasible orientations. Since it is not affected by the sensor data, the heuristic can be precomputed offline and later transformed to align with the active goal pose.
The second heuristic is complementary, disregarding vehicle kinematics and instead uses the obstacle map to estimate the shortest 2D collision-free path to the goal via dynamic programming. The experimental results [16] show that the first heuristic can reduce node expansions by approximately one order of magnitude relative to using Euclidean distance alone.
Analytic expansions are introduced to address a limitation of the discretized control space used during forward search which leads to never reaching the exact continuous goal state. To improve both goal accuracy and search efficiency,

the planner incorporates analytic expansion based on the Reed-Shepp motion model. On the base search process, tree nodes are expanded by simulating the vehicle's kinematic model using a sampled control input for a short time interval, determined by the discretization resolution. On addition to these standard expansion, selected nodes are augmented with an additional child generated by computing an optimal Reed-Shepp path from the current state directly to the goal, assuming no obstacle.
Since most analytic paths computed far from the goal are typically invalid due to obstacles, applying the Reed-Shepp expansion at every node would be computationally inefficient, for this reason it is applied selectively, at a rate of one out of every N nodes, where N decreases as the heuristic cost-to-goal decreases, increasing the frequency of analytic expansions as the search approaches the goal.
The path goal is defined using a potential function referred to as the **Voronoi Field**, designed to balance path length against obstacle clearance. The field is formulated as:

$$\rho_V(x,y) = \begin{cases} \left(\frac{\alpha}{\alpha+d_O(x,y)}\right)\left(\frac{d_V(x,y)}{d_O(x,y)+d_V(x,y)}\right)\frac{(d_O-d_O^{max})^2}{(d_O^{max})^2}, & d_O \leq d_O^{max} \\ 0, & d_O > d_O^{max} \end{cases} \tag{1.23}$$

- d_O and d_V denote the distances to the nearest obstacle and to the nearest edge of the Generalized Vonoroi Diagram, respectively.
- $\alpha 0$ and $d_O^{max} 0$ regulate the decay rate and maximum influence range of the potential.
- the fiald is defined only for $d_O \leq d_O^{max}$, otherwise $\rho_V(x,y) = 0$.

The potential field had the following properties:

1. it evaluates to zero when $d_O \geq d_O^{max}$
2. it is continuous and bounded in the interval $[0,1]$, since d_O and d_V can not vanish simultaneously
3. it reaches its maximum exclusively at obstacle locations
4. it reaches its minimum on the edges of the GVD

An advantages of the Vonoroi Field compared to standard potential fields lies in its normalization by available clearance: field intensity scales relative to the width of free space. This property prevents narrow passages from being saturated by high potential values, preserving navigability even in confined regions. Paths generated by hybrid-state A* are typically feasible but not yet optimal. Although they are drivable, they often contain unnecessary steering variations and irregular curvature. For this reason, the hybrid-A* output undergoes a two-stage optimization process:

1. First stage is a formulation of a non-linear optimization problem over the path vertices to reduce the path length and increase smoothness

2. Second stage performs non-parametric interpolation by applying an additional conjugate-gradient optimization over a higher-resolution discretization of the path

Given a path represented as a sequence of 2D vertices, each vertex is associated with the nearest obstacle location, the displacement to the previous vertex and the change in tangent direction between adjacent segments. These quantities are used to define a multi-term objective function that balances obstacle avoidance, vehicle curvature limits and geometric smoothness.
The cost function includes four main components:

$$w_\rho \sum_{i=1}^{N} \rho_V(x_i, y_i) + w_o \sum_{i=1}^{N} \sigma_o(\|x_i - o_i\| - d_{\max}) + w_\kappa \sum_{i=1}^{N-1} \sigma_\kappa \left(\frac{\Delta\phi_i}{\|\Delta x_i\|} - \kappa_{\max} \right) + w_s \sum_{i=1}^{N-1} \|\Delta x_{i+1} - \Delta x_i\|^2 \tag{1.24}$$

1. $w_\rho \sum_{i=1}^{N} \rho_V(x_i, y_i)$ encourages distance from obstacle using the Vonoroi Field, supporting safe navigation in both narrow and wide passages.
2. $w_o \sum_{i=1}^{N} \sigma_o(\|x_i - o_i\| - d_{\max})$penalizes direct proximity or collision with obstacle.
3. $w_\kappa \sum_{i=1}^{N-1} \sigma_\kappa \left(\frac{\Delta\phi_i}{\|\Delta x_i\|} - \kappa_{\max} \right)$ constraints the curvature between path segments to respect the maximum steering limits of the vehicle, enforcing non-holonomic feasibility.
4. $w_s \sum_{i=1}^{N-1} \|\Delta x_{i+1} - \Delta x_i\|^2$ minimizes changes in consecutive displacement vectors to promote smoothness

To support optimization, the gradient of each term is computed analytically. The Vonoroi Field gradient depends on the relative position of each vertex to both the nearest obstacle and Vonoroi-edge points are updated to each iteration using a kd-tree structure. Collision penalties use a quadratic formulation when a vertex falls within a safety threshold around an obstacle. Curvature constraints depends on three consecutive vertices, requiring derivatives with respect to each of them. The curvature variation is expressed using angular deviation between path segments, with derivatives constructed through orthogonal vector projections.
After optimization, the results is a significantly smoother trajectory compared to the original A* output. However, the path remains piecewise linear, with vertex spacing on the order of several tens of centimeters, which can still results in abrupt steering commands on real vehicles. To further improve motion quality, the methods applies a non-parametric interpolation stage. Instead of relying on spline-based fitting, which can amplify noise and create oscillations, the path is densely resampled by inserting intermediate vertices while keeping the original ones fixed. A final conjugate-gradient smoothing steps is then applied to minimize curvatures across the un-sampled path.
The **State Lattice** in based on a discrete state space structured through a pregenerated set of kinematically feasible motion primitives. Unlike Hybrid-A*, the lattice planner used a fixed library of motion primitives aligned to a discretized grid of position and heading. Its performance is comparable to Hybrid-A*, with

planning times ranging from 10 to 350 ms depending on environment complexity, and as low as 1–3 ms with cached heuristics. Similar to the previous planner, the state lattice incorporates analytic expansion and cost-aware heuristic search to reduce unnecessary expansion and to improve the path quality. The planner produces high-quality paths without requiring complex smoothing techniques. It differs from the Hybrid-A* due to a lightweight descent smoother in order to minimize curvature and improve continuity while preserving adherence to environmental constraints. The SmacPlannerLattice used in Nav2 refers to [17], which is not implemented precisely, due to the fact the control set based on [18].

1.4.2.2 NavFn planner

The NavFn planner, originally based on the Global Dynamic approach [19], remained the sole path planning algorithm available in ROS until recently and it continues to be the default one. It operates on maps represented as discrete rectangular grids, stored in two-dimensional integer matrices, where each cell holds a traversal cost. The cost values range from 50 (COST_NEUTRAL) to 254 (COST_OBS), with 254 indicating a lethal or impassible obstacle. Given such a grid map, along with the specified start and goal positions and a defined step length, the planner generates a path that meets the navigation requirements.
From the *navfn.cpp* source file in the Navigation2 repository [20] it is possible to analyze the NavFn algorithm:

1. In the first phase, referred to as *calCPotential*, the algorithm computes an auxiliary navigation function in the form of a potential field across the map, assigning lower potential values to cells closer to the goal.

2. In the second phase, called *calcPath*, this potential field is used to determine the path by tracing the steepest descent from the start position to the goal

The navigation function is calculated using either Dijkstra's algorithm or A*, based on user preference. In both cases, the computation initiates from the goal and progress outward until it reaches the starting position.
The pseudocode for computing the navigation potential begins by initializing the potential matrix with infinite values. A threshold is set as the sum of the Euclidean distance between the start and the goal and the COST_OBS value. Three separate FIFO queues (*curr*, *next* and *over*) are maintained to organize cells based on their estimated cost.
The algorithm starts by assigning zero potential to the cell and unqueuing its four cardinal neighbors into *curr*. During each iteration, up to a maximum number of cycles, cells from *curr* are processed. For each cell p, if the associated cost below is below COST_OBS, the algorithm computes a propagates potential v from neighboring cells. If v improves upon the current potential at p, it updates *navfn[p]* and consider the cardinal neighbors of p. Each neighbors p' is evaluated using a cost estimation that combines the new potential v, the distance from p to the start and a scaled cell cost. If the estimated cost is below the threshold p', is added to the *next* queue, otherwise to the *over* queue. Once all elements of *curr* have been processed, the queue are rotated and *next* becomes *curr* and if *curr* is empty, the threshold is increased and over is reinserted as

the new *curr*. This process repeats until the start position is assigned a finite potential or the maximum number of iteration is reached.
The A* variant in NavFn does not utilize a standard priority queue and it organizes cells across multiple FIFO queues based on whether their estimated total path cost falls below or above a dynamical adjusted threshold. The *curr*, *next* and *over* queue serve to group nodes for expansion and the threshold mechanism controls when higher-cost nodes are considered.

Algorithm 1 NavFn Algorithm (Potential Calculation)

```
1: function navFn(map : matrix⟨int⟩, start, goal : pose, potIterMax, pathIterMax : int) → list⟨pose⟩
2:     navfn ← calcPotential(map, start, goal, potIterMax)
3:     if navfn[start] < ∞ then
4:         return calcPath(map, start, goal, navfn, pathIterMax)
5:     else
6:         return no path
7:     end if
8: end function
9: function calcPotential(map : matrix⟨int⟩, start, goal : pose, ncycles : int) → matrix⟨float⟩
10:    threshold ← d²(start, goal) + COST_OBS
11:    navfn ← matrix of same size as map filled with ∞
12:    curr, next, over ← empty queues of poses
13:    navfn[goal] ← 0
14:    append the cardinal neighbors (N, S, E, W) of goal to curr
15:    repeatup to ncycles times
16:        while curr is not empty do
17:            p ← curr.extractFirst()
18:            if map[p] < COST_OBS then                    ▷ skip obstacles
19:                v ← propagated potential from neighbors
20:                if v < navfn[p] then
21:                    navfn[p] ← v
22:                    for all cardinal neighbors p′ of p do
23:                        if v + d²(p, start) + 1/√2 · map[p′] < navfn[p′] then
24:                            if v + d²(p, start) < threshold then
25:                                append p′ to next
26:                            else
27:                                append p′ to over
28:                            end if
29:                        end if
30:                    end for
31:                end if
32:            end if
33:        end while
34:        swap curr with next
35:        if curr is empty then
36:            threshold ← threshold + 2 · COST_NEUTRAL
37:            swap curr with over
38:        end if
39:    until navfn[start] < ∞
40:    return navfn
41: end function
```

Once the potential map has been computed as in 1, the *calcPath* function extracts a feasible path by following the descending gradient of the potential values from the start to the goal (cell with potential zero). in each iteration the algorithm either:

1. moves the neighboring cell with the lowest potential value.

2. advances a fractional distance in the direction of the interpolated gradient, depending on the presence of nearby obstacles and the current behavior f the path.

This stepwise descent continues until either the goal is reached or a defined iteration limit is exceeded. the path is built incrementally and if the algorithm reaches a location with potential below the cost COST_NEUTRAL value, it concludes successfully by appending the goal to the path.
To address potential oscillations the implementation checks whether the current location coincides with the position two steps prior. If such pattern is detected or if a neighbor has an infinite potential, the algorithm resets to the neighbor with the lowest available potential. However, this comparison uses strict floating-point equality and may miss longer or subtler cycles, which can result in incomplete path-finding and unnecessary path extensions before oscillations is recognized. The step size is constrained so that offset values for each axis do not exceed one cell in magnitude. Although the navigation function is designed to be free of local minima, these oscillation cases represent exceptions where the algorithm must apply additional logic to recover.

Algorithm 2 NavFn Algorithm (Path Extraction)

function CALCPATH(map : matrix⟨int⟩, start, goal : pose, navfn : matrix⟨float⟩, ncycles : int) → list⟨pose⟩
  path ← empty list of poses
  p ← start ▷ current position (integer)
  dx, dy : float ← 0.0 ▷ current offset (fractional)
  for ncycles rounds **do**
    if navfn[p + rounded (dx, dy)] < COST_NEUTRAL **then**
      append goal to path
      return path
    end if
    append p + (dx, dy) to path
    oscillation ← —path— > 2 ∧ path[—path—−1] = path[—path—−3]
    if there exists neighbor c of p (including diagonals) such that navfn[c] = ∞ ∨ oscillation **then**
      c ← neighbor cell with lowest potential
      p, dx, dy ← c, 0, 0
      if navfn[c] = ∞ **then**
        return no path
      end if
    else
      $\vec{g} \leftarrow -\nabla$navfn[p + (dx, dy)] ▷ interpolated gradient
      if $\vec{g} = (0, 0)$ **then**
        return no path
      end if
      dx, dy ← (dx, dy) + `stepLength` $\cdot \frac{\vec{g}}{\|\vec{g}\|}$
      normalize p, dx, dy so that |dx|, |dy| ≤ 1
    end if
  end for
  return no path
end function

1.5 Controllers

1.5.1 MPPI

The MPPI controller is a local trajectory planner that implements the Model Predictive Path Integral algorithm, which combines the best of the hierarchical and optimal control paradigms. The algorithm is based on a stochastic optimal control framework using fundamental relationships between the information theoretic notions of free energy and relative entropy [21].

1.5.1.1 Stochastic trajectory optimization

The path integral control method provides a robust mathematical methodology for creating optimal control algorithms based on the stochastic sampling of trajectories. The advantages which this method provides are the flexibility and robustness during the computation of the system dynamics and designing cost functions due to the independence from them. To derive a path integral control

generally is obtained using an exponential transformation of the value function of the optimal control problem and it requires that the noise in the system only affects directly actuated states. Through this assumption it is possible to transform the stochastic Hamilton-Jacobi-Bellman equation into a linear partial differential equation which can then be transformed into a path integral using the Feynman-Kac lemma. The resulting path integral represent 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. The resulting alternative for the formulation offers two advantages:

- the relaxed relationship between noise and control authority allows the noise in both directly and indirectly actuated states.
- it provide a formula for the optimal controls over the entire time horizon (the traditional formulation allows only the initial time for the time horizon).

It is then possible to solve the optimal control problem by minimizing the relative entropy between the optimal distribution and the distribution induced by the controller.
The stochastic dynamical systems considered is the one with the state and controls at time t denoted by $x_t \in \mathbb{R}^n$ and $u_t \in \mathbb{R}^m$, respectively, these dynamics are affected by a Brownian motion $dw \in \mathbb{R}^p$. We define $u(\cdot) : [t_0, T] \to \mathbb{R}^m$ as the function that maps each time instant to a control input, and $\tau : [t_0, T] \to \mathbb{R}^n$ as the corresponding state trajectory of the system. In the classical stochastic optimal control framework, the goal is to determine a control sequence $u(\cdot)$ that minimizes an expected cost functional of the form:

$$u^*(\cdot) = \arg\min_{u(\cdot)} \mathbb{E}_Q \left[\phi(x_T, T) + \int_{t_0}^{T} \mathcal{L}(x_t, u_t, t)\, dt \right]. \tag{1.25}$$

The expectation is evaluated with respect to the dynamics: $dx = F(x_t, u_t, t)dt + B(x_t, t)dw$ and the cost function $\mathcal{L}(x_t, u_t, t)$ is the composition of an arbitrary state-dependent term and a quadratic control loss:

$$\mathcal{L}(x_t, u_t, t) = q(x_t, t_t, t) + \frac{1}{2} u_t^T R(x_t, t) u_t \tag{1.26}$$

While the dynamics are defined in controls as:

$$F(x_t, u_t, t) = f(x_t, t) + G(x_t, t)u_t \tag{1.27}$$

In accordance with [22], an interpretation of Path Integral Control is provided, based on the concepts of free energy $\mathcal{F}$ and relative entropy $\mathcal{D}_{KL}$. This interpretation is formalized by the following equality:

$$-\lambda \mathcal{F}(\mathcal{S}(\tau)) = \inf_{\mathcal{Q}} \left[\mathbb{E}_{\mathcal{Q}}[\mathcal{S}(\tau)] + \lambda \mathcal{D}_{KL}(\mathcal{Q} \| \mathcal{P}) \right] \tag{1.28}$$

Where $\lambda \in \mathbb{R}^+$ is the weighting parameter, and $\mathcal{S}(\tau)$ is the state-dependent cost-to-go term, defined as $\mathcal{S}(\tau) = \int_{t_0}^{\mathcal{T}} q(x_t, t)\, dt$. The function $\mathcal{F}(\mathcal{S}(\tau))$ denotes the free energy and is defined as $\log(\mathbb{E}_{\mathcal{P}}[\exp(-\frac{1}{\lambda}\mathcal{S}(\tau))])$. The measure $\mathcal{P}$ is the

probability measure over the path space induced by the uncontrolled stochastic dynamics:

$$dx(t,t) = f(x_t,t)dt + B(x_t,t)dw, \tag{1.29}$$

And $\mathcal{Q}$ is any probability measure over the path space that is absolutely continuous with respect to $\mathcal{P}$. The Kullback-Leibler divergence (relative entropy) $\mathcal{D}_{KL}(\mathcal{Q}\|\mathcal{P})$ measures the difference between the distributions and is defined as $\mathbb{E}_{\mathcal{Q}}[\log(\frac{d\mathcal{Q}}{d\mathcal{P}})]$.
The controlled dynamics induce a different probability measure, $\mathcal{Q}(u)$, on the space of trajectories. The relative entropy $\mathcal{D}_{KL}(\mathcal{Q}(u)\|\mathcal{P})$ between the uncontrolled distribution $\mathcal{P}$ and the controlled distribution $\mathcal{Q}(u)$ can be computed by applying Girsanov's theorem, yielding:

$$\mathcal{D}_{KL}(\mathcal{Q}(u)\|\mathcal{P}) = \frac{1}{2}\int_{t_0}^{\mathcal{T}} u_t^T G(x_t,t)^T \Sigma(x_t,t)^{-1} G(x_t,t) u_t \, dt \tag{1.30}$$

Note that $\Sigma(x_t,t) = B(x_t,t)B(x_t,t)^T$. Assuming the control cost matrix takes the form:

$$R(x_t,t) = \lambda G(x_t,t)^T \Sigma(x_t,t)^{-1} G(x_t,t) \tag{1.31}$$

We establish the following correspondence with the right-hand sides of (1.26) and (1.28):

$$\mathbb{E}_{\mathcal{Q}(u)}[\mathcal{S}(\tau)] + \lambda \mathcal{D}_{KL}(\mathcal{Q}\|\mathcal{P}) = \mathbb{E}_{\mathcal{Q}(u)}[\mathcal{S}(\tau)] + \frac{1}{2}\int_{t_0}^{\mathcal{T}} u_t^T R(x_t,t) u_t \, dt \tag{1.32}$$

Furthermore, [22] explicitly derives the form of the optimal probability measure $\mathcal{Q}^*$ in terms of the Radon-Nikodym derivative with respect to the uncontrolled dynamics $\mathcal{P}$, which takes the form:

$$\frac{d\mathcal{Q}^*}{d\mathcal{P}} = \frac{\exp\left(-\frac{1}{\lambda}\mathcal{S}(\tau)\right)}{\mathbb{E}_{\mathcal{P}}\left[\exp\left(-\frac{1}{\lambda}\mathcal{S}(\tau)\right)\right]} \tag{1.33}$$

The previous work [22] does not provide a method for computing a control law independent of the Hamilton-Jacobi-Bellman equation, which is proposed in [21]. Rather than directly solving the optimal control problem by computing the solution to the stochastic HJB equation, it is possible to track the minimization problem by optimizing the probability distribution generated by the controller $\mathcal{Q}(u)$ as close as possible to the optimal probability measured $\mathcal{Q}^*$, given by the Radon-Nikodym derivative of (1.33).
The optimization problem is described by the following minimization problem:

$$u^*(\cdot) = \arg\min_{u(\cdot)} D_{KL}\big(\mathcal{Q}^* \,\|\, \mathcal{Q}(u)\big) \tag{1.34}$$

The advantages of this formulation are attributed to the optimal probability measure $\mathcal{Q}^*$, which resembles a soft-max function with temperature λ. If a trajectory's cost is low, then $\frac{d\mathcal{Q}^*}{d\mathcal{P}}$ will have a high value, and as a result, if the trajectory is sampled from $\mathcal{Q}^*$, it will likely have a low cost and if the tracking of $\mathcal{Q}(u)$ matches as close as possible to $\mathcal{Q}^*$, it will result in a low-cost trajectory.

The relative entropy terms in equation (1.34) are the following:

$$D_{KL}\big(\mathcal{Q}^* \parallel \mathcal{Q}(u)\big) = \mathbb{E}_{\mathcal{Q}^*}\left[\log\left(\frac{d\mathcal{Q}^*}{d\mathcal{Q}(u)}\right)\right] \tag{1.35}$$

To optimize the function, an expression for the Radon-Nikodym derivative of (1.33) must be found and through the chain rule property of Radon-Nikodym derivatives:

$$\frac{d\mathcal{Q}^*}{d\mathcal{Q}(u)} = \frac{d\mathcal{Q}^*}{d\mathcal{P}}\frac{d\mathcal{P}}{d\mathcal{Q}(u)} \tag{1.36}$$

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

$$\frac{dP}{dQ(u)} = exp(D(\tau, u(\cdot))) \tag{1.37}$$

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

$$\begin{aligned} D(\tau, u(\cdot)) = &- \int_0^T u_t^T G(x_t,t)^T \Sigma(x_t,t)^{-1} B(x_t,t)\, dw(0) \\ &+ \frac{1}{2}\int_0^T u_t^T G(x_t,t)^T \Sigma(x_t,t)^{-1} G(x_t,t) u_t\, dt \end{aligned} \tag{1.38}$$

The term in (1.38) $dw^{(0)}$ represents a Brownian motion with respect to P.
The combination of equation (1.38) with (1.33) leads to the following.

$$D_{KL}\big(\mathcal{Q}^* \parallel \mathcal{Q}(u)\big) = \mathbb{E}_{\mathcal{Q}^*}\left[\log\left(\frac{\exp\big(-\frac{1}{\lambda}S(\tau)\big)\cdot \exp(D(\tau, u(\cdot)))}{\mathbb{E}_{\mathcal{P}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\big]}\right)\right] \tag{1.39}$$

This equation can also be written as:

$$D_{KL}\big(\mathcal{Q}^* \parallel \mathcal{Q}(u)\big) = \mathbb{E}_{\mathcal{Q}^*}\left[-\frac{S(\tau)}{\lambda} + D(\tau, u(\cdot)) - \log \mathbb{E}_{\mathcal{P}}\left(\exp\left(-\frac{S(\tau)}{\lambda}\right)\right)\right] \tag{1.40}$$

Since the optimization problem is to minimize $D_{KL}\big(Q^* \parallel Q(u)\big)$ with respect to $u(\cdot)$ and since $S(\tau)$ is independent of the control $u(\cdot)$, the equation can be formulated as follows.

$$\arg\min_{u(\cdot)} D_{KL}\big(\mathcal{Q}^* \parallel \mathcal{Q}(u)\big) = \arg\min_{u(\cdot)} \mathbb{E}_{\mathcal{Q}^*}[D(\tau, u(\cdot))] \tag{1.41}$$

The aim is to determine a function $u^*(\cdot)$ that minimizes the equation (1.41) and since control is typically applied at discrete time intervals, it is viable to consider only a step function:

$$u_t = \begin{cases} \vdots \\ u_j, \quad \text{if } j\Delta t \le t \le (j+1)\Delta t \\ \vdots \end{cases} \tag{1.42}$$

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

$$D(\tau, u(\cdot)) = -\sum_{j=0}^{N} u_j^{\mathrm{T}} \int_{t_j}^{t_{j+1}} G(x_t, t)\, dw_t^{(0)} + \frac{1}{2} u_j^{\mathrm{T}} \int_{t_j}^{t_{j+1}} H(x_t, t)\, dt\, u_j. \quad (1.43)$$

- $G(x,t) = G(x,t)^T \Sigma(x,t)^{-1} B(x,t)$
- $H(x,t) = G(x,t)^T \Sigma(x,t)^{-1} G(x,t)$
- $N = \frac{T}{\Delta t}$

The independency of u_j to the trajectory, through the expectation operator, becomes:

$$\mathbb{E}_Q[D(\tau, u(\cdot))] = -\sum_{j=0}^{N} u_j^{\mathrm{T}} \mathbb{E}_Q\left[\int_{t_j}^{t_{j+1}} G(x_t, t)\, dw_t^{(0)}\right] + \sum_{j=0}^{N} \frac{1}{2} u_j^{\mathrm{T}} \mathbb{E}_Q\left[\int_{t_j}^{t_{j+1}} H(x_t, t)\, dt\right] u_j. \quad (1.44)$$

It is possible to notice how the formulation used is convex with respect to each u_j. The gradient of the equation is set to zero to find the optimal u_j.

$$u_j^* = \mathbb{E}_Q\left[\int_{t_j}^{t_{j+1}} H(x_t, t)\, dt\right]^{-1} \mathbb{E}_Q\left[\int_{t_j}^{t_{j+1}} G(x_t, t)\, dw_t^{(0)}\right]. \quad (1.45)$$

The equation (1.45) can be approximated considering small Δt.

$$\begin{aligned} \int_{t_j}^{t_{j+1}} H(x_t, t)\, dt &\approx H(x_{t_j}, t_j)\, \Delta t, \\ \int_{t_j}^{t_{j+1}} G(x_t, t)\, dw_t^{(0)} &\approx G(x_{t_j}, t_j) \int_{t_j}^{t_{j+1}} dw_t^{(0)}. \end{aligned} \quad (1.46)$$

This leads to:

$$u_j^* = \frac{1}{\Delta t} \mathbb{E}_Q\big[H(x_{t_j}, t_j)\big]^{-1} \mathbb{E}_Q\left[G(x_{t_j}, t_j) \int_{t_j}^{t_{j+1}} dw_t^{(0)}\right]. \quad (1.47)$$

Since sampling from Q^* is not feasible, it is necessary to change the expectation to be with respect to the uncontrolled dynamics $\mathbb{P}$ and to approximate the controls, the trajectory can be sampled from that.

$$u_j^* = \frac{1}{\Delta t} \mathbb{E}_P\left[\frac{\exp\big(-\frac{1}{\lambda} S(\tau)\big)\, H(x_{t_j}, t_j)}{\mathbb{E}_P\big[\exp\big(-\frac{1}{\lambda} S(\tau)\big)\big]}\right]^{-1} \mathbb{E}_P\left[\frac{\exp\big(-\frac{1}{\lambda} S(\tau)\big)\, G(x_{t_j}, t_j) \int_{t_j}^{t_{j+1}} dw_t^{(0)}}{\mathbb{E}_P\big[\exp\big(-\frac{1}{\lambda} S(\tau)\big)\big]}\right]. \quad (1.48)$$

1.5.1.2 State independent control matrix

Beside the assumption that no correlations between noise in both directly and indirectly actuated states are present and the diffusion for the directly one is state independent the diffusion matrix and the control matric have the form:

$$B(x_t) = \begin{pmatrix} B_a(x_t) & 0 \\ 0 & B_c \end{pmatrix}, \qquad G = \begin{pmatrix} 0 \\ G_c \end{pmatrix}. \tag{1.49}$$

Consequently, the covariance matrix is:

$$\Sigma(x) = \begin{pmatrix} B_a(x)B_a(x)^{\mathrm{T}} & 0 \\ 0 & B_c B_c^{\mathrm{T}} \end{pmatrix}. \tag{1.50}$$

Then the terms $\mathcal{H}(x_{t_j})$ and $\mathcal{G}(x_t)$ are longer exhibit state dependence and reduce to:

$$\mathcal{H} = G_c^{\mathrm{T}}(B_c B_c^{\mathrm{T}})^{-1} G_c, \qquad \mathcal{G} = G_c^{\mathrm{T}}(B_c B_c^{\mathrm{T}})^{-1} B_c. \tag{1.51}$$

From the expectation it is possible to extract these matrix, resulting in:

$$u_j^* = \frac{1}{\Delta t}\,\mathcal{H}^{-1}\mathcal{G}\left(\mathbb{E}_P\left[\frac{\displaystyle\int_{t_j}^{t_{j+1}} \exp\!\left(-\frac{1}{\lambda}S(\tau)\right)\, dw_t^{(0)}}{\mathbb{E}_P\left[\exp\!\left(-\frac{1}{\lambda}S(\tau)\right)\right]}\right]\right). \tag{1.52}$$

1.5.1.3 Numerical approximation

Once get the equation (1.52), a numerical approximation is needed. However, two distinct challenges must be adressed:

1. Reformulating the equation for discrete-time sampling
2. Importance sampling is needed for the equation due to the fact the expectation is with respect to the uncontrolled dynamics $\mathbb{P}$ which can lead to very inefficient distribution to sample from.

In the discrete-time the dynamics of the system becomes:

$$dx_t = f(x_{t_j}, t_j)\,\Delta t + G(x_{t_j}, t_j)u_j\,\Delta t + B(x_{t_j}, t_j)\,\varepsilon_j\sqrt{\Delta t}, \tag{1.53}$$

Where ε_j denotes a random vector whose components are standard normal variables. The equation (1.52) results now in:

$$u_j^* = \frac{1}{\Delta t}\,\mathcal{H}^{-1}\mathcal{G}\left(\mathbb{E}_P\left[\frac{\exp\!\left(-\frac{1}{\lambda}S(\tau)\right)\varepsilon_j\sqrt{\Delta t}}{\mathbb{E}_P\left[\exp\!\left(-\frac{1}{\lambda}S(\tau)\right)\right]}\right]\right). \tag{1.54}$$

Under this formulation, the probability distribution p corresponds to the dynamics uncontrolled dynamics. Sampling from another probability distribution different form p, as q_u^*, the system's dynamics can be expressed as:

$$dx_t = f(x_{t_j}, t_j)\,\Delta t + G(x_{t_j}, t_j)u_t\,\Delta t + B_E(x_{t_j}, t_j)\varepsilon_j\sqrt{\Delta t}. \tag{1.55}$$

$B_E(x_t)$ is the new diffusion matrix, defined as:

$$B_E(x_t) = \begin{pmatrix} B_a(x_t) & 0 \\ 0 & \nu B_c \end{pmatrix}, \qquad \nu \geq 1. \tag{1.56}$$

The designer can select:

- The initial controls around which sampling is centered
- The magnitude of the exploration variance (ν)

The choice of sampling from q_u^* is necessary to compute the likelihood ratio between the two distributions [23]. Incorporating the likelihood ration corresponds to modifying the running cost terms from $q(x_t, u_t, t)$ to:

$$\begin{aligned}\tilde{q}(x_t, u_t, \varepsilon_t, t) = q(x_t, t) + \frac{1}{2} u_t^{\mathrm{T}} R u_t + \lambda u_t^{\mathrm{T}} G_c \frac{\varepsilon_t}{\sqrt{\Delta t}} \\ + \frac{\lambda}{2}(1 - \nu^{-1}) \frac{\varepsilon_t^{\mathrm{T}}}{\sqrt{\Delta t}} B_c^{\mathrm{T}} (B_c B_c^{\mathrm{T}})^{-1} B_c \frac{\varepsilon_t}{\sqrt{\Delta t}}\end{aligned} \tag{1.57}$$

- $\frac{1}{2} u_t^{\mathrm{T}} R u_t$ and $\lambda u_t^{\mathrm{T}} G_c \frac{\varepsilon_t}{\sqrt{\Delta t}}$ are penalties for shifting the mean of the exploration away from zero.
- $\frac{\lambda}{2}(1-\nu^{-1}) \frac{\varepsilon_t^{\mathrm{T}}}{\sqrt{\Delta t}} B_c^{\mathrm{T}} (B_c B_c^{\mathrm{T}})^{-1} B_c \frac{\varepsilon_t}{\sqrt{\Delta t}}$ is a term that penalizes sampling from an overly aggressive variance.
- The term $B_c^{\mathrm{T}} (B_c B_c^{\mathrm{T}})^{-1} B_c \frac{\varepsilon_t}{\sqrt{\Delta t}}$ is the effective change in control input caused by noise.

Since sampling comes from a distribution with non-zero control input, the update law (1.52) changes.
Denoting $\tilde{S}(\tau)$ as:

$$\tilde{S}(\tau) = \phi(x_T, T) + \sum_{j=0}^{N} \tilde{q}(x_t, u_t, \varepsilon_t, t)\, \Delta t. \tag{1.58}$$

We can obtain the iterative update rule:

$$u_j^* = u_j + \mathcal{H}^{-1} \mathcal{G} \left(\mathbb{E}_{q_u^*} \left[\frac{\exp\left(-\frac{1}{\lambda} \tilde{S}(\tau)\right) \varepsilon_j / \sqrt{\Delta t}}{\mathbb{E}_{q_u^*}\left[\exp\left(-\frac{1}{\lambda} \tilde{S}(\tau)\right)\right]} \right] \right). \tag{1.59}$$

The expectation terms in equation (1.59) is estimated as:

$$\sum_{k=1}^{K} \frac{\exp\left(-\frac{1}{\lambda} \tilde{S}(\tau_k)\right) \varepsilon_{j,k} / \sqrt{\Delta t}}{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda} \tilde{S}(\tau_k)\right)}. \tag{1.60}$$

where each trajectory τ_k is drawn from the sampling dynamics (1.55). This iterative update law can be interpreted as computing the new control derived from a reward-weighted average over trajectories.

1.5.1.4 Model predictive control algorithm

The final equation (1.59) can be applied in a model predictive control setting since provides an iterative update law. In this way the optimization is performed dynamically: the trajectory is optimized, a single control input is executed and then the optimization is repeated.

The derivation of the path integral control inherently yields a formula for optimizing the entire sequence of control inputs over the prediction horizon, a key distinction from methods that only optimize the action for the current time instance. This feature is leveraged to implement a highly effective warm-start strategy. In fact the un-executed remainder of the optimal control sequence computed at the previous time step is re-used to warm start the optimization. This mechanism is crucial for the performance of the algorithm since for complex system operating at a reasonable control frequency it is possible to perform only small number of iterations every timestep. One of the advantages of the cited and described implementation is that the bulk of the computation used in the update rule can be performed in parallel. The parallelization implies parallel trajectory sampling on a GPU is an efficient operation, allowing to perform and handle thousands of trajectories from complex non-linear dynamics.
The MPPI algorithm is given in Algorithm 3 and to run this algorithm, the simplest method is to parallelize the sampling for-loop by employing one thread per sample, in order to provide a massive uplift in the time required to perform the algorithm.

Algorithm 3 MPPI optimization sequence

Given:
K: Number of samples
N: Number of time-steps
$(u_0, u_1, \ldots, u_{N-1})$: Initial control sequence
$\Delta t, x_0, f, G, B, B_E$: System/sampling dynamics
ϕ, q, R, λ: Cost parameters
u_{init}: Value to initialize new controls
while task not completed **do**
  $x = x_0$
  for $k = 0$ **to** $K - 1$ **do**
    for $i = 0$ **to** $N - 1$ **do**
      $x_{i+1} = x_i + (f + Gu_i)\Delta t + B_E \varepsilon_{k,i} \sqrt{\Delta t}$
      $\tilde{S}(\tau_k) = S(\tau_k) + \tilde{q}(x_i, u_i, \varepsilon_{k,i}, t_i)$
    end for
  end for
  for $i = 0$ **to** $N - 1$ **do**
    $u_i = u_i + \mathcal{H}^{-1}\mathcal{G}\left(\frac{\sum_k \exp(-\frac{1}{\lambda}\tilde{S}(\tau_k))\,\varepsilon_{k,i}/\sqrt{\Delta t}}{\sum_k \exp(-\frac{1}{\lambda}\tilde{S}(\tau_k))}\right)$
  end for
  Send (u_0) to actuators
  for $i = 0$ **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

1.5.1.5 Comparison Between Classical MPC and MPPI

Although Model Predictive Control (MPC) represents a general and well-established framework for optimal control, classical MPC formulations typically rely on the solution of a deterministic or stochastic optimization problem using gradient-based or sequential quadratic programming methods. In most implementations, only the first control input of the optimized sequence is applied to the system, following the receding horizon principle, and the optimization problem is repeatedly solved at each control cycle, often requiring local linearization of the system dynamics around a nominal trajectory [6].

In contrast, the Model Predictive Path Integral (MPPI) controller formulates the optimal control problem as a stochastic optimization over entire control sequences. The solution is derived from a path integral formulation of stochastic optimal control, where the optimal control sequence is computed as a weighted average of sampled trajectories, with weights obtained through an exponential transformation of the trajectory costs. This approach explicitly provides control inputs over the whole prediction horizon, rather than only for the first timestep. A key difference with respect to classical MPC lies in the absence of local convexity or differentiability requirements. MPPI does not require explicit gradient information nor system linearization, which makes it particularly suitable for highly nonlinear systems and non-smooth cost functions. Furthermore, the stochastic formulation relaxes the strict relationship between control authority and noise typically assumed in classical path integral control formulations.

1.5.1.6 Suitability of MPPI for Autonomous Navigation

The properties of MPPI make it particularly well-suited for autonomous navigation tasks, where systems operate in dynamic and uncertain environments. In classical hierarchical navigation architectures, global path planning and local control are typically decoupled, which can result in conservative behaviors, as the local controller is required to track reference trajectories that may not fully account for the vehicle's dynamic constraints.

MPPI addresses this limitation by directly reasoning over dynamically feasible trajectories generated through stochastic sampling. System constraints and navigation objectives, such as obstacle avoidance, smoothness, and comfort, can be encoded directly in the cost function, allowing the controller to implicitly balance these objectives during trajectory evaluation.

Moreover, the stochastic nature of MPPI allows the controller to explicitly handle uncertainty and modeling errors by propagating sampled control perturbations through the system dynamics. This capability is particularly advantageous in navigation scenarios involving dynamic obstacles or social environments, where prediction uncertainty plays a crucial role.

Finally, the ability of MPPI to generate novel control behaviors online, without relying on precomputed trajectories or extensive offline optimization, enables reactive and adaptive navigation strategies. These characteristics make MPPI a suitable control framework for modern autonomous navigation systems, including those based on ROS2 Navigation2, where robustness, real-time performance, and flexibility in cost design are fundamental requirements.

1.5.2 DWB

In the ROS2 Navigation2 framework, the Dynamic Window Approach is implemented through the *Dynamic Window Based* (DWB) local planner. While DWB is conceptually rooted in the original DWA formulation, it introduces a more modular and extensible architecture that significantly improves flexibility and adaptability for autonomous navigation tasks.

1.5.2.1 Difference between DWA and DWB

The fundamental similarity between DWA and DWB lies in the search strategy: both planners operate directly in the velocity space (ν, ω) and evaluate dynamically feasible velocity commands over a short prediction horizon, following the receding horizon principle. However, unlike the classical DWA formulation, DWB decouples trajectory generation from trajectory evaluation. Velocity samples are first generated according to kinematic and dynamic constraints, and then evaluated through a set of independent cost critics.
In DWB, the objective function is no longer expressed as a fixed weighted sum of predefined terms, as in Equation (1.9). Instead, the overall trajectory score is computed as the weighted combination of multiple cost critics, each representing a specific navigation objective, such as obstacle avoidance, goal alignment, path following, or motion smoothness. This modular structure allows individual critics to be enabled, disabled, or reweighted without modifying the core planning algorithm.
Another key difference concerns trajectory evaluation. While classical DWA evaluates only instantaneous circular arcs corresponding to constant velocity commands, DWB explicitly simulates short-horizon trajectories by forward propagating the robot state over multiple timesteps. This enables a more accurate assessment of future robot behavior, especially in cluttered environments or near obstacles, and reduces the likelihood of oscillatory or overly reactive motions.
From a practical standpoint, DWB provides improved tunability and maintainability within large-scale navigation systems. The critic-based architecture allows the planner to be adapted to different robot platforms and operational contexts by adjusting cost functions rather than altering the planner logic itself. This design choice aligns well with modern autonomous navigation requirements, where robots must operate robustly across diverse environments and task constraints.
Overall, while DWA represents a foundational approach to local motion planning, DWB extends its principles into a more general and flexible framework. These enhancements make DWB better suited for autonomous navigation in real-world scenarios, particularly when integrated into complex navigation stacks such as ROS 2 Navigation2, where modularity, extensibility, and robustness are essential.

Comfort-oriented critic and Benchmark

The Nav2 navigation framework offers a modular controller architecture, including the MPPI and DWB, allowing the implementation of user-defined critics to evaluate and rank candidate trajectories. A comfort-oriented critic has been developed, implemented and tested in order to improve the pre-existing controller of Nav2, such as the MPPI. The planners used are the *NavFn*, *SmacPlannerHybrid* and *SmacPlannerLattice* and their parameters have been maintained constant for all the simulations. After recording the ROS 2 bag files, the metrics were extracted to evaluate comfort during the simulations. Most of the default critics in Nav2 aim to optimize navigation success and do not explicitly model comfort-related dynamic quantities such as acceleration and over-acceleration.

2.1 Critic implementation

The ComfortMPPICritic is a trajectory scoring function designed to penalize trajectories that would result in uncomfortable motions, discouraging sudden or extreme maneuvers by the robot. High accelerations, abrupt braking, and jerky motions are well-known sources of discomfort for passengers. Therefore, the critic introduces cost penalties for motions that exceed predefined comfort thresholds for linear and angular velocity, acceleration, and over-acceleration, steering the controller toward smoother trajectories. The critic evaluates each sampled trajectory by analyzing its kinematic profile over the prediction horizon. Linear velocity and angular velocity are directly available from the MPPI internal state, while acceleration and over-acceleration are numerically estimated from the velocity signals. At each time step, the critic checks whether any of these quantities exceed the corresponding comfort thresholds and, if so, applies a penalty to the trajectory cost. Penalization is implemented using a hinge-quadratic loss function. For each kinematic quantity, the cost contribution is zero within the comfort band and grows quadratically once the threshold is exceeded. This design ensures that trajectories remain unaffected when operating within acceptable comfort limits, while increasingly penalizing uncomfortable motions. Acceleration and over-acceleration are estimated using a central finite-difference scheme applied to the velocity profiles. To reduce numerical noise and spurious oscillations, the resulting signals are clamped using saturation limits and then smoothed with an exponential moving average filter. This approach prevents the critic from reacting to high-frequency numerical artifacts while preserving physically meaningful peaks.
Velocity penalties are symmetric and applied only when the magnitude of linear or angular velocity exceeds the corresponding comfort threshold.

$$c_v \;=\; w_{\text{vel,lin}} \left(\max\{0,\; |v| - v_{\max}\}\right)^2 \tag{2.1}$$

$$c_w \;=\; w_{\text{vel,ang}} \left(\max\{0,\; |w| - w_{\max}\}\right)^2 \tag{2.2}$$

Acceleration penalties limit rapid changes of speed for both translation and rotation. A direction-dependent weights are applied due to the asymmetric human perception of comfort, which perceive abrupt braking as harsher. This allows to penalize hard stop more than equally strong speeding-up.
The structure for acceleration and over-acceleration penalty differs from the velocity term due to the asymmetric weights given to these dynamic quantities. The important design feature of this critic is the asymmetric treatment for positive and negative value of these terms. This separation reflect the fact that human perception of comfort is not symmetric for acceleration and braking [24]: hard braking is often felt as more uncomfortable or startling than equally strong acceleration, due to the way passengers experience forces; in fact is less "comfortable to be thrown forward against a seatbelt versus being pressed back into the seat.

$$c_{\dot{v}} = \begin{cases} w^{+}_{\text{acc,lin}} \left(\max\{0,\ |\dot{v}| - a_{\max}\}\right)^2, & \dot{v} \geq 0, \\ w^{-}_{\text{acc,lin}} \left(\max\{0,\ |\dot{v}| - a_{\max}\}\right)^2, & \dot{v} < 0, \end{cases} \tag{2.3}$$

$$c_{\dot{w}} = \begin{cases} w^{+}_{\text{acc,ang}} \left(\max\{0,\ |\dot{w}| - \alpha_{\max}\}\right)^2, & \dot{w} \geq 0, \\ w^{-}_{\text{acc,ang}} \left(\max\{0,\ |\dot{w}| - \alpha_{\max}\}\right)^2, & \dot{w} < 0, \end{cases} \tag{2.4}$$

$$c_{\ddot{v}} = \begin{cases} w^{+}_{\text{over-acceleration,lin}} \left(\max\{0,\ |\ddot{v}| - j^{\text{lin}}_{\max}\}\right)^2, & \ddot{v} \geq 0, \\ w^{-}_{\text{over-acceleration,lin}} \left(\max\{0,\ |\ddot{v}| - j^{\text{lin}}_{\max}\}\right)^2, & \ddot{v} < 0, \end{cases} \tag{2.5}$$

$$c_{\ddot{w}} = \begin{cases} w^{+}_{\text{over-acceleration,ang}} \left(\max\{0,\ |\ddot{w}| - j^{\text{ang}}_{\max}\}\right)^2, & \ddot{w} \geq 0, \\ w^{-}_{\text{over-acceleration,ang}} \left(\max\{0,\ |\ddot{w}| - j^{\text{ang}}_{\max}\}\right)^2, & \ddot{w} < 0, \end{cases} \tag{2.6}$$

For each trajectory, the total comfort cost is obtained by summing the velocity, acceleration, and over-acceleration penalties over the entire prediction horizon. This total cost is then added to the trajectory cost computed by the MPPI controller and influences the ranking and selection of the optimal control sequence.

$$c_t = c_v + c_w + c_{\dot{v}} + c_{\dot{w}} + c_{\ddot{v}} + c_{\ddot{w}} \tag{2.7}$$

$$C = \sum_{t=1}^{T} c_t \tag{2.8}$$

Parameter	**Threshold**
Linear velocity	$v_{\max} = 0.80\,\text{m/s}$
Angular velocity	$\omega_{\max} = 0.60\,\text{rad/s}$
Linear acceleration	$a_{\max} = 0.10\,\text{m/s}^2$
Angular acceleration	$\alpha_{\max} = 0.40\,\text{rad/s}^2$
Linear over-acceleration	$j_{\max} = 0.40\,\text{m/s}^3$
Angular over-acceleration	$\zeta_{\max} = 0.40\,\text{rad/s}^3$

Table 2.1: Thresholds used in ComfortCritic

The algorithm 4 summarizes the implemented scoring pipeline:

Algorithm 4 ComfortMPPICritic Scoring Logic

function SCORE(data)
  Get v_x, ω_z from `data.state`
  Get Δt from `data.model`
  Initialize $c \leftarrow 0$
  Compute $\dot{v}$, $\dot{\omega}$ via central difference
  Apply EMA smoothing to $\dot{v}$, $\dot{\omega}$
  Compute $\ddot{v}$, $\ddot{\omega}$ via central difference
  Apply EMA smoothing to $\ddot{v}$, $\ddot{\omega}$
  for each timestep t **do**
    Add velocity cost if $|v_x|$ or $|\omega_z|$ exceed thresholds
    Add acceleration cost if $|\dot{v}|$ or $|\dot{\omega}|$ exceed thresholds
    Add over-acceleration cost if $|\ddot{v}|$ or $|\ddot{\omega}|$ exceed thresholds
  end for
  Add total c to trajectory cost
end function

Although the benchmark includes both DWB and MPPI controllers combined with different planners, the comfort-oriented critic was implemented exclusively within the MPPI controller. MPPI provides direct access to ime-discretized velocity profiles over the prediction horizon, making it possible to estimate acceleration and over-acceleration during trajectory evaluation. In contrast, DWB approximates trajectories as sequences of constant-velocity circular or straight arcs, which do not allow reliable computation of acceleration and over-acceleration. While a velocity-only comfort penalty could theoretically be applied to DWB, the maximum feasible velocities produced by the controller are already below the comfort thresholds used in this work, resulting in negligible effects. Nevertheless, after the simulations, acceleration and over-acceleration can still be computed offline from recorded bag files. This allows a posteriori comparison of comfort metrics between DWB-based and MPPI-based navigation, even though the comfort critic actively influences only the MPPI controller.

2.2 Benchmark creation

The next step of this thesis is to test and evaluate the different combination of controllers and planners. Before testing on a real robot, simulations are used to observe the behavior and different software and packages are used to perform the simulations.

2.2.1 Gazebo environment

Gazebo have been chosen as simulator for the robot and the agents due to it flexibility and compatibility with other software used in this thesis. Gazebo allows to add models to create a customize environment for the simulation. The Figure 2.1 shows the world used for the simulation and take's inspiration from Caselle, the Torino's airport, in order to have large space and allows to implement crowd with different movement.

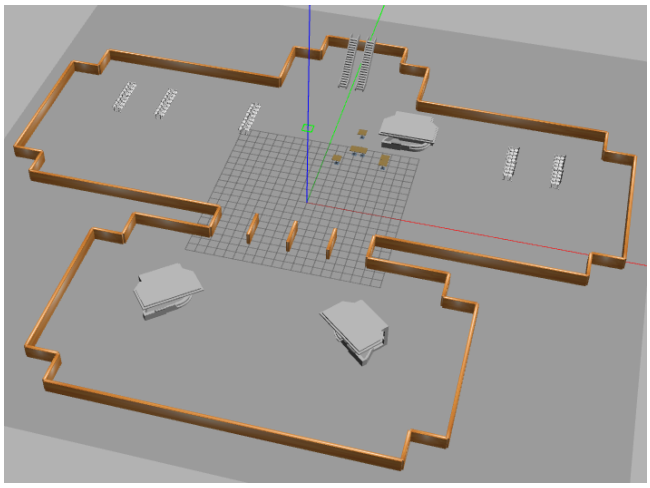

Figure 2.1: Gazebo world used in the simulation for the benchmark

The map is known by the robot and this represent the statics obstacles the planner have to deal with during the planning of the path. An additional obstacles will be introduced and more precisely different dynamic agents will be introduced to evaluate also the behavior with unknown obstacles. To evaluate better the behavior of robot with the dynamics obstacles, the map have been filled with three walls in order to split the the passage from the bottom to the top area of the airport; this introduction allows to add agents with different behavior in different corridor and simulate the episodes in different condition.

2.2.2 HunavSim

The implementation of dynamics obstacles is done through a ROS 2-based simulator for human navigation, the package HuNavSim [25], which allows to introduce agents and to control their behavior. The navigation behavior of the human agents spawned in a regular physics simulator can be controlled by HuNavSim. Therefore, HuNavSim can be "connected" to a popular simulators used in Robotics like Gazebo, Webots or Isaac Sim. At each simulation step, the wrapper must collect the current state of the human agents and the robot (positions and velocities), and send them to HuNavSim. The HuNavSim controller will compute and return the next state of the agents. Finally, the wrapper must update the agents' state in the simulation.
The behavior of the agents is not just an object moving in a space which its only objective is to go to initial position to the goal (or to goal to the successive goal), but they act as a real human and interact with the world around them and react to the presence of the robot. Different set of realistic human navigation reactions is provided by the packages:

- **Regular**: the human treats the robot like another human.
- **Impassive**: the human deals with the robot like a static obstacle.
- **Surprised**: when the human sees the robot, he/she stops walking and starts to look at the robot.

- **Curious**: the human abandons the current navigation goal for a while and starts to approach the robot slowly.
- **Scared**: the human tries to stay far from the robot.
- **Threatening**: the human tries to block the path of the robot by walking in front of it.

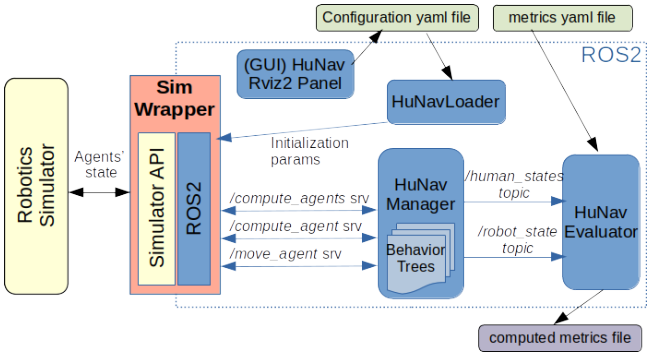

Figure 2.2: [25] HunavSim architecture

The initial parameters of the agents can be read from the "/hunav_loader" ROS2 node, which load the agents data from the yaml file in the configuration directory of the package hunav_agent_manager
The communication with HuNvavSim is done through different ROS2 services, which uses the messages of the package hunav_sim:

- "/compute_agents"

2.2.3 Gazebo test

Gazebo test is a repository containing utilities and script for testing using Gazebo in an automated way. The repository contains two packages:

- gazebo_test which is the main package and contains the test framework and utilities
- gazebo_sim which is a packages containing Gazebo simulation models and worlds for testing.

The advantages of running simulation with this packages is the capacity of running simulation and repeating them, generating then a report for each experiment.

2.2.4 Social evaluation graphs

Once the file .db3 containing all the data of the topic selected during the simulation, the package social_evaluation_graph is used in order to extract the data and evaluate the test, generating two .pickle file, one for the extraction data from .db3 and one for extracting the metrics from the .pickle. The high degree of configurability and customizability of this package allows to add from the default metrics of the package also the one needed for our simulation.

2.2.5 Scenarios

To obtain a comprehensive view of the possible situations in which an EPW could be involved in an airport environment, two agents file are used, in order to use the least number of agents for the scenario and to keep the simulation lightweight.

- south-north
- east-west

In the south–north configuration three scenarios are considered:

1. **Scenario 1** implements a vertical traversal of the environment, encountering along the trajectory agents that move perpendicularly to the robot.
2. **Scenario 2** implements a vertical traversal of the environment, encountering along the trajectory agents that move parallel to the robot.
3. **Scenario 3** implements a diagonal traversal through the corridors in the middle of the map, where two agents are present and move perpendicularly and parallel to the robot.

In the east–west configuration three scenarios are considered:

1. **Scenario 4** implements a diagonal traversal of the environment in order to evaluate the behavior during a curved path. Along the trajectory, agents moving in the same direction as the robot are present.
2. **Scenario 5** implements a horizontal traversal from the east side of the map to the west side, with agents moving in the same direction as the robot.
3. **Scenario 6** implements a diagonal traversal through the corridors in the middle of the map, where two agents are present and move both perpendicularly and parallel to the robot.

Through the six scenarios created it is possible to evaluate the metrics of the comfort and also other data, such as the path length and the time to reach the goal.

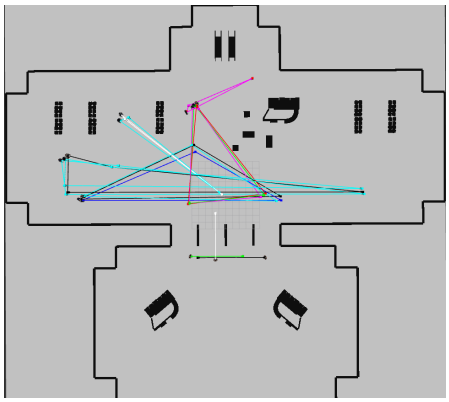

Figure 2.3: Agents configuration in the scenario east-west

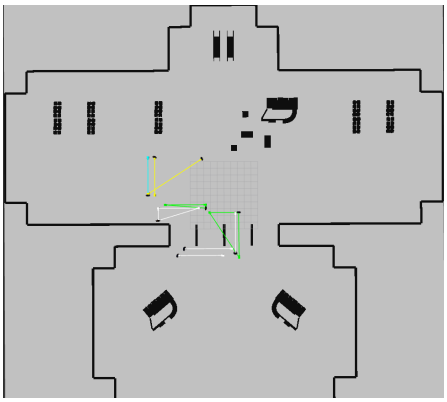

Figure 2.4: Agents configuration in the scenario south-north

To simulate the presence of the agents in the environment, two different configuration files are used to populate the regions of interest of the map. As shown in Figures 2.3 and 2.4, the agents move continuously along the predefined path, computed by the planner, with the purpose of encountering the robot and observing its behavior. The figures also illustrate how the agents are grouped to reflects realistic crowd dynamics, where individuals with similar goals tend to move in the same direction or hinder the path.

2.2.6 Metrics evaluated

Once the data are extracted and converted into a `.pickle` file containing only the required metrics, all files are loaded into a Jupyter notebook to generate the plots. Since the ComfortCritic explicitly targets motion smoothness, the evaluation focuses on dynamic quantities, but also other metrics are evaluated in terms of safety and navigation performance. The considered metrics are grouped as follows:

- **Average kinematic quantities**:
  - *Average linear velocity, acceleration, and over-acceleration*: mean values computed over the entire navigation task by averaging the linear velocity, linear acceleration, and linear over-acceleration across all collected trajectory samples. These metrics provide an overall indication of the typical longitudinal motion profile experienced by the robot.
  - *Average angular velocity, acceleration, and over-acceleration*: mean values of angular velocity, angular acceleration, and angular over-acceleration computed over the full navigation duration. They describe the average rotational behavior of the robot and are particularly relevant for assessing turning smoothness.
- **Maximum dynamic values**:
  - *Maximum linear acceleration*: the highest linear acceleration value observed during the navigation task, representing the most intense longitudinal acceleration experienced by the robot.
  - *Maximum angular acceleration*: the peak angular acceleration measured along the trajectory, indicating the most abrupt rotational motion.
  - *Maximum linear over-acceleration*: the maximum value of linear over-acceleration recorded, capturing the sharpest changes in linear acceleration, which are closely related to perceived discomfort.
  - *Maximum angular over-acceleration*: the largest angular over-acceleration encountered during navigation, reflecting sudden variations in rotational acceleration.
- **Threshold exceedances**:
  - *Number of linear acceleration threshold violations*: the total number of samples in which the linear acceleration exceeds a predefined comfort threshold, used to quantify how often longitudinal comfort limits are violated.

- *Number of angular acceleration threshold violations*: the number of occurrences where angular acceleration surpasses the defined threshold, indicating potentially uncomfortable rotational motions.
- *Number of linear over-acceleration threshold violations*: the count of samples where linear over-acceleration exceeds the allowed comfort limit, highlighting abrupt changes in linear motion.
- *Number of angular over-acceleration threshold violations*: the total number of instances in which angular over-acceleration goes beyond the selected threshold, capturing sudden rotational dynamics that may negatively impact comfort.

- **Navigation performance metrics**:
  - *Time to reach goal*: the total elapsed time from the start of the navigation task until the robot successfully reaches the goal position, providing a direct measure of navigation efficiency.
  - *Path length*: the total length of the executed trajectory, computed by integrating the traveled distance along the path, and used to evaluate path efficiency and detours.

- **Safety and social interaction metrics**:
  - *Average distance to obstacles and people*: the mean distance between the robot and nearby obstacles or dynamic agents over the entire navigation task, indicating the typical safety margin maintained during motion.
  - *Minimum distance to obstacles and people*: the smallest distance recorded between the robot and any obstacle or person during navigation, representing the worst-case proximity event and a critical safety indicator.

For each scenario, all reported metrics are computed as the mean values over 15 independent runs.
The majority of the metrics evaluated are already present in the package, while the metrics for the maximum values and for counting the number of times over the thresholds are added.

```
def compute_lin_acc_over-acceleration(df_robot, period=5, alpha
    =0.05):
    df = df_robot.copy()
    df["accel"] = df.velocity_x.diff(period) / df.time.diff(period)
    df["over-acceleration"] = df["accel"].diff(period) / df.time.
        diff(period)
    df["accel_smooth"] = df["accel"].ewm(alpha=alpha).mean()
    df["over-acceleration_smooth"] = df["over-acceleration"].ewm(
        alpha=alpha).mean()
    df.dropna(inplace=True)
    return df

@metric("max_acceleration")
def max_acceleration(df_agents_list, df_robot, *args, period=5,
    alpha=0.05, **kwargs):
    df = compute_lin_acc_over-acceleration(df_robot, period, alpha)
    return df["accel_smooth"].max(), df[["time", "accel_smooth"]]
```

```
14
15 @metric("count_linear_acceleration_threshold")
16 def count_linear_acceleration_threshold(df_agents_list, df_robot, *
       args, threshold=0.1, period=5, alpha=0.05, **kwargs):
17     df = compute_lin_acc_over-acceleration(df_robot, period, alpha)
18     count = (df["accel_smooth"].abs() > threshold).sum()
19     return count, df[["time", "accel_smooth"]]
```

Listing 2.1: Linear acceleration and over-acceleration metrics

Listing 2.1 shows the Python functions used to compute metrics related to linear velocity. The `compute_lin_acc_over-acceleration` function takes the robot's velocity over time and derives:

- The linear acceleration via central finite difference over a configurable window (`period = 5`)
- The linear over-acceleration as the rate of change of acceleration
- EMA-smoothing is then applied to both signals using a low smoothing factor `\alpha = 0.05` to reduce noise without distorting peaks

The angular versions follow the same logic and are computed using angular velocity instead of linear velocity.
Furthermore, other metrics are evaluated that are not parameter of the critic, but are indirectly related to the comfort of the passenger, such as the time to reach the goal: if to go to an initial pose to the goal in 10 times the time needed to the robot without the critic the passenger could probably prefer to be faster to the detriment of the comfort. The other metrics evaluated are the following:

- time to reach the goal
- path lenght
- average distance to people
- minimum distance to people
- average distance to obstacle
- minimum distance to obstacle

Results of simulations

To evaluate the comfort for each planner-controller combination different metrics are considered, but the main metrics are:

- The maximum values
- The number of times over the threshold
- The minimum distance with people and obstacles

Since the map is larger and the critical situation, where the robots approach static and dynamic obstacles affect for a small amount of time with respect to the complete simulation, the impact of the critic is not significant. The path length and the time to reach goal metrics are evaluated only in terms of how much the critics affect them, allowing the robot to spent more time and to generate longer path. In fact few seconds and meters does not affect the comfort of a navigation, so during the comparison this two metrics became important only for large difference. In the results the maximum linear and angular speed are not evaluated due to the dynamics limits of the robot parameter, set in the controller parameter. In fact the maximum linear speed is 0.4 and the threshold is set to 0.8
Each episode of each planner-controller combination is simulated 15 times, in order to guarantee a sufficient number of simulation for the evaluation. It is important to stress how each simulation differs from the other with the same planner-controller combination and the same parameter and to reduce this determinism,
Each simulation for each episode differs from the other with the same planner-controller combination and the same parameter and to reduce this stochastic behavior, a sufficient number of simulation are done to cover different possible scenario for each episode, although from a modeling perspective, the main components of the systems are deterministic when initialized with identical parameters and initial condition. The only systems which introduce non deterministic behavior is the MPPI which is stochastic from design, but with a random seed fixed in become deterministic. In practice, each simulation run evolves slightly different due to microscopic execution-level variation that are unavoidable in complex robotic systems. These variation originate from several sources, as floating-point arithmetic, asynchronous execution of ROS2 nodes and timing differences in message passing. These extremely small differences are amplified over time by the closed-loop nature of the navigation systems and in particular by the MPPI controller. As a results, the overall system exhibits behavior that can be described as deterministic but chaotic, in the sense that in contains no explicit stochastic processes, yet is highly sensitive to initial conditions and executive-level-perturbations.

3.1 Parameters tuning

Once all parameters are set, the critic's tuning is done thought several simulation to reach an improvement of the comfort during the navigation.The tuning is done

with the SmacPlannerLattice and once the parameters leads to an improvement, they are tested with all the planners in all scenarios:

1. SmacPlannerLattice + MPPI with and without the critic
2. SmacPlannerHybrid+ MPPI with and without the critic
3. NavFn A* + MPPI with and without the critic

The critic `PreferForwardCritic` is removed form the plugin and the `TwirlingCritic` is added. The reason behind this modification is the possible conflict between the `PreferForwardCritic` and the ComfortCritic due to the fact the first push the robot to move forward, in the direction to the goal reducing the time to reach the goal, but this could generate high dynamics values, which is the opposite of the wanted behavior. The `TwirlingCritic` is added due to the reduction of angular acceleration and overacceleration during the tuning. This critic is not added to the default simulation due to the fact, after different simulation, it affects negatively the linear acceleration and overacceleration.
Counter-intuitively an increment of the weight of the parameter does not leads to a comfortable navigation since it introduce higher peaks in some dynamics quantities. This behavior is caused by the lower cost of an higher peaks with respect to several smaller peaks. An aspect counterintuitive is the improvement of the quality comfort of the navigation with the critic penalizing the dynamics quantities not from the threshold value, but with respect to an higher cost.

3.2 Metrics evaluation

3.2.1 Straight navigation in narrow corridor with crossing agents

The **scenario 1** simulates a straight navigation into a narrow corridor with crossing agents, yielding a stop-and-go pattern due to the presence of agents along the path, blocking the navigation and inducing peaks of acceleration.

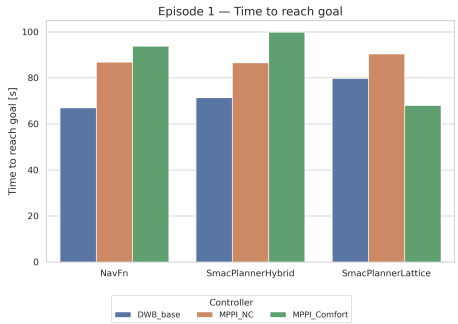

Figure 3.1: Time to reach goal metric for scenario 1

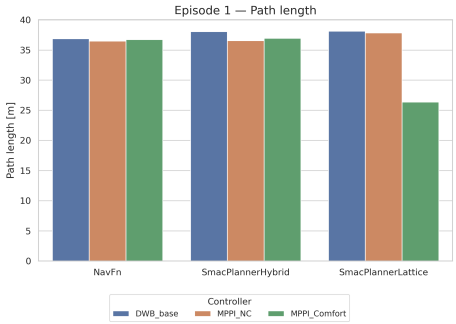

Figure 3.2: Path length metric for scenario 1

The time to reach the goal is, as expected, lower for the DWB controller, particularly when combined with NavFn and SmacPlannerHybrid with respectively 67 s and 71.4 s. In contrast, the default configuration of MPPI exhibits longer execution times, with a noticeable increase when the comfort critic is enabled. Notably, the implementation of the critic reduces this metric compared to the MPPI-based configurations by 24.7%.
Path lengths are comparable across all controller–planner combinations, ranging from 26.4 m to 38.1 m, suggesting that longer times to reach the goal do not result in significantly different paths. This metric follows the same trend as the *time to reach goal* metric, exhibiting a slight increase for NavFn and SmacPlannerHybrid with MPPI-comfort, while SmacPlannerLattice shows a reduction.

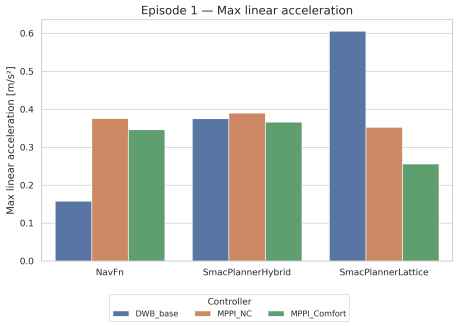

Figure 3.3: Maximum linear acceleration metric for scenario 1

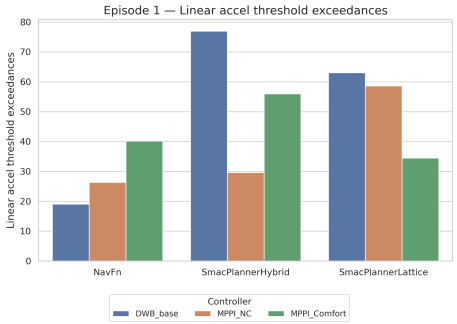

Figure 3.4: Count of linear acceleration threshold exceedances for scenario 1

Across all combinations, DWB–SmacPlannerLattice shows the highest peak acceleration with $0.61\,m/s^2$, indicating a more aggressive linear control compared to NavFn and SmacPlannerHybrid. DWB–NavFn, with $0.16\,m/s^2$, shows the lowest value in this scenario.
The implementation of the comfort critic consistently reduces maximum linear acceleration across all planners, especially for SmacPlannerLattice, with a reduction of 27%.
Due to the stop-and-go pattern in this scenario, multiple threshold exceedances are expected and are clearly planner-dependent. The only planner that significantly benefits from the comfort critic is SmacPlannerLattice, with a reduction of 41%, decreasing from 58.6 to 34.5.

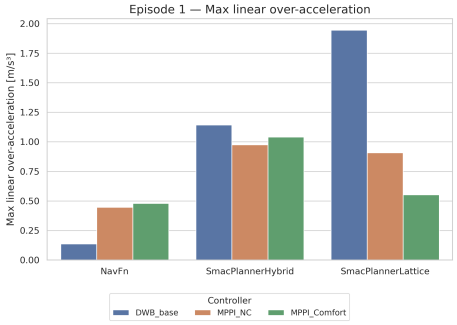

Figure 3.5: Maximum linear over-acceleration metric for scenario 1

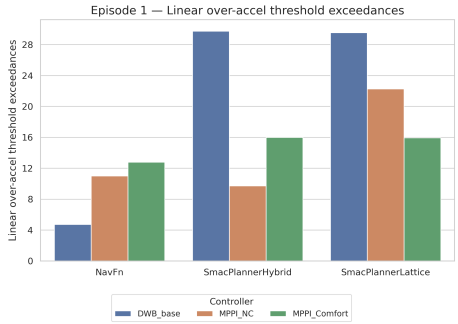

Figure 3.6: Count of linear over-acceleration threshold exceedances for scenario 1

For the maximum linear over-acceleration metric, the same pattern observed for linear acceleration is repeated, with the DWB controller showing both the highest and lowest peaks when combined with SmacPlannerLattice ($1.94\,m/s^3$) and NavFn ($0.14\,m/s^3$), respectively. The comfort critic mitigates maximum linear over-acceleration, with a pronounced effect for SmacPlannerLattice corresponding to a reduction of approximately 39%.
The combination of DWB and SmacPlanners exhibits the highest number of threshold exceedances, while NavFn reports the lowest count. For the MPPI controller, the comfort critic shows an improvement only when combined with SmacPlannerLattice, while results with NavFn remain comparable.

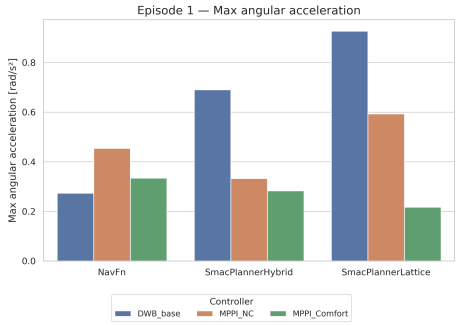

Figure 3.7: Maximum angular acceleration metric for scenario 1

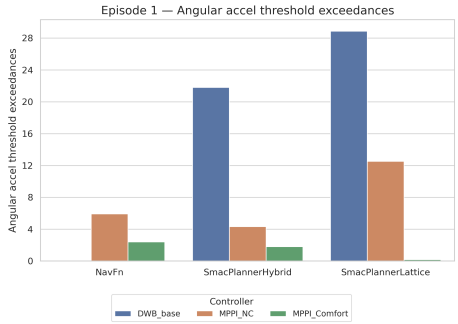

Figure 3.8: Count of angular acceleration threshold exceedances for scenario 1

Despite the straight navigation between the initial pose and the goal, the presence of crossing agents induces peaks of angular acceleration to avoid collisions. The combinations of DWB with SmacPlanners produce higher peaks, indicating abrupt and aggressive rotational maneuvers. DWB–NavFn exhibits the best behavior among DWB-based configurations.
For MPPI results, the influence of the comfort critic is significant and consistently reduces maximum angular acceleration across all planners. The most pronounced improvement is observed with SmacPlannerLattice, where the peak drops from $0.59\,rad/s^2$ to $0.22\,rad/s^2$, corresponding to a reduction of 63%.
Across all planners, MPPI-comfort exhibits the best overall behavior, maintaining the number of threshold exceedances below 5. While DWB–NavFn performs well for its respective planner, performance deteriorates when combined with SmacPlanners.

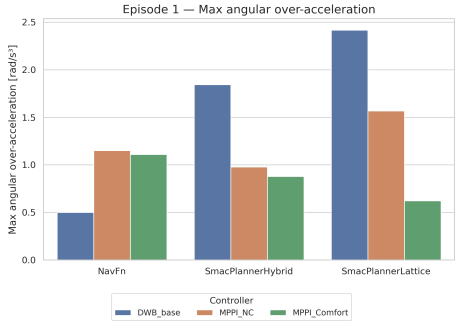

Figure 3.9: Maximum angular over-acceleration metric for scenario 1

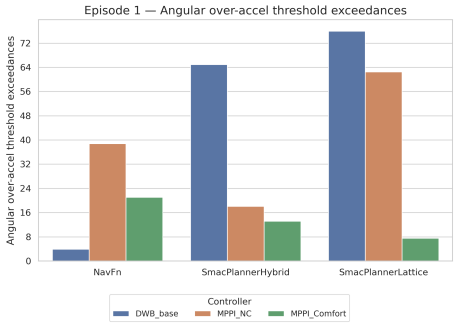

Figure 3.10: Count of angular over-acceleration threshold exceedances for scenario 1

The same trend observed for angular acceleration is repeated for angular over-acceleration. DWB combined with NavFn exhibits the best behavior, while performance degrades when combined with SmacPlanners. Across all planners, the comfort critic reduces peak values with respect to MPPI-based configurations, particularly for SmacPlannerLattice, where a reduction of approximately 60% is achieved. These results confirm the effectiveness of the comfort critic in damping impulsive rotational transients and promoting smoother orientation changes.

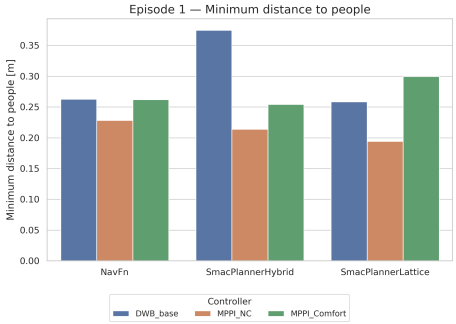

Figure 3.11: Minimum distance to people for scenario 1

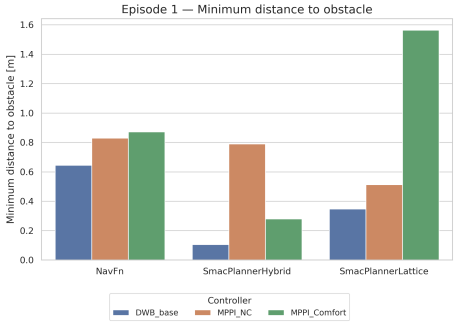

Figure 3.12: Minimum distance to obstacles for scenario 1

Regarding passenger safety, the minimum distance to people and obstacles is evaluated. For the minimum distance to people, DWB configurations show moderate clearance, with SmacPlannerHybrid achieving the largest value. The comfort critic slightly improves this metric, although the improvement is limited to a few centimeters. For the minimum distance to obstacles, DWB exhibits the worst performance across all planner combinations, while the comfort critic provides a significant improvement when combined with SmacPlannerLattice, but leads to a degradation when used with SmacPlannerHybrid.

Algorithm	**Success rate**
DWB + NavFn	100%
DWB + SmacPlannerHybrid	100%
DWB + SmacPlannerLattice	100%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	100%
MPPI-NC + SmacPlannerLattice	100%
MPPI-Comfort + NavFn	100%
MPPI-Comfort + SmacPlannerHybrid	93.3%
MPPI-Comfort + SmacPlannerLattice	100%

Table 3.1: Success rate — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	67.0	86.9	93.8	+7.9
SmacPlannerHybrid	71.4	86.6	99.8	+15.2
SmacPlannerLattice	79.8	90.4	68.1	-24.7

Table 3.2: Time to reach goal (s) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	36.9	36.5	36.7	+0.5
SmacPlannerHybrid	38.1	36.5	37.0	+1.4
SmacPlannerLattice	38.1	37.8	26.4	-30.2

Table 3.3: Path length (m) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	1.50	3.05	2.86	-6.2
SmacPlannerHybrid	7.31	2.24	2.70	+20.5
SmacPlannerLattice	12.1	5.06	1.91	-62.3

Table 3.4: Cumulative heading changes (rad) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	0.158	0.376	0.346	-8.0
SmacPlannerHybrid	0.375	0.390	0.366	-6.2
SmacPlannerLattice	0.606	0.352	0.256	-27.3

Table 3.5: Maximum linear acceleration (m/s^2) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	19.1	26.3	40.2	+52.9
SmacPlannerHybrid	76.9	29.7	55.9	+88.2
SmacPlannerLattice	63.0	58.6	34.5	-41.1

Table 3.6: Linear acceleration threshold exceedances — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	0.136	0.446	0.480	+7.6
SmacPlannerHybrid	1.14	0.976	1.04	+6.6
SmacPlannerLattice	1.94	0.907	0.553	-39.0

Table 3.7: Maximum linear overacceleration (m/s^3) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	4.73	11.0	12.8	+16.4
SmacPlannerHybrid	29.7	9.73	16.0	+64.5
SmacPlannerLattice	29.5	22.3	15.9	-28.7

Table 3.8: Linear overacceleration threshold exceedances — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	0.274	0.454	0.335	-26.2
SmacPlannerHybrid	0.690	0.332	0.283	-14.8
SmacPlannerLattice	0.927	0.593	0.217	-63.4

Table 3.9: Maximum angular acceleration (rad/s^2) — Episode 1

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	0.0	5.93	2.40	-59.5
SmacPlannerHybrid	21.8	4.33	1.80	-58.4
SmacPlannerLattice	28.9	12.5	0.20	-98.4

Table 3.10: Angular acceleration threshold exceedances — Episode 1

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.50	1.15	1.11	-3.5
SmacPlannerHybrid	1.84	0.978	0.879	-10.1
SmacPlannerLattice	2.42	1.57	0.623	-60.3

Table 3.11: Maximum angular overacceleration (rad/s^3) — Episode 1

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	3.93	38.8	21.1	-45.6
SmacPlannerHybrid	64.9	18.1	13.1	-27.6
SmacPlannerLattice	75.9	62.5	7.53	-88.0

Table 3.12: Angular overacceleration threshold exceedances — Episode 1

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.263	0.228	0.262	+14.9
SmacPlannerHybrid	0.375	0.214	0.254	+18.7
SmacPlannerLattice	0.259	0.194	0.299	+54.1

Table 3.13: Minimum distance to people (m) — Episode 1

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.645	0.830	0.872	+5.1
SmacPlannerHybrid	0.106	0.790	0.280	-64.6
SmacPlannerLattice	0.348	0.512	1.560	+204.7

Table 3.14: Minimum distance to obstacles (m) — Episode 1

3.2.2 Straight navigation in narrow corridor with passing agents

The **scenario 2** simulates the same situation as **scenario 1**, with the difference of a parallel flow of agents instead of a crossing flow. Lower linear dynamic values are expected due to the absence of a stop-and-go pattern, while angular dynamics may increase to avoid collisions with people.

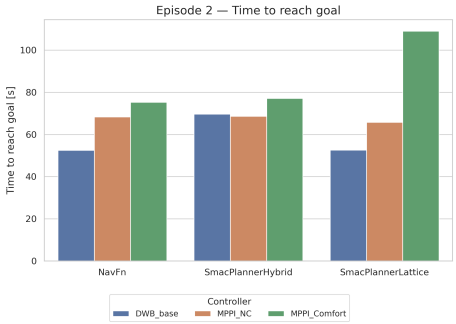

Figure 3.13: Time to reach goal metric for scenario 2

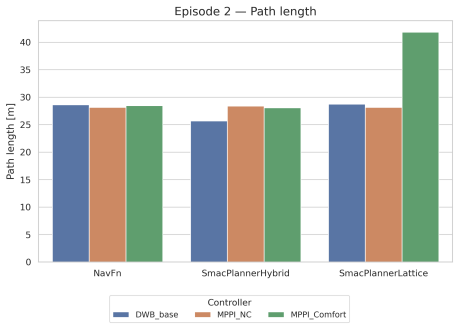

Figure 3.14: Path length metric for scenario 2

The lowest time to reach the goal is achieved by the DWB controller across all configurations, while the implementation of the comfort critic increases the execution time, especially when combined with SmacPlannerLattice. As discussed at the beginning of the chapter, this metric does not directly influence passenger comfort. The path length remains comparable across NavFn and SmacPlannerHybrid, while it increases for MPPI-comfort combined with SmacPlannerLattice, suggesting a different path selection with respect to the other controller–planner combinations.

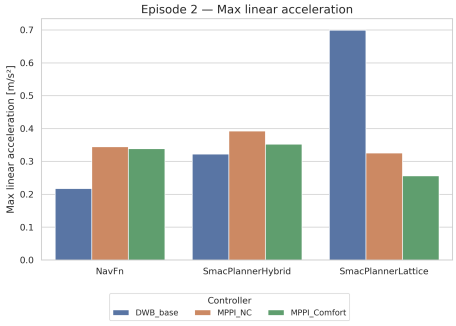

Figure 3.15: Maximum linear acceleration metric for scenario 2

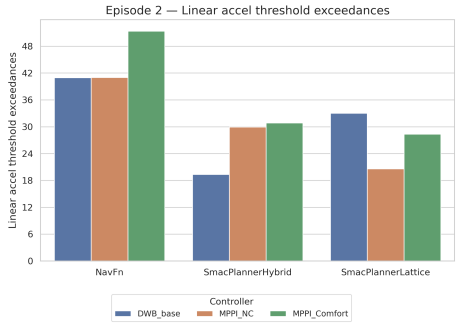

Figure 3.16: Count of linear acceleration threshold exceedances for scenario 2

Similarly to scenario 1, the combination DWB–SmacPlannerLattice yields the highest peak of linear acceleration ($0.70\,m/s^2$), while the use of MPPI as controller reduces the peaks by approximately half. The comfort critic slightly reduces peak values across all planners. For the NavFn planner, DWB remains the best-performing controller in terms of peak linear acceleration.
The number of threshold exceedances is planner-dependent:

- **NavFn:** $\iota oggle DWB and MPPI show comparable behavior$; **SmacPlannerHybrid:** $MPPI exhibits a hig$

- **SmacPlannerLattice:** MPPI results in fewer exceedances compared to DWB.

It can be observed that, for all planners, the implementation of the comfort critic slightly increases the number of threshold exceedances.

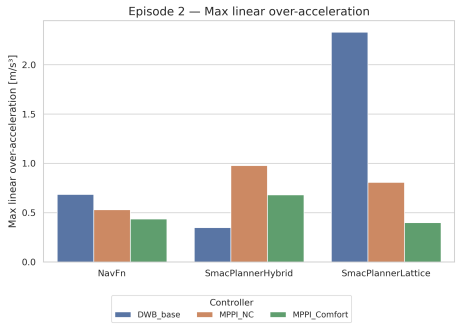

Figure 3.17: Maximum linear over-acceleration metric for scenario 2

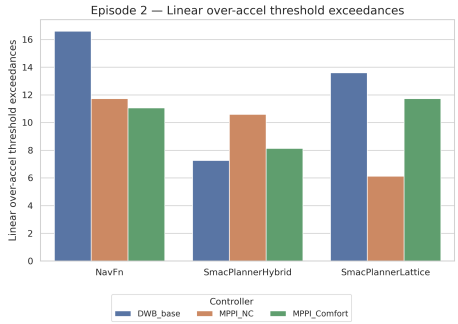

Figure 3.18: Count of linear over-acceleration threshold exceedances for scenario 2

Within the DWB configurations, a strong planner dependency emerges, yielding the highest peak when combined with NavFn and, in particular, SmacPlanner-Lattice, reaching $2.33\,m/s^3$. In contrast, the combination with SmacPlanner-Hybrid represents the best option among DWB-based controllers. The MPPI-comfort configuration systematically lowers linear over-acceleration peaks across all planners.
Across all planners, DWB records the highest number of threshold exceedances when combined with NavFn and SmacPlannerLattice. MPPI provides a better trade-off overall, and the implementation of the comfort critic further reduces the count for NavFn and SmacPlannerHybrid.

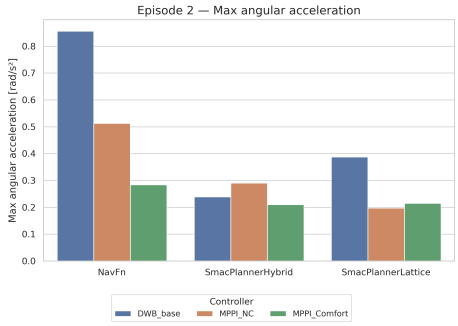

Figure 3.19: Maximum angular acceleration metric for scenario 2

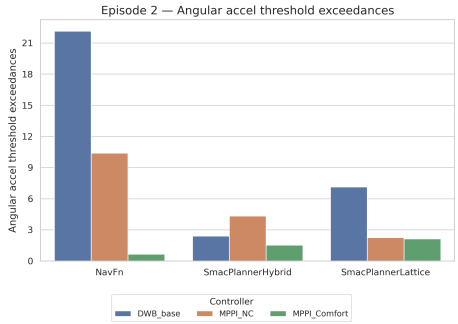

Figure 3.20: Count of angular acceleration threshold exceedances for scenario 2

The MPPI-comfort configuration exhibits the best performance for both angular acceleration metrics. In particular, peak angular acceleration values are reduced for NavFn and SmacPlannerHybrid, while results for SmacPlannerLattice are comparable with MPPI without the critic. DWB shows high peaks when combined with NavFn and SmacPlannerLattice, while achieving better performance than MPPI-based configurations when used with SmacPlannerHybrid.

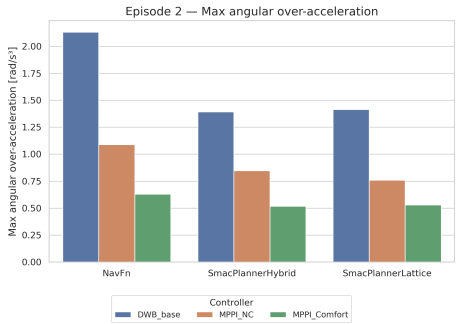

Figure 3.21: Maximum angular over-acceleration metric for scenario 2

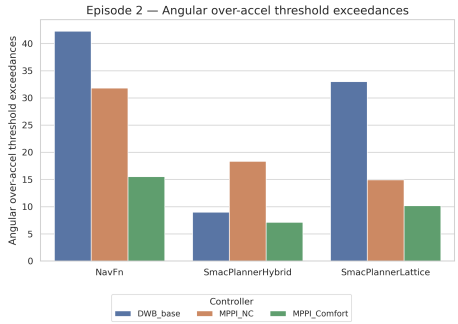

Figure 3.22: Count of angular over-acceleration threshold exceedances for scenario 2

The combination DWB–NavFn exhibits the highest peaks across all controller–

planner configurations, both in magnitude and frequency of angular over-acceleration events. In this scenario, DWB shows aggressive behavior when combined with all planners. Conversely, MPPI significantly reduces peak values (approximately 40% for all planners), while also lowering the number of exceedances for NavFn and SmacPlannerLattice. The comfort critic further reduces peaks by an additional 30–40% and decreases the number of threshold exceedances across all planners, confirming it as the best-performing configuration for this metric in scenario 2.

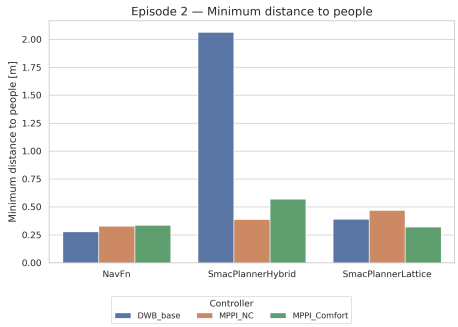

Figure 3.23: Minimum distance to people for scenario 2

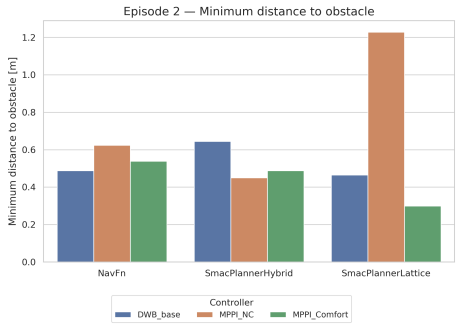

Figure 3.24: Minimum distance to obstacles for scenario 2

For the minimum distance to people and obstacles, results are comparable across most controller–planner combinations, with two notable outliers: MPPI-based configurations combined with SmacPlannerLattice and DWB combined with SmacPlannerHybrid.

Algorithm	Success rate
DWB + NavFn	100%
DWB + SmacPlannerHybrid	86.7%
DWB + SmacPlannerLattice	100%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	100%
MPPI-NC + SmacPlannerLattice	100%
MPPI-Comfort + NavFn	100%
MPPI-Comfort + SmacPlannerHybrid	100%
MPPI-Comfort + SmacPlannerLattice	100%

Table 3.15: Success rate — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	52.5	68.4	75.2	+9.9
SmacPlannerHybrid	69.6	68.7	77.2	+12.4
SmacPlannerLattice	52.6	65.7	109.0	+65.9

Table 3.16: Time to reach goal (s) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	28.6	28.1	28.5	+1.4
SmacPlannerHybrid	25.7	28.4	28.1	-1.1
SmacPlannerLattice	28.7	28.2	41.8	+48.2

Table 3.17: Path length (m) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	4.17	3.19	2.80	-12.2
SmacPlannerHybrid	2.11	4.15	1.78	-57.1
SmacPlannerLattice	3.44	2.35	3.25	+38.3

Table 3.18: Cumulative heading changes (rad) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.217	0.345	0.339	-1.7
SmacPlannerHybrid	0.323	0.393	0.353	-10.2
SmacPlannerLattice	0.699	0.326	0.257	-21.2

Table 3.19: Maximum linear acceleration (m/s^2) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	40.9	41.0	51.3	+25.1
SmacPlannerHybrid	19.3	29.9	30.9	+3.3
SmacPlannerLattice	33.0	20.6	28.3	+37.4

Table 3.20: Linear acceleration threshold exceedances — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.685	0.530	0.436	-17.7
SmacPlannerHybrid	0.350	0.980	0.682	-30.4
SmacPlannerLattice	2.33	0.809	0.399	-50.7

Table 3.21: Maximum linear overacceleration (m/s^3) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	16.6	11.7	11.1	-5.1
SmacPlannerHybrid	7.27	10.6	8.13	-23.3
SmacPlannerLattice	13.6	6.13	11.7	+90.9

Table 3.22: Linear overacceleration threshold exceedances — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.856	0.512	0.284	-44.5
SmacPlannerHybrid	0.239	0.291	0.210	-27.8
SmacPlannerLattice	0.388	0.197	0.215	+9.1

Table 3.23: Maximum angular acceleration (rad/s^2) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	22.1	10.4	0.667	-93.6
SmacPlannerHybrid	2.4	4.33	1.53	-64.7
SmacPlannerLattice	7.13	2.27	2.13	-6.2

Table 3.24: Angular acceleration threshold exceedances — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.13	1.09	0.629	-42.3
SmacPlannerHybrid	1.39	0.848	0.518	-38.9
SmacPlannerLattice	1.42	0.760	0.529	-30.4

Table 3.25: Maximum angular overacceleration (rad/s^3) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	42.3	31.8	15.5	-51.3
SmacPlannerHybrid	9.0	18.3	7.13	-61.0
SmacPlannerLattice	33.0	14.9	10.2	-31.5

Table 3.26: Angular overacceleration threshold exceedances — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.277	0.329	0.337	+2.4
SmacPlannerHybrid	2.06	0.389	0.571	+46.8
SmacPlannerLattice	0.391	0.470	0.320	-31.9

Table 3.27: Minimum distance to people (m) — Episode 2

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.488	0.623	0.538	-13.6
SmacPlannerHybrid	0.644	0.450	0.488	+8.4
SmacPlannerLattice	0.465	1.23	0.299	-75.7

Table 3.28: Minimum distance to obstacles (m) — Episode 2

3.2.3 Curve trajectory in narrow corridor

The **scenario 3** simulates a vertical crossing with curvature, without the presence of agents. The main difficulty arises from the combination of curvature and a narrow corridor. Acceleration peaks may be generated to avoid collisions with static obstacles encountered during navigation.

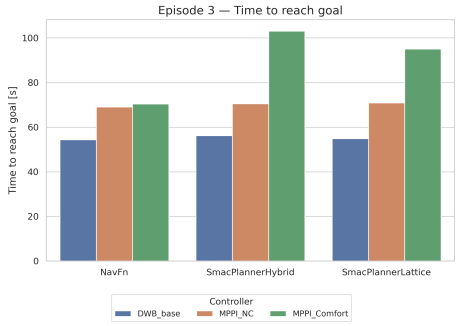

Figure 3.25: Time to reach goal metric for scenario 3

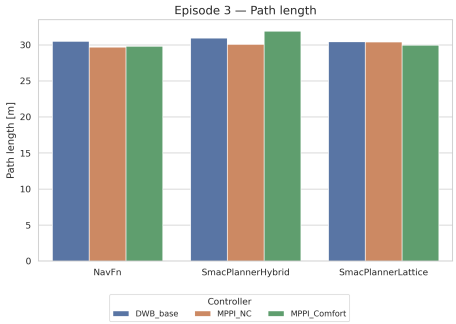

Figure 3.26: Path length metric for scenario 3

This complex scenario, in terms of collision avoidance and path computation along a curved trajectory within a narrow corridor, exhibits results that are expected *a priori*. DWB achieves the lowest time to reach the goal for all planners, while the use of the comfort critic and the absence of the *PreferForwardCritic* increase the execution time.
Figure 3.26 shows that the executed path length is comparable across all controller–planner combinations, ranging between $29.7\,m$ and $31.9\,m$.

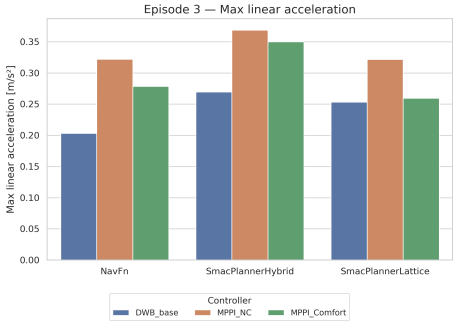

Figure 3.27: Maximum linear acceleration metric for scenario 3

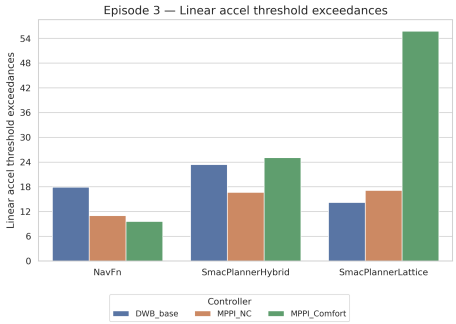

Figure 3.28: Count of linear acceleration threshold exceedances for scenario 3

DWB configurations exhibit moderate peak linear accelerations, with the com-

bination with NavFn representing the best trade-off. MPPI introduces higher peak values across all planners; however, the implementation of the comfort critic slightly reduces these peaks. Nevertheless, this reduction is not sufficient to match the performance achieved by DWB.
In contrast to peak acceleration values, the count of linear acceleration threshold exceedances shows better performance for MPPI across all planners. The comfort critic introduces a significant increase in threshold exceedances when combined with SmacPlannerLattice.

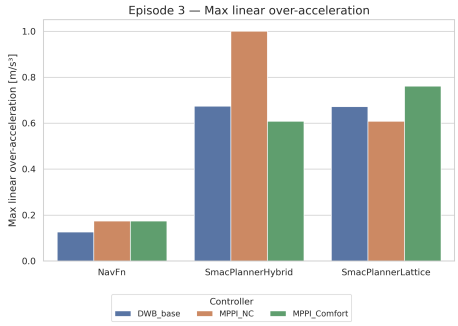

Figure 3.29: Maximum linear over-acceleration metric for scenario 3

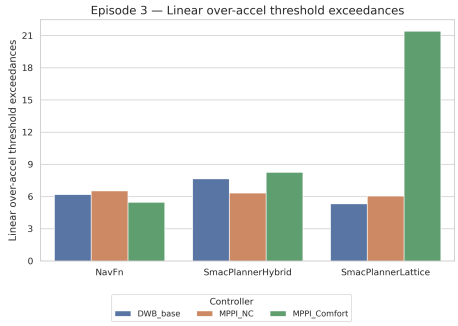

Figure 3.30: Count of linear over-acceleration threshold exceedances for scenario 3

DWB configurations generally exhibit lower peak over-acceleration values when combined with NavFn, while higher peaks emerge with SmacPlannerHybrid and SmacPlannerLattice, indicating a stronger sensitivity to sudden velocity variations. MPPI-based configurations show a wider spread of values, with some planners exhibiting higher peak over-acceleration, reflecting a more dynamic but potentially less impulsive control strategy.
When considering the number of linear over-acceleration threshold exceedances, differences between configurations become more evident. Most controller–planner combinations show comparable counts, suggesting that impulsive linear events occur with similar frequency. However, MPPI-comfort combined with SmacPlannerLattice stands out with a substantially higher number of exceedances, indicating frequent high-frequency acceleration transients despite moderate peak values.

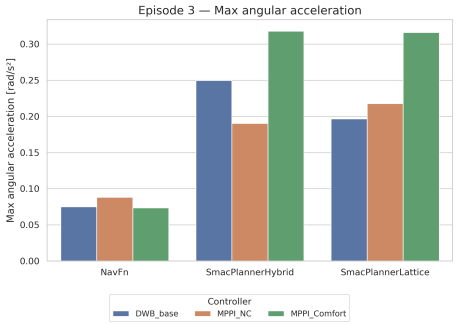

Figure 3.31: Maximum angular acceleration metric for scenario 3

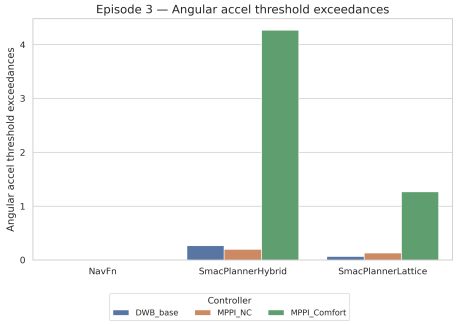

Figure 3.32: Count of angular acceleration threshold exceedances for scenario 3

For this metric, the implementation of the comfort critic does not provide benefits in terms of passenger comfort. In fact, it leads to higher peak values when combined with SmacPlanners and degrades MPPI performance. In this scenario, NavFn represents the best trade-off for angular acceleration-related metrics.

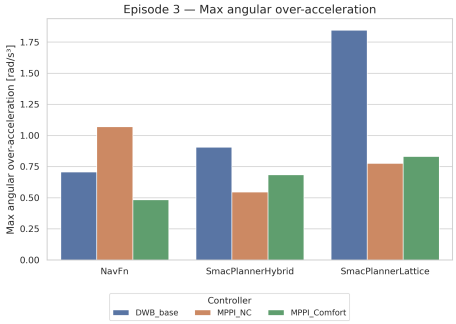

Figure 3.33: Maximum angular over-acceleration metric for scenario 3

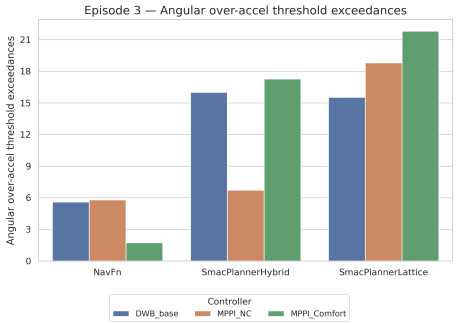

Figure 3.34: Count of angular over-acceleration threshold exceedances for scenario 3

The behavior of DWB differs from that observed for angular acceleration. It exhibits the highest angular over-acceleration peaks when combined with SmacPlanners ($1.84\,m/s^3$ and $0.91\,m/s^3$), while the influence of the comfort critic significantly improves performance only when combined with the NavFn planner for both metrics.

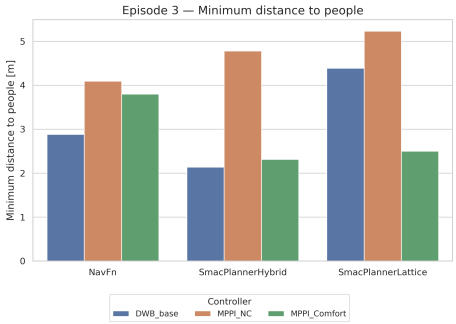

Figure 3.35: Minimum distance to people for scenario 3

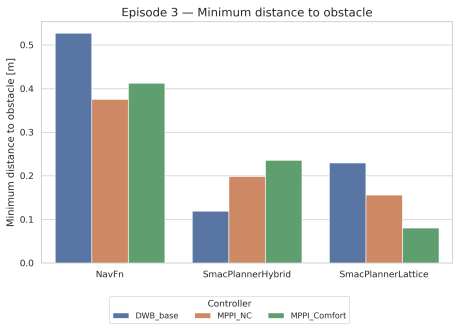

Figure 3.36: Minimum distance to obstacles for scenario 3

Regarding minimum distances to people and obstacles, MPPI exhibits the best overall behavior, maintaining larger clearances from potential collisions, particularly with dynamic agents. The implementation of the comfort critic reduces

these distances by approximately half when combined with SmacPlanners, while remaining comparable to NavFn-based configurations. For minimum distances to obstacles, overall values are smaller due to the structural constraints of the scenario, and the comfort critic yields improved performance when combined with SmacPlannerHybrid.

Algorithm	**Success rate**
DWB + NavFn	100%
DWB + SmacPlannerHybrid	100%
DWB + SmacPlannerLattice	100%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	93.3%
MPPI-NC + SmacPlannerLattice	93.3%
MPPI-Comfort + NavFn	100%
MPPI-Comfort + SmacPlannerHybrid	93.3%
MPPI-Comfort + SmacPlannerLattice	100%

Table 3.29: Success rate — Episode 3

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	54.3	69.1	70.4	+1.9
SmacPlannerHybrid	56.2	70.5	103.0	+46.1
SmacPlannerLattice	54.9	70.9	95.0	+34.0

Table 3.30: Time to reach goal (s) — Episode 3

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	30.5	29.7	29.8	+0.3
SmacPlannerHybrid	31.0	30.1	31.9	+6.0
SmacPlannerLattice	30.4	30.4	30.0	-1.3

Table 3.31: Path length (m) — Episode 3

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	1.70	1.31	1.25	-4.6
SmacPlannerHybrid	2.55	2.25	3.93	+74.7
SmacPlannerLattice	2.39	3.12	5.32	+70.5

Table 3.32: Cumulative heading changes (rad) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.203	0.322	0.278	-13.7
SmacPlannerHybrid	0.269	0.369	0.350	-5.1
SmacPlannerLattice	0.253	0.322	0.259	-19.6

Table 3.33: Maximum linear acceleration (m/s^2) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	17.9	11.0	9.6	-12.7
SmacPlannerHybrid	23.4	16.7	25.1	+50.3
SmacPlannerLattice	14.2	17.1	55.7	+225.7

Table 3.34: Linear acceleration threshold exceedances — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.127	0.175	0.174	-0.6
SmacPlannerHybrid	0.674	1.00	0.609	-39.1
SmacPlannerLattice	0.673	0.608	0.761	+25.2

Table 3.35: Maximum linear overacceleration (m/s^3) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	6.20	6.53	5.47	-16.2
SmacPlannerHybrid	7.67	6.33	8.27	+30.7
SmacPlannerLattice	5.33	6.07	21.4	+252.6

Table 3.36: Linear overacceleration threshold exceedances — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.075	0.088	0.074	-16.0
SmacPlannerHybrid	0.250	0.191	0.318	+66.5
SmacPlannerLattice	0.197	0.218	0.316	+45.0

Table 3.37: Maximum angular acceleration (rad/s^2) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.0	0.0	0.0	0.0
SmacPlannerHybrid	0.27	0.20	4.27	+2035.0
SmacPlannerLattice	0.07	0.13	1.27	+876.9

Table 3.38: Angular acceleration threshold exceedances — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.707	1.07	0.484	-54.8
SmacPlannerHybrid	0.906	0.546	0.684	+25.3
SmacPlannerLattice	1.84	0.776	0.831	+7.1

Table 3.39: Maximum angular overacceleration (rad/s^3) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	5.6	5.8	1.73	-70.2
SmacPlannerHybrid	16.0	6.73	17.3	+157.1
SmacPlannerLattice	15.5	18.8	21.8	+16.0

Table 3.40: Angular overacceleration threshold exceedances — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.88	4.09	3.80	-7.1
SmacPlannerHybrid	2.14	4.78	2.31	-51.7
SmacPlannerLattice	4.39	5.23	2.50	-52.2

Table 3.41: Minimum distance to people (m) — Episode 3

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.527	0.376	0.413	+9.8
SmacPlannerHybrid	0.119	0.199	0.236	+18.6
SmacPlannerLattice	0.230	0.156	0.0806	-48.3

Table 3.42: Minimum distance to obstacles (m) — Episode 3

3.2.4 Curve trajectory with crossing agents

The **scenario 4** simulates a simple curved path with agents moving in the same direction as the robot. This scenario is simpler compared to the previous ones, but it remains relevant as it represents a common situation encountered in autonomous navigation.

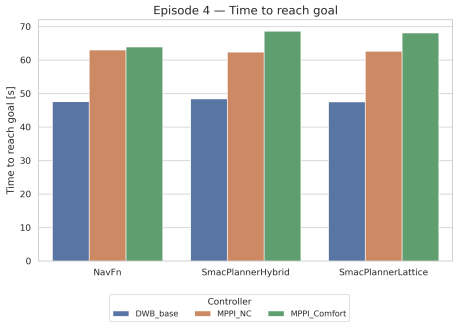

Figure 3.37: Time to reach goal metric for scenario 4

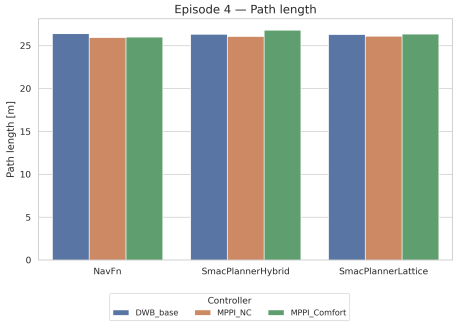

Figure 3.38: Path length metric for scenario 4

As expected, DWB achieves the lowest time to reach the goal for all planners, while the implementation of the comfort critic slightly increases the execution time compared to MPPI by approximately 10%. Regarding path length, a high level of uniformity is observed across all controller–planner combinations, with values ranging between 26.0 m and 26.8 m.

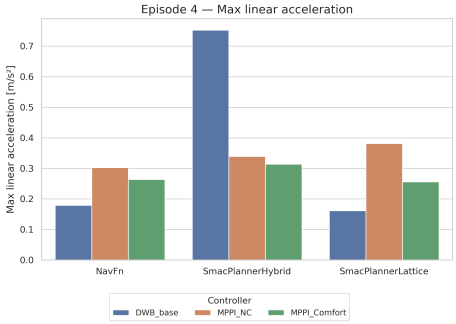

Figure 3.39: Maximum linear acceleration metric for scenario 4

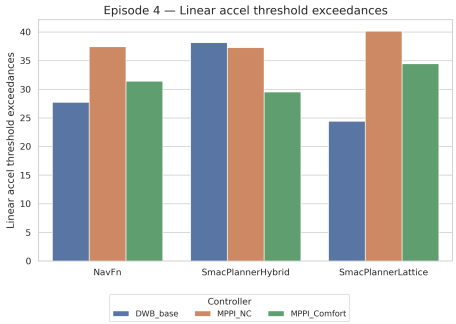

Figure 3.40: Count of linear acceleration threshold exceedances for scenario 4

The DWB controller shows a strong planner dependency, exhibiting the highest peak linear acceleration across configurations, reaching $0.75\,m/s^2$. However, when combined with NavFn and SmacPlannerLattice, DWB achieves better performance than MPPI-based configurations. Nevertheless, the comfort critic consistently reduces peak linear acceleration for MPPI across all planners for both metrics, indicating smoother navigation behavior.

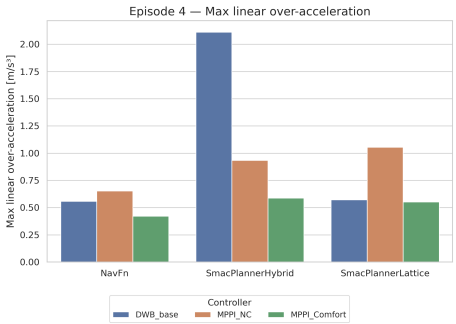

Figure 3.41: Maximum linear over-acceleration metric for scenario 4

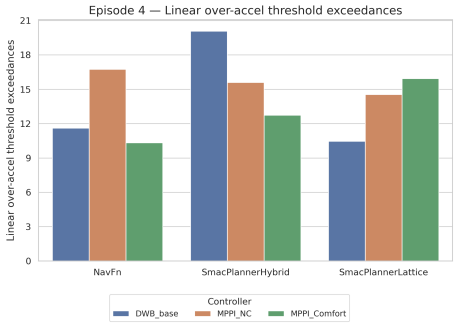

Figure 3.42: Count of linear over-acceleration threshold exceedances for scenario 4

A behavior similar to that observed for linear acceleration is reported for lin-

ear over-acceleration. DWB–SmacPlannerHybrid exhibits the highest peak, exceeding $2\,m/s^3$, while better performance is achieved when combined with NavFn and SmacPlannerLattice. The comfort critic reduces peak linear over-acceleration for MPPI across all planners for both metrics, except for the count of threshold exceedances when combined with SmacPlannerLattice, where a slight increase is observed.

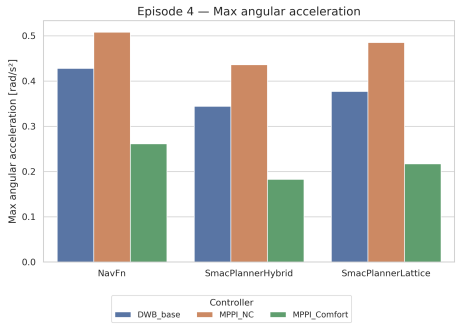

Figure 3.43: Maximum angular acceleration metric for scenario 4

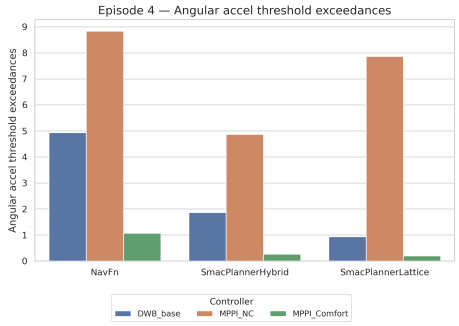

Figure 3.44: Count of angular acceleration threshold exceedances for scenario 4

Angular metrics become particularly relevant in this scenario, as the robot navigates a curved path. As shown in Figure 3.43, MPPI-comfort significantly reduces peak angular acceleration and almost completely eliminates threshold exceedances, resulting in low-magnitude and low-frequency angular transients. In contrast, MPPI-based configurations without the comfort critic exhibit worse performance than DWB across all planner combinations for both metrics.

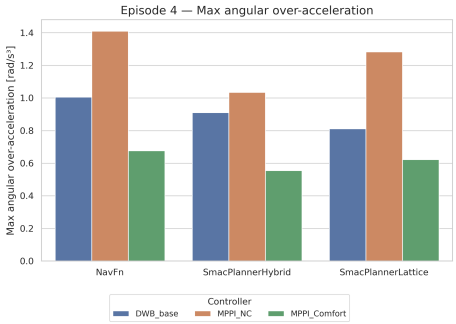

Figure 3.45: Maximum angular over-acceleration metric for scenario 4

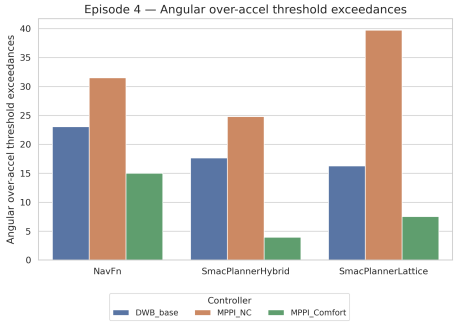

Figure 3.46: Count of angular over-acceleration threshold exceedances for scenario 4

The same trend observed for angular acceleration is also present for angular over-acceleration. It is important to emphasize the consistent reduction achieved by the comfort critic across all controller–planner combinations for both metrics, leading to the best overall performance in this scenario.

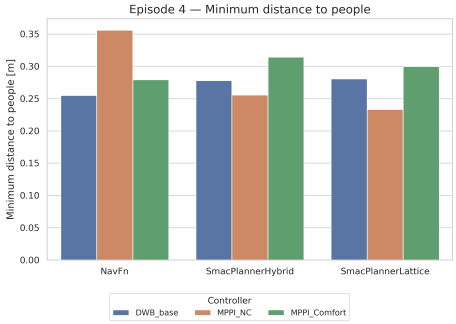

Figure 3.47: Minimum distance to people for scenario 4

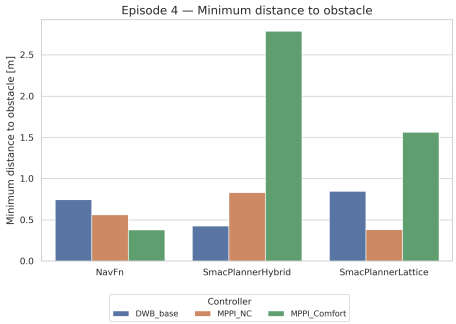

Figure 3.48: Minimum distance to obstacles for scenario 4

Regarding minimum distances to people and obstacles, controller–planner combinations exhibit very similar behavior with respect to distances from people, with MPPI maintaining slightly larger clearances than DWB, both with and without the comfort critic. For minimum distances to obstacles, a noticeable increase is observed for MPPI-comfort when combined with SmacPlanners, suggesting an improved sense of safety in this scenario.

Algorithm	**Success rate**
DWB + NavFn	100%
DWB + SmacPlannerHybrid	100%
DWB + SmacPlannerLattice	100%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	100%
MPPI-NC + SmacPlannerLattice	100%
MPPI-Comfort + NavFn	100%
MPPI-Comfort + SmacPlannerHybrid	100%
MPPI-Comfort + SmacPlannerLattice	100%

Table 3.43: Success rate — Episode 4

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	47.6	63.0	63.9	+1.4
SmacPlannerHybrid	48.4	62.4	68.6	+9.9
SmacPlannerLattice	47.5	62.6	68.1	+8.8

Table 3.44: Time to reach goal (s) — Episode 4

Planner	**DWB**	**MPPI-NC**	**MPPI-Comfort**	**Δ[%]**
NavFn	26.4	26.0	26.0	+0.0
SmacPlannerHybrid	26.3	26.1	26.8	+2.7
SmacPlannerLattice	26.3	26.1	26.4	+1.1

Table 3.45: Path length (m) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.32	2.97	2.02	-32.0
SmacPlannerHybrid	2.00	2.63	1.56	-40.7
SmacPlannerLattice	1.68	3.06	1.91	-37.6

Table 3.46: Cumulative heading changes (rad) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.179	0.302	0.264	-12.6
SmacPlannerHybrid	0.752	0.339	0.314	-7.4
SmacPlannerLattice	0.161	0.381	0.256	-32.8

Table 3.47: Maximum linear acceleration (m/s^2) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	27.7	37.4	31.4	-16.0
SmacPlannerHybrid	38.1	37.3	29.5	-20.9
SmacPlannerLattice	24.4	40.1	34.5	-14.0

Table 3.48: Linear acceleration threshold exceedances — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.558	0.653	0.422	-35.4
SmacPlannerHybrid	2.11	0.934	0.587	-37.1
SmacPlannerLattice	0.571	1.05	0.553	-47.3

Table 3.49: Maximum linear overacceleration (m/s^3) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	11.6	16.8	10.3	-38.7
SmacPlannerHybrid	20.1	15.6	12.7	-18.6
SmacPlannerLattice	10.5	14.5	15.9	+9.7

Table 3.50: Linear overacceleration threshold exceedances — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.428	0.508	0.262	-48.4
SmacPlannerHybrid	0.344	0.436	0.183	-58.0
SmacPlannerLattice	0.377	0.485	0.217	-55.3

Table 3.51: Maximum angular acceleration (rad/s^2) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	4.93	8.83	1.07	-87.9
SmacPlannerHybrid	1.87	4.87	0.267	-94.5
SmacPlannerLattice	0.933	7.87	0.20	-97.5

Table 3.52: Angular acceleration threshold exceedances — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	1.01	1.41	0.676	-52.1
SmacPlannerHybrid	0.911	1.03	0.555	-46.1
SmacPlannerLattice	0.811	1.28	0.623	-51.3

Table 3.53: Maximum angular overacceleration (rad/s^3) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	23.1	31.5	15.0	-52.4
SmacPlannerHybrid	17.7	24.8	3.93	-84.2
SmacPlannerLattice	16.3	39.7	7.53	-81.0

Table 3.54: Angular overacceleration threshold exceedances — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.255	0.356	0.279	-21.6
SmacPlannerHybrid	0.278	0.256	0.314	+22.7
SmacPlannerLattice	0.281	0.233	0.299	+28.3

Table 3.55: Minimum distance to people (m) — Episode 4

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.746	0.563	0.380	-32.5
SmacPlannerHybrid	0.427	0.832	2.79	+235.3
SmacPlannerLattice	0.846	0.382	1.56	+308.4

Table 3.56: Minimum distance to obstacles (m) — Episode 4

3.2.5 Straight navigation with crossing and passing agents

The **scenario 5** simulates a horizontal traverse with agents and static obstacles such as coffee tables. During the simulation, the path passes close to a table, and it may happen that an agent gets stuck against the table, starting to oscillate and blocking the robot. These events are not considered for the evaluation of the metrics.

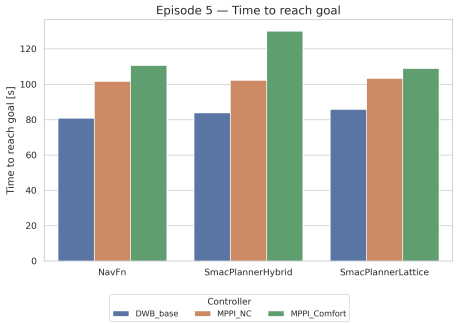

Figure 3.49: Time to reach goal metric for scenario 5

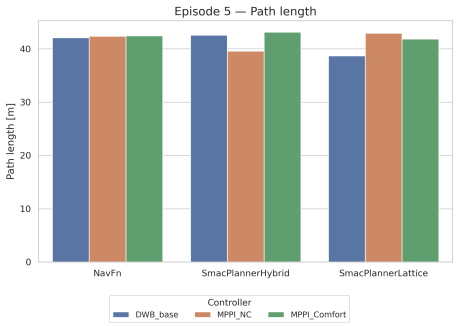

Figure 3.50: Path length metric for scenario 5

As expected, the DWB controller achieves the shortest time to reach the goal, while MPPI and MPPI-comfort increase the execution time. The path length remains comparable across all controller–planner combinations.

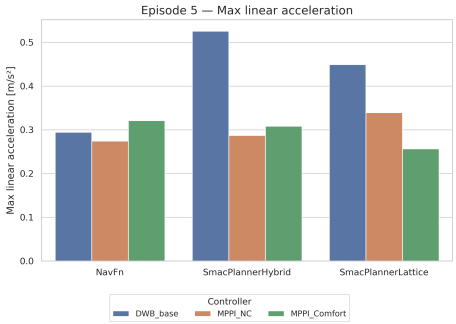

Figure 3.51: Maximum linear acceleration metric for scenario 5

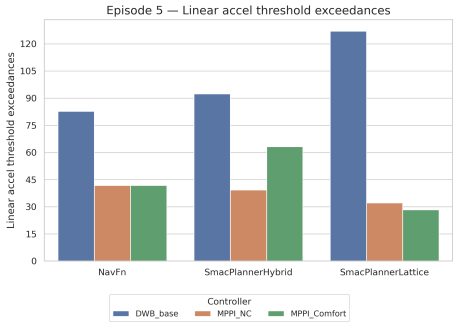

Figure 3.52: Count of linear acceleration threshold exceedances for scenario 5

Results highlight worse performance for the DWB controller when combined with SmacPlanners, reaching $0.53\,m/s^2$ and $0.45\,m/s^2$ for SmacPlannerHybrid and SmacPlannerLattice, respectively. In contrast, MPPI across all planners reaches $0.34\,m/s^2$ in the worst case. The influence of the comfort critic does not significantly affect maximum linear acceleration, with a slight increase when combined with NavFn and SmacPlannerHybrid. Regarding the number of threshold exceedances, DWB exhibits high-frequency violations compared to MPPI, suggesting a less comfortable trajectory. MPPI provides better performance, while the implementation of the comfort critic does not further improve this metric.

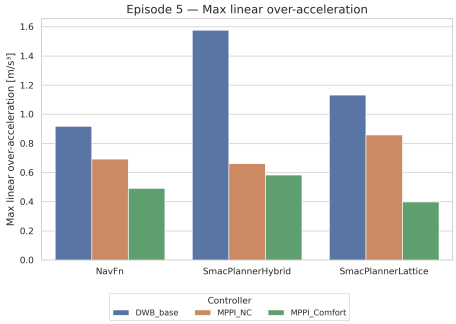

Figure 3.53: Maximum linear over-acceleration metric for scenario 5

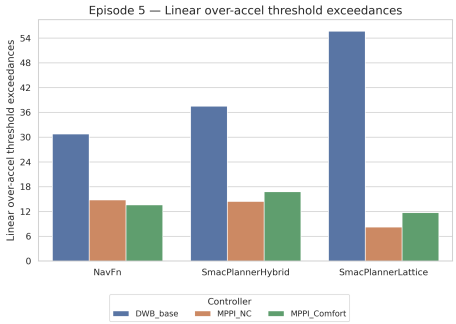

Figure 3.54: Count of linear over-acceleration threshold exceedances for scenario 5

In contrast to linear acceleration, linear over-acceleration peaks are reduced by the comfort critic, particularly when combined with SmacPlannerLattice, where a reduction of 53% is achieved, reaching the lowest value for this metric. The DWB controller exhibits high peak values across all planners, especially when combined with SmacPlannerHybrid, reaching $1.58\,m/s^3$.

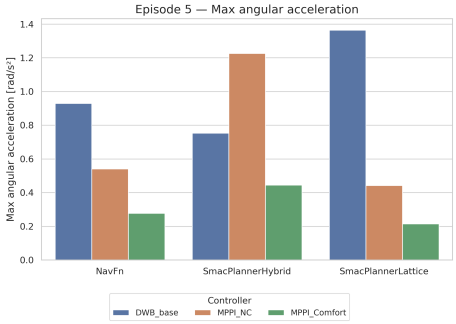

Figure 3.55: Maximum angular acceleration metric for scenario 5

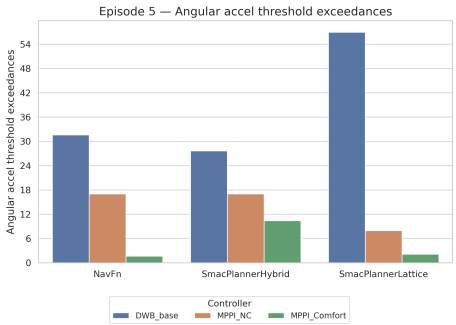

Figure 3.56: Count of angular acceleration threshold exceedances for scenario 5

Despite the absence of a predefined curved path, high angular acceleration peaks are observed. In particular, the DWB controller exhibits the minimum peak

when combined with SmacPlannerHybrid ($0.75\,rad/s^2$) and the maximum peak with SmacPlannerLattice ($1.36\,rad/s^2$). MPPI reduces these peaks, except when combined with SmacPlannerHybrid, where a peak of $1.23\,rad/s^2$ is observed. The implementation of the comfort critic drastically reduces both peak values and the number of threshold exceedances, demonstrating its effectiveness in limiting angular acceleration, promoting smoother rotational behavior, and strongly suppressing frequent high-angular-acceleration events.

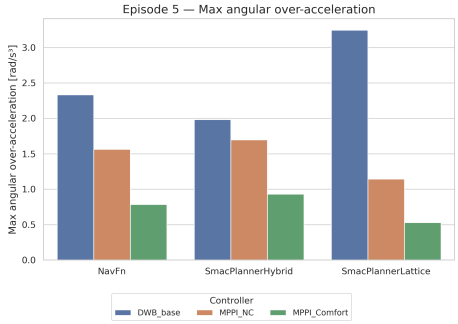

Figure 3.57: Maximum angular over-acceleration metric for scenario 5

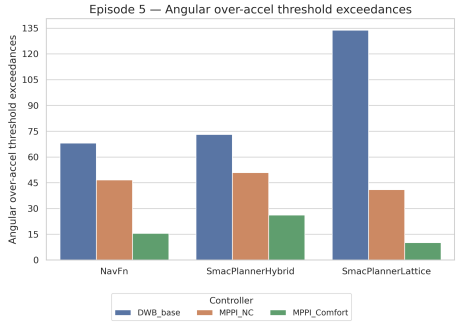

Figure 3.58: Count of angular over-acceleration threshold exceedances for scenario 5

As observed for angular acceleration metrics, angular over-acceleration follows the same trend. DWB exhibits the highest peak values for all planners, while MPPI and, in particular, MPPI-comfort significantly reduce peak magnitudes, suggesting smoother navigation. The strongest reductions are observed for SmacPlannerLattice and NavFn, with decreases of 54% and 50%, respectively. A similar pattern is observed for the number of angular over-acceleration threshold exceedances, where a substantial reduction is achieved, especially for SmacPlannerLattice:

- 134.0 counts for DWB
- 41.0 counts for MPPI
- 10.2 counts for MPPI-comfort

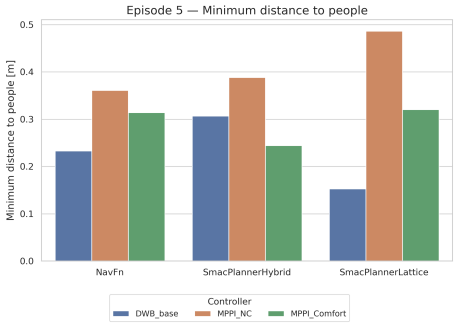

Figure 3.59: Minimum distance to people for scenario 5

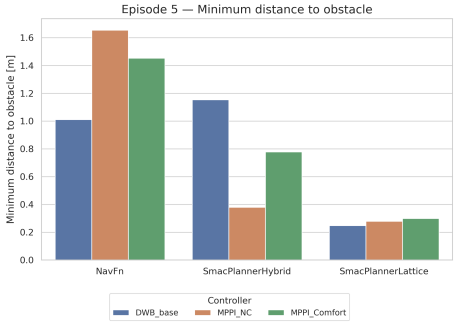

Figure 3.60: Minimum distance to obstacles for scenario 5

For minimum distance metrics, the implementation of the comfort critic generally reduces distances; however, in absolute terms, differences are limited to a few tens of centimeters and can be considered negligible. Across most controller–planner combinations, MPPI without the comfort critic represents the best trade-off in maintaining larger distances from both dynamic and static obstacles. The only exception is SmacPlannerHybrid, where DWB and MPPI-comfort maintain larger distances of $1.15\,m$ and $0.78\,m$, respectively, while MPPI achieves approximately $0.38\,m$.

Algorithm	**Success rate**
DWB + NavFn	93.3%
DWB + SmacPlannerHybrid	93.3%
DWB + SmacPlannerLattice	73.3%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	86.7%
MPPI-NC + SmacPlannerLattice	100%
MPPI-Comfort + NavFn	86.7%
MPPI-Comfort + SmacPlannerHybrid	93.3%
MPPI-Comfort + SmacPlannerLattice	93.3%

Table 3.57: Success rate — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	80.8	102.0	111.0	+8.8
SmacPlannerHybrid	83.8	102.0	130.0	+27.5
SmacPlannerLattice	85.8	103.0	109.0	+5.8

Table 3.58: Time to reach goal (s) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	42.1	42.4	42.5	+0.2
SmacPlannerHybrid	42.5	39.6	43.1	+8.8
SmacPlannerLattice	38.7	42.9	41.8	-2.6

Table 3.59: Path length (m) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	6.27	4.05	2.98	-26.4
SmacPlannerHybrid	5.95	12.1	5.89	-51.3
SmacPlannerLattice	14.7	4.29	3.25	-24.2

Table 3.60: Cumulative heading changes (rad) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.294	0.274	0.321	+17.2
SmacPlannerHybrid	0.525	0.287	0.308	+7.3
SmacPlannerLattice	0.449	0.339	0.257	-24.2

Table 3.61: Maximum linear acceleration (m/s^2) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	82.7	41.8	41.8	+0.0
SmacPlannerHybrid	92.3	39.3	63.2	+60.8
SmacPlannerLattice	127.0	32.2	28.3	-12.1

Table 3.62: Linear acceleration threshold exceedances — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.918	0.693	0.492	-29.0
SmacPlannerHybrid	1.58	0.661	0.584	-11.6
SmacPlannerLattice	1.13	0.858	0.399	-53.5

Table 3.63: Maximum linear overacceleration (m/s^3) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	30.8	14.8	13.6	-8.1
SmacPlannerHybrid	37.5	14.5	16.8	+15.9
SmacPlannerLattice	55.7	8.27	11.7	+41.5

Table 3.64: Linear overacceleration threshold exceedances — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.93	0.541	0.277	-48.8
SmacPlannerHybrid	0.753	1.23	0.445	-63.8
SmacPlannerLattice	1.36	0.442	0.215	-51.4

Table 3.65: Maximum angular acceleration (rad/s^2) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	31.6	17.0	1.67	-90.2
SmacPlannerHybrid	27.7	17.0	10.4	-38.8
SmacPlannerLattice	56.9	7.93	2.13	-73.1

Table 3.66: Angular acceleration threshold exceedances — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.33	1.56	0.785	-49.7
SmacPlannerHybrid	1.98	1.70	0.931	-45.2
SmacPlannerLattice	3.24	1.14	0.529	-53.6

Table 3.67: Maximum angular overacceleration (rad/s^3) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	68.1	46.7	15.5	-66.8
SmacPlannerHybrid	73.1	50.9	26.3	-48.3
SmacPlannerLattice	134.0	41.0	10.2	-75.1

Table 3.68: Angular overacceleration threshold exceedances — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.233	0.361	0.314	-13.0
SmacPlannerHybrid	0.307	0.388	0.244	-37.1
SmacPlannerLattice	0.153	0.486	0.320	-34.2

Table 3.69: Minimum distance to people (m) — Episode 5

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	1.01	1.65	1.45	-12.1
SmacPlannerHybrid	1.15	0.379	0.778	+105.3
SmacPlannerLattice	0.248	0.279	0.299	+7.2

Table 3.70: Minimum distance to obstacles (m) — Episode 5

3.2.6 Curve in narrow corridor with crossing agents

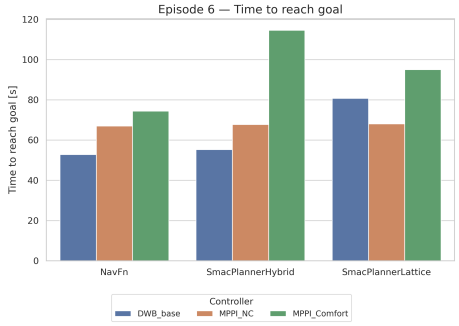

Figure 3.61: Time to reach goal metric for scenario 6

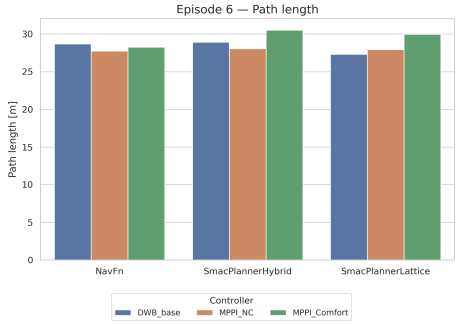

Figure 3.62: Path length metric for scenario 6

As expected, the DWB controller achieves the lowest time to reach the goal across all planners, while MPPI-comfort yields the highest values, suggesting that the comfort critic does not push the robot to move aggressively when this conflicts with passenger comfort. Regarding path length, controller–planner combinations yield values in the range between $27.7\,m$ and $30.5\,m$, with negligible differences.

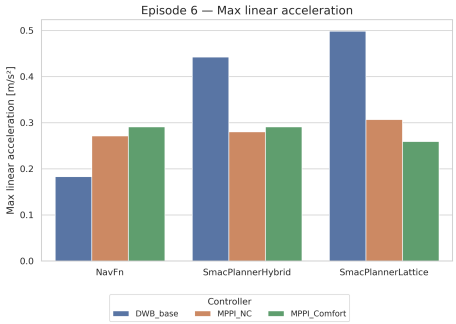

Figure 3.63: Maximum linear acceleration metric for scenario 6

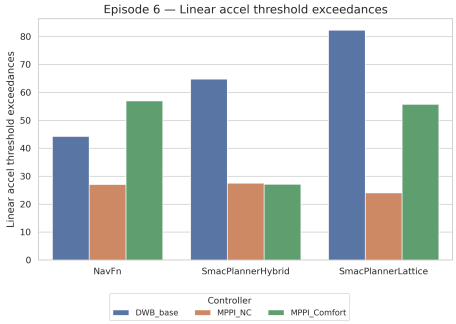

Figure 3.64: Count of linear acceleration threshold exceedances for scenario 6

DWB–NavFn achieves the lowest maximum linear acceleration across all combinations, while DWB combined with SmacPlanners yields the highest values. The implementation of the comfort critic does not lead to significant improvements or reductions in peak acceleration. In contrast, MPPI-comfort doubles the number of threshold exceedances with respect to MPPI without the critic, suggesting similar magnitudes of acceleration peaks occurring more frequently.

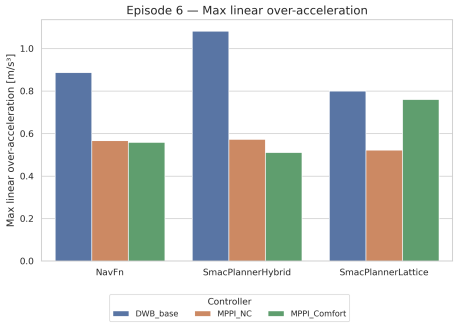

Figure 3.65: Maximum linear over-acceleration metric for scenario 6

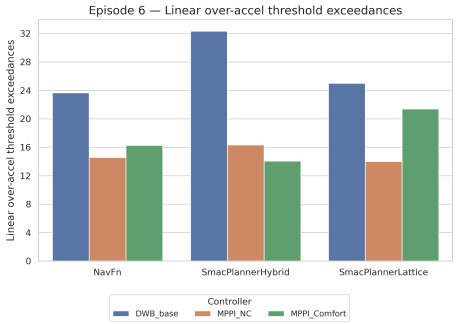

Figure 3.66: Count of linear over-acceleration threshold exceedances for scenario 6

Similarly, for linear over-acceleration metrics, the implementation of the comfort critic does not provide significant improvements. In particular, when combined with SmacPlannerLattice, both metrics increase by approximately 50%. DWB exhibits the worst behavior for both metrics.

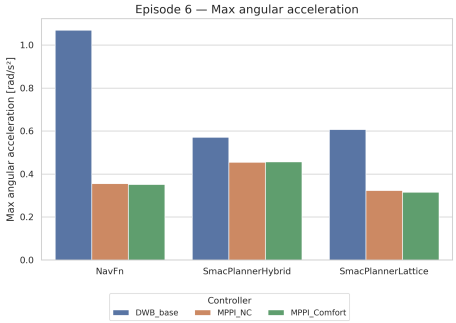

Figure 3.67: Maximum angular acceleration metric for scenario 6

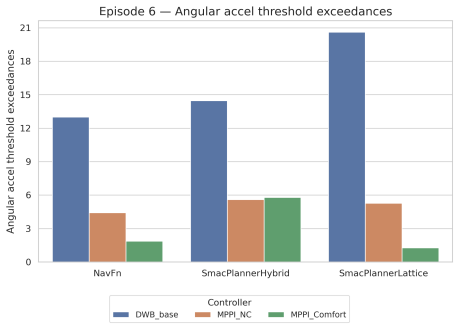

Figure 3.68: Count of angular acceleration threshold exceedances for scenario 6

DWB exhibits the worst behavior also in terms of angular acceleration. For maximum angular acceleration, MPPI without the critic and MPPI-comfort yield comparable peak values. However, despite similar peak magnitudes, the implementation of the comfort critic significantly reduces the number of threshold

exceedances for NavFn and SmacPlannerLattice by 58% and 76%, respectively, while SmacPlannerHybrid is not significantly affected.

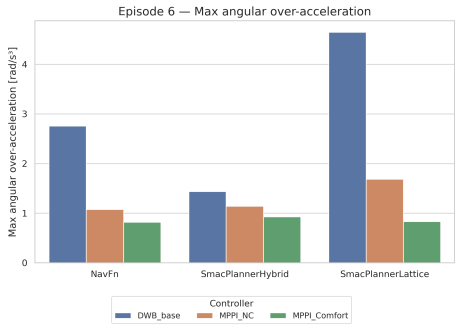

Figure 3.69: Maximum angular over-acceleration metric for scenario 6

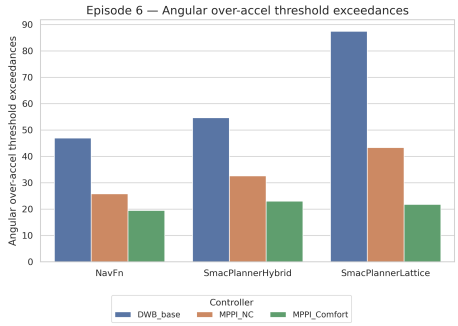

Figure 3.70: Count of angular over-acceleration threshold exceedances for scenario 6

For angular over-acceleration, the implementation of the comfort critic yields improvements for both metrics, reducing peak values for all planners. The most significant improvement is observed with SmacPlannerLattice, where MPPI exhibits a peak of $1.69\,m/s^3$ and MPPI-comfort reduces it to $0.83\,m/s^3$. DWB shows higher peaks across all planners, particularly when combined with NavFn and SmacPlannerLattice, reaching $2.76\,m/s^3$ and $4.65\,m/s^3$, respectively.

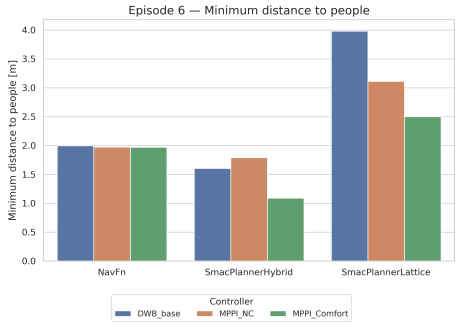

Figure 3.71: Minimum distance to people for scenario 6

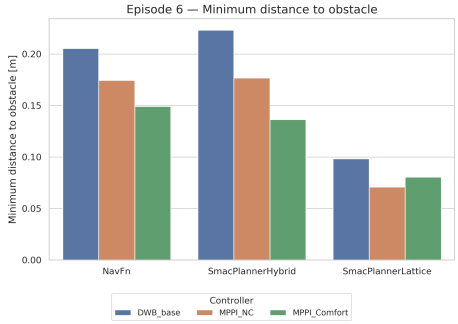

Figure 3.72: Minimum distance to obstacles for scenario 6

For minimum distance metrics, DWB generally maintains larger distances from people and obstacles, while MPPI-comfort reduces these distances by a few centimeters across all planners. An exception is SmacPlannerHybrid, where the distance decreases from $1.79\,m$ to $1.09\,m$ when using MPPI-comfort.

Algorithm	**Success rate**
DWB + NavFn	100%
DWB + SmacPlannerHybrid	100%
DWB + SmacPlannerLattice	86.7%
MPPI-NC + NavFn	100%
MPPI-NC + SmacPlannerHybrid	100%
MPPI-NC + SmacPlannerLattice	100%
MPPI-Comfort + NavFn	93.3%
MPPI-Comfort + SmacPlannerHybrid	100%
MPPI-Comfort + SmacPlannerLattice	100%

Table 3.71: Success rate — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	52.9	67.0	74.4	+11.0
SmacPlannerHybrid	55.3	67.7	115.0	+69.9
SmacPlannerLattice	80.7	68.1	95.0	+39.5

Table 3.72: Time to reach goal (s) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	28.7	27.7	28.3	+2.2
SmacPlannerHybrid	28.9	28.1	30.5	+8.5
SmacPlannerLattice	27.3	28.0	30.0	+7.1

Table 3.73: Path length (m) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	4.05	2.65	3.36	+26.8
SmacPlannerHybrid	6.35	3.18	4.32	+35.8
SmacPlannerLattice	10.4	3.96	5.32	+34.3

Table 3.74: Cumulative heading changes (rad) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.183	0.271	0.291	+7.4
SmacPlannerHybrid	0.442	0.281	0.291	+3.6
SmacPlannerLattice	0.498	0.307	0.259	-15.6

Table 3.75: Maximum linear acceleration (m/s^2) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	44.3	27.1	57.0	+110.3
SmacPlannerHybrid	64.8	27.5	27.1	-1.5
SmacPlannerLattice	82.3	24.1	55.7	+131.1

Table 3.76: Linear acceleration threshold exceedances — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.887	0.567	0.559	-1.4
SmacPlannerHybrid	1.08	0.573	0.511	-10.8
SmacPlannerLattice	0.800	0.523	0.761	+45.5

Table 3.77: Maximum linear overacceleration (m/s^3) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	23.7	14.6	16.3	+11.6
SmacPlannerHybrid	32.3	16.3	14.1	-13.5
SmacPlannerLattice	25.0	14.0	21.4	+52.9

Table 3.78: Linear overacceleration threshold exceedances — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	1.07	0.356	0.351	-1.4
SmacPlannerHybrid	0.572	0.455	0.457	+0.4
SmacPlannerLattice	0.607	0.323	0.316	-2.2

Table 3.79: Maximum angular acceleration (rad/s^2) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	13.0	4.42	1.87	-57.7
SmacPlannerHybrid	14.5	5.60	5.80	+3.6
SmacPlannerLattice	20.6	5.27	1.27	-75.9

Table 3.80: Angular acceleration threshold exceedances — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.76	1.07	0.817	-23.6
SmacPlannerHybrid	1.44	1.14	0.928	-18.6
SmacPlannerLattice	4.65	1.69	0.831	-50.8

Table 3.81: Maximum angular overacceleration (rad/s^3) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	47.0	25.8	19.5	-24.4
SmacPlannerHybrid	54.7	32.7	23.0	-29.7
SmacPlannerLattice	87.5	43.4	21.8	-49.8

Table 3.82: Angular overacceleration threshold exceedances — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	2.00	1.97	1.97	+0.0
SmacPlannerHybrid	1.60	1.79	1.09	-39.1
SmacPlannerLattice	3.98	3.11	2.50	-19.6

Table 3.83: Minimum distance to people (m) — Episode 6

Planner	DWB	MPPI-NC	MPPI-Comfort	Δ[%]
NavFn	0.206	0.174	0.149	-14.4
SmacPlannerHybrid	0.223	0.177	0.136	-23.2
SmacPlannerLattice	0.098	0.0709	0.0806	+13.7

Table 3.84: Minimum distance to obstacles (m) — Episode 6

3.2.7 Composite Score Evaluation

In order to provide an overall comparison across the six scenarios among the selected navigation strategies, three composite indicators were defined:

- Comfort score
- Safety score
- Performance score

Each score aggregates multiple metrics into a single normalized index in the range $[0, 1]$, where higher values correspond to better behavior.
Given a metric measured for a planner P and controller configuration M in a specific episode, min–max normalization is applied:

$$\text{norm}_{episode}(x_{P,M,run}) = \frac{x_{P,M,run} - \min\limits_{P,M}(x)}{\max\limits_{P,M}(x) - \min\limits_{P,M}(x)} \tag{3.1}$$

This transformation maps the worst algorithm to 0 and the best to 1 within the same scenario, preserving the relative ranking while removing scale dependency. For quantities where lower values represent better behavior (e.g., accelerations or time-to-goal), the normalized value is inverted:

$$\text{score}_{episode}(x) = 1 - \text{norm}_{episode}(x) \tag{3.2}$$

Thus, all normalized quantities consistently follow a *higher-is-better* interpretation.
The score is first computed by aggregating the normalized metrics within each run:

$$Score_{episode,run}(P, M) = \sum_{m=1}^{K} w_m \cdot \text{score}_{episode}\left(x^{(m)}_{P,M,run}\right) \tag{3.3}$$

Then, the results are averaged across runs of the same episode:

$$Score_{episode}(P, M) = \frac{1}{R} \sum_{run=1}^{R} Score_{episode,run}(P, M) \tag{3.4}$$

Finally, the global score is obtained by averaging across the six scenarios:

$$Score(P, M) = \frac{1}{6} \sum_{episode=1}^{6} Score_{episode}(P, M) \tag{3.5}$$

where R is the number of runs per scenario and K is the number of metrics. This guarantees that each scenario contributes equally to the final evaluation.

3.2.7.1 Average Comfort Score

The comfort score evaluates motion smoothness and passenger-perceived naturalness. It includes peak accelerations, over-accelerations, and threshold exceedances capturing repeated abrupt motions. Angular components are explicitly considered due to their higher perceptual impact in indoor assistive navigation.
The weights used for the comfort score are reported in Table 3.85.

Table 3.85: Weights used in the comfort score

Metric	Weight
Max linear over-acceleration	0.22
Max angular over-acceleration	0.22
Max linear acceleration	0.14
Max angular acceleration	0.14
Linear acceleration threshold exceedances	0.07
Angular acceleration threshold exceedances	0.07
Linear over-acceleration exceedances	0.07
Angular over-acceleration exceedances	0.07

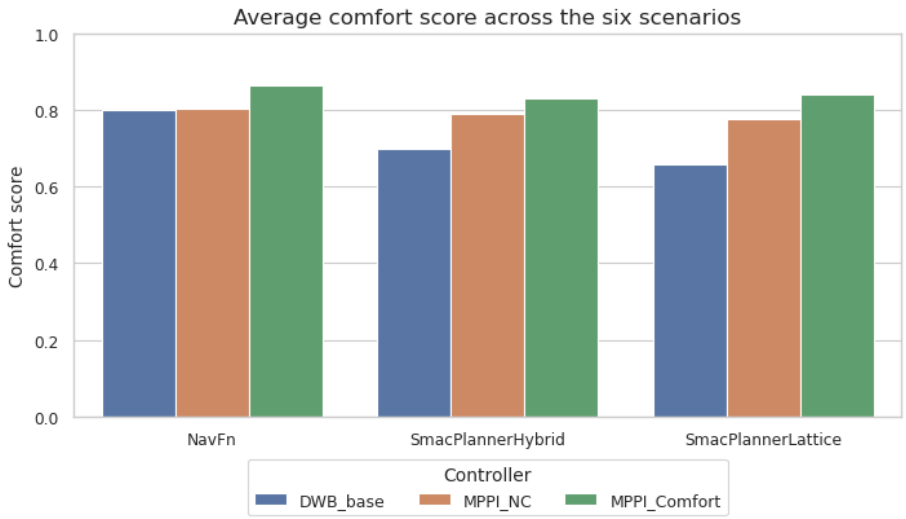

Figure 3.73: Average comfort score across the six navigation scenarios

Figure 3.73 shows a consistent improvement across all global planners when combined with the MPPI controller equipped with the proposed comfort critic. The critic significantly improves motion smoothness while preserving comparable navigation efficiency.

3.2.7.2 Average Safety Score

The safety score evaluates the navigation behavior in terms of proximity to humans and environmental obstacles. It is computed using the minimum distance

to people and the minimum distance to obstacles measured during navigation. Since larger distances correspond to safer behavior, no inversion is applied after normalization.
The weights used for the safety score are reported in Table 3.86.

Table 3.86: Weights used in the safety score

Metric	Weight
Minimum distance to people	0.70
Minimum distance to obstacle	0.30

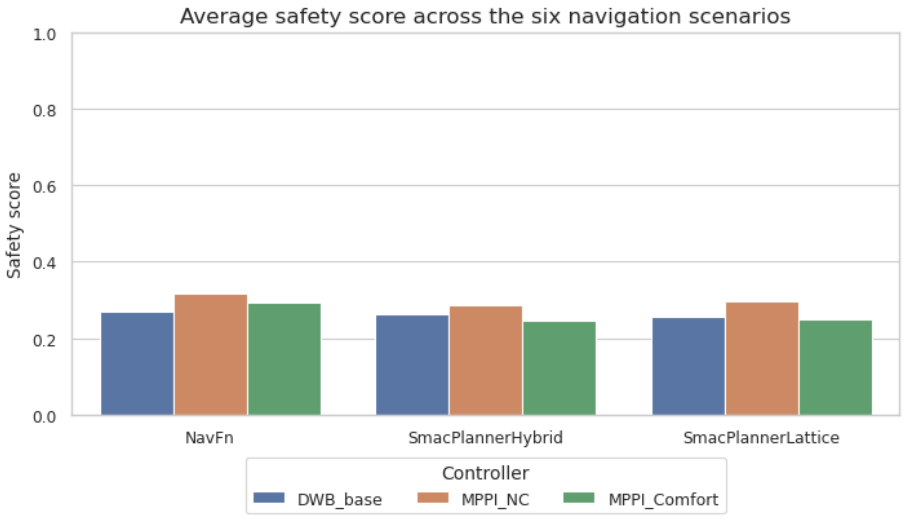

Figure 3.74: Average safety score across the six navigation scenarios

Figure 3.74 shows that the MPPI-based configurations maintain safety margins comparable to or larger than the baseline DWB controller. This indicates safe behavior in crowded environments. The comfort-oriented critic does not degrade safety and preserves consistent distances from surrounding agents across all planners.

3.2.7.3 Average Performance Score

The performance score evaluates navigation efficiency and is computed using the time required to reach the goal and the resulting path length. Since lower values of these quantities correspond to better performance, the normalized metrics are inverted to maintain the higher-is-better interpretation of the score.
The weights used for the performance score are reported in Table 3.87.

Table 3.87: Weights used in the performance score

Metric	Weight
Time to reach goal	0.65
Path length	0.35

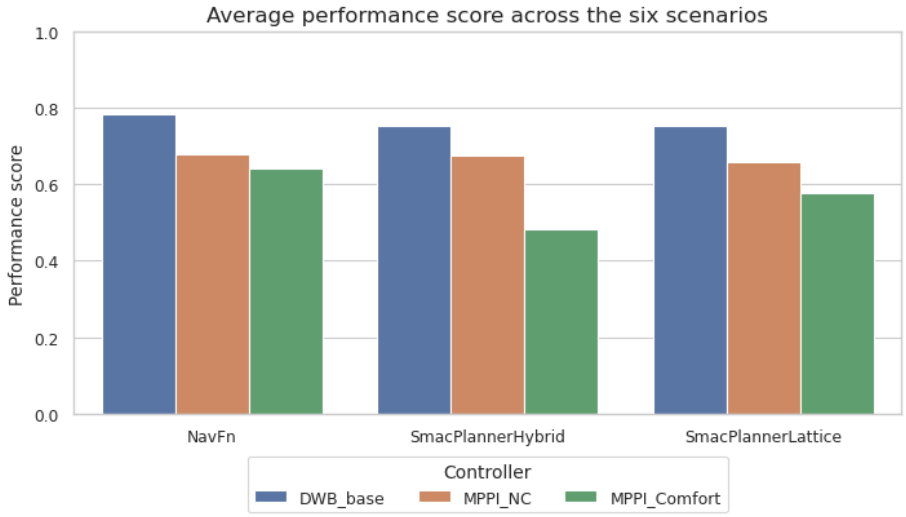

Figure 3.75: Average performance score across the six navigation scenarios

Figure 3.75 shows that the baseline DWB controller achieves the highest efficiency across all planners. The MPPI configurations slightly reduce performance due to smoother and more cautious motion generation. The comfort-oriented critic further emphasizes this behavior, highlighting the expected trade-off between motion naturalness and navigation efficiency.

3.2.8 Temporal metrics evaluation (MPPI+NavFn)

Once the global metrics are evaluated, a temporal evaluation of a run of an episode is proposed and specifically the **third run** of the **episode 1**. The evaluation is done comparing the behavior of the robot with and without the comfort-critic, with MPPI-NavFn as configuration of controller-planner.

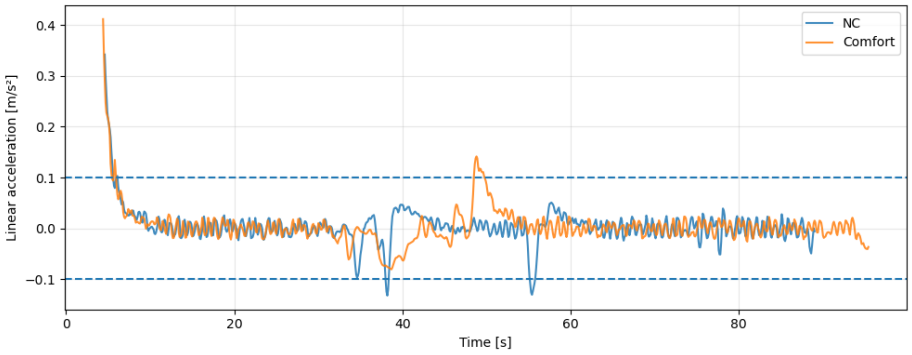

Figure 3.76: Linear acceleration temporal evaluation for episode 1

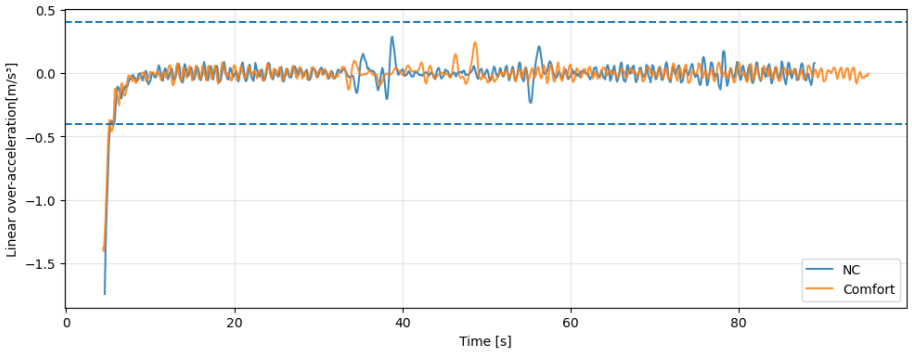

Figure 3.77: Linear over-acceleration temporal evaluation for episode 1

ì

The evaluation of the figure 3.76 can be split into two main part: the beginning of the navigation and the temporal behavior around 30-60 s, where the robot crosses the agents. In both configurations the robot start its navigation at the same time with an important difference: the comfort configuration has an higher linear acceleration , around 0.40 m/s^2, while the basic MPPI is around $0.35m/s^2$. The critical part is when the robot cross the agents and comparing the two behavior, around 40s the MPPI-NC shows several sharp negative acceleration peaks, where they also exceeds the threshold. In contrast the implementation of the critic smooth the behavior, remaining within the threshold bounds for most of the trajectory and a single positive acceleration is observed around 45 s and 55 s. Apart from this event, the profile exhibits reduced peak magnitude and fewer abrupt variations and the peak is positive, which is preferable respect to a negative one, as the MPPI-NC configurations. For the 3.77 both configurations show no exceedances and exhibit a strong initial transient, characterized by a large negative over-acceleration peak (around -1.6 m/s^3, corresponding tho the rapid change in acceleration at motion onset and this transient rapidly decays in few seconds. When the robot crosses the agents, the NC shows more pronounced and frequent spikes, especially in negative direction, while the MPPI-comfort maintains a more regular and attenuated profile,

with a limited number of mild positive peaks, smaller in magnitude than those of the MPPI-NC.

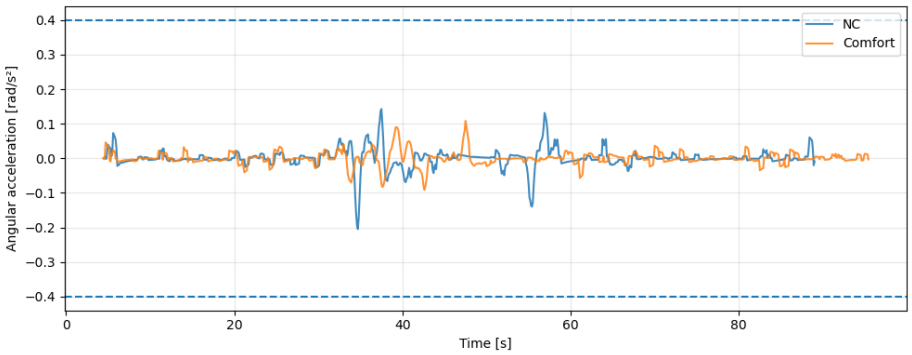

Figure 3.78: Angular acceleration temporal evaluation for episode 1

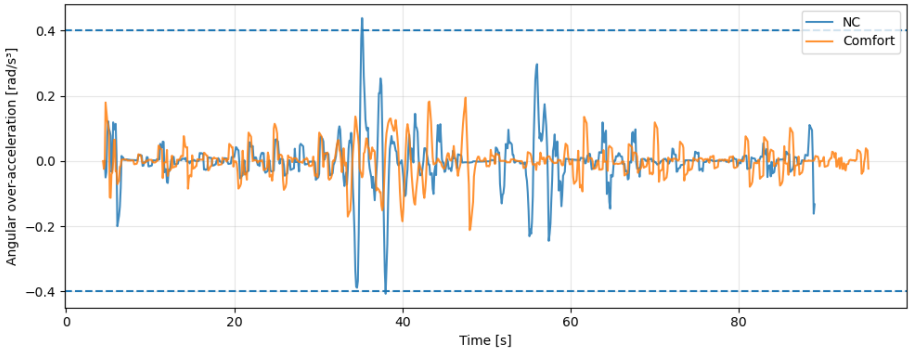

Figure 3.79: Angular over-acceleration temporal evaluation for episode 1

During the initial transient both signals stabilize around zero, as expected due to the absence of curve, obstacles and agents near the robot. Around 30 s and 65 s, the magnitude and frequency of peaks increase for both configurations, in particular for the MPPI-NC configuration, which shows several sharp acceleration peaks, in particular two of them in the negative direction. In contrast, the implementation of the critic exhibits a smoother angular response, with reduced peak in terms of magnitude and frequency and more evenly distributed over time when compared to the MPPI-NC case. For the MPPI-comfort configuation, peaks shows around 35 s and 50 s, while for the MPPI-NC around 35 s and 60 s. The figure 3.79 shows the angular over-acceleration over time and the differences between the two configuration become relevant. During the initial transient both exhibits oscillatory behavior in a range between -0.2 rad/s^3 and + 0.2 rad/s^3, but the MPPI-comfort tends to produce positive peaks, while the MPPI-NC more negative. The critical part is around 35 s and 65 s, where the MPPI-configuration presents several large-amplitude spikes, exceeding the threshold in both negative and positive limits. THe implementation of the critic yields to a more attenuated and regular profile, with smaller oscillations and no

threshold exceedances. Although some peaks are still present in the same time interval, their magnitude is consistently reduced an in particular negative peaks are drastically reduced.

Conclusion

This thesis presented a comprehensive benchmarking study of navigation strategy for mobile robot in socially populated environments as an airport. The evaluation focused on the combined effect of controller architecture and global planner available on the Nav2 stack framework, comparing three controller and three planners. The controller used in this thesis are the DWB, the MPPI and the MPPI implemented with a comfort-oriented critic, while the planners are the NavFn, SmacPlannerHybrid and SmacPlannerLattice. The benchmark was conducted evaluating six distinct episodes, designed to simulate different navigation conditions and interaction patterns with obstacles and agents and the performances are assessed using a set of kinematic and comfort-related metrics, including linear and angular acceleration and over-acceleration, and safety related-distances. The behavior of a run of an episode is evaluated over time in order to get a more comprehensive analysis. The results of the simulation highlight clear and consistent trends, showing that the DWB controller generally achieves lower time-to-goal values and compared to the other controller, confirming its competitively in terms of simplicity and reactivity when combined with simpler planners as the NavFn. DWB-NavFn achieves also the lowest value in terms of maximum peak across all controller-planner combination, at the cost of higher maximum angular values. Generally the combination of DWB controller and complex planners as the SmacPlanners yields to uncomfortable navigation. In contrast the standard MPPI controller, does not lead to the best metric value across the majority of the episode, but yields to produce a better trade-off between all the metrics across all the planners. The main part of the thesis is the evaluation of the influence of the critic to improve the comfort of the passenger during the navigation and the results shows a general improvement with respect to the standard MPPI controller, reducing the peaks on the kinematic metrics at the cost of higher time to reach the goal. The `PreferForwardCritic` pushed the robot to move to the goal, reducing the comfort of the passenger and the elimination of the critic in combination to the `ComfortCritic` improve significantly the comfortability of the navigation. Furthermore the asymmetric weighting reduce the abrupt brakes and could lead to mild positive spike to compensate them, which is also the scope of the critic due to the fact that the same magnitude of peaks, lead to worst sense of comfort when it is negative. Across all controller-planner combinations the one that achieves the best trade-off across all the metrics is the MPPI-NavFn combined with the `ComfortCritic`, the remotion of the `PreferForwardCritic` and the addition of the `TwirlingCritic`, but despite showing the higher average comfort score across, the MPPI-SmacPlannerLattice configuration achieves the lowest value in the angular metrics for 5 out of 6 episodes, drastically augmenting the comfort of the passenger, while for the linear metrics reaches the lower value in two episodes for the acceleration metric and in four for the over-acceleration metric. It is important to emphasize that **the tuning of the parameter is made with the SmacPlannerLattice**, yielding to the be the best global planner when combined with this configuration, but a different tuning could lead to different results, suggesting other planner as the most comfortable. A common behavior exhibited by all three controllers is an unhuman-like reaction when

agents cross the robot's path. During agent–robot interactions, the controller modifies the robot's trajectory to avoid collisions, but frequently curves in the same direction as the agent's motion. As a consequence, this strategy can result in near-collision situations,forcing the robot to brake abruptly and wait until the agent has passed. Addressing this limitation represents a relevant direction for future work aimed at improving passenger comfort. In particular, incorporating agent trajectory prediction could allow the robot to adapt its path proactively, for example by steering in the opposite direction of the agents' motion, thereby enabling smoother and more human-like avoidance behaviors.

Overall, this thesis provides a systematic evaluation of navigation strategies for socially aware mobile robots, highlighting the importance of controller design and comfort-oriented critics in improving passenger experience. The presented results contribute to a better understanding of the trade-offs between efficiency, safety, comfort, and can serve as a foundation for the development of more socially acceptable and human-centered navigation systems in real-world environments.

Bibliography

[1] Eurostat, *Past and future population ageing trends in the eu*, https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Population_structure_and_ageing, Source dataset: demo_pjangroup. Accessed: 18 January 2026, 2024.

[2] S. Galan, *Life expectancy at birth worldwide 1950–2100*, https://www.statista.com/statistics/805060/life-expectancy-at-birth-worldwide/, Accessed: 18 January 2026, Nov. 2025.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics / Sebastian Thrun, Wolfram Burgard, Dieter Fox* (Intelligent robotics and autonomous agents), eng. Cambridge (Mass.): MIT Press, 2005, ISBN: 0-262-20162-3.

[4] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," eng, *IEEE robotics automation magazine*, vol. 4, no. 1, pp. 23–33, 1997, ISSN: 1070-9932.

[5] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," eng, in *Proceedings - IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 1986–1991, ISBN: 1424406013.

[6] E. F. Camacho and C. Bordons, *Model predictive control / E.F. Camacho, C. Bordons* (Advanced textbooks in control and signal processing), eng, 2nd ed. London: Springer, 2004, ISBN: 1-85233-694-3.

[7] C. Mavrogiannis, F. Baldini, A. Wang, *et al.*, "Core challenges of social robot navigation: A survey," eng, 2021.

[8] Y. Morales, N. Kallakuri, K. Shinozawa, T. Miyashita, and N. Hagita, "Human-comfortable navigation for an autonomous robotic wheelchair," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 2737–2743, ISBN: 1467363588.

[9] Y. Morales, T. Miyashita, and N. Hagita, "Social robotic wheelchair centered on passenger and pedestrian comfort," *Robotics and Autonomous Systems*, vol. 87, pp. 355–362, 2017, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2016.09.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092188901630570X.

[10] Y. Morales, J. Even, N. Kallakuri, *et al.*, "Visibility analysis for autonomous vehicle comfortable navigation," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2197–2202. DOI: 10.1109/ICRA.2014.6907162.

[11] Y. Isono, H. Yoshitake, M. Shino, *et al.*, "Autonomous navigation method considering passenger comfort recognition for personal mobility vehicles in crowded pedestrian spaces," eng, in *Computer Vision, Imaging and Computer Graphics Theory and Applications*, ser. Communications in Computer and Information Science, vol. 1815, Switzerland: Springer, 2023, pp. 182–202, ISBN: 9783031457241.

[12] Open Source Robotics Foundation, *Ros concepts*, https://wiki.ros.org/ROS/Concepts, Accessed: 14 October 2025, ROS Wiki, 2025.

[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[14] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2718–2725. DOI: 10.1109/IROS45743.2020.9341207.

[15] S. Macenski, M. Booker, J. Wallace, and T. Fischer, *Open-source, cost-aware kinematically feasible planning for mobile and surface robotics*, 2025. arXiv: 2401.13078 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2401.13078.

[16] D. A. Dolgov, "Practical search techniques in path planning for autonomous driving," 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:16143233.

[17] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Optimal, smooth, nonholonomic mobile robot motion planning in state lattices," Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-15, May 2007.

[18] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," eng, in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005, pp. 3231–3237, ISBN: 0780389123.

[19] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, 341–346 vol.1. DOI: 10.1109/ROBOT.1999.770002.

[20] S. Macenski, T. Moore, D. V. 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," *Robotics and Autonomous Systems*, vol. 168, p. 104 493, Oct. 2023, ISSN: 0921-8890. DOI: 10.1016/j.robot.2023.104493. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2023.104493.

[21] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. 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.

[22] E. A. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and kl control," eng, in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, IEEE, 2012, pp. 1466–1473, ISBN: 9781467320658.

[23] W. Zhang, H. Wang, C. Hartmann, M. Weber, and C. Schütte, "Applications of the cross-entropy method to importance sampling and optimal control of diffusions," eng, *SIAM journal on scientific computing*, vol. 36, no. 6, A2654–A2672, 2014, ISSN: 1064-8275.

[24] I. Bae, J. Moon, J. Jhung, *et al.*, *Self-driving like a human driver instead of a robocar: Personalized comfortable driving experience for autonomous vehicles*, 2022. arXiv: 2001.03908 [eess.SY]. [Online]. Available: https://arxiv.org/abs/2001.03908.

[25] N. Pérez-Higueras, R. Otero, F. Caballero, and L. Merino, "Hunavsim: A ros 2 human navigation simulator for benchmarking human-aware robot navigation," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7130–7137, Sep. 2023, ISSN: 2377-3766. DOI: 10.1109/LRA.2023.3316072.