Topology-Driven Parallel Trajectory Optimization in Dynamic Environments

Oscar de Groot, Laura Ferranti, Dariu M. Gavrila, Javier Alonso-Mora

***Abstract*—Ground robots navigating in complex, dynamic environments must compute collision-free trajectories to avoid obstacles safely and efficiently. Nonconvex optimization is a popular method to compute a trajectory in real-time. However, these methods often converge to locally optimal solutions and frequently switch between different local minima, leading to inefficient and unsafe robot motion. In this work, we propose a novel topology-driven trajectory optimization strategy for dynamic environments that plans multiple distinct evasive trajectories to enhance the robot's behavior and efficiency. A global planner iteratively generates trajectories in distinct homotopy classes. These trajectories are then optimized by local planners working in parallel. While each planner shares the same navigation objectives, they are locally constrained to a specific homotopy class, meaning each local planner attempts a different evasive maneuver. The robot then executes the feasible trajectory with the lowest cost in a receding horizon manner. We demonstrate, on a mobile robot navigating among pedestrians, that our approach leads to faster trajectories than existing planners.**

***Index Terms*—Motion and Path Planning, Optimization and Optimal Control, Collision Avoidance, Constrained Motion Planning**

I. INTRODUCTION

MOBILE robots are being deployed in increasingly more complex environments, for example, to automate logistics in warehouses [1] or mobility through self-driving cars [2]. However, it remains challenging to safely and efficiently navigate complex dynamic environments.

In dynamic environments, a robot must make both high-level and low-level decisions. High-level decisions involve, for example, choosing the general direction for safely avoiding obstacles (e.g., going left or right). Low-level decisions involve, for example, determining the exact shape of a trajectory that is both collision-free and dynamically feasible. While these decisions operate on separate levels of the planning problem, they are often not differentiated, which can degrade planner performance in terms of time efficiency and safety. Existing methods make the high-level decision implicitly [3]–[5], do not distinguish the high-level and low-level decisions [6]−[7], only consider static obstacles in the high-level decision [8] or require a structured environment to make the high-level decision [9]–[11]. We propose a trajectory optimization algorithm that accounts for these two levels of the planning problem explicitly.

The authors are with the Dept. of Cognitive Robotics, TU Delft, 2628 CD Delft, The Netherlands. Email: o.m.degroot@tudelft.nl

This work received support from the Dutch Science Foundation NWO-TTW within the Veni project HARMONIA (18165), and the European Union within the ERC Starting Grant INTERACT (101041863) and the EVENTS project (101069614). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them.

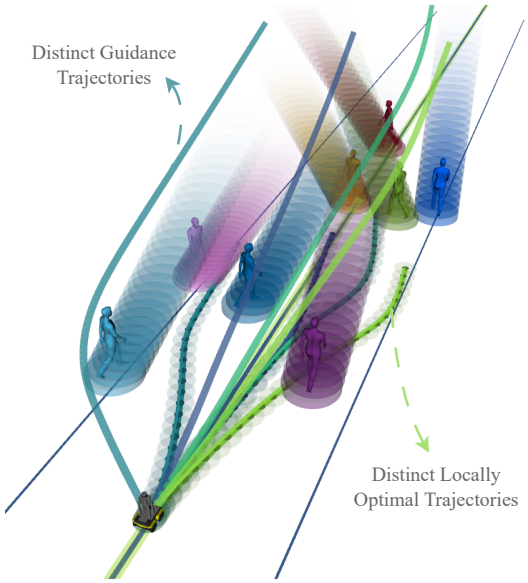


Fig. 1: T-MPC first computes distinct guidance trajectories in the state space (time is visualized in the upwards direction). Each guidance trajectory initializes a local planner, resulting in several distinct locally optimized trajectories. The locally optimized trajectories each pass the obstacles (predicted future motion visualized as cylinders) in a distinct way.

Widely used *optimization-based local planners*, such as Model Predictive Control (MPC) [3], formulate the motion planning problem as an optimization problem that can efficiently compute trajectories satisfying dynamic and collision avoidance constraints. Optimization-based planners make high-level decisions implicitly through the initialization of the optimization and tuning of the cost function. The planner explores only a small set of possible motion plans near the initial guess. An inadequate initial guess may lead to a poor (i.e., slow or non-smooth) trajectory, slow convergence, or infeasibility. When the initial guess is not consistent over multiple planner cycles, the planner can repeatedly switch its high-level decision, leading to indecisive behavior.

Alternatively, *global planners*, such as Randomly exploring Random Trees (RRT*) [7] and motion primitives [6], generate many feasible trajectories, evaluating safety and performance for each. Because they do not distinguish between high-level and low-level decisions, many redundant poor trajectories may be generated, leading to poor-quality motion plans under strict computational limits. This issue is especially prevalent in highly dynamic environments where trajectories need to be computed fast.

In this work, we present a planning framework, which we

refer to as Topology-driven Model Predictive Control (T-MPC) (see Fig. 1), that leverages the strengths of optimization-based planners and global planners.

We present a global planner that identifies several distinct, high-level navigation options by considering the topology of the dynamic collision-free space. The underlying topology allows us to distinguish between similar and distinct trajectories. We then use each high-level trajectory as initialization for an optimization-based planner. The low-level planning problems are independent and are solved in parallel. Our framework does not modify the cost function of the optimization-based planner and can select the executed trajectory by comparing their optimal costs.

II. RELATED WORK AND CONTRIBUTION

Motion planning methods can be divided into local and global planning methods and combinations of these methods.

1) Local Planners: Local planners such as Model Predictive Control (MPC) [3], [12] typically formulate trajectory planning as a nonlinear optimization problem where performance (e.g., progress and smoothness) is optimized under constraints (e.g., dynamic constraints and collision avoidance). MPC can plan time efficient and smooth trajectories and handles various constraints, for example, to account for uncertainty in human behavior [4], [13]–[15]. However, because the collision-free space is nonconvex (obstacle regions are excluded), the optimized trajectory is locally optimal, and there may therefore exist a lower-cost motion plan than the returned solution. This may occasionally result in poor (e.g., slow, non-smooth) trajectories and can prevent MPC from returning a feasible trajectory in time.

To mitigate infeasibility with MPC, some authors propose to use two trajectories where one features as a contingency plan [16], [17] improving planner safety. The planner may still perform poorly when the contingency plan is activated. Alternatively, robustness can be improved by running several optimizations in parallel. For example, in [18], an MPC is parallelized over goal locations, but requires a structured environment and a specific cost function and constraint set.

2) Global Planners: In contrast with local planners, *global planners* do not rely on nonconvex optimization and therefore do not get trapped in local optima. Sampling-based global planners such as RRT* [7] and Probabilistic Roadmaps (PRM) [19] plan by randomly sampling and connecting states in the configuration space until a goal configuration is reached. These methods typically consider static obstacles. In dynamic environments, RRT^x [20] continuously rewires the graph. Recent work [21] greatly improved the computational efficiency of sampling-based planners for high-dimensional problems by using topological abstraction over fiber bundles. Unfortunately, these methods remain computationally demanding when dynamic constraints and collision avoidance are imposed on the problem and may return non-smooth trajectories.

Motion primitive planners (e.g., [6], [22]) instead generate a large number of trajectories that are dynamically feasible by construction. The best trajectory is identified by scoring each trajectory. Motion primitives planners efficiently compute smooth trajectories, but discretize the possible maneuvers which can lead to infeasibility and inefficient robot motion. For static environments, [23] presents a smooth global planner that repairs dynamic mismatches between global plans through trajectory optimization. It is however computationally demanding. Similarly, PiP-X [24] combines graph-search with funnels to find robust dynamically feasible paths but may return inefficient trajectories. In [25], the motion planning problem with collision avoidance is solved via a convex optimization by utilizing Graphs of Convex Sets (GCS). This approach is promising, but is not real-time yet and imposes limitations on the trajectory end point, supported dynamics and constraints.

3) Guidance Planners: Local planners typically receive an initial trajectory or reference path from a global planner. This global planner takes into account static obstacles and the overall route to the goal, which helps prevent the local planner from encountering deadlocks [9], [26]. The performance can be further enhanced by incorporating dynamic obstacles into the global planner. We refer to global planners that consider dynamic obstacles as 'guidance planners'. For example, for self-driving vehicle applications, [27] initializes an MPC in the desired behavior with a simplified Mixed-Integer Linear Program (MILP). In [28], a behavior planner based on a Partially-Observable Markov Decision Process (POMDP) guides a local motion planner in interactive scenarios for a self-driving vehicle. Both methods rely on a structured environment.

To compute a suitable initial guess for a local planner considering obstacles, several authors [8], [11], [29]–[35] have noted that local optima related to collision avoidance link to the topology of trajectories through the collision-free space. Roughly speaking, two trajectories are in the same *homotopy* class if they can be smoothly transformed into each other in the collision-free space [30] (e.g., when they evade the obstacles on the same side). Unfortunately, homotopy classes of trajectories are difficult to compute in general. If the environment can be consistently represented as a graph, then homotopy classes of trajectories can be identified through distinct paths over the graph [29]. This applies, for example, in structured autonomous driving applications through the lane structure of the road network [11], [31].

Without structure in the environment, it is difficult to compute a trajectory in each homotopy class. Graphs can be constructed from static obstacles. In [36], Delauney triangulation is used to identify passable gaps between dynamic obstacles in a global planner. Voronoi graphs are used in [34] to identify homotopy classes with respect to static obstacles and in [8] include each dynamic obstacle and their predicted motion as a static obstacle. Trajectories are generated from the homotopy class description in [37] by modeling interactions as a physical vortex system. These graph-based and generative approaches can exhaust the possible homotopy classes, but scale poorly to crowded environments (as noted in [8] and [37]).

Instead, several works, such as [8], [30], [33], compute distinct trajectories by filtering out homotopy equivalent trajectories during planning. For 3-D navigation among static obstacles, [33] introduces Universal Visibility Deformation (UVD) to compare trajectories. Trajectories are UVD

equivalent if they can be connected without collision at several intermediate times. The authors present a visibility-PRM [38] to generate UVD-distinct trajectories. In 2D dynamic environments, homotopy classes are typically compared via winding numbers [39] or the H-signature [30]. *Winding numbers* track the relative angle between the robot and dynamic obstacles over their trajectories. They were used in [34] to distinguish homotopy classes of trajectories with respect to dynamic obstacles. In [35], an MPC with winding numbers in the cost function was proposed to motivate passing progress. The *H-signature* uses *homology* classes as an approximation for homotopy classes. The work in [8], which relates most closely to this work, applied this approximation for 2-D navigation among static obstacles. Their planner, Time Elastic Band (TEB) Local Planner, identifies several trajectories in distinct homology classes (using regular PRM) and uses each to initialize a soft-constrained optimization-based planner. TEB has, however, three main limitations that can hinder its performance in dynamic environments. Firstly, the trajectory topology is confined to the static workspace, treating dynamic obstacles and their future motion as static obstacles. Secondly, the guidance planner is designed to reach a single goal. Lastly, the local planner lacks hard constraints.

In this work, we introduce a topology-guided planner that is different from these existing works in four ways. First, we consider homotopy classes in the *dynamic* collision-free space, that includes time, to incorporate the motion of dynamic obstacles (contrary to [8], [33]). Second, our framework does not modify the cost function (i.e., the performance criteria) of the local planner (contrary to [18], [25], [35]). Third, our planner does not rely on a structured environment (contrary to [11], [28], [31]). Finally, our guidance planner can handle the case where its goal is blocked (contrary to [8], [33]) by considering multiple goal positions.

In addition, we *enforce* the final trajectories to be in distinct homotopy classes using constraints in the local planner and we show that it is not sufficient to initialize the solver in a homotopy class (contrary to [8]). By consistently planning distinct trajectories, we can reidentify trajectories of prior planning iterations and use this information to make the planner more consistent and decisive. Our method furthermore supports the H-signature, winding numbers and UVD for comparing homotopy classes.

A. Contribution

In summary, our topology-driven parallel planning framework, T-MPC, contributes to the state of the art as follows:

1) A planning framework for dynamic environments that optimizes trajectories in multiple distinct homotopy classes in parallel. Our framework extends existing optimization-based local planners, improving their time efficiency, safety and consistency.
2) A fast guidance planner that computes homotopy distinct trajectories through the dynamic collision-free space towards multiple goal positions.

We validate our proposed framework in simulation on a mobile robot navigating among interactive pedestrians. We show how our framework can accommodate different trajectory optimization approaches (e.g., [3] in the nominal case, and [4] to accommodate Gaussian uncertainties added to the motion of the dynamic obstacles). We show how our framework enhances the performance of [3], [4] out of the box and we compare against three additional baselines ([6], [8], [32]). We finally demonstrate our planner in the real world on a mobile robot navigating among five pedestrians. Our C++/ROS implementation of T-MPC will be released open source.

This work is an extension of our earlier conference publication [32]. In [32], we computed a single guidance trajectory and followed it with a local planner by adding a tracking term. Compared to [32], we compute and optimize multiple distinct guidance trajectories in parallel. In addition, we derive constraints from the guidance trajectory such that the cost of the optimization is unmodified and can be used to compare optimized trajectories. Finally, we improved the robustness and consistency of the guidance planner and extended the experimental evaluation.

The rest of this work is organized as follows. We introduce the planning problem in Sec. III. The planning framework is described and analyzed in Sec. IV. Simulation and real-world results are presented in Sec. V and Sec. VI, respectively, followed by a discussion in Sec. VII.

III. PROBLEM FORMULATION

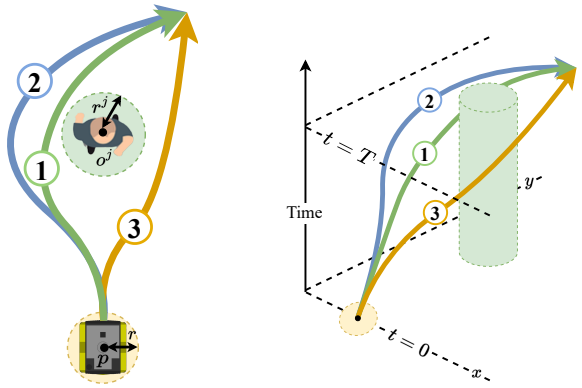
We consider discrete-time nonlinear robot dynamics

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{1}$$

where $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$ and $\boldsymbol{u}_k \in \mathbb{R}^{n_u}$ are the state and input at discrete time instance k, n_x and n_u are the state and input dimensions respectively and the state contains the 2-D position of the robot $\boldsymbol{p}_k = (x_k, y_k) \in \mathbb{R}^2 \subseteq \mathbb{R}^{n_x}$.

The robot must avoid moving obstacles in the environment. The position of obstacle j at time $k = 0$ is denoted $\boldsymbol{o}_0^j \in \mathbb{R}^2$ and we assume that for each obstacle, predictions of its positions over the next N time steps are provided to the planner (i.e., $\boldsymbol{o}_1^j, \dots, \boldsymbol{o}_N^j$) at each time instance. The collision region of the robot is modeled by a disc of radius r and that of each obstacle j by a disc with radius r^j (see Fig. 2a).

For high-level planning with dynamic collision avoidance, we consider the simplified state space $\mathcal{X} \coloneqq \mathbb{R}^2 \times [0, T]$, with $[0, T]$ a continuous finite time domain (see Fig. 2b). The area of the workspace occupied by the union of obstacles at time t is denoted by $\mathcal{O}_t \subset \mathbb{R}^2$ and the obstacle set in the state space is thus $\mathcal{O} \coloneqq \bigcup_{\forall t \in [0,T]} (\mathcal{O}_t, t) \subset \mathcal{X}$. The collision free state space (or *free space*) is denoted $\mathcal{C} \coloneqq \mathcal{X} \backslash \mathcal{O}$. A trajectory is a continuous path through the state space, $\boldsymbol{\tau} : [0, 1] \to \mathcal{X}$. The goal of the robot is to traverse along a given reference path $\boldsymbol{\gamma} : [0, 1] \to \mathbb{R}^2$ without colliding with the obstacles while tracking a reference velocity v_{ref}. It is allowed to deviate from the path.



(a) Planning scene. (b) State space view.

Fig. 2: (a) Depiction of the planning problem and (b) equivalent in the state-space. Trajectory 1 and 2 are in the same homotopy class while trajectory 1 and 3 are in distinct homotopy classes.

A. Optimization Problem

We formalize the planning problem as the following trajectory optimization problem over a horizon of N steps

$$\min_{\boldsymbol{u}\in\mathbb{U},\boldsymbol{x}\in\mathbb{X}} \quad \sum_{k=0}^{N} J(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ \forall k \tag{2b}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \tag{2c}$$

$$g(\boldsymbol{x}_k, \boldsymbol{o}_k^j) \leq 0,\ \forall k, j, \tag{2d}$$

where the cost function J in (2a) expresses the planning objectives (e.g., following reference path $\boldsymbol{\gamma}$). Robot dynamics and initial conditions are imposed by (2b) and (2c), respectively and collision avoidance constraints are imposed by (2d).
Because dynamic obstacles puncture holes in the free space, the free space associated with the constraints (2d) is nonconvex. Nonlinear optimization algorithms, solving this problem, return just one of possibly many local optimal trajectories. The initial guess provided to them determines which local optimal trajectory is returned. It is generally unclear how close this trajectory is to the globally optimal trajectory (i.e., the best trajectory under the specified cost). In this work, we want to leverage this weakness to explore in parallel multiple locally optimal trajectories (provided as initial guesses on $\boldsymbol{x}$) that evade obstacles in a distinct way.

B. Homotopic Trajectories

To achieve the goal above, we rely on the concept of homotopic trajectories, which can be formalized as follows:

Definition 1. [30] (Homotopic Trajectories) Two paths connecting the same start and end points $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ respectively, are homotopic if they can be continuously deformed into each other without intersecting any obstacle. Formally, if $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2 \in T$ represent two trajectories, with $\boldsymbol{\tau}_1(0) = \boldsymbol{\tau}_2(0) = \boldsymbol{x}_s$ and $\boldsymbol{\tau}_1(1) = \boldsymbol{\tau}_2(1) = \boldsymbol{x}_g$, then $\boldsymbol{\tau}_1$ is homotopic to $\boldsymbol{\tau}_2$ iff there exists a continuous map $\eta : [0,1] \times [0,1] \to \mathcal{C}$ such that $\eta(\alpha, 0) = \tau_1(\alpha) \forall \alpha \in [0,1],\ \eta(\beta, 1) = \tau_2(\beta), \forall \beta \in [0,1]$ and $\eta(0, \gamma) = \boldsymbol{x}_s,\ \eta(1, \gamma) = \boldsymbol{x}_g \forall \gamma \in [0,1]$.

If two trajectories are homotopic, they are said to be in the same homotopy class. An example is depicted in Fig. 2. To distinguish between trajectories in different homotopy classes, we make use of the homotopy comparison function

$$\mathcal{H}(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j, \mathcal{O}) = \begin{cases} 1, & \boldsymbol{\tau}_i, \boldsymbol{\tau}_j \text{ in the same homotopy class} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Verifying whether two trajectories are in the same homotopy class can be computationally inefficient. We support the H-signature [30], winding numbers [39] and UVD [33] that allow us to approximately perform this verification in real-time. Details of the three methods are provided in Appendix A.

IV. TOPOLOGY-DRIVEN MODEL PREDICTIVE CONTROL

In this section, we propose T-MPC, a topology-guided planner that optimizes trajectories in multiple distinct homotopy classes in parallel.
Our planner consists of two components: a high-level guidance planner and multiple identical low-level local planners (see Fig. 3). The *guidance planner* G generates homotopy distinct trajectories through the free space

$$G(\boldsymbol{x}_0, \mathcal{P}_g, \mathcal{C}) = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_P\} =: \mathcal{T}_P, \tag{4}$$

where $\boldsymbol{x}_0$ denotes the robot initial state and $\mathcal{P}_g$ denotes a set of goal positions. Each *local planner* is initialized with one of the guidance trajectories and optimizes the trajectory in the same homotopy class. With N the horizon of the guidance and local planners[1], each local planner defines a mapping $L : \mathcal{X}^N \to \mathcal{X}^N$,

$$L(\boldsymbol{\tau}_i) = \boldsymbol{\tau}_i^*. \tag{5}$$

To ensure that the local planner optimizes in the provided homotopy class, we append a set of constraints derived from the guidance trajectory. These constraints are appended to existing collision avoidance constraints to adapt the planner to the globalized framework. The proposed planner computes locally optimal trajectories $\mathcal{T}_P^* := \{\boldsymbol{\tau}_1^*, \ldots, \boldsymbol{\tau}_P^*\}$ in several distinct homotopy classes.

A. Guidance Planner - Overview

The goal of the guidance planner is to quickly compute several homotopy distinct trajectories through the free space. Similarly to [8], [33], we perform this search via Visibility-Probabilistic RoadMaps (Visibility-PRM [38]), a sampling-based global planner. The modifications that we make ensure that the graph remains consistent over successive iterations.
The guidance planner is outlined in Algorithm 1 and visualized in Fig. 4. Details of the algorithm are given in Sec. IV-B. We give a high-level overview here. First, **Visibility-PRM** constructs a sparse graph through the state space from the robot position to a set of goals, where each connection is

[1]The guidance planner horizon could extend beyond the horizon of the local planner. We set them equal for simplicity.

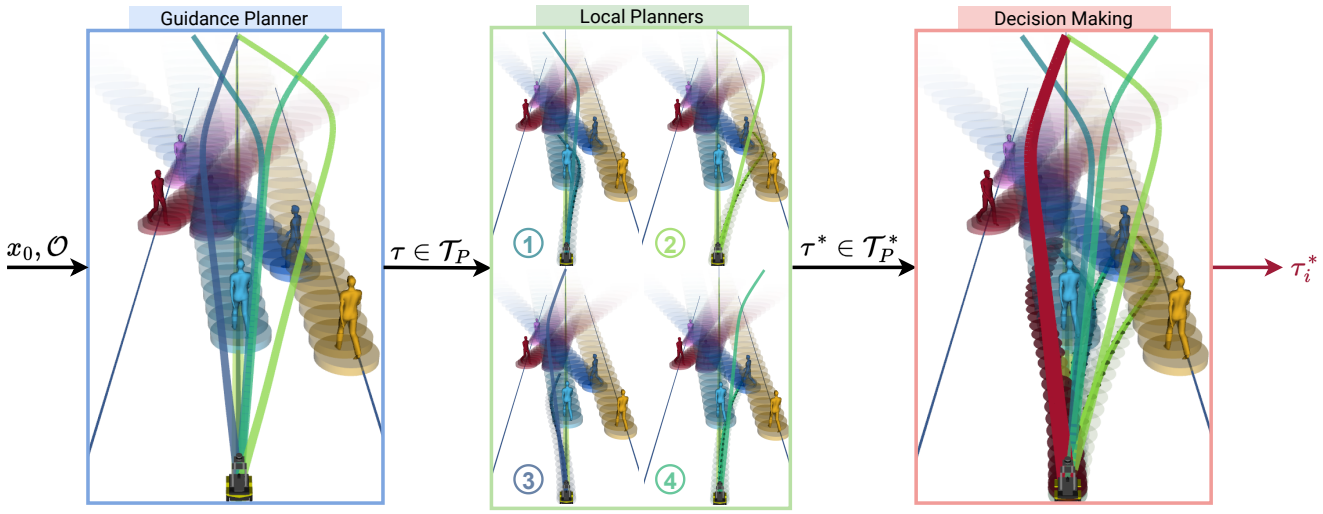
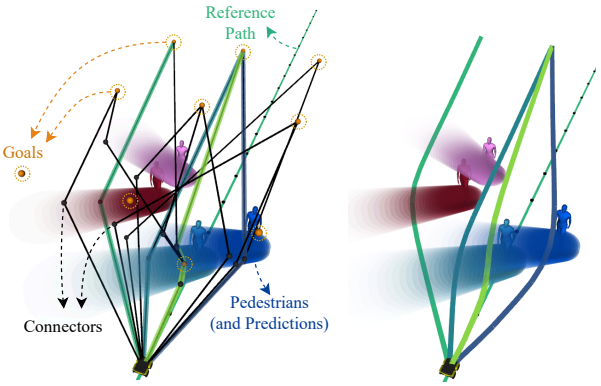


Fig. 3: Schematic of T-MPC. An environment with several obstacles and a robot is visualized in x, y, t (time in the upwards axis). Obstacle motion predictions are denoted with cylinders. (1) A guidance planner (Sec. IV-A) finds $P = 4$ trajectories (visualized with colored lines) from the robot initial position to one of the goals. Each of these trajectories is in a distinct homotopy class in the state space. (2) Each trajectory guides a local planner (Sec. IV-C) as initial guess and through a set of homotopy constraints. Four guidance trajectories and optimized trajectories (as occupied regions for each step) are visualized. (3) The optimized trajectories are compared through their objective value (Sec. IV-E) and a single trajectory (in red) is excuted by the robot.



(a) (b)

Fig. 4: Illustration of the guidance planner in the state-space (time in the upwards axis). Visualization follows Fig. 3. (a) The visibility-PRM graph (black lines and dots) explores the free space toward the goals placed at $t = T$ around the reference path (orange dots). The homotopy distinct guidance paths (colored lines) are obtained by searching the graph. (b) The final trajectories are smoothened.

homotopy distinct (line 1). The goals represent end points for the guidance planner and are placed along the reference path. For each goal, **DepthFirstSearch** (line 2) searches in this graph for the shortest P trajectories that reach it. Any homotopy equivalent trajectories are filtered out by **FilterAndSelect** (line 3), ensuring that the remaining trajectories are in distinct homotopy classes. The P trajectories that seem most promising are selected by a heuristic that prefers its goal to be as close as possible to the reference path at the reference velocity (see Fig. 4a). **IdentifyAndPropagate** (line 4) verifies

Algorithm 1: Guidance Planner

Input: $\mathcal{C}$, $\boldsymbol{x}_0$, $\boldsymbol{x}_N$, previous graph $\mathcal{G}^-$, previous trajectories $\mathcal{T}_P^-$

1 $\mathcal{G} \leftarrow$ **Visibility-PRM**$(\mathcal{C}, \boldsymbol{x}_0, \boldsymbol{x}_N, \mathcal{G}^-)$

2 $\{\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_{N_{GS}}\} \leftarrow$ **DepthFirstSearch**$(\mathcal{G})$

3 $\mathcal{T}_P = \{\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_P\} \leftarrow$ **FilterAndSelect**$(\{\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_{N_{GS}}\})$

4 $\mathcal{G}^- \leftarrow$ **IdentifyAndPropagate**$(\{\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_P\}, \mathcal{T}_P^-)$

Output: $\mathcal{T}_P$

if any of the selected trajectories are equivalent to trajectories of the previous planning iteration. This reidentification makes it possible to follow the same passing behavior over multiple planning iterations. We finally propagate the nodes in the Visibility-PRM graph by lowering their time state by the planning time step.

Through this process, we obtain in each iteration P piecewise linear trajectories $\mathcal{T}_P = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_P\}$ that each connects the robot position to one of the goals (see Fig. 3). We finally smoothen these trajectories and fit cubic splines to make them differentiable (see Fig. 4b). More details can be found in [32]. The smoothening procedure produces only a small displacement in trajectories to maintain their homotopy class. These trajectories serve as initializations for the local planners, described in Sec. IV-C.

B. Guidance Planner - Detailed Description

We detail each step of Algorithm 1 in the following.

Visibility-PRM computes sparse paths through the free space by randomly sampling positions and creating either a *Guard* or *Connector* node at the sampled position. The type of node depends on the number of Guards that it can directly connect to without colliding (i.e., which Guards are *visible*). A *Guard*

is added if no other Guards are visible. A *Connector* (see black dots in Fig. 4) is added when exactly two Guards are visible and its connection to the Guards is feasible (e.g., satisfying velocity and acceleration limits). Similar to [33], we also check if any Connectors link to the same Guards (referred to as *neighbors*). If there are neighbors, we keep the new connection if it is distinct from existing connections, which we verify with homotopy comparison function (3). If it is equivalent and more efficient than the existing connection (e.g., if its connection is shorter), then we replace the existing connector with the new connector. In regular visibility-PRM [38], the graph is initialized with a Guard at the start and goal positions and new nodes are drawn up to a time or node limit. More details of the algorithm can be found in [32, Algorithm 1].

Multiple Goals in Visibility-PRM: In this work, we address the limitation that a single goal must be reached by Visibility-PRM, which causes the planner to fail when that goal cannot be reached. We propose to add a Goal node type to Visibility-PRM. Goals inherit the properties of Guards but are inserted initially and are likely visible to each other. When a Connector can connect to multiple Goals, we single out the Goal with the lowest distance to the point on the reference path reached with the reference velocity (i.e., our ideal goal). By supporting multiple goals, we increase the robustness of the guidance planner. In practice, we deploy a grid of goals centered around the reference path (see Fig. 4).

Homotopy Comparison: We use the homotopy comparison function (3) to verify if two trajectories are in the same homotopy class. We implemented (3) with the H-signature [30], winding numbers [39] and UVD [33]. Appendix A provides details on these methods. In our experiments, we use the H-signature that joins the two trajectories to be compared into a loop and verifies if that loop encircles any moving obstacles. If it does, then the two trajectories pass obstacles differently and belong to different homotopy classes.

DepthFirstSearch searches for P paths to each goal, with each search implemented similar to [8, Algorithm 1].

FilterAndSelect uses homotopy comparison function (3) to remove equivalent trajectories to different Goals found by DepthFirstSearch. The set of filtered trajectories $\mathcal{T}_F$ therefore satisfy

$$\mathcal{H}(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j, \mathcal{O}) = 0, \ \forall i, j, i \neq j, \ \boldsymbol{\tau}_i, \boldsymbol{\tau}_j \in \mathcal{T}_F. \tag{6}$$

The P lowest cost trajectories in $\mathcal{T}_F$ constitute the output $\mathcal{T}_P$.

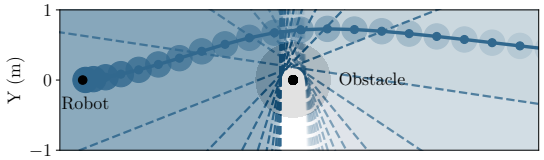
IdentifyAndPropagate uses homotopy comparison function (3) to link new trajectories to trajectories found in the previous iteration. It checks for each previous trajectory $\boldsymbol{\tau}_i^- \in \mathcal{T}_P^-$ if

$$\exists \boldsymbol{\tau}_j \in \mathcal{T}_P, \mathcal{H}(\boldsymbol{\tau}_i^-, \boldsymbol{\tau}_j) = 1. \tag{7}$$

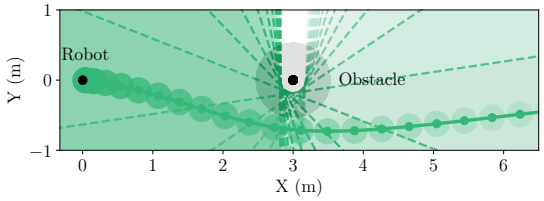
A unique identifier, tied to the homotopy class, is passed from $\boldsymbol{\tau}_i^-$ to $\boldsymbol{\tau}_j$ if the latter exists. We can use this identifier to decide which trajectory to follow (see Sec. IV-E).

C. Local Planner

To refine the trajectories of the guidance planner, we apply P local planners in parallel. Each local planner refines one of the guidance trajectories $\boldsymbol{\tau}_i$ and needs to ensure that the final trajectory is dynamically feasible and that it satisfies any other imposed constraints. We pose the following definition:



(a) Local planner 1 plans to evade the obstacle left.



(b) Local planner 2 plans to evade the obstacle right.

Fig. 5: Two distinct locally planned trajectories for a robot (black dot) evading an obstacle (black region and dot) that is *static* ($\boldsymbol{o}_k = \boldsymbol{o}_0, \ k = 1, \ldots, N$). For both planners, we depict the topology constraints for each time step in their respective colors showing the constraint boundaries (broken lines) and their feasible region (colored regions with increasing transparency over time).

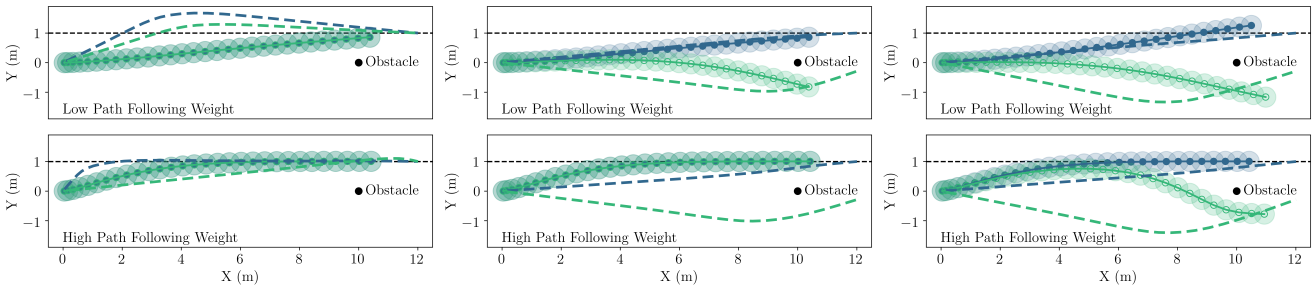
Definition 2. (Local Planner) The local planner is an algorithm $L : \mathcal{X}^N \to \mathcal{X}^N$ that respects constraints.

This definition captures many existing optimization-based planners. In this work, we define the local planner through the trajectory optimization in Eq. (2), where we make two modifications to ensure that the optimized trajectories are in the homotopy class of the associated guidance trajectory. First, the trajectory optimization of each local planner uses its guidance trajectory as the initial guess for $\boldsymbol{x}$. The initial guess speeds up convergence but does not guarantee that the optimized trajectory remains in the same homotopy class when there are obstacles. In the next section (Sec. IV-D), we provide an example where initialization in distinct homotopy classes still leads to identical optimized trajectories.

To ensure that the homotopy class of the guidance trajectory is respected, we add to each local planner a set of constraints $g_H(\boldsymbol{x}_k, \boldsymbol{o}_k^j, \boldsymbol{\tau}_{i,k})$. For this purpose, we construct for each time instance k and obstacle j a linear constraint between the guidance trajectory and obstacle position (see Fig. 5). With guidance trajectory $\boldsymbol{\tau}_i$ and obstacle trajectory $\boldsymbol{o}$, these constraints are given by $\boldsymbol{A}_k \boldsymbol{x}_k \leq b_k$, where

$$\boldsymbol{A}_k = \frac{\boldsymbol{o}_k - \boldsymbol{\tau}_{i,k}}{||\boldsymbol{o}_k - \boldsymbol{\tau}_{i,k}||}, \quad b_k = \boldsymbol{A}_k^T (\boldsymbol{o}_k - A_k(\beta(r + r_{\text{obs}}))). \tag{8}$$

The relaxation factor $0 \leq \beta \leq 1$ scales the distance that the constraints enforce from each obstacle. A key observation is that, with other collision avoidance constraints in place, the topology constraints can be relaxed ($\beta \approx 0$) such that they are inactive at the obstacle boundary. Since the constraints do ensure that the trajectory remains on the same side of each



(a) Without homotopy comparison (3). (b) Without constraints (9e). (c) Our proposed method.

Fig. 6: Planned trajectories (lines with shaded discs) tracking a reference path (dashed black line) while avoiding a static obstacle with two guidance trajectories (dashed lines) for a low (0.01) and high (0.3) path following weight. (a) Without homotopy comparison (3), guidance trajectories are not distinct and optimized trajectories are identical. (b) Without homotopy constraints, increasing the path following weight results in identical trajectories. (c) With homotopy comparison (3) and homotopy constraints (9e), optimized trajectories are distinct.

obstacle, the optimized trajectory is in the same homotopy class as the initialization provided by the guidance planner.
The resulting homotopy preserving local planner is given by

$$J_i^* = \min_{\boldsymbol{u}\in\mathbb{U},\boldsymbol{x}\in\mathbb{X}} \quad \sum_{k=0}^{N} J(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{9a}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ \forall k, \tag{9b}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \tag{9c}$$

$$g(\boldsymbol{x}_k, \boldsymbol{o}_k^j) \leq 0\ \forall k, j \tag{9d}$$

$$g_H(\boldsymbol{x}_k, \boldsymbol{o}_k^j, \boldsymbol{\tau}_{i,k}) \leq 0\ \forall k, j. \tag{9e}$$

The topology constraints and initialization of the optimization realize the local planning mapping of Eq. (5) which, as a function of the guidance trajectory, returns a distinct local optimal trajectory (see Fig. 3).

D. Enforcing Consistency over Time

Our proposed method computes distinct trajectories through two algorithmic features: The guidance trajectories are distinct with respect to homotopy comparison function (3) and the homotopy constraints (9e) ensure that the local planner does not change the homotopy class during optimization. We illustrate the necessity of these two components with an example.
Consider a planning scenario with a robot and static obstacle (both at $y = 0$) and the reference path at $y = 1$. We plan the robot's trajectory for a low and high weight on following the path[2]. Fig. 6 shows the planned trajectories after optimization. Without homotopy comparison (3) (see Fig. 6a), guidance trajectories are not distinct and lead to identical optimized trajectories. Without homotopy constraints (9e) (see Fig. 6b), trajectories are distinct for a low path following weight but become identical by increasing the path following weight. This is possible as the final state of the optimized trajectory is free to move to the other side of the obstacle. Our proposed method (see Fig. 6c) maintains the two trajectories in both cases, irrespective of the tuning of the objective function.

[2]The path following weight is the contouring weight from [3].

E. Decision Making

The robot can only execute one trajectory. Since the cost function of the local planners (9a) matches that of the original trajectory optimization (2a), the quality of the guided plans are directly comparable[3] through their optimal costs J_i^*.
The local planners output P optimized trajectories

$$\mathcal{T}_P^* = \{\boldsymbol{\tau}_1^*, \ldots, \boldsymbol{\tau}_P^*\}. \tag{10}$$

Since each local planner minimizes the same cost function, the lowest cost trajectory[4]

$$\boldsymbol{\tau}_i^*, \quad i = \text{argmin}_i\ J_i^*, \tag{11}$$

is the best trajectory under the specified objective. We refer to executing $\boldsymbol{\tau}_i^*$ as obtained from (11) as the *minimal cost* decision.
In practice, frequently switching the homotopy class of the executed trajectory can degrade motion planning performance and lead to collisions even if, in each time instance, the selected trajectory attains the lowest cost. We therefore consider a generalization of the decision-making process where the previously selected trajectory is given precedence. This is possible as we maintain a consistent set of trajectories in distinct homotopy classes where the previously executed trajectory is marked. This *consistent* decision is given by

$$\boldsymbol{\tau}_i^*, \quad i = \text{argmin}_i\ w_i J_i^*, \tag{12}$$

where $w_i = c_i$ if this trajectory was previously selected, with c_i a constant $0 \leq c_i \leq 1$, and $w_i = 1$ otherwise. If $c_i = 0$, then the planner will pick the trajectory with the same homotopy class of the previous iteration, while for $c_i = 1$, we recover the minimal cost decision. In practice, this decision-making scheme improves navigation behavior over consecutive iterations.

[3]As trajectory end points can be distinct, their quality needs to be represented in the cost function. We include a terminal cost that accounts for the deviation of the end point from the reference path.

[4]We set $J_i^* = \infty$ when the optimization is infeasible.

F. Theoretical Analysis

In the following, we formalize to what extent our proposed planner resolves the nonconvexity of the free space. First, note that due to the cost function and/or nonlinear robot dynamics, the trajectory optimization in Eq. (2) remains nonconvex, even when it is constrained to stay in a single homotopy class. There may therefore be multiple local optima in each homotopy class. This means that the proposed planner does not provably return a globally optimal solution to the optimization in Eq. (2). We propose instead a weaker notion of globalization[5].

Definition 3. (Homotopy Globally Optimal) Denote the highest-cost local-optimum of optimization (2) in homotopy class i as $\boldsymbol{\tau}_i^-$. A trajectory $\boldsymbol{\tau}$ is said to be a Homotopy Globally Optimal (HGO) solution if its cost is lower or equal to that solution in each homotopy class, that is, if $J(\boldsymbol{\tau}) \leq J(\boldsymbol{\tau}_i^-),\ \forall i$, for all homotopy classes that admit a feasible trajectory.

To prove when the proposed scheme computes an HGO solution, we pose three conditions. These conditions link the solution of Eq. (9) to that of Eq. (2).

Condition 1. The homotopy constraints are not active ($g_H(\boldsymbol{x}_k, \boldsymbol{o}_k^j, \boldsymbol{\tau}_{i,k}) < 0\ \ \forall i, k, j$) in the final solution of (9).

Condition 2. The guidance planner finds a trajectory in each homotopy class where a dynamically feasible trajectory exists.

Condition 3. The executed trajectory is selected via (11).

Theorem 1. If Conditions 1-3 hold, T-MPC is HGO for optimization problem (2).

Proof. Under Condition 1, the solution for each optimization (9), $\boldsymbol{\tau}_i^*$, is locally optimal for (2) since homotopy constraints (9e) are the only distinction between the two problems. Therefore, $J(\boldsymbol{\tau}_i^*) \leq J(\boldsymbol{\tau}_i^-)$. If Condition 2 is satisfied, then $\mathcal{T}_P$ contains a guidance trajectory in every feasible homotopy class. Therefore, under Condition 3, the final trajectory $\boldsymbol{\tau}^*$ executed by T-MPC satisfies $J(\boldsymbol{\tau}^*) \leq J(\boldsymbol{\tau}_i^*) \leq J(\boldsymbol{\tau}_i^-),\ \forall i$ and we obtain the HGO property. □

This shows under what conditions the proposed planner finds a provably HGO trajectory. Although these conditions are useful for analysis, they are not necessarily satisfied in practice. Condition 1 can fail if there is no local optimum in a homotopy class or when the linearization around the guidance trajectory restricts the optimization.

Condition 2 is hard to guarantee in crowded environments. In 2-D navigation with M obstacles, there can be 2^M homotopy classes that do not wind around obstacles. Although robot dynamic constraints and bundled obstacles may reduce this amount in practice, the number of classes can still be too large. Limiting the planner to P classes allows us to plan in real-time, but the executed trajectory may not be HGO.

Condition 3 ensures that the lowest cost trajectory is executed but can lead to non-smooth driving behavior over consecutive iterations and it may be preferable to use (12) instead.

While these conditions may not always be satisfied and the HGO property is not provably obtained in each iteration, we will show that the proposed planner always improves on the local planner in isolation.

[5]In the following, with some abuse of notation, $J(\boldsymbol{\tau})$ refers to the optimal cost of the optimization initialized with trajectory $\boldsymbol{\tau}$.

G. Non-Guided Local Planner in Parallel

The constraints and initialization provided by the guidance planner allow the local planner to escape poor local optima. Once the planner is in the correct homotopy class, the restrictions imposed by the guidance planner (i.e., homotopy constraints) may degrade performance. For this reason, we consider an extension of the proposed planner where the regular local planner without guidance (i.e., the optimization in Eq. (2)) is added to the set of parallel guided local planners. Since this planner is less restricted and does not rely on the global planner, it can occasionally find a better solution.

Next to practical benefits, this allows us to trivially establish that the proposed scheme does not achieve a higher cost solution than the local planner in isolation.

Theorem 2. Consider the planner in Fig. 3 that includes a non-guided planner with solution $\bar{\tau}^*$. If a trajectory is selected according to (11), then $J(\boldsymbol{\tau}^*) \leq J(\bar{\boldsymbol{\tau}}^*)$.

Proof. Decision (11) picks the lowest cost solution from $J(\boldsymbol{\tau}_0^*), \ldots, J(\boldsymbol{\tau}_P^*), J(\bar{\boldsymbol{\tau}}^*)$, which cannot exceed $J(\bar{\boldsymbol{\tau}}^*)$. □

If the guided plans are always higher or equal cost compared to the non-guided planner, then this planner architecture reduces to the local planner (the non-guided planner is always selected). If they ever have a lower cost, then guidance *must* improve the planner in the sense that it reduces the cost of the executed trajectory. We will show in the following section that the latter holds true. We refer to the method where the non-guided local planner is added in parallel as T-MPC++.

H. Computation Time Analysis

T-MPC plans guidance trajectories before optimization. In the following, we analyze the computational complexity of the guidance planner (see Algorithm 1), considering the number of PRM samples n, obstacles M and distinct trajectories P.

Visibility-PRM: The time complexity of regular Visibility-PRM is dominated by the visibility check. When adding a node, it checks its visibility in the worst case to all nodes (if all nodes are Guards), where each visibility check considers all obstacles. Its time complexity therefore is $O(n^2M)$. We additionally verify that new connections are distinct. The time complexity of a single homotopy comparison is $O(M)$: the H-signature or winding numbers are evaluated for each obstacle. The homotopy class is compared roughly n times if a new distinct connection neighbors all connectors. Its time complexity therefore is $O(n^2M)$ and does not change the time complexity of Visibility-PRM.

DepthFirstSearch: Each node links to at most one goal. Hence searching the graph for at most P paths to each goal at worst considers each node once. Its time complexity is $O(n)$.

FilterAndSelect: Sorting P trajectories has time complexity $O(P \log P)$. Filtering homotopy distinct trajectories from the sorted list must compare a trajectory to P others in the worst case and has time complexity $O(P^2M)$.

IdentifyAndPropagate: Similarly, comparing the homotopy class of new and existing trajectories has time complexity $O(P^2M)$. Propagating the graph has time complexity $O(n)$. *Total:* The time complexity of the guidance planner is $O((n^2+P^2)M)$. In practice, this time complexity can be approximated by $O(n^2M)$, given that the number of relevant homotopy classes is typically small (i.e., $n \gg P$). Thanks to the propagation of nodes from the previous iteration, we find that n can be relatively small as well (e.g., $n < 100$). For our use case, a relatively small n and P are usually sufficient to construct a sparse graph from which the relevant homotopy classes can be extracted.

V. SIMULATION RESULTS

In the following, we compare our planner against several baselines on a mobile robot navigating among pedestrians.

A. Implementation

Our implementation for T-MPC is written in C++/ROS and will be released open source[6]. We will also release the guidance planner as a standalone package.

For the deterministic simulations we implement the optimization-based planner LMPCC [3] as local planner. The robot dynamics follow second-order unicycle dynamics [40]. Its objective, with weights w, is given by[7]

$$J = w_c J_c + w_l J_l + w_v J_v + w_\omega J_\omega + w_a J_a \qquad (13)$$

for each time instance k in the horizon N. Herein, J_c, J_l are the contour and lag error used to follow the reference path, $J_v = ||v - v_{\text{ref}}||_2^2$ tracks a desired velocity and $J_\omega = ||\omega||_2^2$, $J_a = ||a||_2^2$ weigh the control inputs consisting of the rotational velocity ω and acceleration a. Collision avoidance constraints are imposed with $g(\boldsymbol{x}_k, \boldsymbol{o}_k^j) \leq 0$,

$$g(\boldsymbol{x}_k, \boldsymbol{o}_k^j) = 1 - (\Delta \boldsymbol{p}_k^j)^T \boldsymbol{R}(\phi)^T \begin{bmatrix} \frac{1}{r^2} & 0 \\ 0 & \frac{1}{r^2} \end{bmatrix} \boldsymbol{R}(\phi)(\Delta \boldsymbol{p}_k^j), \quad (14)$$

here $\Delta \boldsymbol{p}_k^j = \boldsymbol{p}_k - \boldsymbol{o}_k^j$, $\boldsymbol{R}(\phi)$ is a rotation matrix with orientation ϕ of the robot and $r = r_{\text{robot}} + r_{\text{obs}}$. These nonconvex constraints directly formulate that the robot region should not overlap with that of the obstacles. We solve each parallel local optimization with Forces Pro [41]. Parameters of the full planner are listed in Table I. Weights of the guidance and local planners are manually tuned. The planning scheme, including guidance and local planners, is updated in each iteration in a receding horizon manner.

B. Simulation Environment

The first simulation environment (see Fig. 1) consists of a mobile robot (Clearpath Jackal) moving through a 6 m wide corridor with up to 12 pedestrians. The robot follows the centerline with a reference velocity of 2 m/s and is controlled at 20 Hz. The pedestrians follow the social forces model [42] using implementation [43]. They interact with other pedestrians and the robot and are aware of the walls. We use a constant velocity model to predict the future pedestrian positions for the planner. The pedestrians have a radius of 0.3 m. We specify a radius of 0.4 m in the planners to account for discretization effects, allowing us to clearly identify collisions. Pedestrians spawn on two sides of the corridor with the objective to traverse the corridor. The random start and goal locations are the same for each planner.

C. Comparison to Baselines

We compare T-MPC and T-MPC++ against four baselines. Baselines are selected on the availability of an open-source implementation and their application to navigation in 2D dynamic environments. We consider the following baselines:

- **Motion Primitives** *(global planner)* [6]: A non-optimization-based global planner that respects the robot dynamics.
- **TEB Local Planner** *(topology-guided planner)* [8]: One of the most used local planners in the ROS navigation stack [44] that considers multiple homotopy classes.
- **LMPCC** *(local planner)* [3]: An open-source non parallelized MPC (see Sec. V-A). We supply the previous solution shifted forward in time as the initial guess of the optimization.
- **Guidance-MPCC** *(topology-guided planner)* [32]: Our previous conference work. We updated the guidance planner to that used in this work to make it more competitive with T-MPC++.

We use the same weights for the MPC planners (LMPCC, Guidance-MPCC, T-MPC, T-MPC++). Baseline actuation limits and tracking objectives are adapted to match the MPC objectives. TEB Local Planner tuning uses its default but with increased collision avoidance weight (from 10 to 20) to decrease collisions in crowded environments.

We perform the simulations with $4, 8$ and 12 pedestrians.

Evaluation Metrics. We compare the planners on the following metrics.

1) *Task Duration:* The time it takes to reach the end of the corridor.

TABLE I: Experimental settings.

Parameter Name	Parameter Value	Parameter Description
N	30	Global and local planner horizon
ΔT	0.2 s	Integration time step
h	0.05 s	Planning time step
n	30	Visibility-PRM sample limit
T_{max}	10 ms	Visibility-PRM time limit
Eq. 3	H-signature	Homotopy comparison function
P	4	# of distinct guidance trajectories
G	5×5	Grid of goals (longitudinal $\times$ lateral)
r	0.725 m	Combined obstacle and robot radius
w_c	0.05	Optimization contouring weight
w_l	0.75	Optimization lag weight
w_v	0.55	Optimization velocity tracking weight
w_ω	0.85	Optimization rotational velocity weight
w_a	0.34	Optimization acceleration weight
Decision	Eq. (12)	Type of decision-making
c_i	0.75	Discount factor for trajectory in previously followed homotopy class

[6]See https://github.com/tud-amr/mpc_planner

[7]We do not use the repulsive forces around obstacles from [3] as they lead to more conservative plans and slow down the optimization problem.

TABLE II: Quantative results for interactive navigation simulations of Sec. V-C over 200 experiments with pedestrian motion *prediction* following a constant velocity model. Task duration (Dur.) and runtime are reported as "mean (std. dev.)". Without obstacles, the task duration is 12.9 s. Best planner performances per column are denoted in **bold**. Underlined results indicate that T-MPC++ significantly outperforms the respective method as tested with a U-test for a significance value of $p = 0.001$.

# Ped.	Method	Dur. [s]	Safe (%)	Runtime [ms]
0	-	12.9 (0.0)	-	-
4	Frenét-Planner [6]	14.0 (0.9)	77	11.6 (2.6)
	TEB Local Planner [8]	**13.0** (1.1)	**100**	**5.8** (5.2)
	LMPCC [3]	13.1 (0.4)	98	11.3 (2.6)
	Guidance-MPCC [32]	**13.0** (0.4)	92	13.9 (1.3)
	T-MPC (ours)	**13.0** (0.2)	**100**	18.3 (3.5)
	T-MPC++ (ours)	**13.0** (0.1)	**100**	19.4 (3.7)
8	Frenét-Planner [6]	15.1 (1.7)	64	11.6 (2.4)
	TEB Local Planner [8]	13.8 (1.7)	**98**	**7.5** (5.1)
	LMPCC [3]	13.8 (1.3)	96	13.7 (4.2)
	Guidance-MPCC [32]	**13.2** (0.7)	92	13.4 (1.4)
	T-MPC (ours)	13.3 (0.7)	96	20.2 (4.8)
	T-MPC++ (ours)	**13.2** (0.6)	96	21.4 (4.9)
12	Frenét-Planner [6]	16.5 (2.4)	42	14.1 (6.3)
	TEB Local Planner [8]	14.9 (2.4)	92	**8.7** (5.4)
	LMPCC [3]	14.0 (1.5)	90	12.9 (4.5)
	Guidance-MPCC [32]	**13.6** (1.1)	86	13.6 (1.6)
	T-MPC (ours)	14.1 (1.3)	90	18.3 (5.1)
	T-MPC++ (ours)	**13.6** (1.0)	**93**	20.1 (5.4)

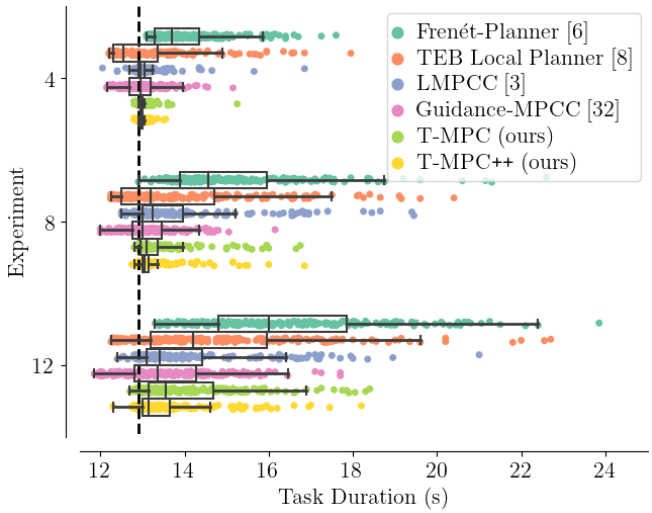
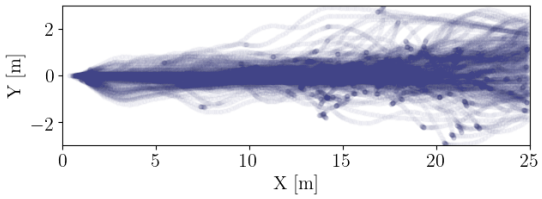


Fig. 7: Visualization of the task duration (i.e., the time taken to reach the goal) in Table II. The dashed vertical line denotes the task duration without obstacles. Our method achieves the smallest variation in task duration and the shortest task duration in crowded environments.

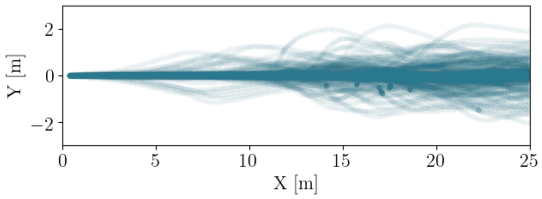
2) *Safety:* The percentage of experiments in which the robot does not collide with pedestrians or the corridor bounds.
3) *Runtime:* Computation time of the control loop.

We note that collisions in simulation may not correspond to collisions in practice but do provide insight into the safety of the planners (more details in Sec. VII).

The simulations are performed on a laptop with an Intel i9 CPU@2.4GHz 16 core CPU. Our implementation of T-MPC and T-MPC++ use P and $P + 1$ CPU threads, respectively.



(a) TEB-Planner [8].



(b) T-MPC++ (ours)

Fig. 8: Trajectories of 200 experiments with 12 pedestrians for the TEB-Planner and T-MPC++. Our method results in smoother and more consistent robot navigation.

We terminate threads when they exceed the control period of 50 ms and in this case select the best trajectory out of the completed optimization problems.

The results over 200 experiments are summarized in Table II and the task duration is visualized in Fig. 7. The Motion Primitives planner is not safe and has a significantly longer task duration than the other planners even in the least crowded case. In the two more crowded environments, LMPCC has a significantly longer task duration than T-MPC++ and is less safe, in part because LMPCC plans a single trajectory. When the optimization becomes infeasible, it often does not recover fast enough to avoid oncoming obstacles. TEB Local Planner is marginally safer than LMPCC and maintains competitive average task durations to the other methods in the four and eight pedestrian scenarios under less computational cost. In the crowded scenario, T-MPC++ completes the task significantly faster. Additionally, in all cases, T-MPC++ has a much smaller standard deviation of the task duration indicating that its behavior is more consistent (visible also in Fig. 7). The TEB Local Planner soft constrains collision avoidance which, in crowded environments, leads the robot into poor behaviors (e.g., reversing) due to the shape of the cost function. To further compare T-MPC++ and the TEB Local Planner, we visualize their trajectories in Fig. 8. Our proposed planner results in smoother and more consistent trajectories and can follow the reference path more closely. We quantitatively compare the smoothness of the planners through the standard deviation on second-order input commands. The deviation on acceleration (σ_a) and rotational acceleration (σ_α) for TEB Local Planner are higher ($\sigma_a = 0.16$, $\sigma_\alpha = 0.12$) than for T-MPC++ ($\sigma_a = 0.04$, $\sigma_\alpha = 0.05$).

Out of the guidance planners, Guidance-MPCC attains the same mean task duration as T-MPC++ but is less consistent (high std. dev.) because the guidance planner, which does not account for the robot dynamics, determines the planner's behavior. This results in larger tracking errors under the same cost function, for example in the 12 pedestrian case, the

TABLE III: Quantative results in crowded environment of Sec. V-D over 200 experiments. Notation follows that of Table II. Without obstacles, the task duration is 20.6 s. The runtime is denoted as "mean (max)".

# Ped.	Method	Dur. [s]	Safe (%)	Runtime* [ms]
0	-	20.6 (0.0)	-	-
50	TEB Local Planner [8]	22.4 (2.4)	**92**	**9.4** (177.1)
	T-MPC++ (ours)	**21.0** (0.9)	**92**	21.2 (46.9)

mean velocity and path errors of Guidance-MPCC are 0.45m and 0.47m/s, respectively, compared to values of 0.20m and 0.42m/s for T-MPC++. It also leads to more collisions than LMPCC. T-MPC is generally faster than LMPCC, except for the 12 pedestrian case. In this environment, the local planner may find solutions that the guidance planner did not, given that the space is cluttered. T-MPC++ demonstrates superior navigation performance over the other planners: it is significantly faster (with the exception of Guidance-MPCC), varies less in its task duration (lower std. dev.) and is safer in almost all cases (the TEB Local Planner is safer in the 8 pedestrian case). T-MPC++ has higher computational demands than the other planners. Compared to the other MPC planners, T-MPC++ first computes guidance plans. We measured that this step takes approximately 5 ms on average (included in the runtime of Table II) in all scenarios.

D. Crowded Baseline Comparison

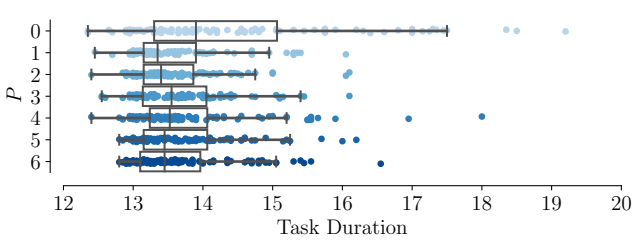
We further compare T-MPC++ against TEB Local Planner in a square-shaped crowded environment with 50 pedestrians. The robot's task is to move diagonally through the environment at $v_{\text{ref}} = 1.5$ m/s. We consider the 12 pedestrians closest to the plan, with preference for nearby pedestrians in both methods. Pedestrians are removed when they reach the goal to prevent unpredictable turns. Table III presents the results. T-MPC++ is significantly faster and the standard deviation of the task duration is less than half that of TEB Local Planner. TEB Local Planner is on average computationally faster as it does not compute a new plan in each iteration. When it does compute a plan, its computation time can exceed the planning frequency of 20 Hz as shown by the maximum in Table III (it exceeded 50 ms in 0.26% of its iterations). In contrast, T-MPC++ is explicitly limited to 50 ms such that its maximum computation time remains below 50 ms.

E. Sensitivity Studies

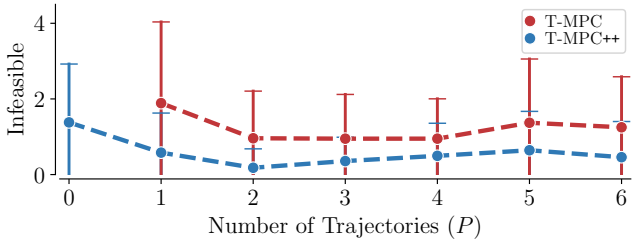
To provide more insight into the key parameters of our approach, we study their sensitivity.

1) Sensitivity to the Number of Trajectories P: To study how the number of guidance trajectories impacts the task duration, we run 100 experiments in the 12 pedestrians environment and compute $P = 0, \dots, 6$ guidance trajectories. The $P = 0$ case corresponds to the non-guided local planner, LMPCC [3].

Fig. 9 displays statistics on task duration. We observe from Fig. 9a that guidance trajectories reduce the task duration compared to the local planner. Fig. 9b shows for both T-MPC and T-MPC++ how often the planner becomes infeasible. It indicates that the availability of at least two plans makes



(a) Task duration of T-MPC++.



(b) Infeasible iterations (mean and std) per experiment.

Fig. 9: Sensitivity study of the number of guidance trajectories P. (a) Task duration of T-MPC++, individual experiments denoted by dots. (b) Mean and standard deviation of the number of control iterations in which the optimization is infeasible per experiment (out of approximately 280 iterations each) for the same simulations.

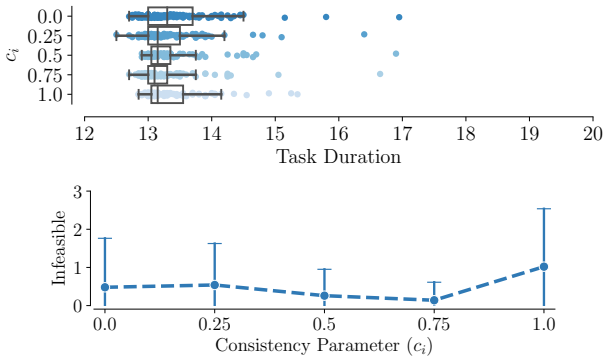


Fig. 10: Sensitivity study of the consistency parameter c_i comparing the task duration and infeasibility of the planner. Note that a lower c_i makes the planner *more* consistent.

it more likely that a trajectory is found and shows that the non-guided planner added in T-MPC++ further improves feasibility.

2) Sensitivity to the Consistency Parameter c_i: We recall that c_i (see Eq. (12)) expresses our preference to follow the trajectory with the same passing behavior as in the previous planning iteration. We vary $c_i \in [0, 1]$ within its range, including $c_i = 0$ that enforces the robot to follow the trajectory in the previous homotopy class, if it still exists, and $c_i = 1$ that expresses no preference.

Fig. 10 compares task duration and infeasibility. Both extreme values show poor performance. For $c_i = 0$, the task duration increases, while for $c_i = 1$, the planner becomes infeasible more often (it is indecisive). We deployed $c_i = 0.75$, which best retains the feasibility of the optimization.

TABLE IV: Comparison of the attained cost in the crowded environment of Sec. V-D over 50 experiments. Notation, including the cost that is tested for significance, follows that of Table II. The cost excludes infeasible planner iterations.

# Ped.	Method	Dur. [s]	Safe (%)	Cost	Runtime [ms]
50	LMPCC [3]	21.9 (1.7)	84	2.25 (9.36)	**7.1** (4.8)
	T-MPC (ours)	21.6 (1.5)	84	1.60 (6.21)	20.6 (7.7)
	T-MPC++ (ours)	**21.0** (0.9)	**96**	**1.05** (4.67)	21.7 (7.4)

F. Empirical Cost Comparison

To verify that T-MPC++ is able to find lower-cost local optima than the local planner in isolation (as discussed in Sec. IV-F), we compare the optimal cost of the executed trajectory attained by LMPCC (local planner), T-MPC and T-MPC++ that use identical cost functions. We perform this study in the crowded environment of Sec. V-D over 50 experiments. Table IV indicates that T-MPC finds lower-cost local optima than LMPCC and does so consistently (lower std. dev.). T-MPC++ further reduces this cost and its deviation. The average cost of T-MPC++ is less than half that of the non-guided planner.

G. T-MPC Under Obstacle Uncertainty

To illustrate that T-MPC applies to different local planner formulations, we deploy T-MPC on top of CC-MPC [4]. CC-MPC is a local planner that considers the probability of collision with obstacles when their motion is represented by a Gaussian distribution at each time step. We assume that the motion of the obstacles follows the uncertain dynamics

$$\boldsymbol{o}_{k+1}^j = \boldsymbol{o}_k^j + (\boldsymbol{v}_k^j + \boldsymbol{\eta}_k^j)dt,\ \boldsymbol{\eta}_k^j \sim \mathbb{P}_k^j, \tag{15}$$

Here $\boldsymbol{v}_k$ is the velocity that follows the social forces model as in previous experiments. The distribution of $\boldsymbol{\eta}_k^j \in \mathbb{R}^2$ follows a bivariate Gaussian distribution, $\boldsymbol{\eta}_k^j \sim \mathcal{N}(\boldsymbol{\mu}_k^j, \boldsymbol{\Sigma}_k^j)$, where $\boldsymbol{\mu}_k^j = \mathbf{0}$ and $\boldsymbol{\Sigma}_k^j = \sigma \boldsymbol{I}$. In this simulation we set $\sigma = 0.3$. Instead of deterministic collision avoidance constraints (14), we formulate a chance constraint with risk $0 < \epsilon < 1$

$$\mathbb{P}\left[||\boldsymbol{p}_k - \boldsymbol{o}_k^j||_2^2 \geq r\right] \geq 1 - \epsilon,\ \forall k, j \tag{16}$$

that specifies collision avoidance to hold with a probability of $1 - \epsilon$ for each agent and time instance. CC-MPC [4] reformulates this constraint using the Gaussian 1-D CDF in the direction of the obstacle. In this formulation, the collision avoidance constraint is linearized and reduces to

$$(\boldsymbol{A}_k^j)^T(\boldsymbol{p}_k - \boldsymbol{o}_k^j) - r - r^j \geq \text{erf}^{-1}(1-2\epsilon)\sqrt{2(\boldsymbol{A}_k^j)^T\boldsymbol{\Sigma}_k^j(\boldsymbol{A}_k^j)},$$

with $\boldsymbol{A}_k^j$ as in (8) and where erf^{-1} is the inverse standard error function.

We apply T-MPC with the non-guided CC-MPC in parallel (referred to as TCC-MPC++). The uncertainty directly affects the local planner, while the guidance planner only avoids the mean obstacle trajectories. It may happen that some guidance trajectories are not feasible for the local planner.

We deploy both planners in the scenario with 12 randomized pedestrians and compare the planners under high ($\epsilon = 0.1$), medium ($\epsilon = 0.01$) and low ($\epsilon = 0.001$) risk settings. The results are shown in Table V and visualized in Fig. 11. TCC-MPC++ consistently outperforms CC-MPC in isolation, leading to significantly faster and more consistent task completion and fewer collisions. The local planner often collides when it becomes infeasible since it cannot recover. The initialization provided by the guidance planner resolves infeasibility and improves robustness.

TABLE V: Quantative results for simulations with uncertain obstacle motion of Sec. V-G over 200 experiments. Notation follows that of Table II.

#	Method	Task Duration [s]	Safe (%)	Runtime [ms]
High Risk	CC-MPC [4]	15.8 (2.3)	91	**17.0** (8.7)
	TCC-MPC++ (ours)	**14.1** (0.8)	**96**	34.5 (7.9)
Medium Risk	CC-MPC [4]	16.5 (2.7)	92	**17.4** (9.7)
	TCC-MPC++ (ours)	**15.1** (1.4)	**93**	35.8 (8.2)
Low Risk	CC-MPC [4]	17.2 (2.5)	90	**18.5** (10.2)
	TCC-MPC++ (ours)	**16.1** (1.3)	**97**	38.1 (7.9)

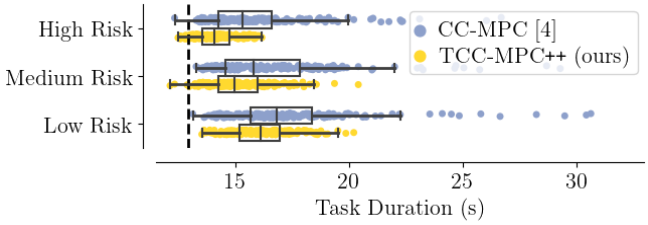


Fig. 11: Visualization of the task duration in Table V.

VI. REAL-WORLD EXPERIMENTS

We demonstrate the proposed planner in a real-world setting on a mobile robot driving among pedestrians.

A. Experimental Setup

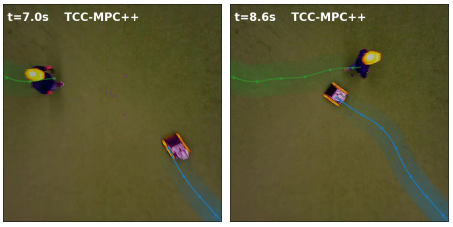
The experiment takes place in a 5m×8m square environment where participants walk among the robot. The robot and pedestrian positions are detected by a motion capture system at 20Hz. The pedestrian positions are passed through a Kalman filter and constant velocity predictions are passed to the planner. The robot is given a reference path between two opposite corners and turns around once a corner is reached.

B. One Pedestrian

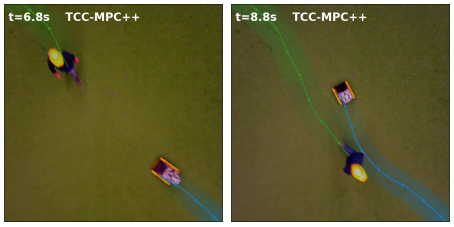
In the first set of experiments, a single pedestrian interacts with the robot. We run TCC-MPC++ to evade the pedestrian. Fig. 12 shows the results of two experiments. In Fig. 12a, the pedestrian turns and speeds up to pass the robot in front. The robot changes its behavior from passing in front to passing behind to let the pedestrian pass. In Fig. 12b, the pedestrian changes its intended passing side from the right to the left side of the robot. The planner detects the change in direction and switches sides, passing the pedestrian smoothly.

C. Five Pedestrians

In the second set of experiments, we run LMPCC, T-MPC++, CC-MPC and TCC-MPC++ for 5 minutes each with 5 pedestrians in the space. The participants were instructed to walk naturally toward a point on the other side of the lab. Before starting the recorded experiments, participants were asked to

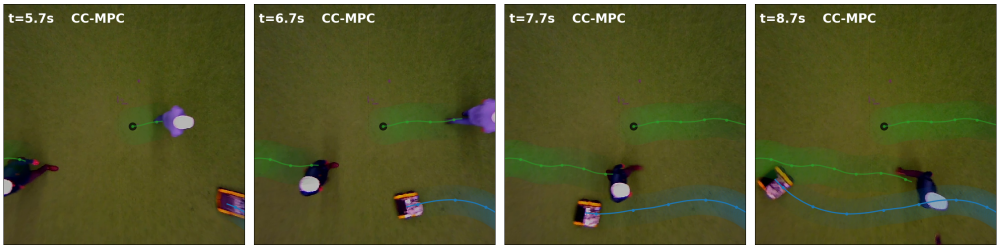


(a) Encounter with a turning pedestrian.

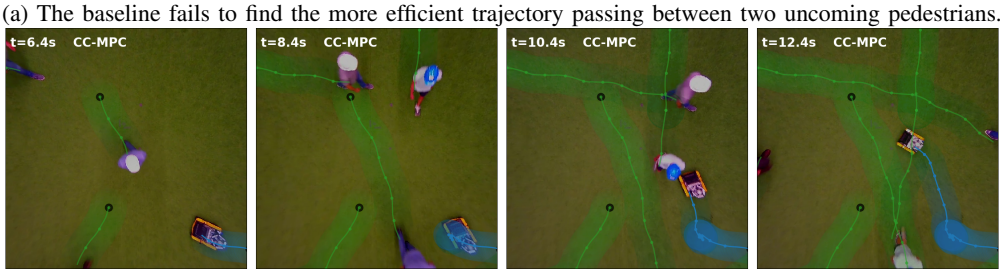


(b) Pedestrian switching intended passing side.

Fig. 12: Overlayed top view camera images of real-world experiments of TCC-MPC++ with one pedestrian. Blue and green overlays denote the robot and pedestrian trajectories, respectively. Timestamps of each image denoted in the upper left corner.



(a) The baseline fails to find the more efficient trajectory passing between two uncoming pedestrians.



(b) The baseline becomes infeasible and does not recover quickly (note the time scale).

Fig. 13: Trajectories of baseline CC-MPC [4] overlayed on camera images. Black dots denote pedestrian start positions.

walk for 3 minutes without the robot, to get used to the environment. The participants were not aware of the planner running in each experiment and the order of planners was randomized. Several runs of CC-MPC and TCC-MPC++ are visualized in Fig. 13 and Fig. 14, respectively. The reader is encouraged to watch the associated video [45]. Fig. 13a highlights a case where CC-MPC conservatively avoids two pedestrians, not detecting the pathway in between the pedestrians. In Fig. 13b, CC-MPC gets infeasible and is not able to replan fast enough, requiring the robot to wait for a pedestrian to pass. In all experiments of Fig. 14, the robot passes pedestrians efficiently and smoothly. Fig. 14a and 14b highlight cases where the planner has to navigate through the crowd and does so successfully. In Fig. 14c, we observe the planner falling in line with a pedestrian to pass another pedestrian.

After the experiments, the participants unanimously preferred the planners running in experiments 1 and 4, which were both guided planners. In general, participants reported that the guided planners felt safer and more predictable than the non-guided planners.

VII. DISCUSSION

Our results have indicated that our proposed global and local planning framework can improve the safety, consistency and time efficiency of the planner. We discuss further insights related to our planning framework in the following.

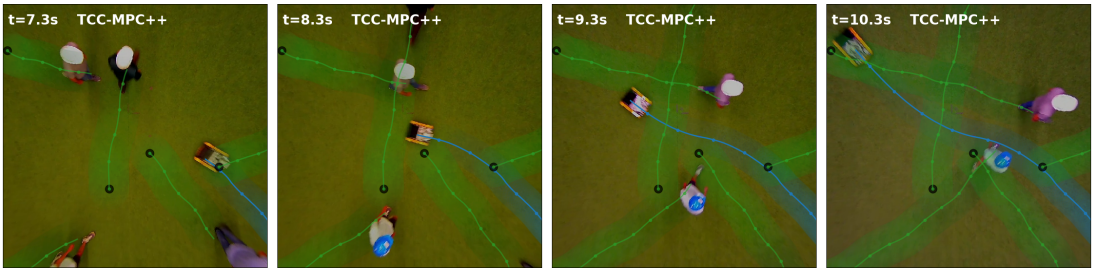
A. Safety in Dynamic Environments

Planners deployed in the real world must be safe (i.e., collisions are unacceptable) and should not impede humans more than necessary. Table II indicated that all planners collided at least once in simulation. To identify the source of collisions, we repeated the experiments using the social forces model for both the pedestrian simulation model and the prediction model of the planner (i.e., removing prediction mismatch). The results are summarized in Table VI. Collisions are almost reduced to zero for T-MPC++ in this case, which shows that prediction mismatch causes most of the collisions.

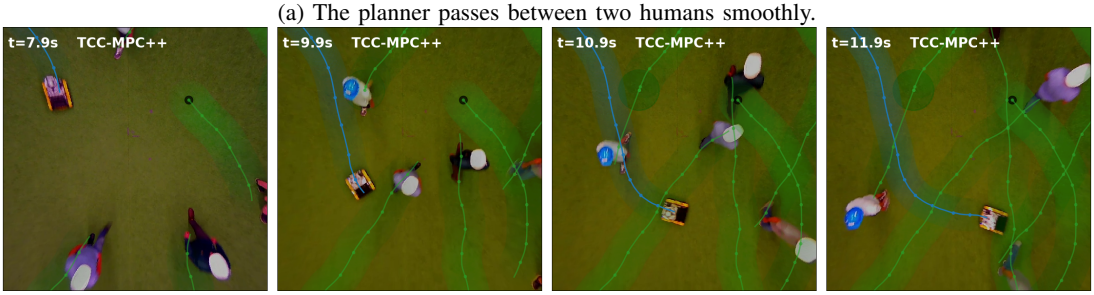
In the real-world experiments, we did not observe collisions. We analyzed the video of the experiments and visually annotated the following instances where pedestrians had to take

TABLE VI: Quantative results for the simulations of Sec. V-C repeated with the pedestrian motion predictions following the social forces model.

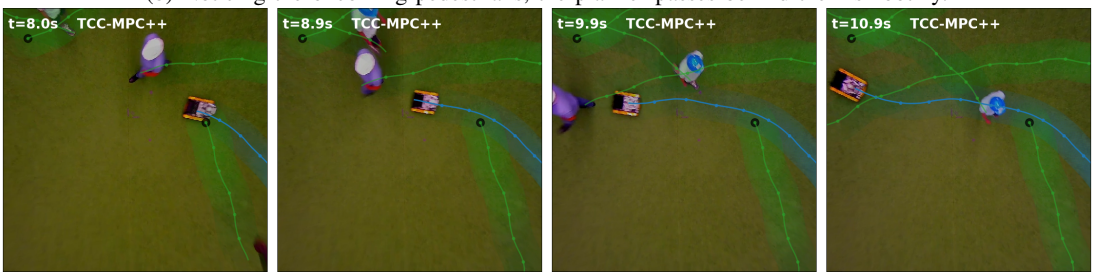
# Peds.	Method	Task Duration [s]	Safe (%)	Runtime [ms]
4	T-MPC++ (ours)	13.0 (0.1)	100	25.4 (4.9)
8	T-MPC++ (ours)	13.2 (0.4)	100	27.7 (6.2)
12	T-MPC++ (ours)	13.6 (0.7)	98	26.7 (7.0)



(a) The planner passes between two humans smoothly.



(b) Noticing the oncoming pedestrians, the planner passes behind them smoothly.



(c) The planner follows another passing pedestrian to pass.

Fig. 14: Trajectories of TCC-MPC++ overlayed on camera images for three examples.

evasive action: 3 out of 63 interactions using LMPCC, 4 out of 61 interactions using CC-MPC, 0 out of 61 interactions for T-MPC++ and 0 out of 60 interactions for TCC-MPC++. This indicates that pedestrians take direct evasive action when the robot impedes their safety, deviating from the social forces model. Collisions in simulation therefore seem to correspond to cases where the human must take evasive action in practice. We also observed in the real world that evasive action was necessary when the baseline planner became infeasible. Safety guarantees provided through constraints only hold when the optimization problem is feasible and can impose danger when no solution is found in time. Our proposed approach reduced this danger by planning more than a single trajectory and we did not observe dangerous cases of infeasibility for T-MPC.

B. Advantages of Parallel Optimization

Deploying several local planners in parallel makes it more likely that the planner returns a trajectory (in time) as a feasible trajectory can be provided by not a single, but several optimization problems. We suspect that this effect is more pronounced when the planning problems are more diverse. In addition, parallelization reduces the maximum computation times. The fastest solved optimization immediately provides a trajectory and other problems can be ignored if necessary. The parallel planner computation time is, at worst, equal to that of a single planner but is almost always faster. Several CPU cores are necessary to parallelize the planner but are usually available. These two advantages inherently improve all performance metrics (e.g., safety and time efficiency) as a solution is more often available. Because redundancy and reduced computation times are key for real-world applications, parallelization may be the key to safely and efficiently deploying optimization-based planners in practice.

C. Selection of the Homotopy Class

The decision-making in Sec. IV-E used the optimal costs of the local planners to decide which trajectory to execute and preferred the homotopy class of the last followed trajectory. We observed in practice that the robot stayed closer to the reference path and velocity, and passed pedestrians behind rather than in front when necessary. We additionally observed that due to measurement and prediction noise, making a more consistent decision led to better navigation. With high consistency, the robot switches behavior only if the new one is significantly better than the current one. In this way we can ensure a more robust estimation of the cost of the trajectory. While our proposed decision-making led to fast navigation, it ignored social norms. The decision-making could be made more socially compliant by learning to pick the homotopy class that humans take from data (see e.g., [34],

[46], [47]). Finally, as noted by [8], homotopy classes merge and split when obstacles are passed or appear in the planning horizon. Reacting to these events could make the planner more responsive in practice.

D. Limitations and Future Work

One of the remaining limitations of the framework is the lack of interaction between the humans and the robot. The social forces model that we used in simulation is interactive but does not accurately model human-robot interactions. It may be possible to reduce the complexity of interaction with humans to an explicit decision on the topology class of interaction (along the lines of [48]) that simplifies the planning problem. Additionally, T-MPC++ can possibly be extended to 3D navigation in dynamic environments, for instance using higher-dimensional H-signatures, and the guidance planner could be extended to incorporate non-Gaussian (i.e., multi-modal) uncertainty in obstacle motion (e.g., [15]).

VIII. CONCLUSION

We presented in this paper a two-fold planning approach to address the inherent local optimality of optimization-based planners. Our planner consisted of a high-level global planner and a low-level optimization-based planner. By accounting for the topology classes of trajectories in the dynamic free space, we generated trajectories with distinct passing behaviors that we then used to guide several local optimization-based planners in parallel.

We simulated a mobile robot navigating among pedestrians and showed that the proposed guided planner resulted in faster and more consistent robot motion than existing planners, including a state-of-the-art topology-guided planner. We qualitatively observed the same improvement in the real world, where we navigated successfully among five pedestrians.

In future work, we aim to deploy the proposed method, considering uncertainty in obstacle motion, on a self-driving vehicle navigating in urban environments.

APPENDIX A
HOMOTOPY COMPARISON

This appendix details and compares three implementations of homotopy comparison function (3) for 2D motion planning in dynamic environments.

A. H-signature

The H-signature [30] approximates homotopy classes by homology classes, formally defined as follows.

Definition 4. [30] (Homologous Trajectories) Two trajectories $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2 \in T$ connecting the same start and end points $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ respectively, are homologous iff $\boldsymbol{\tau}_1$ together with $\boldsymbol{\tau}_2$ (the latter in the opposite direction) forms the complete boundary of a 2-dimensional manifold embedded in $\mathcal{X}$ not containing or intersecting any of the obstacles.

If two trajectories are homotopic, they are homologous. The reverse does not hold. To compute the H-signature in the 3D space composed of x, y-position and time, each obstacle and its prediction is virtually modeled as a current-carrying wire (see Fig. 15). The H-signature of Obstacle j, $h^j(\boldsymbol{\tau})$ is defined by the virtual magnetic field resulting from Obstacle j's current loop, integrated over trajectory $\boldsymbol{\tau}$. If two trajectories $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2$ (see Fig. 15) enclose the loop of obstacle j, then $h^j(\boldsymbol{\tau}_1) \neq h^j(\boldsymbol{\tau}_2)$. Hence, two trajectories are equivalent if $h^j(\boldsymbol{\tau}_1) = h^j(\boldsymbol{\tau}_2)\ \forall j$ and are distinct otherwise. We compute the H-signature for a finite-time state space, by constructing a 1D skeleton of the obstacle prediction that is looped outside of the workspace and time horizon (see Fig. 15). The skeleton is composed of the following lines. ① The obstacle's prediction for $0 < t < T$. ② A line upwards to $t = T + \epsilon$ where $\epsilon > 0$ is a small constant and a line going outside of the workspace. ③ A line down to $t = -\epsilon$. ④ A line to the obstacle position at $t = 0$. This skeleton ensures that trajectories can only enclose the predicted obstacle motion for $0 < t < T$.

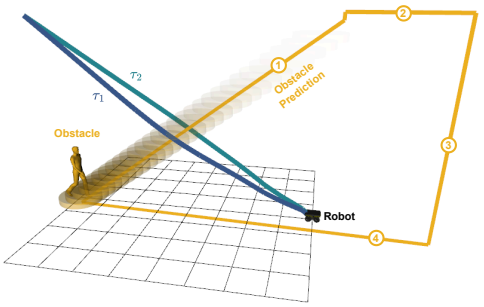


Fig. 15: Illustrating example for the H-signature [30]. The two trajectories $\boldsymbol{\tau}_1$ (blue) and $\boldsymbol{\tau}_2$ (green) are in distinct homology classes as the loop that they form contains obstacle skeleton (orange). In practice, link (3) is placed far away.

We assume that obstacle trajectories are piecewise linear (e.g., discrete-time trajectories). The integration of the magnetic field $\boldsymbol{B}$ can then be computed analytically (see [30]) per segment i of Obstacle j's skeleton $\overline{\boldsymbol{o}_i^j \boldsymbol{o}_i^{j'}}$ as follows:

$$\boldsymbol{p} = \boldsymbol{o}_i^j - \boldsymbol{r},\ \boldsymbol{p}' = \boldsymbol{o}_i^{j'} - \boldsymbol{r},\ \boldsymbol{d} = \frac{(\boldsymbol{o}_i^{j'} - \boldsymbol{o}_i^j) \times (\boldsymbol{p} \times \boldsymbol{p}')}{||\boldsymbol{o}_i^{j'} - \boldsymbol{o}_i^j||^2},$$

$$\boldsymbol{\Phi}(\boldsymbol{o}_i^j, \boldsymbol{o}_i^{j'}, \boldsymbol{r}) = \frac{1}{||\boldsymbol{d}||^2} \left(\frac{\boldsymbol{d} \times \boldsymbol{p}'}{||\boldsymbol{p}'||} - \frac{\boldsymbol{d} \times \boldsymbol{p}}{||\boldsymbol{p}||} \right),$$

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{1}{4\pi} \sum_{i=0}^{I} \boldsymbol{\Phi}(\boldsymbol{o}_i^j, \boldsymbol{o}_i^{j'}, \boldsymbol{r}),$$

with I the number of segments in Obstacle j's skeleton. The integral $\int_{\boldsymbol{l}} \boldsymbol{B}(\boldsymbol{r}) d\boldsymbol{r}$ over looped robot trajectories $\boldsymbol{l}$ yields 1 if the obstacle is enclosed and 0 otherwise. To compare trajectories that reach different goals in our guidance planner, we connect their end points directly at $t = T$ with an additional line. We use the GSL library [49] to perform the integration and cache computed H-signature for each trajectory to prevent re-computation.

B. Winding Number

The winding number [39] is a topological invariant that indicates how the robot and obstacle j are rotated around

each other. It is computed as follows. The relative position of obstacle j to the robot for time step k is $\boldsymbol{d}_k^j = \boldsymbol{p}_k - \boldsymbol{o}_k^j$. The relative angle $\angle\theta_k^j$ is the angle of $\boldsymbol{d}_k^j$ in a fixed global frame. Between time steps k and $k+1$, the relative angle changes by $\Delta\theta_k^j = \theta_{k+1}^j - \theta_k^j$. The winding number accumulates these changes over all time steps, $\lambda(\boldsymbol{\tau}, \boldsymbol{o}^j) = \frac{1}{2\pi}\sum_{k=1}^{N} \Delta\theta_k^j$. The sign of the winding number λ indicates the passing direction, its magnitude denotes passing progress. We consider a trajectory to pass obstacle j if $|\lambda^j| \geq \lambda_{\text{pass}}$, where by default $\lambda_{\text{pass}} = \frac{1}{4\pi}$. We consider two trajectories distinct if there exists at least one obstacle that the trajectories pass on different sides and consider them equivalent otherwise[8]. We cache computed winding numbers to prevent recomputation.

C. Universal Visibility Deformation

Universal Visibility Deformation (UVD) was proposed for static obstacle avoidance in 3D and therefore does not exactly capture the local optima for collision avoidance in 2D dynamic environments. Two trajectories are in the same UVD class if points along the trajectories can be connected, without intersecting with obstacles.

Definition 5. [33] Two trajectories $\boldsymbol{\tau}_1(s), \boldsymbol{\tau}_2(s)$ parameterized by $s \in [0,1]$ and satisfying $\boldsymbol{\tau}_1(0) = \boldsymbol{\tau}_2(0)$, $\boldsymbol{\tau}_1(1) = \boldsymbol{\tau}_2(1)$, belong to the same uniform visibility deformation class, if for all s, line $\overline{\boldsymbol{\tau}_1(s)\boldsymbol{\tau}_2(s)}$ is collision-free.

In practice, we check collisions for s at discrete intervals along the trajectories.

D. Comparison

We compare the homotopy comparison functions in simulation on the scenario of Sec. V-D over 100 experiments. Table VII indicates that UVD degrades navigation performance, likely because it is not designed for dynamic environments and may lead to duplicate trajectories in practice. The H-signature and winding numbers show similar navigation performance. Winding numbers are computationally more efficient but require a minimum passing angle to be tuned. Since the H-signature generalizes to higher dimensions and both methods are still real-time, we opted to use the H-signature in this paper.

TABLE VII: Comparison between homotopy comparison implementations. Notation follows that of Table II.

#	Method	Dur. [s]	Safe (%)	Homotopy Comparison Time [ms]
Homotopy Comparison	Winding Angles	21.2 (0.9)	**93**	**0.3** (0.7)
	H-Signature	21.2 (1.0)	92	2.1 (4.0)
	UVD	**21.1** (0.9)	88	3.7 (8.5)

[8]Future work could also use winding numbers to distinguish between passing and non-passing trajectories.

REFERENCES

[1] M. Simon, *Inside the Amazon Warehouse Where Humans and Machines Become One*, 2019. [Online]. Available: https://www.wired.com/story/amazon-warehouse-robots/

[2] J. Walker, *The Self-Driving Car Timeline - Predictions from the Top 11 Global Automakers*, 2019. [Online]. Available: https://emerj.com/ai-adoption-timelines/self-driving-car-timeline-themselves-top-11-automakers/

[3] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4459–4466, Oct. 2019.

[4] H. Zhu and J. Alonso-Mora, "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 776–783, Apr. 2019.

[5] M. Everett, Y. F. Chen, and J. P. How, "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning," in *IEEE Int. Conf. Intell. Robots Syst.*, Oct. 2018, pp. 3052–3059.

[6] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame," in *IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 987–993.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[8] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robot. Auton. Syst.*, vol. 88, pp. 142–153, Feb. 2017.

[9] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knöppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making Bertha Drive—An Autonomous Journey on a Historic Route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, 2014.

[10] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter, F. Gritschneder, A. Scheel, M. Stübler, M. Bach, P. Hatzelmann, C. Wild, and K. Dietmayer, "Autonomous driving at Ulm University: A modular, robust, and sensor-independent fusion approach," in *IEEE Intell. Veh. Symp.*, Jun. 2015, pp. 666–673, iSSN: 1931-0587.

[11] F. Altché and A. de La Fortelle, "Partitioning of the free space-time for on-road navigation of autonomous ground vehicles," in *IEEE Conf. Decis. Control.*, Dec. 2017, pp. 2126–2133.

[12] L. Ferranti, B. Brito, E. Pool, Y. Zheng, R. M. Ensing, R. Happee, B. Shyrokau, J. F. P. Kooij, J. Alonso-Mora, and D. M. Gavrila, "SafeVRU: A Research Platform for the Interaction of Self-Driving Vehicles with Vulnerable Road Users," in *IEEE Intelligent Vehicles*, 2019, pp. 1660–1666.

[13] A. Wang, X. Huang, A. Jasour, and B. Williams, "Fast Risk Assessment for Autonomous Vehicles Using Learned Models of Agent Futures," in *Robotics: Science and Systems*, Jul. 2020.

[14] O. de Groot, B. Brito, L. Ferranti, D. Gavrila, and J. Alonso-Mora, "Scenario-Based Trajectory Optimization in Uncertain Dynamic Environments," *IEEE Robot. Autom. Lett.*, pp. 5389 – 5396, 2021.

[15] O. de Groot, L. Ferranti, D. Gavrila, and J. Alonso-Mora, "Scenario-Based Motion Planning with Bounded Probability of Collision," Jul. 2023. [Online]. Available: https://arxiv.org/pdf/2307.01070.pdf

[16] C. Pek and M. Althoff, "Fail-Safe Motion Planning for Online Verification of Autonomous Vehicles Using Convex Optimization," *IEEE Trans. Robot.*, vol. 37, no. 3, pp. 798–814, Jun. 2021.

[17] J. P. Alsterda, M. Brown, and J. C. Gerdes, "Contingency Model Predictive Control for Automated Vehicles," in *IEEE Am. Control Conf.*, Philadelphia, PA, USA, Jul. 2019, pp. 717–722.

[18] V. K. Adajania, A. Sharma, A. Gupta, H. Masnavi, K. M. Krishna, and A. K. Singh, "Multi-Modal Model Predictive Control Through Batch Non-Holonomic Trajectory Optimization: Application to Highway Driving," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 4220–4227, Apr. 2022.

[19] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Robot. Autom. Lett.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[20] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 797–822, Jun. 2016.

[21] A. Orthey, S. Akbar, and M. Toussaint, "Multilevel motion planning: A fiber bundle formulation," *Int. J. Robot. Res.*, vol. 43, no. 1, pp. 3–33, Jan. 2024.

[22] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Multilayer Graph-Based Trajectory Planning for Race Vehicles in Dynamic Scenarios," in *IEEE Intell. Transp. Syst. Conf.*, Oct. 2019, pp. 3149–3154.

[23] J. Ortiz-Haro, W. Hoenig, V. N. Hartmann, and M. Toussaint, "iDb-A*: Iterative Search and Optimization for Optimal Kinodynamic Motion Planning," Nov. 2023. [Online]. Available: http://arxiv.org/abs/2311.03553

[24] M. K. M Jaffar and M. Otte, "PiP-X: Online feedback motion planning/replanning in dynamic environments using invariant funnels," *Int. J. Robot. Res.*, Nov. 2023.
[25] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion Planning around Obstacles with Convex Optimization," May 2022, arXiv:2205.04422 [cs]. [Online]. Available: http://arxiv.org/abs/2205.04422
[26] K. Zheng, "ROS Navigation Tuning Guide," in *Robot Operating System (ROS): The Complete Reference (Volume 6)*, ser. Studies in Computational Intelligence, A. Koubaa, Ed. Springer International Publishing, 2021, pp. 197–226. [Online]. Available: https://doi.org/10.1007/978-3-030-75472-3_6
[27] F. Eiras, M. Hawasly, S. V. Albrecht, and S. Ramamoorthy, "A Two-Stage Optimization-Based Motion Planner for Safe Urban Driving," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 822–834, Apr. 2022.
[28] W. Ding, L. Zhang, J. Chen, and S. Shen, "EPSILON: An Efficient Planning System for Automated Vehicles in Highly Interactive Environments," *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 1118–1138, Apr. 2022.
[29] J. Park, S. Karumanchi, and K. Iagnemma, "Homotopy-Based Divide-and-Conquer Strategy for Optimal Trajectory Planning via Mixed-Integer Programming," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1101–1115, Oct. 2015.
[30] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Auton. Robots*, vol. 33, no. 3, pp. 273–290, Oct. 2012.
[31] B. Yi, P. Bender, F. Bonarens, and C. Stiller, "Model Predictive Trajectory Planning for Automated Driving," *IEEE Trans. Intell. Veh.*, vol. 4, no. 1, pp. 24–38, Mar. 2019.
[32] O. De Groot, L. Ferranti, D. Gavrila, and J. Alonso–Mora, "Globally Guided Trajectory Planning in Dynamic Environments," in *IEEE Int. Conf. Robot. Autom.*, May 2023, pp. 10 118–10 124.
[33] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust Real-time UAV Replanning Using Guided Gradient-based Optimization and Topological Paths," in *IEEE Int. Conf. Robot. Autom.*, May 2020, pp. 1208–1214.
[34] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *Int. J. Robot. Res.*, vol. 35, no. 11, pp. 1289–1307, Sep. 2016.
[35] C. Mavrogiannis, K. Balasubramanian, S. Poddar, A. Gandra, and S. S. Srinivasa, "Winding Through: Crowd Navigation via Topological Invariance," *IEEE Robot. Autom. Lett.*, vol. 8, no. 1, pp. 121–128, Jan. 2023.
[36] C. Cao, P. Trautman, and S. Iba, "Dynamic Channel: A Planning Framework for Crowd Navigation," in *IEEE Int. Conf. Robot. Autom.*, May 2019, pp. 5551–5557.
[37] C. Mavrogiannis and R. A. Knepper, "Hamiltonian coordination primitives for decentralized multiagent navigation," *Int. J. Robot. Res.*, vol. 40, no. 10-11, pp. 1234–1254, Sep. 2021.
[38] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, Jan. 2000.
[39] M. A. Berger, "Topological Invariants in Braid Theory," *Letters in Mathematical Physics*, vol. 55, no. 3, pp. 181–192, Mar. 2001.
[40] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, 2nd ed. The MIT Press, 2011.
[41] A. Domahidi and J. Jerez, *FORCES Professional*. Embotech AG, Jul. 2014. [Online]. Available: https://embotech.com/FORCES-Pro
[42] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, pp. 4282–4286, May 1995.
[43] C. Gloor, "Pedsim: Pedestrian crowd simulation," 2016. [Online]. Available: https://github.com/chgloor/pedsim
[44] C. Rösmann, "ROS Package teb_local_planner," Nov. 2023. [Online]. Available: https://github.com/rst-tu-dortmund/teb_local_planner
[45] O. de Groot, L. Ferranti, D. Gavrila, and J. Alonso-Mora, "Video Topology-Driven Parallel Trajectory Optimization in Dynamic Environments," Jan. 2024. [Online]. Available: https://youtu.be/kXUAldQXrNk
[46] D. Martinez-Baselga, O. de Groot, L. Knoedler, L. Riazuelo, J. Alonso-Mora, and L. Montano, "SHINE: Social Homology Identification for Navigation in Crowded Environments," Apr. 2024. [Online]. Available: http://arxiv.org/abs/2404.16705
[47] C. Rösmann, M. Oeljeklaus, F. Hoffmann, and T. Bertram, "Online trajectory prediction and planning for social robot navigation," in *IEEE Int. Conf. on Adv. Int. Mech.*, Jul. 2017, pp. 1255–1260.
[48] C. I. Mavrogiannis and R. A. Knepper, "Multi-agent path topology in support of socially competent navigation planning," *Int. J. Robot. Res.*, vol. 38, no. 2-3, pp. 338–356, Mar. 2019.
[49] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, "GNU Scientific Library Reference Manual (3rd Ed.)," Aug. 2019.

Oscar de Groot received both the B.Sc. degree in electrical engineering, in 2016, and the M.Sc. degree in systems & control, in 2019, from the Delft University of Technology, Delft, The Netherlands. He is currently pursuing a Ph.D. in motion planning for autonomous vehicles in urban environments at the department of Cognitive Robotics at the Delft University of Technology. His research interests include probabilistic safe motion planning, scenario optimization, model predictive control and self-driving vehicles.

Laura Ferranti received her PhD from Delft University of Technology, Delft, The Netherlands, in 2017. She is currently an assistant professor in the Cognitive Robotics (CoR) Department, Delft University of Technology, Delft, The Netherlands. She is the recipient of an NWO Veni Grant from The Netherlands Organisation for Scientific Research (2020), and of the Best Paper Award in Multi-robot Systems at ICRA 2019. Her research interests include optimization and optimal control, model predictive control, reinforcement learning, embedded optimization-based control with application in flight control, maritime transportation, robotics, and automotive.

Dariu Gavrila received the Ph.D. degree in computer science from Univ. of Maryland at College Park, USA, in 1996. From 1997 until 2016, he was with Daimler R&D, Ulm, Germany, where he became a Distinguished Scientist. He led the vision-based pedestrian detection research, which was commercialized 2013-2014 in various Mercedes-Benz models. In 2016, he moved to TU Delft, where he since heads the Intelligent Vehicles group as a Full Professor. His current research deals with sensor-based detection of humans and analysis of behavior in the context of self-driving vehicles. He was awarded the Outstanding Application Award 2014 and the Outstanding Researcher Award 2019, both from the IEEE Intelligent Transportation Systems Society.

Javier Alonso-Mora is an Associate Professor at the Cognitive Robotics department of the Delft University of Technology, and a Principal Investigator at the Amsterdam Institute for Advanced Metropolitan Solutions (AMS Institute). Before joining TU Delft, Dr. Alonso-Mora was a Postdoctoral Associate at the Massachusetts Institute of Technology (MIT). He received his Ph.D. degree in robotics from ETH Zurich,

His main research interest is in navigation, motion planning and control of autonomous mobile robots, with a special emphasis on multi-robot systems, on-demand transportation and robots that interact with other robots and humans in dynamic and uncertain environments. He is the recipient of an ERC Starting Grant (2021), the ICRA Best Paper Award on Multi-Robot Systems (2019), an Amazon Research Award (2019) and a talent scheme VENI Grant from the Netherlands Organisation for Scientific Research (2017).