# DECENTRALIZED REAL-TIME TRAJECTORY PLANNING FOR MULTI-ROBOT NAVIGATION IN CLUTTERED ENVIRONMENTS

by

Baskın Burak Şenbaşlar

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2023

## Dedication

To my father, İbrahim Yaşar, who ignited my passion for computers. This dissertation is dedicated in his memory.

# Acknowledgements

I express my heartfelt gratitude to everyone who supported, inspired, mentored, and sometimes pushed me throughout this journey. Your guidance has been invaluable in navigating the numerous forks on the road to completing this dissertation.

I am grateful to have had the opportunity to work with two amazing advisors, Prof. Nora Ayanian and Prof. Gaurav S. Sukhatme. They were always supportive, allowed and encouraged me to pursue my own ideas, and created a safe research environment to allow me such independence. The independence I had thanks to them enabled me to think deeply about the nature of the problems I was attempting to solve and make the best contributions I could ever make in this period. They allowed many students to work with me, who not only were helpful for my research, but also helped me become a better mentor.

I thank Dr. Wolfgang Hönig, who mentored me during my MSc., gave me the first problem in this problem domain, which not only was an important problem instance by itself, but also became the seed of everything I have done in this dissertation, and continued collaborating with me throughout my PhD.

I am also grateful to my committee members, Prof. Sven Koenig, Prof. Mihailo R. Jovanovic, and Prof. Satish Kumar Thittamaranahalli, for their helpful comments, insights, and suggestions.

I also would like to thank Prof. Erol Şahin for introducing me to systems of multiple robots during my undergraduate studies. The mentorship he provided formed my initial inspiration for pursuing graduate studies in this domain.

I was fortunate to have mentored many students, some of whose work is a part of dissertation while some are not. I thank Pilar Luiz, Alp Çevikel, Terry Lu, Brendan Glascock, Andrew-Richard Opem, Nikhil Bola Kamath, and Silas John for their hard work and commitment. I also would like to thank all my other co-authors, Sumeet Batra, Eric Ewing, Zhehui Huang, Rahul Pramod Krupani, Jingyao Ren, Vikraman Sathiyanarayanan, Zhaojing Yang, on papers not discussed in the context of this dissertation, but helped me understand some parallel problem domains better. I am also grateful to the other members of the Automatic Coordination of Teams (ACT) Lab and Robotic and Embedded Systems Lab (RESL) of USC for all the friendship, discussions, support, and fun.

I thank the organizations funding my studies and research during my PhD: University of Southern California, National Science Foundation (NSF) and United States Army. The funding they provided allowed me to focus on research for almost the entirety of my studies.

I thank my family for all the support, love, encouragement, and inspiration. I thank my aunt, Ayşe Filiz, who taught me how to be a hard-working person, and my mother, Nur, who showed me the power of persistence. I thank my wife, Sevgi, for her unconditional love, support, and endless joy, who made my life amazing.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

Multi-robot collision-free and deadlock-free navigation in cluttered environments with static and dynamic obstacles is a fundamental problem for many real-world applications. Dynamic obstacles can additionally be interactive, i.e., changing their behaviors depending on the behaviors of other objects. We focus on decision making algorithms, with a particular emphasis on decentralized real-time trajectory planning, to enable multi-robot navigation in such environments.

Practicality of the developed approaches is a central focus of ours, such that we design our systems and algorithms under assumptions that can be realized in the real world. Central concerns of our treatment are i) embracing on-board compute, memory, and storage limitations of robotic systems, ii) not relying on communication for safe operation, and explicitly account for communication imperfections, iii) allowing navigation with imperfect a priori knowledge, iv) embracing controller trajectory tracking errors and accounting for them, v) working with minimal sensing and estimation capabilities, and vi) achieving highly reactive collision avoidance behavior. We introduce i) two decentralized real-time multi robot trajectory planning algorithms to allow static obstacle, interactive dynamic obstacle, and teammate avoidance, ii) a constraint generation, overconstraining, and constraint-discarding scheme to ensure inter-robot collision avoidance under asynchronous planning that is inherent in decentralized systems, which we use within one of the proposed planners, and iii) a multi-robot aware planning and control stack that allows collision-free and deadlock-free navigation in diverse types of environments, which combines three qualitatively different decision making approaches in a hierarchical manner.

# Chapter 1

# Introduction

Collision-free (multi-)robot navigation in cluttered environments is a fundamental enabler for many applications, including autonomous driving [14], autonomous last-mile delivery [55], automated warehouses with [42] and without [127] humans, automated intersection management [27], and robotic search and rescue [88]. The complexity of the navigation task in such applications varies: some contain multiple robots to control, i.e., teammates, some contain no obstacles, while others contain static or dynamic obstacles. Dynamic obstacles can be interactive, i.e., they may react to movements of other objects. As the environment clutter, i.e., the space occupied by static or dynamic obstacles or the number of robots, increases, it becomes more and more challenging for teammates to navigate without collisions. In this dissertation, we focus on decision making algorithms, with a particular emphasis on decentralized real-time trajectory planning algorithms, to enable (multi-)robot navigation in cluttered environments with static and dynamic obstacles, which can optionally be interactive.

When there are multiple robots to control, explicitly reasoning about inter-robot interactions during decision making for cooperative navigation allows stronger inter-robot collision avoidance guarantees compared to modeling teammates as non-cooperative entities. We categorize decision making approaches for cooperative multi-robot collision-free navigation in two ways based on their i) deployments and ii) planning horizons.

**Taxonomy Based on Deployment.** The decision making approaches for cooperative multi-robot collision-free navigation can be categorized into centralized and decentralized, differing in where they are deployed. Centralized algorithms make navigation decisions for multiple robots using global information on a centralized computing system. The computed plans are communicated back to the robots for them to execute. Centralized algorithms often provide strong completeness and global optimality guarantees because i) they utilize complete knowledge about the environment beforehand, ii) they generally run on powerful centralized computers, and iii) there are generally less restrictive in time limits on their execution. However, in some cases, such a central entity is undesirable because of the communication link that must be maintained between each robot and the central entity. If the obstacles in the environment are not known a priori, building and relaying the observed obstacles to the central entity through the communication channel adds further challenges. In some cases, it is impractical to have such a central entity because it cannot react to robot trajectory tracking errors and obstacle updates fast enough due to communication and computation delays. In practice, robots must operate safely even if there is limited communication available. Decentralized algorithms delegate the computation of trajectories, allowing on-board autonomy: each robot plans for itself using local information and optionally interacts with other robots explicitly through communication or implicitly through behavioral assumptions, e.g., each robot running the same algorithm. Decentralized algorithms generally forgo strong theoretical guarantees in favor of fast computation because i) they are often used when complete information of the environment is not known beforehand, and ii) they have to work in real-time on-board. Decentralized algorithms are desirable when i) the obstacles in the environment are not known a priori before navigation, ii) reliable low latency communication maintenance to a central system is not possible, iii) there are unforeseen trajectory tracking imperfections causing robots to divert from their plans, which requires real-time re-planning for robust execution, or iv) there is no central system.

**Taxonomy Based on Planning Horizon.** Cooperative multi-robot navigation decision making algorithms can also be categorized into long, medium, and short horizon, differing in the planning horizon used to compute an output. Long horizon navigation decision making (LH-NDM) algorithms are concerned with computing plans for the full navigation task, where planning horizon in distance or duration may grow to infinity. LH-NDM approaches are generally centrally deployed because i) on-board LH-NDM is not possible for all environments because of the potentially infinite planning horizon and on-board compute and memory constraints, and ii) LH-NDM requires global environment representation, which is hard to maintain on-board. Real-time reasoning about obstacles that are not known a priori in centrally deployed LH-NDM approaches is generally not viable when reliable communication maintenance is not possible. Medium horizon navigation decision making (MH-NDM) approaches generate plans to reach intermediate goals, producing plans for the medium horizon future, e.g., next two to ten seconds. Because of this, they are executed in a receding horizon fashion, in which produced plans are executed for a short duration and new plans are produced. MH-NDM approaches typically run at 1 Hz to 20 Hz on-board, allowing real-time execution. Since they can run on-board, they allow integration of new information during navigation. On-board real-time execution of MH-NDM algorithms allows decentralized deployment. Short horizon navigation decision making (SH-NDM) algorithms compute only the next safe action to execute in every decision making iteration, hence, they typically run at a high frequency, e.g., 50 Hz to 200 Hz. Since their reasoning horizon is short and they can be executed at a high frequency, they allow on-board decentralized deployment. However, robots navigating solely with SH-NDM approaches tend to deadlock, i.e., cannot reach to their goal positions, since the reasoning horizon of SH-NDM is short.

In this work, we introduce i) two decentralized real-time multi-robot trajectory planning algorithms deployed as MH-NDM modules, to allow static obstacle, interactive dynamic obstacle, and teammate avoidance, ii) a constraint generation, overconstraining, and constraint-discarding scheme to ensure inter-robot collision avoidance under asynchronous planning that is inherent in decentralized deployments, which we use within one of the proposed planners, and iii) a multi-robot aware planning and control stack that allows collision-free and deadlock-free navigation in diverse types of environments, which combines LH-NDM, MH-NDM and SH-NDM modules in a hierarchical manner. Central concerns of our discussion include

- embracing on-board compute, memory, and storage limitations of robotic systems, which prohibit them from i) generating plans for the full navigation task on-board in real-time, or ii) maintaining a global environment representation on-board,
- not relying on communication (inter-robot or robot-central system) for safe operation, and explicitly accounting for communication imperfections whenever robots communicate,
- working with imperfect a priori knowledge, which prohibits generating completely collision-free plans offline,
- ensuring dynamic feasibility of plans and embracing controller trajectory tracking errors, which results in divergence from reference trajectories, and accounting for them for safe operation,
- working with minimal sensing and estimation capabilities, as each additional estimation requirement increases the error associated with the estimation algorithms, decreasing the robustness of the decision making system, and
- achieving highly reactive collision avoidance behavior, which allows the robots to react to unforeseen situations switfly.

## 1.1 Contributions

In this dissertation, we make the following contributions:

1. We introduce a decentralized real-time trajectory planning algorithm (RLSS, short for real-time trajectory planning using linear spatial separations), which allows collision-free multi-robot navigation

in environments with static obstacles without any reliance on communication, laying the groundwork non-reliance on communication property. RLSS uses position/geometry only sensing of teammates and static obstacles, and does not require higher order derivative estimations or prediction of teammate trajectories. It assumes synchronously planning teammates, which we address in the next contribution. The details of RLSS are discussed in Chapter 4 and published as an Autonomous Robots paper [99].

2. When multiple robots make navigation decisions on-board in a decentralized fashion, their planning start and end times do not match when they are not explicitly synchronized. Decentralized algorithms that assume synchronized execution for safety may result in collisions under such asynchronous execution. We introduce a constraint generation, overconstraining and constraint-discarding scheme that allows each pair of teammates' plans to share a mutually excluding constraint under asynchronous planning. The method uses communication between teammates, but can handle communication imperfections, including message drops, delays, and re-orderings. The details of the method are discussed in Chapter 5 and published as an IROS paper [102].

3. We introduce a decentralized real-time trajectory planning algorithm (DREAM, short for decentralized real-time asynchronous probabilistic trajectory planning algorithm for multi-robot teams) for static obstacle, interactive dynamic obstacle, and asynchronously planning teammate avoidance. DREAM allows probabilistic collision avoidance, in which, uncertainty stems from static obstacle perception and dynamic obstacle prediction inaccuracies. It uses a novel vector field based representation for interactive dynamic obstacle behavior, which allows fast querying of interactions during decision making. For teammate avoidance, it uses the constraint generation, overconstraining, and constraing-discarding scheme presented in Chapter 5, allowing inter-robot collision avoidance under asynchronous planning and communication imperfections. We also propose three online prediction methods to generate probabilistic interactive predictions for dynamic obstacles. The details

of DREAM are discussed in Chapter 6, submitted to IEEE Transactions on Robotics, and available on arXiv [103].

4. We propose a multi-robot aware planning and control stack (MRNAV, short for multi-robot navigation stack) that allows collision-free and deadlock-free navigation in diverse types of environments. MRNAV integrates LH-NDM, MH-NDM, and SH-NDM modules, and assigns responsibilities to them. It defines interfaces to the perception, localization, mapping, and prediction subsystems that might exist in a full robotic stack. We implement MRNAV for multiple simulated quadrotor flight with motor speed control, in which we use single-robot kinematic $A^*$ search as LH-NDM, DREAM as MH-NDM, and a safety barrier certificates (SBC) based decision making module as SH-NDM; and show that all components are needed for effective navigation in diverse types of environments. The details of MRNAV are discussed in Chapter 7, will be submitted to IEEE Robotics and Automation Letters, and are available on arXiv [101].

# Chapter 2

# Related Work

## 2.1 Navigation Decision Making

The decision making approaches for (multi-)robot navigation are investigated in many domains using different abstractions, assumptions, formalisms, and methods. In this chapter, we discuss them using the planning horizon taxonomy. We classify navigation decision making approaches into three categories: long (Section 2.1.1), medium (Section 2.1.2), and short horizon (Section 2.1.3), differing in the planning horizon used to compute an output. The same algorithms can be used for long, medium, or short horizon decision making in some cases; differing in their planning horizon.

### 2.1.1 Long Horizon Navigation Decision Making

LH-NDM approaches are concerned with making decisions for the full navigation task, planning horizon in terms of distance or duration growing up to infinity. These approaches are sometimes called global planners. They can produce trajectories or paths from start to goal states, or they can solve joint task and motion planning (TAMP) problems. Completeness is one of the central aims during LH-NDM, meaning that the algorithm produces a solution whenever the navigation problem is feasible, albeit not in real-time in all environments, especially if dynamic feasibility is required. Because of this, using LH-NDM approaches with completeness guarantees online in real-time is not viable for the general navigation task.

The outputs of these algorithms can be kinematically or kinodynamically feasible trajectories. Kinematic feasibility typically refers to static obstacle and/or teammate avoidance. Dynamic obstacle avoidance is generally not pursued in these approaches, since knowing the existence and behaviors dynamic obstacles before navigation, i.e., offline, is not a realistic expectation and incorporating them to decision making is often unnecessary since by the time long horizon plans are executed, dynamic objects may have already moved contrary to their predictions in many navigation applications.

Many planning algorithms can be used for single-robot LH-NDM, e.g., RRT* [46], A* [124], or D* [108]. Combining planning algorithms with trajectory optimization methods to ensure the dynamic feasibility of the final plans is also proposed [94].

Multi-agent path finding (MAPF) [109] and multi-robot path planning (MRPP) [132] are concerned with finding synchronized paths for multiple agents on graphs such that no two agents occupy the same vertex or traverse the same edge in opposite directions at the same step, i.e., they do not collide. MAPF algorithms can be used for LH-NDM, since they are typically concerned with the full navigation task and provide completeness guarantees. The permutation-invariant MAPF problem, in which goal vertices can be exchanged between agents, is polynomial time solvable for makespan optimization [131]. When goal vertices are assigned to the agents, MAPF problem becomes NP-hard to solve for sum-of-costs or makespan optimization [133]. Many optimal [130, 105, 110, 52, 106], bounded suboptimal [1, 5, 56], or unbounded suboptimal algorithms [107, 63, 62, 120] for NP-Hard variants are proposed. Algorithm selection techniques for optimal MAPF solvers are also proposed [92, 2], which reason about problem types for which individual MAPF solvers produce a solution fast. Generally, agent dynamics are omitted during path planning. Combining MAPF algorithms with trajectory optimization allows dynamic feasibility of the produced plans [111, 39, 78, 24]. Some multi-robot motion planning (MRMP) approaches compute dynamically feasible plans without decoupling path planning from robot dynamics [51, 26, 18, 25].

### 2.1.2 Medium Horizon Navigation Decision Making

MH-NDM approaches generate plans for reaching intermediate goals during navigation, and are sometimes called local planners. Given the current state of the the environment, they generate plans for the medium horizon future of the navigation task, e.g., the next two to ten seconds. They are typically used in a receding horizon fashion, generating plans that are executed for a short duration followed by re-planning. The re-planning frequency of MH-NDM approaches typically vary from 1 Hz to 20 Hz. MH-NDM algorithms are expected to run in real-time because of the receding horizon execution, requiring on-board execution unless realiable communication maintenance from a centralized computer to the robots is possible. The real-time planning requirement makes it difficult to have completeness guarantees within the re-planning period. On-board MH-NDM allows integration of new information about the environment that may be collected during navigation, making them a viable choice in unknown or partially known environments and tasks with imperfect communication media. In addition, since their decision making horizon is not long, the robot does not need to maintain the full representation of the environment, allowing on-board local environment representation maintenance. On-board execution complements with decentralization of planning when there are multiple robots to be controlled, in which, each robot computes its own plan, while possibly coordinating with other robots through explicit communication or implicit behavioral assumptions, e.g., assuming that teammates run the same algorithm. Dynamic feasibility of the generated plans are frequently pursued in MH-NDM. Generally, the feasibility of most MH-NDM algorithm is not guaranteed, i.e., planning iterations might fail. In those cases, robot continues using the plan from the previous planning iteration.

Many single robot collision-free polynomial trajectory generation methods for static obstacle avoidance can be used for MH-NDM. Richter, Bry, and Roy [94] use RRT* [46] to find a collision-free path in environments with only static obstacles, followed by solving a quadratic program to smoothen the path to a dynamically feasible piecewise polynomial trajectory. Polynomial spline trajectory generation based on local trajectory optimization in which collision avoidance against static obstacles is integrated into the cost function [73] is proposed, which makes safety a soft contraint. Chen, Liu, and Shen [17] use standard A* search where octrees [41] are used for static obstacle representation to plan a kinematically safe path, and smooths it safely by containing the final plan in a safe navigation corridor (SNC) lying in the empty octree cells. Computing a collision-free discrete path using jump point search (JPS) [37] followed by SNC based polynomial optimization is also proposed [58]. Usenko et al. [118] propose a B-spline [83] generation algorithm using locally built maps, which are represented with 3D circular buffers. A piecewise Bézier curve [85] based polynomial trajectory generation algorithm [33] in which an initial collision-free trajectory is computed using fast marching [104], followed by optimization within an SNC is also proposed. Tordesillas et al. [115] combine JPS with SNC construction and solve an optimization problem to compute a piecewise polynomial that allows navigation in unknown environments.

Decision making approaches for avoiding static and dynamic obstacles and integrating uncertainty associated with several sources (e.g., unmodeled system dynamics, state estimation inaccuracy, perception noise, or prediction inaccuracies) are proposed, which can be used as MH-NDM modules. Tordesillas and How [114] propose a polynomial trajectory planner to avoid static obstacles and dynamic obstacles given their predicted trajectories along with a maximum prediction error. Qi et al. [87] combine motion primitive search with spline optimization for static and dynamic obstacle avoidance. Chance constrained RRT (CC-RRT) [59] plans trajectories to avoid static and dynamic obstacles, limiting the probability of collisions under Gaussian system and prediction noise. Aoude et al. [4] perform trajectory prediction using Gaussian mixture models to estimate motion models of dynamic obstacles, and use these models within a RRT variant to predict their trajectories as a set of trajectory particles within CC-RRT to compute and limit collision probabilities. Zhu and Alonso-Mora [139] propose a chance-constrained model-predictive control (MPC) formulation for static and dynamic obstacle avoidance where uncertainty stems from Gaussian system model and state estimation noise, and dynamic obstacle model noise where dynamic obstacles are modeled using constant velocities with Gaussian acceleration noise. RAST [16] is a risk-aware planner that does not require segmenting obstacles into static and dynamic, but uses a particle based occupancy map in which each particle is associated with a predicted velocity; and Nair, Tseng, and Borrelli [70] propose a MPC-based collision avoidance method where uncertainty stems from system noise of the robot and prediction noise for dynamic obstacles. Janson, Schmerling, and Pavone [43] use a Monte Carlo sampling to compute collision probabilities of trajectories under system uncertainty.

Many decentralized decision making approaches have been proposed for cooperative navigation of multiple robots, which can be deployed as MH-NDM modules. Zhou et al. [135] utilize buffered Voronoi cells (BVC) within a model predictive control (MPC) framework, where each robot stays within its cell in each planning iteration. BVC requires position-only sensing and does not depend on inter-robot communication. Wang et al. [123] present a distributed model predictive control (DMPC) scheme that requires full state sensing between robots. Utilizing a DMPC scheme with full plan communication is also proposed [61, 60]. Differential flatness [69] of the underlying systems is utilized to plan in the output space instead of the input space by many planners, which allows planning continuous splines with limited derivative magnitudes to ensure dynamic feasibility. RTE [100] uses buffered Voronoi cells in a spline optimization framework and combines the optimization with discrete planning to locally resolve deadlocks. Obstacle avoidance is ensured using safe navigation corridors (SNC) during optimization. MADER [114] combines discrete planning with spline optimization, treating SNC constraints as decision variables in a non-linear optimization problem. MADER explicitly accounts for asynchronous planning using communication, yet it assumes instantaneous perfect communication between robots. RMADER [50] extends MADER to handle communication delays with known bounds between teammates. RSFC [79, 77] plans for piecewise splines with Bézier curve [29] pieces where safety between robots is ensured by making sure that their relative trajectories are in a safe set, where trajectories between robots are shared with instantaneous perfect communication. LSC [76] extends RSFC by using linear safety constraints without slack variables, which may cause the final solution to be unsafe in RSFC. DLSC [80] extends LSC to dynamic obstacles, and maze-like environments. Ego-swarm [136] formulate the collision avoidance as a cost function, which they optimize using gradient based local optimization. TASC [117, 116] uses SNCs, which it computes between the communicated plans of other robots and the last plan of the planning robot. TASC accounts for bounded communication delays between robots. In Peterson et al. [82]'s approach, which operates in a receding horizon fashion, robot tasks are defined as time window temporal logic specifications. The method can plan collision-free trajectories with finite temporal relaxations of the task specifications and provably avoid deadlocks, but requires communication of planned trajectories between robots.

### 2.1.3 Short Horizon Navigation Decision Making

SH-NDM approaches generate only the next actions to execute in each decision making iteration. They aim for collision-free operation after applying generated actions to the system and some maintain the invariance of a safe set [10], ensuring that there will always be safe actions to execute for all future time. Ensuring the feasibility of the algorithm is another objective, which is closely related to maintaining invariance of the safe set, in which, the robot never fails to compute a safe action or, in some cases, an action that drives the system eventually to safety. Since these approaches produce only the next action, they typically run at higher frequency than the medium horizon approaches, e.g., from 50 Hz to 200 Hz, depending on the complexity of the robot dynamics. Since their reasoning horizons are short and a high frequency execution is required, they typically run on-board using local environment information to ensure safety. Robots navigating solely using these approaches tend to deadlock, since they generally do not consider task completion during decision making. Integrating learned models for reducing deadlocks is proposed [96, 20], but, all existing methods are prone to deadlocks. Robots navigating with SH-NDM algorithms may cooperate with each other explicitly via communication, or implicitly via behavioral assumptions, e.g., teammates running the same algorithm, similar to MH-NDM.

Optimal reciprocal collision avoidance (ORCA) [8, 3], which is based on velocity obstacles [31], computes safe velocities that are as close as possible to desired velocities, allowing static and dynamic obstacle and teammate avoidance. Wang, Ames, and Egerstedt [121] present a safety barrier certificates (SBC) [84] based method computing accelerations that are as close as possible to desired accelerations, providing collision avoidance guarantee against static obstacles, teammates, and dynamic obstacles moving with constant velocities. Both ORCA and SBC are *minimally invasive* algorithms, making desired control actions safe by minimally altering them. Wang, Ames, and Egerstedt [122] make reference trajectories for quadrotors safe in a miniminally invasive manner using SBCs by explicitly exploiting the differential flatness of quadrotors [66]. Wu and Sreenath [126] compute forces and moments directly for obstacle avoidance in quadrotors. Zhou et al. [135]'s approach can be used as a short horizon decision making module by limiting the number of planned inputs by one. GLAS [96] combines a learned network that is trained to mimic a long horizon planner [39] with a theoretical safety module to provide collision-free navigation capability while decreasing deadlocks. Batra et al. [7] utilize reinforcement learning (RL) to train neural networks with direct thrust command outputs for quadrotors, allowing robot-robot collision avoidance. Cui et al. [20] combine a RL trained network with a safety barrier certificates based module to guarantee the collision-free operation. Several methods utilize RL to train neural networks to navigate robots on grid maps, which is a special case of the MAPF problem, in which action space is simplified to directions on a grid [97, 21, 119, 57].

Artificial potential fields [47] are applied to (multi-)robot collision avoidance, which can be deployed as SH-NDM modules. In the most generic sense, artificial potential fields are functions from states to actions, describing the behavior of robots and have been extensively studied in robot navigation; obstacles are modeled using repulsive fields and navigation goals are modeled using attractive fields. Navigation functions [95], a special case of artificial potential fields, can solve the exact robot navigation problem with perfect information where obstacles are spherical; subsequent work has applied them to dynamic obstacle avoidance [34]. When perfect information is not available, local minima may cause robots to deadlock. Some approaches attempt to decrease deadlocks by adding virtual obstacles [81], switching between multiple fields [30], or using simulated annealing [140]. Yongjie and Yan [129] utilize artificial potential fields for decentralized multi-robot collision avoidance. Dynamically adjusting artificial potential fields to escape local minima is also investigated for multi-robot formation control [134]. Many flocking algorithms can be considered artificial potential field based methods and can be used as a short horizon decision making module, in which robots use simple rules to avoid collisions, navigate to goal positions, and produce coherent swarm behavior. Boids [93] is one of the oldest flocking approaches, which is initially developed for swarm behavior in computer graphics. The agents compute a linear combination of three simple rules, i.e., collision avoidance, flock centering, velocity matching, to produce a coherent swarm behavior. Tolstaya et al. [112] proposes using graph neural networks (GNNs) to produce flocking behavior, in which, GNNs are used to propogate information about far away agents using local communication.

## 2.2 Predicting Dynamic Obstacle Behaviors

While some decision making approaches utilize only the current states of dynamic obstacles to avoid them, other set of approaches require prediction of the behaviors of dynamic obstacles. Predicting future states of dynamical systems is studied extensively. Many recent approaches have been developed in the autonomous vehicles domain specifically for dynamic obstacles, which typically constitute humans or other vehicles.

Wiest et al. [125] use Gaussian mixture models to estimate a Gaussian distribution over future states of a vehicle given its past states. Lee et al. [53] predict future trajectories of dynamic obstacles by learning a posterior distribution over future dynamic obstacle trajectories given past trajectories. Multi-modal prediction for vehicles to tackle bias against unlikely future trajectories during training is also investigated [48]. Bartoli et al. [6] present a method for human movement prediction using context information modeling human-human and human-static obstacle interactions. Zhou et al. [138] generate multi-modal pedestrian predictions utilizing and modeling social interactions between humans and human intentions. Goal directed prediction [90, 91, 64] converts the prediction problem to a planning problem, in which, the behavior of predicted objects are modeled as a distribution on future states conditioned on goal distributions.

# Chapter 3

# Preliminaries

In this chapter, we introduce some essential concepts that are used repeatedly in the following chapters.

## 3.1 Hyperplanes, Half-spaces, Convex Sets, Convex Polytopes, and Convex Hulls

**Definition 1.** *An affine hyperplane, or shortly a* hyperplane *in our discussion, $\mathcal{H} \in \mathcal{H}^d$ can be defined by a normal vector $\mathcal{H}_{\mathbf{n}} \in \mathbb{R}^d$ and an offset $\mathcal{H}_a$ as $\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^d \mid \mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a = 0\}$ where $\mathcal{H}^d$ is the set of all hyperplanes in $\mathbb{R}^d$ and $\mathcal{H}_{\mathbf{n}} \neq \mathbf{0}$.*

**Definition 2.** *A (closed)* half-space *$\mathcal{H}^- \subseteq \mathbb{R}^d$ is the subset of $\mathbb{R}^d$ bounded by a hyperplane such that $\mathcal{H}^- = \{\mathbf{x} \in \mathbb{R}^d \mid \mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \leq 0\}$ where $\mathcal{H}_{\mathbf{n}} \neq \mathbf{0}$.*

**Definition 3.** *A set $A \subseteq \mathbb{R}^d$ is convex, if convex combination $t\mathbf{x} + (1-t)\mathbf{y} \in A$ for all $\mathbf{x}, \mathbf{y} \in A$, $t \in [0, 1]$.*

**Lemma 1.** *Half-spaces are convex sets.*

*Proof.* See Section 2.2.1 in [13]. □

**Lemma 2.** *Intersection of any number of convex sets is a convex set.*

*Proof.* See Section 2.3.1 in [13]. □

**Definition 4.** *A* convex polytope *is an intersection of a finite number of half-spaces, and hence a convex set.*

**Definition 5.** Convex hull $ConvexHull(A) \subseteq \mathbb{R}^d$ *of a set* $A \subseteq \mathbb{R}^d$ *is the smallest convex set containing* $A$*. Equivalently, it can be defined as the set containing all convex combinations* $\sum \alpha_i \mathbf{x}_i, \alpha_i \geq 0, \sum \alpha_i = 1$ *of all points* $\mathbf{x}_i \in A$*. Convex hull of a finite set of points is a convex polytope.*

**Definition 6.** *Two sets* $A, B \subseteq \mathbb{R}^d$ *are called* linearly separable *if there is a hyperplane* $\mathcal{H}$ *that separates them, i.e.,* $\mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \leq 0$ *for all* $\mathbf{x} \in A$ *and* $\mathcal{H}_{\mathbf{x}}^\top \mathbf{x} + \mathcal{H}_a \geq 0$ *for all* $\mathbf{x} \in B$*. They are called* strictly linearly separable *if inequalities are strict.*

**Theorem 1.** ***Separating hyperplane theorem.*** *Let* $A, B \subseteq \mathbb{R}^d$ *be two disjoint non-empty convex sets. Then, they are linearly separable, i.e., there exist a hyperplane* $\mathcal{H} \in \mathcal{H}^d$ *such that* $\mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \leq 0$ *for all* $\mathbf{x} \in A$ *and* $\mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \geq 0$ *for all* $\mathbf{x} \in B$ *where* $\mathcal{H}_{\mathbf{n}} \neq \mathbf{0}$*. If, in addition,* $A$ *and* $B$ *are closed, and at least one of them is compact, they are strictly linearly separable.*

*Proof.* See Section 2.5.1 in [13]. □

### 3.1.1 Hard-margin Support Vector Machines

Let $A, B \subseteq \mathbb{R}^d$ be two strictly linearly separable sets as per Definition 6. The hard-margin support vector machine hyperplane [19] between $A$ and $B$ separates $A$ and $B$ such that the minimum distance between $A$ and $B$ along the normal of the hyperplane, i.e., the margin, is maximal. Formally, it is the solution to the following convex quadratic optimization problem[*]

$$\min_{\mathcal{H}_{\mathbf{n}}, \mathcal{H}_a} \|\mathcal{H}_{\mathbf{n}}\|_2^2 \ s.t.$$

$$\mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \leq -1 \ \forall \mathbf{x} \in A$$

$$\mathcal{H}_{\mathbf{n}}^\top \mathbf{x} + \mathcal{H}_a \geq 1 \ \forall \mathbf{x} \in B.$$

[*] We discuss convex optimization problems in Section 3.2.1.

Since the problem is convex, it has a unique global minimum. In addition, the hard-margin support vector machine problem as defined above has a unique solution [11] when the sets are linearly separable.

### 3.1.2 Voronoi Tesselation

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a finite non-empty set of distinct points. The Voronoi tesselation [23] of $\mathbb{R}^d$ associated $\mathcal{X}$ is the partitioning of $\mathbb{R}^d$ to $|\mathcal{X}|$ regions $\mathcal{V}_1, \dots, \mathcal{V}_{|\mathcal{X}|}$, one for each point $\mathbf{x}_i \in \mathcal{X}$, such that points in $\mathcal{V}_i$ are closer or equidistant to $\mathbf{x}_i$ than any other $\mathbf{x}_j$ $\forall j \neq i$ in L2 distance. Formally,

$$\mathcal{V}_i = \{\mathbf{x} \in \mathbb{R}^d \mid \|\mathbf{x} - \mathbf{x}_i\|_2 \leq \|\mathbf{x} - \mathbf{x}_j\|_2 \ \forall j \neq i\}.$$

$\mathcal{V}_i$ is called the Voronoi cell of point $\mathbf{x}_i$.

Each inequality $\|\mathbf{x} - \mathbf{x}_i\|_2 \leq \|\mathbf{x} - \mathbf{x}_j\|_2$ for a $j \neq i$ is the half-space with normal $\mathcal{H}_{\mathbf{n}} = \mathbf{x}_j - \mathbf{x}_i$ and offset $\mathcal{H}_a = \frac{\mathbf{x}_i^\top \mathbf{x}_i - \mathbf{x}_j^\top \mathbf{x}_j}{2}$. Hence, Voronoi cells are intersections of finite number of half-spaces, and therefore convex polytopes.

## 3.2 Continuous Optimization

In the general sense, a continuous optimization problem seeks the minimum value of a real valued function of one or more real decision variables subject to inequality constraints, and has the form

$$\min_{\mathbf{x}} f(\mathbf{x}) \ s.t. \tag{3.1}$$

$$g_i(\mathbf{x}) \leq 0 \ \forall i \in \{1, \dots, M\} \tag{3.2}$$

where $\mathbf{x} \in \mathbb{R}^d$ are the decision variables, $f : \mathbb{R}^d \to \mathbb{R}$ is the objective function, $g_i : \mathbb{R}^d \to \mathbb{R}, i \in \{1, \dots, M\}$ are $M$ constraint functions. It is possible to use other equivalent forms, e.g., with equality constraints.

A *globally optimal solution*, or simply a solution, to the optimization problem constitute values $\mathbf{x}$ minimizing $f(\mathbf{x})$ among the values satisfying the constraints $g_i(\mathbf{x}) \leq 0 \; \forall i \in \{1, \ldots, M\}$. Finding a globally optimal solution for general classes of $f$ and $g_i$ is often intractable. However, *convex optimization problems* permit efficient algorithms for finding globally optimal solutions.

### 3.2.1 Convexity of Optimization Problems

**Definition 7.** *A real valued function $f : \mathbb{R}^d \to \mathbb{R}$ of one or more real valued variables is called convex if it satisfies, $f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}) \; \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d \; \forall t \in [0, 1]$.*

**Definition 8.** *A continuous optimization problem of Equations* (3.1) *and* (3.2) *is called convex, if objective function $f$, and constraint functions $g_i, i \in \{1, \ldots, M\}$ are convex.*

Globally optimal solutions for convex optimization problems can be found in polynomial time in dimension and accuracy, using, for instance, interior point methods [71].

### 3.2.2 Linear and Quadratic Optimization Problems

Linear optimization problems, or linear programs (LP), are continuous optimization problems in which the objective function $f$ and constraint functions $g_i$ are affine functions of the decision variables in the Equations (3.1) and (3.2) form. Linear optimization problems are convex by definition, because affine functions are convex. While solution methods for convex optimization can be readily used to solve LPs, there are many specialized solution methods exploiting the linear structure of the problem, e.g., the simplex method [22].

Quadratic optimization problems, or quadratic programs (QP), are continuous optimization problems in which the objective function $f$ is a quadratic and constraint functions $g_i$ are affine functions of the decision variables. Unlike LPs, QPs are not necessarily convex. Solving general QPs is known to be NP-Hard [75] and only locally optimal solutions are typically computed. However, a globally optimal solution for convex QPs can be found in polynomial time, and specialized solution methods exist [128] that exploit the structure of the problem.

## 3.3 Bézier Curves

A Bézier curve $\mathbf{f} : [0, T] \to \mathbb{R}^d$ of degree $h$ is defined by $h + 1$ control points $\mathbf{P}_0, \ldots, \mathbf{P}_h \in \mathbb{R}^d$ as follows:

$$\mathbf{f}(t) = \sum_{i=0}^{h} \mathbf{P}_i \binom{h}{i} \left(\frac{t}{T}\right)^i \left(1 - \frac{t}{T}\right)^{(h-i)}.$$

Any Bézier curve $\mathbf{f}$ satisfies $\mathbf{f}(0) = \mathbf{P}_0$ and $\mathbf{f}(T) = \mathbf{P}_h$, i.e., it starts at the first control point and ends at the last. Other points guide the curve from $\mathbf{P}_0$ to $\mathbf{P}_h$. Since any Bézier curve $\mathbf{f}$ is a polynomial of degree $h$, it is infinitely differentiable and smooth, meaning $\mathbf{f} \in C^\infty$.

### 3.3.1 Convex Hull Property of Bézier Curves

Bézier curves lie within the convex hull of their control points, i.e., $\mathbf{f}(t) \in ConvexHull(\{\mathbf{P}_0, \ldots, \mathbf{P}_h\})\, \forall t \in [0, T]$ [29]. Tordesillas and How [113] discuss the conservativeness of the convex hulls of control points of Bézier curves and show that the convex hulls are considerably less conservative than those of B-Splines [83], which are another type of curve with the convex hull property. Yet, they also show that convex hulls of the control points of the Bézier curves can be considerably more conservative compared to the smallest possible convex sets containing the curves.

## 3.4 Differential Flatness of Dynamical Systems

A system is called differentially flat [69] when its states and inputs can be expressed in terms of the outputs and finite derivatives of them. When robots are differentially-flat, their dynamics can be accounted for by i) imposing continuity up to the required degree on the flat outputs, and ii) imposing constraints on the maximum derivative magnitudes of the flat outputs. Differential flatness is common for many kinds of mobile robots, including differential drive robots [15], car-like robots [68], omnidirectional robots [44], and quadrotors [66].

# Chapter 4

# Decentralized Real-time Networkless Multi-robot Trajectory Planning for Static Obstacle and Synchronous Teammate Avoidance

Trajectory planning for multiple robots in shared environments is a challenging problem especially when there is limited communication available or no central entity. In this chapter, we present Real-time planning using Linear Spatial Separations, or RLSS: a real-time decentralized trajectory planning algorithm for cooperative multi-robot teams in static environments. The algorithm requires relatively few robot capabilities, namely i) sensing the positions of robots and obstacles without higher-order derivatives and ii) the ability of distinguishing robots from obstacles. There is no communication requirement and the robots' dynamic limits are taken into account. RLSS generates and solves quadratic optimization problems that are kinematically feasible and guarantees collision avoidance if the resulting problems are feasible. We demonstrate the algorithm's performance in real-time in simulations and on physical robots. We compare RLSS to two state-of-the-art planners and show empirically that RLSS does avoid deadlocks and collisions in forest-like and maze-like environments, significantly improving prior work, which result in collisions and deadlocks in such environments.

This chapter is based on Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. "RLSS: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations". In: *Autonomous Robots* (2023), pp. 1–26. DOI: 10.1007/s10514-023-10104-w.

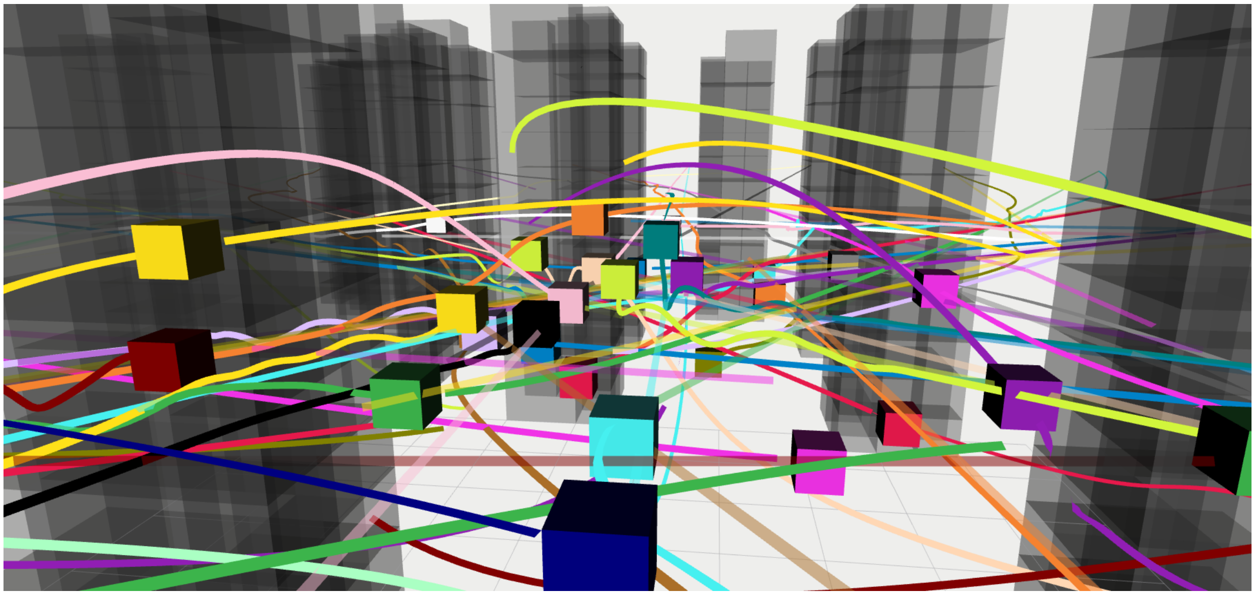


Figure 4.1: RLSS runs in real-time in dense environments. Each robot plans a trajectory by itself using only the position information of other robots and obstacles.

## 4.1 Introduction

In this chapter, we introduce RLSS, a real-time decentralized trajectory planning algorithm for multiple robots in a shared environment that requires no communication between robots and requires relatively few sensing capabilities: each robot senses the relative positions of other robots and obstacles along with their geometric shapes in the environment, and is able to distinguish robots from obstacles. RLSS requires relatively few robot capabilities than most state-of-the-art decentralized planners used for multi-robot navigation, which typically require communication [60, 114, 123], higher order derivative estimates [77, 123], or predicted trajectories of objects [77]. However, the ability to distinguish robots from obstacles is not required by some state-of-the-art algorithms [135], which require modelling obstacles as robots. RLSS is cooperative in the sense that we assume each robot stays within its current cell of a tessellation of the space until the next planning iteration. We assume obstacles are static, which is required to guarantee collision avoidance.

RLSS explicitly accounts for the dynamic limits of the robots and enforces safety with hard constraints, reporting when they cannot be enforced, thus guaranteeing collision avoidance when it succeeds. The planning algorithm can be guided with desired trajectories, thus it can be used in conjunction with long horizon planners. If no long horizon planner or plan is available, RLSS can be used on its own, without any central guidance, by setting the desired trajectories to line segments directly connecting robots' start positions to their goal positions.

There are four stages in RLSS.

1. Select a goal position to plan toward on the desired trajectory;

2. Plan a discrete path toward the selected goal position;

3. Formulate and solve a kinematically feasible optimization problem to compute a safe smooth trajectory guided by the discrete plan;

4. Check if the trajectory obeys the dynamic limits of the robot, and temporally rescale the trajectory if not, and go to step 3.

RLSS works in the receding horizon fashion: it plans a long trajectory, executes it for a short duration, and replans at the next iteration. It utilizes separating hyperplanes, i.e., linear spatial separations, between robot shapes, obstacles, and sweep geometries (the subsets of space swept by robots while traversing straight line segments) to enforce safety during trajectory optimization.

We demonstrate, through simulation and experiments on physical robots, that RLSS works in dense environments in real-time (Figure 4.1). We compare our approach to two state-of-the-art receding horizon decentralized multi-robot trajectory planning algorithms in 3D. In the first, introduced by Zhou et al. [135], robots plan for actions using a model predictive control-style optimization formulation while enforcing that each robot stays inside its buffered Voronoi cell at every iteration. We refer to this method as BVC. The original BVC formulation works only for discrete single-integrator dynamics and environments without obstacles. We extend the BVC formulation to discrete time-invariant linear systems with position outputs and environments with obstacles, and call the extended version eBVC (short for extended BVC). RLSS and eBVC have similar properties: they require no communication between robots, and require position sensing of other objects in the environment. However, unlike RLSS, eBVC does not require that robots are able to distinguish robots from obstacles, as it treats each obstacle as a robot. In the second, introduced by Park and Kim [77], robots plan for trajectories by utilizing relative safe navigation corridors, which they execute for a short duration, and replan. We refer to this method as RSFC. RSFC requires no communication between robots, and utilizes positions as well as velocities of the objects in the environment, thus requires more sensing capabilities than RLSS. We demonstrate empirically that RLSS results in no deadlocks or collisions in our experiments in forest-like and maze-like environments, while eBVC is prone to deadlocks and RSFC results in collisions in such environments. However, RLSS results in slightly longer navigation durations compared to both eBVC and RSFC.

The contribution of this chapter can be summarized as follows:

- A carefully designed, numerically stable, and effective real-time decentralized planning algorithm for multiple robots in shared environments with static obstacles with relatively few requirements: no communication between robots, position-only sensing of robots and obstacles, and ability to distinguish robots from obstacles.

- An extension (eBVC) of a baseline planner (BVC) to environments with obstacles and a richer set of dynamics than only single-integrators.

- The first comparison of more than two state-of-the-art communication-free decentralized multi-robot trajectory planning algorithms, namely, RSFC, eBVC, and RLSS, in complicated forest-like and maze-like environments, some of which containing more than 2000 obstacles.

## 4.2 Situating within Related Work

The pipeline of RLSS contains three stages (discrete planning, safe navigation corridor construction, and trajectory optimization) that are employed by several existing single-robot trajectory planning algorithms. Richter, Bry, and Roy [94] present a single-robot trajectory planning method for aggressive quadrotor flight which utilizes RRT* [46] to find a kinematically feasible discrete path and formulates an unconstrained quadratic program over polynomial splines guided by the discrete path. Collisions are checked after optimization, and additional decision points are added and optimization is re-run if there is collision. Chen, Liu, and Shen [17] present a method that utilizes octree representation [41] of the environment during discrete search. They find a discrete path using unoccupied grid cells of the octree. Then, they maximally inflate unoccupied grid cells to create a safe navigation corridor that they use as constraints in the subsequent polynomial spline optimization. Liu et al. [58] uses Jump Point Search (JPS) [37] as the discrete planner, and construct safe navigation corridors that are used as constraints in the optimization stage. Our planning system uses these three stages (discrete planning, safe navigation corridor construction, and corridor-constrained optimization) and extends it to multi-robot scenarios in a decentralized way. We handle robot-robot safety by cooperatively computing a linear partitioning of the environment, and enforcing that each robot stays within its own cell of the partition.

RLSS falls into the medium horizon navigation decision making algorithms that are deployed in a decentralized fashion. To recap, these approaches employ receding horizon planning: they plan medium horizon trajectories, execute them for a short duration, and replan. Examples include [135], which presents a model predictive control (MPC) based approach in which robots plan for a sequence of actions while enforcing each robot to stay within its buffered Voronoi cell. It requires no inter-robot communication and depends on sensing of other robots' positions only. Another MPC-based approach approximates robots' controller behaviors under given desired states as a linear system [60]. A smooth Bézier curve for each robot is computed by solving an optimization problem, in which samples of desired states are drawn from the curve and fed into the model of the system. The approach requires communication of future states for collision avoidance. Another MPC-based approach generates plans using motion primitives to compute time-optimal trajectories, then trains a neural network to approximate the behavior of the planner, which is used during operation for fast planning [123]. It requires sensing or communicating the full states of robots. Tordesillas and How [114] present a method for dynamic obstacle avoidance, handling asynchronous planning between agents in a principled way. However, the approach requires instant communication of planned trajectories between robots to ensure safety. Park and Kim [77] plan a piecewise Bézier curve by formulating an optimization problem. It utilizes relative safe flight corridors (RSFCs) for collision avoidance and requires no communication between robots, but requires position and velocity sensing. In Peterson et al. [82]'s approach, robot tasks are defined as time window temporal logic specifications. Robots plan trajectories to achieve their tasks while avoiding each other. The method provably avoids deadlocks, but requires communication of planned trajectories between robots.

RLSS requires sensing only the positions of other robots and obstacles, and does not require any communication between robots. The work of Zhou et al. [135] (BVC) is the only other approach with these properties. Different from their approach, RLSS uses discrete planning to avoid local minima, plans piecewise Bézier trajectories instead of MPC-style input planning, and does not constrain the full trajectory to be inside a constraining polytope. These result in better deadlock and collision avoidance as we show in the evaluation section.

In terms of inter-robot collision avoidance, RLSS belongs to a family of algorithms that ensure multi-robot collision avoidance without inter-robot communication by using the fact that each robot in the team runs the same algorithm. In these algorithms, robots share the responsibility of collision avoidance by computing feasible action sets that would not result in collisions with others when others compute their own feasible action sets using the same algorithm. Examples include BVC [135], which partitions position space between robots and makes sure that each robot stays within its own cell in the partition, ORCA [3], which computes feasible velocity sets for robots such that there can be no collisions between robots as long as each robot chooses a velocity command from their corresponding sets, and SBC [121], which does the same with accelerations.

BVC and RLSS also belong to a family of decentralized multi-robot trajectory planning algorithms that utilize mutually computable separating hyperplanes for inter-robot safety that we define in Chapter 5. These algorithms do not require any inter-robot communication and utilize only position/geometry sensing between robots. The inter-robot safety is enforced by making sure that each pair of robots can compute the *same separating hyperplane* and enforce themselves to be in the different sides of this hyperplane at every planning iteration. Separating hyperplanes that can be mutually computed by robots using only position/geometry sensing and no communication are defined as mutually computable. BVC uses Voronoi hyperplanes and RLSS uses support vector machine hyperplanes, both of which are mutually computable.

RLSS is an extension to our previous conference paper [100]. We extend it conceptually by

- supporting robots with any convex geometry instead of only spherical robots; and
- decreasing the failure rate of the algorithm from $3\%$ in complex environments to $0.01\%$ by providing important modifications to the algorithm. These modifications include i) changing the discrete planning to best-effort $A^*$ search, thus allowing robots to plan towards their goals even when goals are not currently reachable; ii) increasing the numerical stability of the algorithm by running discrete planning at each iteration instead of using the trajectory of the previous iteration when it is collision free to define the homotopy class, and adding a *preferred distance cost term* to the optimization problem in order to create a safety zone between objects when possible; and iii) ensuring the kinematic feasibility of the optimization problem generated at the trajectory optimization stage.

We extend our previous work empirically by

- applying our algorithm to a heterogeneous team of differential drive robots in 2D;

- applying our algorithm to quadrotors in 3D; and
- comparing the performance of our algorithm to stronger baselines.

## 4.3 Problem Statement

We first define the multi-robot trajectory planning problem, then, we indicate the specific case of the problem we solve.

Consider a team of $N$ robots for which trajectories must be computed at time $t = 0$. The team can be heterogeneous, i.e., each robot might be of a different type or shape. Let $\mathcal{R}_i$ be the collision shape function of robot $i$ such that $\mathcal{R}_i(\mathbf{p})$ is the collision shape of robot $i$ located at position $\mathbf{p} \in \mathbb{R}^d, d \in \{2, 3\}$, i.e., the subset of space occupied by robot $i$ when placed at position $\mathbf{p}$. We define $\mathcal{R}_i(\mathbf{p}) = \{\mathbf{p} + \mathbf{x} \mid \mathbf{x} \in \mathcal{R}_{i,0}\}$ where $\mathcal{R}_{i,0} \subset \mathbb{R}^d$ is the the space occupied by robot $i$ when placed at the origin. Note that we do not model robot orientation. If a robot can rotate, the collision shape should contain the union of spaces occupied by the robot for each possible orientation at a given position; which is minimally a hypersphere if all orientations are possible.

We assume the robots are differentially flat (Section 3.4) in their position outputs.

Let $\mathcal{O}(t) = \{\mathcal{Q} \subset \mathbb{R}^d\}$ be the set of obstacles in the environment at time $t$. The union $\mathcal{Q}(t) = \cup_{\mathcal{Q} \in \mathcal{O}(t)} \mathcal{Q}$ denotes the occupied space at time $t$. Let $\mathcal{W}_i \subseteq \mathbb{R}^d$ be the workspace that robot $i$ should stay inside of.

Let $\mathbf{d}_i(t) : [0, T_i] \rightarrow \mathbb{R}^d$ be the desired trajectory of duration $T_i$ that robot $i$ should follow as closely as possible. Define $\mathbf{d}_i(t) = \mathbf{d}_i(T_i)\ \forall t \geq T_i$. We do not require that this trajectory is collision-free or even dynamically feasible to track by the robot. If no such desired trajectory is known, it can be initialized, for example, with a straight line to a goal location.

The intent is that each robot $i$ tracks a Euclidean trajectory $\mathbf{f}_i(t) : [0, T] \rightarrow \mathbb{R}^d$ such that $\mathbf{f}_i(t)$ is collision-free, executable according to the robot's dynamics, is as close as possible to the desired trajectory $\mathbf{d}_i(t)$, and ends at $\mathbf{d}_i(T_i)$. Here, $T$ is the navigation duration of the team. We generically define the multi-robot trajectory planning problem as the problem of finding trajectories $\mathbf{f}_1, \ldots, \mathbf{f}_N$ and navigation duration $T$ that optimizes the following formulation:

$$\min_{\mathbf{f}_1,\ldots,\mathbf{f}_N,T} \sum_{i=1}^{N} \int_0^T \left\|\mathbf{f}_i(t) - \mathbf{d}_i(t)\right\|_2 dt \text{ s.t.} \tag{4.1}$$

$$\mathbf{f}_i(t) \in C^{c_i} \qquad \forall i \in \{1, \ldots, N\} \tag{4.2}$$

$$\mathbf{f}_i(T) = \mathbf{d}_i(T_i) \qquad \forall i \in \{1, \ldots, N\} \tag{4.3}$$

$$\frac{d^c\mathbf{f}_i(0)}{dt^c} = \frac{d^c\mathbf{p}_i^0}{dt^c} \qquad \forall i \forall c \in \{0, \ldots, c_i\} \tag{4.4}$$

$$\mathcal{R}_i(\mathbf{f}_i(t)) \cap \mathcal{Q}(t) = \emptyset \qquad \forall i \forall t \in [0, T] \tag{4.5}$$

$$\mathcal{R}_i(\mathbf{f}_i(t)) \cap \mathcal{R}_j(\mathbf{f}_j(t)) = \emptyset \qquad \forall j \neq i \forall t \in [0, T] \tag{4.6}$$

$$\mathcal{R}_i(\mathbf{f}_i(t)) \in \mathcal{W}_i \qquad \forall i \forall t \in [0, T] \tag{4.7}$$

$$\max_{t \in [0,T]} \left\| \frac{d^k\mathbf{f}_i(t)}{dt^k} \right\|_2 \leq \gamma_i^k \qquad \forall i \forall k \in \{1, \ldots, K_i\} \tag{4.8}$$

where $c_i$ is the order of derivative up to which the trajectory of the $i^{th}$ robot must be continuous, $\mathbf{p}_i^0$ is the initial position of robot $i$ (derivatives of which are the initial higher order state components, e.g., velocity, acceleration, etc.), $\gamma_i^k$ is the maximum $k^{th}$ derivative magnitude that the $i^{th}$ robot can execute, and $K_i$ is the maximum derivative degree that robot $i$ has a derivative magnitude limit on.

The cost (4.1) of the optimization problem is a metric for the total deviation from the desired trajectories; it is the sum of position distances between the planned and the desired trajectories. (4.2) enforces that the trajectory of each robot is continuous up to the required degree of derivatives. (4.3) enforces that planned trajectories end at the endpoints of desired trajectories. (4.4) enforces that the planned trajectories have the same initial position and higher order derivatives as the initial states of the robots. (4.5) and (4.6) enforce robot-obstacle and robot-robot collision avoidance, respectively. (4.7) enforces that each robot stays within its defined workspace. Lastly, (4.8) enforces that the dynamic limits of the robot are obeyed.

Note that only constraint (4.6) stems from multiple robots. However, this seemingly simple constraint couples robots' trajectories both spatially and temporally, making the problem much harder. As discussed in Section 2.1.1, solving the multi-agent path finding problem optimally is NP-Hard even for the discrete case while the discrete single-agent path finding problem can be solved optimally with classical search methods in polynomial time. This curse of dimensionality affects continuous motion planning as well, where even geometric variants are known to be PSPACE-Hard [40].

A centralized planner can be used to solve the generic multi-robot trajectory planning problem in one-shot provided that: the current and future obstacle positions are known a priori, computed trajectories can be sent to robots over a communication link, and robots can track these trajectories well enough that they do not violate the spatio-temporal safety of the computed trajectories. In the present work, we aim to approximately solve this problem in the case where obstacles are static, but not known a priori by a central entity, and there is no communication channel between robots or between robots and a central entity. Each robot plans its own trajectory in real-time. They plan at a high frequency to compensate for trajectory tracking errors.

## 4.4 Assumptions

Here, we list our additional assumptions about the problem formulation defined in Section 4.3 and the capabilities of robots.

We assume that obstacles in the environment are static, i.e., $\mathcal{O}(t) = \mathcal{O}\ \forall t \in [0, \infty]$, and convex[*]. Many existing, efficient, and widely-used mapping tools, including occupancy grids [38] and octrees [41], internally store obstacles as convex shapes; such maps can be updated in real-time using visual or RGBD sensors, and use unions of convex axis-aligned boxes to approximate the obstacles in the environment.

Similarly, we assume that the shapes of the robots are convex.

We assume that the workspace $\mathcal{W}_i \subseteq \mathbb{R}^d$ is a convex polytope. It can be set to a bounding box that defines a room that a ground robot must not leave, a half-space that contains vectors with positive $z$ coordinates so that a quadrotor does not hit the ground or simply be set to $\mathbb{R}^d$. A non-convex workspace $\tilde{\mathcal{W}_i}$ can be modeled by a convex workspace $\mathcal{W}_i$ such that $\tilde{\mathcal{W}_i} \subseteq \mathcal{W}_i$ and a static set of convex obstacles $\tilde{\mathcal{O}}$ that block portions of the convex workspace so that $\tilde{\mathcal{W}_i} = \mathcal{W}_i \backslash \left(\cup_{\mathcal{Q} \in \tilde{\mathcal{O}}} \mathcal{Q}\right)$.

To provide a guarantee of collision avoidance, we assume that robots can *perfectly* sense the relative positions of obstacles and robots as well as their shapes in the environment. The approach we propose does not require sensing of higher order state components (e.g., velocity, acceleration, etc.) or planned/estimated trajectories of objects, as the former is generally a noisy signal which cannot be expected to be sensed perfectly and the latter would require either communication or a potentially noisy trajectory estimation.

RLSS treats robots and obstacles differently. It enforces that each robot stays within a spatial cell that is disjoint from the cells of other robots until the next planning iteration to ensure robot-robot collision avoidance. To compute the spatial cell for each robot, RLSS uses positions and shapes of nearby robots, but not obstacles, in the environment. Therefore, robots must be able to distinguish other robots from obstacles. However, we do not require individual identification of robots.

We assume that the team is cooperative in that each robot runs the same algorithm using the same replanning period.

[*] For the purposes of our algorithm, concave obstacles can be described or approximated by a union of finite number of convex shapes provided that the union contains the original obstacle. Using our algorithm with approximations of concave obstacles results in trajectories that avoid the approximations.

Lastly, we assume that planning is synchronized between robots, meaning that each robot plans at the same time instant. The synchronization assumption is needed for ensuring robot-robot collision avoidance when planning succeeds[†].

## 4.5 Approach

Under the given assumptions, we solve the generic multi-robot trajectory planning problem approximately using decentralized receding horizon planning. Each robot plans a trajectory, executes it for a short period of time, and repeats this cycle until it reaches its goal. We call each plan-execute cycle an iteration.

We refer to the planning robot as the ego robot, and henceforth drop indices for the ego robot from our notation as the same algorithm is executed by each robot. Workspace $\mathcal{W}$ is the convex polytope in which the ego robot must remain. $\mathcal{R}$ is the collision shape function of the ego robot such that $\mathcal{R}(\mathbf{p}) \subset \mathbb{R}^d$ is the space occupied by the ego robot when it is placed at position $\mathbf{p}$. The ego robot is given a desired trajectory $\mathbf{d}(t) : [0, T] \to \mathbb{R}^d$ that it should follow. The dynamic limits of the robot are modeled using required derivative degree $c$ up to which the trajectory must be continuous, and maximum derivative magnitudes $\gamma^k$ for required degrees $k \in \{1, \ldots, K\}$.

At every iteration, the ego robot computes a piecewise trajectory $\mathbf{f}(t)$, which is dynamically feasible and safe up to the safety duration $s$, that it executes for the replanning period $\delta t \leq s$, and fully re-plans, i.e., runs the full planning pipeline, for the next iteration. *The only parameter that is shared by all robots is the replanning period $\delta t$.*

Without loss of generality, we assume that navigation begins at $t = 0$, and at the start of planning iteration $u$, the current timestamp is $\tilde{T} = u\delta t$.

[†] In physical deployments, asynchronous planning can cause collisions between robots when they are in close proximity to each other. To handle collisions stemming from asynchronous planning, robot shapes can be artificially inflated according to the maximum possible planning lag between robots (an empirical value) at the expense of conservativeness. Starting Chapter 5, we explicitly account for asynchronous planning in a principled way.

RLSS fits into the planning part of the classical robotics pipeline using perception, planning, and control. The inputs from perception for the ego robot are:

- $\mathcal{S}$: Shapes of other robots[‡]. $\mathcal{S}_j \in \mathcal{S}$ where $j \in \{1, \ldots, i-1, i+1, \ldots, N\}$ is the collision shape of robot $j$ sensed by the ego robot such that $\mathcal{S}_j \subseteq \mathbb{R}^d$.
- $\mathcal{O}$: The set of obstacles in the environment, where each obstacle $\mathcal{Q} \in \mathcal{O}$ is a subset of $\mathbb{R}^d$.
- $\mathbf{p}$: Current position of the ego robot, from which derivatives up to required degree of continuity can be computed[§].

We define $\hat{\mathcal{O}} = \cup_{\mathcal{Q} \in \mathcal{O}} \mathcal{Q}$ as the space occupied by the obstacles, and $\hat{\mathcal{S}} = \cup_{\mathcal{S}' \in \mathcal{S}} \mathcal{S}'$ as the space occupied by the robot shapes. Robots sense the set of obstacles and the set of robot shapes and use those sets in practice. We use spaces occupied by obstacles and robots for brevity in notation.

There are $4$ main stages of RLSS: 1) goal selection, 2) discrete planning, 3) trajectory optimization, and 4) temporal rescaling. The planning pipeline is summarized in Figure 4.2. At each planning iteration, the ego robot executes the four stages to plan the next trajectory. In the goal selection stage, a goal position and the corresponding timestamp of the goal position on the desired trajectory $\mathbf{d}(t)$ is selected. In the discrete planning stage, a discrete path from robot's current position toward the selected goal position is computed and durations are assigned to each segment. In the trajectory optimization stage, discrete segments are smoothed to a piecewise trajectory. In the temporal rescaling stage, the dynamic limits of the robot are checked and duration rescaling is applied if necessary.

Next, we describe each stage in detail. Each stage has access to the workspace $\mathcal{W}$, the collision shape function $\mathcal{R}$, the desired trajectory $\mathbf{d}(t)$ and its duration $T$, the maximum derivative magnitudes $\gamma^k$, and the derivative degree $c$ up to which the trajectory must be continuous. We call these task inputs. They

[‡] Practically, if the ego robot cannot sense a particular robot $j$ because it is not within the sensing range of the ego robot, robot $j$ can be omitted by the ego robot. As long as the sensing range of the ego robot is more than the maximum distance that can be travelled by the ego robot and robot $j$ in duration $\delta t$, omitting robot $j$ does not affect the behavior of the algorithm.

[§] In reality, all required derivatives of the position would be estimated using a state estimation method. We use this definition of $\mathbf{p}$ for notational convenience.

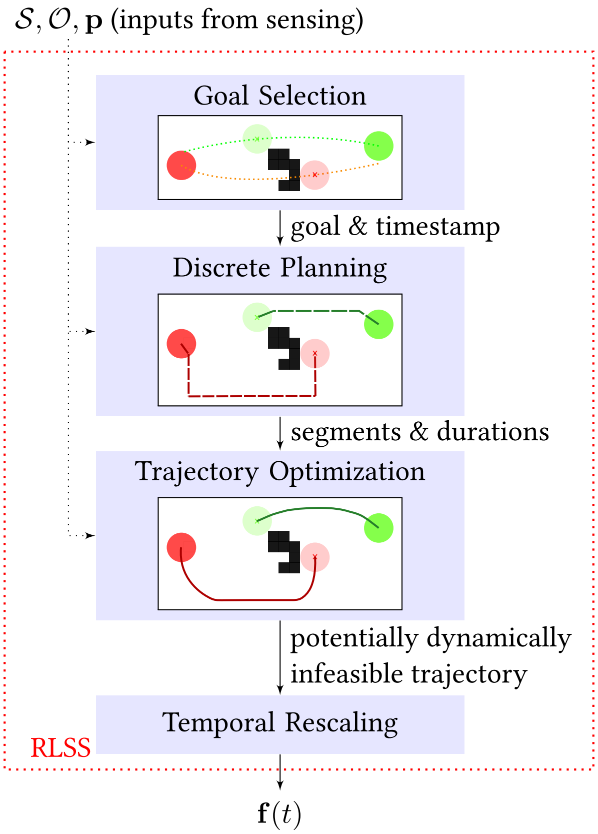
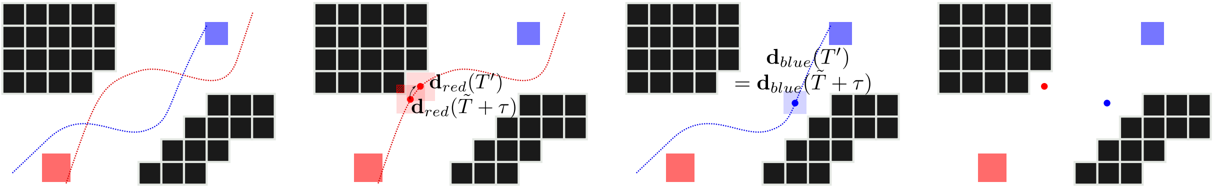


Figure 4.2: RLSS planning pipeline. Based on the sensed robots $\mathcal{S}$, sensed obstacles $\mathcal{O}$, and current position $\mathbf{p}$, the ego robot computes the trajectory $\mathbf{f}(t)$ that is dynamically feasible and safe up to time $s$.


$\mathbf{d}_{red}(T')$
$\mathbf{d}_{red}(\tilde{T}+\tau)$
$\mathbf{d}_{blue}(T')$
$= \mathbf{d}_{blue}(\tilde{T}+\tau)$

(a) Desired Trajectories (b) Red Goal Selection (c) Blue Goal Selection (d) Selected Goals

Figure 4.3: *Goal Selection.* a) Blue and red squares are robots, while obstacles are black boxes. The desired trajectories $\mathbf{d}_{red}(t)$ and $\mathbf{d}_{blue}(t)$ of each robot are given as dotted lines. Safety distance $D$ is set to $0$ for clarity. b) The desired trajectory of the red robot is not collision-free at timestamp $\tilde{T}+\tau$. It selects its goal timestamp $T'$ (and hence its goal position) by solving (4.9), which is the closest timestamp to $\tilde{T}+\tau$ when the robot, when placed on the desired trajectory, would be collision free. c) Since the desired trajectory of blue robot is collision free at timestamp $\tilde{T}+\tau$, it selects its goal timestamp $T' = \tilde{T}+\tau$ after solving (4.9). d) Selected goal positions are shown.

describe the robot shape, robot dynamics, and the task at hand; these are not parameters that can be tuned freely and they do not change during navigation.

### 4.5.1 Goal Selection

At the goal selection stage (Algorithm 1), we find a goal position $\mathbf{g}$ on the desired trajectory $\mathbf{d}(t)$ and a timestamp by which it should be reached. These are required in the subsequent discrete planning stage, which is a goal-oriented search algorithm.

Goal selection has two parameters: the desired planning horizon $\tau$ and safety distance $D$. It uses the robot collision shape function $\mathcal{R}$, desired trajectory $\mathbf{d}(t)$ and its duration $T$, and workspace $\mathcal{W}$ from the task inputs. The inputs of goal selection are the shapes of other robots $\mathcal{S}$, obstacles in the environment $\mathcal{O}$, current position $\mathbf{p}$, and the current timestamp $\tilde{T}$.

At the goal selection stage, the algorithm finds the timestamp $T'$ that is closest to $\tilde{T}+\tau$ (i.e., the timestamp that is one planning horizon away from the current timestamp) when the robot, if placed on the desired trajectory at $T'$, is at least safety distance $D$ away from all objects in the environment. We use the safety distance $D$ as a heuristic to choose goal positions that have free volume around them in order not to command robots into tight goal positions. Note that goal selection only chooses a single point

**Algorithm 1:** GOAL-SELECTION

```
Input      : $\mathcal{S}$ : Set of robot shapes
Input      : $\mathcal{O}$ : Set of obstacles
Input      : $\mathbf{p}$ : Current position
Input      : $\tilde{T}$ : Current timestamp
TaskInput: $\mathcal{R}$ : Collision shape function
TaskInput: $\mathbf{d}(t), T$ : Desired trajectory and its duration
TaskInput: $\mathcal{W}$ : Workspace
Parameter: $\tau$ : Desired planning horizon
Parameter: $D$ : Safety distance
Return     : Goal and timestamp by which it should be reached
$T' \leftarrow$ Solve (4.9) with linear search;
if (4.9) is infeasible then
    return ($\mathbf{p}$, $\tilde{T}$)
else
    $\mathbf{g} \leftarrow \mathbf{d}(T')$;
    return ($\mathbf{g}$, $T'$)
end
```

on the desired trajectory that satisfies the safety distance; the actual trajectory the robot follows will be planned by the rest of the algorithm. Formally, the problem we solve in the goal selection stage is given as follows:

$$\begin{aligned} T' = \arg\min_t |t - (\tilde{T} + \tau)| \ s.t. \\ t \in [0, T] \\ \text{min-dist}(\mathcal{R}(\mathbf{d}(t)), \hat{\mathcal{O}} \cup \hat{\mathcal{S}} \cup \partial\mathcal{W}) \geq D \end{aligned} \tag{4.9}$$

where $\partial\mathcal{W}$ is the boundary of workspace $\mathcal{W}$, and min-dist returns the minimum distance between two sets. We solve (4.9) using linear search on timestamps starting from $\tilde{T} + \tau$ with small increments and decrements. Figure 4.3, demonstrates the goal selection procedure for a particular instance.

If there is no safe point on the desired trajectory, i.e. if the robot is closer than $D$ to objects when it is placed on any point on the desired trajectory, we return the current position and timestamp. This allows us to plan a safe stopping trajectory.

Note that while the selected goal position has free volume around it, it may not be reachable by the ego robot. For example, the goal position may be encircled by obstacles or other robots. Therefore, we

```
Algorithm 2: DISCRETE-PLANNING
Input      : $\mathbf{g}$ : Goal position
Input      : $T'$ : The timestamp that goal position should be (or should have been) reached at
Input      : $\mathcal{S}$ : Set of robot shapes
Input      : $\mathcal{O}$ : Set of obstacles
Input      : $\mathbf{p}$ : Current position
Input      : $\tilde{T}$ : Current timestamp
TaskInput: $\mathcal{R}$ : Collision shape function
TaskInput: $\mathcal{W}$ : Workspace
TaskInput: $\gamma^1$ : Maximum velocity the robot can execute
Parameter: $\sigma$ : Step size of search grid
Parameter: $s$ : Duration up to which computed trajectory must be safe. $s \geq \delta t$ must hold.
Return     : Discrete path and duration assignments to segments
$\mathcal{F} \leftarrow \mathcal{W} \backslash (\hat{\mathcal{O}} \cup \hat{\mathcal{S}})$;
$actions \leftarrow$ BEST-EFFORT-$A^*(\mathbf{p}, \mathbf{g}, \mathcal{R}, \mathcal{F}, \sigma)$;
$\{\mathbf{e}_1, \ldots, \mathbf{e}_L\} \leftarrow$ EXTRACT-SEGMENTS$(actions)$
Prepend $\mathbf{e}_1$ to $\{\mathbf{e}_1, \ldots, \mathbf{e}_L\}$ so that $\mathbf{e}_0 = \mathbf{e}_1$.;
$totalLength \leftarrow \sum_{i=1}^{L} \|\mathbf{e}_i - \mathbf{e}_{i-1}\|_2$;
$fDuration \leftarrow \max\left(T' - \tilde{T}, \frac{totalLength}{\gamma^1}\right)$;
$T_1 \leftarrow s$
for $i = 2 \rightarrow L$ do
    $T_i \leftarrow fDuration \frac{\|\mathbf{e}_i - \mathbf{e}_{i-1}\|_2}{totalLength}$
end
return $\{\mathbf{e}_0, \ldots, \mathbf{e}_L\}, \{T_1, \ldots, T_L\}$
```

use a best-effort search method during discrete planning (as described in Section 4.5.2) that plans a path *towards* the goal position.

The goal and the timestamp are used in the subsequent discrete planning stage as suggestions.

### 4.5.2 Discrete Planning

Discrete planning (Algorithm 2) performs two main tasks: i) it finds a collision-free discrete path from the current position $\mathbf{p}$ towards the goal position $\mathbf{g}$, and ii) it assigns durations to each segment of the discrete path. The discrete path found at the discrete planning stage represents the homotopy class of the final trajectory. Trajectories in the same homotopy class can be smoothly deformed into one another without intersecting obstacles [9]. The subsequent trajectory optimization stage computes a smooth trajectory within the homotopy class. The trajectory optimization stage utilizes the discrete path to i) generate obstacle avoidance constraints and ii) guide the computed trajectory by adding distance cost terms between the discrete path and the computed trajectory. It uses the durations assigned by the discrete planning stage as the piece durations of the piecewise trajectory it computes.

Finding a discrete path from the start position $\mathbf{p}$ to the goal position $\mathbf{g}$ is done with best-effort $A^*$ search (Line 2, Algorithm 2), which we define as follows. If there are valid paths from $\mathbf{p}$ to $\mathbf{g}$, we find the least-cost path among such paths. If no such path exists, we choose the least-cost path to the position that has the lowest heuristic value (i.e., the position that is heuristically closest to $\mathbf{g}$). This modification of $A^*$ search is done due to the fact that the goal position may not always be reachable, since the goal selection stage does not enforce reachability.

The ego robot plans its path in a search grid where the grid has hypercubic cells (i.e. square in 2D, cube in 3D) with edge size $\sigma$, which we call the step size of the search. The grid shifts in the environment with the robot in the sense that the robot's current position always coincides with a grid center. Let $\mathcal{F} = \mathcal{W} \backslash (\hat{\mathcal{O}} \cup \hat{\mathcal{S}})$ be the free space within the workspace (Line 1, Algorithm 2). We do not map free space $\mathcal{F}$ to the grid. Instead, we check if the robot shape swept along any segment on the grid is contained in $\mathcal{F}$ or not, to decide if a movement is valid. This allows us to i) model obstacles and robot shapes independently from the grid's step size $\sigma$, and ii) shift the grid with no additional computation since we do not store occupancy information within the grid.

The states during the search have two components: position $\boldsymbol{\pi}$ and direction $\boldsymbol{\Delta}$. Robots are allowed to move perpendicular or diagonal to the grid. This translates to $8$ directions in 2-dimension, 26 directions in 3-dimension. Goal states are states that have position $\mathbf{g}$ and any direction. We model directions using vectors $\boldsymbol{\Delta}$ of $d$ components where each component is in $\{-1, 0, 1\}$. When the robot moves $1$ step along direction $\boldsymbol{\Delta}$, its position changes by $\sigma\boldsymbol{\Delta}$. The initial state of the search is the ego robot's current position $\mathbf{p}$ and direction $\mathbf{0}$.

Table 4.1: Discrete search actions and their costs.

| Action | Description | Cost |
|---|---|---|
| ROTATE | Change current direction to a new direction. | 1 |
| FORWARD | Move forward from the current position $\boldsymbol{\pi}$ along current direction $\boldsymbol{\Delta}$ by the step size $\sigma$. It is only available when $\boldsymbol{\Delta} \neq \mathbf{0}$. | $\lVert\boldsymbol{\Delta}\rVert_2$ |
| REACHGOAL | Connect the current position $\boldsymbol{\pi}$ to the goal position $\mathbf{g}$ where step size of the grid is $\sigma$. | $1 + \frac{\lVert\boldsymbol{\pi}-\mathbf{g}\rVert_2}{\sigma}$ |

There are 3 actions in the search formulation: ROTATE, FORWARD, and REACHGOAL, summarized in Table 4.1. ROTATE action has a cost of 1. The cost of the FORWARD action is the distance travelled divided by the step size (i.e. cells travelled), which is equal to the size $\lVert\boldsymbol{\Delta}\rVert_2$ of the direction vector. REACHGOAL has the cost of one rotation plus cells travelled from $\boldsymbol{\pi}$ to goal position $\mathbf{g}$: $1 + \frac{\lVert\boldsymbol{\pi}-\mathbf{g}\rVert_2}{\sigma}$. One rotation cost is there because it is almost surely required to do one rotation before going to goal from a cell. ROTATE actions in all directions are always valid whenever the current state is valid. FORWARD and REACHGOAL actions are valid whenever the robot shape $\mathcal{R}$ swept along the movement is contained in free space $\mathcal{F}$.

For any state $(\boldsymbol{\pi}, \boldsymbol{\Delta})$, we use the Euclidean distance from position $\boldsymbol{\pi}$ to the goal position $\mathbf{g}$ divided by step size $\sigma$ (i.e. cells travelled when $\boldsymbol{\pi}$ is connected to $\mathbf{g}$ with a straight line) as the admissible heuristic.

**Lemma 3.** *In the action sequence of the resulting plan*

1. *each ROTATE action must be followed by at least one FORWARD action,*

2. *the first action cannot be a FORWARD action,*

3. *and no action can appear after REACHGOAL action.*

*Proof Sketch.*

1. After each ROTATE action, a FORWARD action must be executed in a least-cost plan because i) a ROTATE action cannot be the last action since goal states accept any direction and removing any ROTATE action from the end would result in a valid lower cost plan, ii) the REACHGOAL action cannot appear

after ROTATE action because REACHGOAL internally assumes robot rotation and removing the ROTATE action would result in a valid lower cost plan, and iii) there cannot be consecutive ROTATE actions in a least-cost path as each rotation has the cost of 1 and removing consecutive rotations and replacing them with a single rotation would result in a valid lower cost plan.

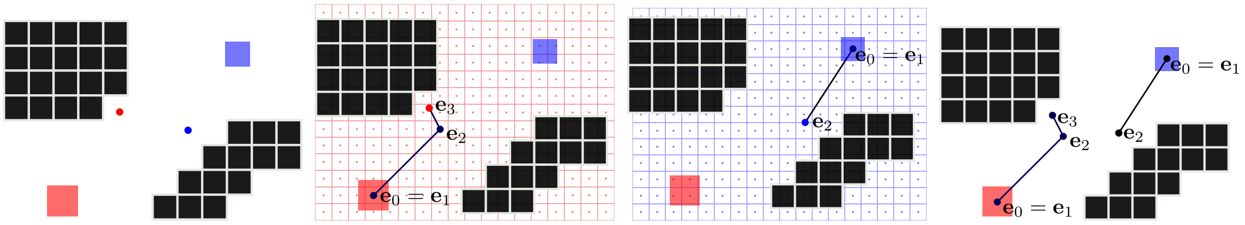
2. The first action cannot be a FORWARD action since initial direction is set to $\mathbf{0}$ and FORWARD action is available only when $\mathbf{\Delta} \neq \mathbf{0}$.

3. No action after a REACHGOAL action can appear in a least cost plan because REACHGOAL connects to the goal position, which is a goal state regardless of the direction. Removing any action after REACHGOAL action would result in a valid lower cost plan.

By Lemma 3, the action sequence can be described by the following regular expression in POSIX-Extended Regular Expression Syntax: ((ROTATE)(FORWARD)$^+$)$^*$(REACHGOAL)$^{\{0,1\}}$.

We collapse consecutive FORWARD actions in the resulting plan and extract discrete segments (Line 3, Algorithm 2). Each (ROTATE)(FORWARD)$^+$ sequence and REACHGOAL becomes a separate discrete segment. Let $\{\mathbf{e}_1, \ldots, \mathbf{e}_L\}$ be the endpoints of discrete segments. We prepend the first endpoint to the endpoint sequence in order to have a 0-length first segment for reasons that we will explain in Section 4.5.3 (Line  4, Algorithm 2). The resulting $L$ segments are described by $L + 1$ endpoints $\{\mathbf{e}_0, \ldots, \mathbf{e}_L\}$ where $\mathbf{e}_0 = \mathbf{e}_1$. Example discrete paths for two robots are shown in Figure 4.4.

Next, we assign durations to each segment. The total duration of the segments is computed using the ego robot's maximum velocity $\gamma^1$, the timestamp $T'$ that the goal position should be reached by, and the current timestamp $\tilde{T}$. We use $T' - \tilde{T}$ as the desired duration of the plan. However, if $T' - \tilde{T}$ is negative (i.e. $T' < \tilde{T}$, meaning that the goal position should been reached in the past), or $T' - \tilde{T}$ is small such that the robot cannot traverse the discrete segments even with its maximum velocity $\gamma^1$, we adjust the desired duration to a feasible one, specifically the total length of segments divided by the maximum velocity (Line 6, Algorithm 2). We distribute the feasible duration to the segments except for the first


$\mathbf{e}_3$
$\mathbf{e}_2$
$\mathbf{e}_0 = \mathbf{e}_1$
$\mathbf{e}_0 = \mathbf{e}_1$
$\mathbf{e}_2$
$\mathbf{e}_3$
$\mathbf{e}_2$
$\mathbf{e}_0 = \mathbf{e}_1$
$\mathbf{e}_0 = \mathbf{e}_1$
$\mathbf{e}_2$

(a) Goal Positions (b) Red Discrete Planning (c) Blue Discrete Planning (d) Discrete Paths

Figure 4.4: *Discrete Planning.* a) Goal positions of two robots computed at the goal selection stage are given. b) Red robot plans a discrete path from its current position to its goal position on a search grid that is aligned on its current position. c) Blue robot plans a discrete path from its current position to its goals position on a search grid that is aligned on its current position. Computed discrete paths are prepended with robot start positions to have 0-length first segments. d) The resulting discrete paths are given.

one, proportional to their lengths (Loop at line 8, Algorithm 2). We set the duration of the first segment, which has zero length, to the safety duration $s$ (Line 7, Algorithm 2); the reason for this will be explained in Section 4.5.3.

The outputs of discrete planning are segments described by endpoints $\{\mathbf{e}_0, \ldots, \mathbf{e}_L\}$ with assigned durations $\{T_1, \ldots, T_L\}$ that are used in the trajectory optimization stage.

### 4.5.3 Trajectory Optimization

In the trajectory optimization stage (Algorithm 3), we formulate a quadratic optimization problem (QP) to compute a piecewise trajectory $\mathbf{f}(t)$ by smoothing discrete segments. The computed trajectory is collision-free and continuous up to the desired degree of derivative. However, it may be dynamically infeasible (i.e., derivative magnitudes may exceed the maximum allowed derivative magnitudes $\gamma^k$); this is resolved during the subsequent temporal rescaling stage.

The decision variables of the optimization problem are the control points of an $L$-piece spline where each piece is a Bèzier curve. The duration of each piece is assigned during discrete planning. Let $T = \sum_{i=1}^{L} T_i$ denote the total duration of the planned trajectory. The degree of the Bézier curves is tuned with the parameter $h$. Let $\mathbf{P}_{i,j} \in \mathbb{R}^d$ be the $j^{th}$ control point of the $i^{th}$ piece where $i \in \{1, \ldots, L\}, j \in \{0, \ldots, h\}$. Let $\mathcal{P} = \{\mathbf{P}_{i,j} \mid i \in \{1, \ldots, L\}, j \in \{0, \ldots, h\}\}$ be the set of all control points.

```
Algorithm 3: TRAJECTORY-OPTIMIZATION
  Input      : $\{\mathbf{e}_0, \dots, \mathbf{e}_L\}$ : Endpoints of discrete segments
  Input      : $\{T_1, \dots, T_L\}$ : Durations of discrete segments
  Input      : $\mathcal{S}$ : Set of robot shapes
  Input      : $\mathcal{O}$ : Set of obstacles
  Input      : $\mathbf{p}$ : Current position
  TaskInput: $\mathcal{R}$ : Collision shape function
  TaskInput: $\mathcal{W}$ : Workspace
  TaskInput: $c$ : Degree of derivative up to which resulting trajectory must be continuous
  Parameter: $h$ : Degree of Bézier curves
  Parameter: $\tilde{o}$ : Obstacle check distance
  Parameter: $\tilde{r}$ : Robot check distance
  Parameter: $\tilde{p}$ : Preferred distance to objects
  Parameter: $\alpha$ : Preferred distance cost weight
  Parameter: $\boldsymbol{\theta}$ : Endpoint cost weights
  Parameter: $\boldsymbol{\lambda}$ : Integrated derivative cost weights
  Return     : Potentially dynamically infeasible trajectory
  $QP$ is a quadratic program with variables $\mathcal{P}$
  $\Upsilon_{\mathcal{W}} \leftarrow$ BUFFER-WORKSPACE($\mathcal{W}, \mathcal{R}$);
  addWorkspaceConstraints($QP$, $\Upsilon_{\mathcal{W}}$)
  $\Upsilon_i \leftarrow \emptyset \ \forall i \in \{1, \dots, L\}$
  for $\forall \mathcal{S}_j \in \mathcal{S}$ min-dist($\mathcal{S}_j, \mathcal{R}(\mathbf{p})) \leq \tilde{r}$ do
      $\Upsilon_1 \leftarrow \Upsilon_1 \cup \{$BUFFERED-SVM($\mathcal{S}_j, \mathcal{R}(\mathbf{p}), \mathcal{R}$)$\}$
  end
  for $i = 1 \rightarrow L$ do
      $\zeta_i \leftarrow$ region swept by $\mathcal{R}$ from $\mathbf{e}_{i-1}$ to $\mathbf{e}_i$
      for $\forall \mathcal{Q} \in \mathcal{O}$ min-dist($\mathcal{Q}, \zeta_i) \leq \tilde{o}$ do
          $\Upsilon_i \leftarrow \Upsilon_i \cup \{$BUFFERED-SVM($\mathcal{Q}, \zeta_i, \mathcal{R}$)$\}$
      end
  end
  addCollAvoidanceConstraints($QP$, $\Upsilon_1, \dots, \Upsilon_L$)
  addContinuityConstraints($QP$, $c$, $\mathbf{p}$, $T_1, \dots, T_L$)
  addEnergyCostTerm($QP$, $\boldsymbol{\lambda}$, $T_1, \dots, T_L$)
  addDeviationCostTerm($QP$, $\boldsymbol{\theta}$, $\mathbf{e}_1, \dots, \mathbf{e}_L$)
  $\overline{\Upsilon}_1 \leftarrow$ SHIFT-HYPERPLANES($\Upsilon_1, \tilde{p}$)
  addPreferredDistanceCostTerm($QP$, $\overline{\Upsilon}_1, \alpha$)
  $\mathbf{f}(t) \leftarrow$ SOLVE-QP($QP$)
  return $\mathbf{f}(t)$
```

#### 4.5.3.1 Constraints

There are $4$ types of constraints on the trajectory; all are linear in the decision variables $\mathcal{P}$.

*1) Workspace constraints:* We shift each bounding hyperplane of workspace $\mathcal{W}$ (there are a finite number of such hyperplanes as $\mathcal{W}$ is a convex polytope) to account for the robot's collision shape $\mathcal{R}$, such that when the robot's position is on the safe side of the shifted hyperplane, the entire robot is on the safe side of the original hyperplane (Line 2, Algorithm 3).

Let $\Upsilon_{\mathcal{W}}$ be the set of shifted hyperplanes of $\mathcal{W}$. We constrain each control point of the trajectory to be on the valid sides of the shifted hyperplanes (Line 3, Algorithm 3). Since Bézier curves are contained in the convex hulls of their control points, constraining control points to be in a convex set automatically constrains the Bézier curves to stay within the same convex set.

*2) Robot-robot collision avoidance constraints:* For robot-robot collision avoidance, recall that each robot replans with the same period $\delta t$ and planning is synchronized between robots.

At each iteration, the ego robot computes its SVM cell within the SVM tessellation of robots using hard-margin SVMs. SVM tessellation is similar to Voronoi tesselation, the only difference is that pairwise SVM hyperplanes are computed between collision shapes instead of Voronoi hyperplanes. We choose SVM tessellation because i) hard-margin SVM is convex, hence pairs of robots can compute the same exact hyperplane under the assumption of perfect sensing, ii) it allows for a richer set of collision shapes than basic Voronoi cells, which is valid only for hyperspherical objects, and iii) SVM cells are always convex unlike generalized Voronoi cells.

We buffer the ego robot's SVM cell to account for its collision shape $\mathcal{R}$, and constrain the first piece of the trajectory to stay inside the buffered SVM cell (BSVM) (loop at Line 5, Algorithm 3). Only the first piece of the trajectory is constrained to remain in the buffered SVM cell, since the entire planning pipeline is run after $\delta t$, which is smaller than the duration $s$ of the first piece. At that time, planning begins at a new location, generating a new first piece that must remain in the new buffered SVM cell.

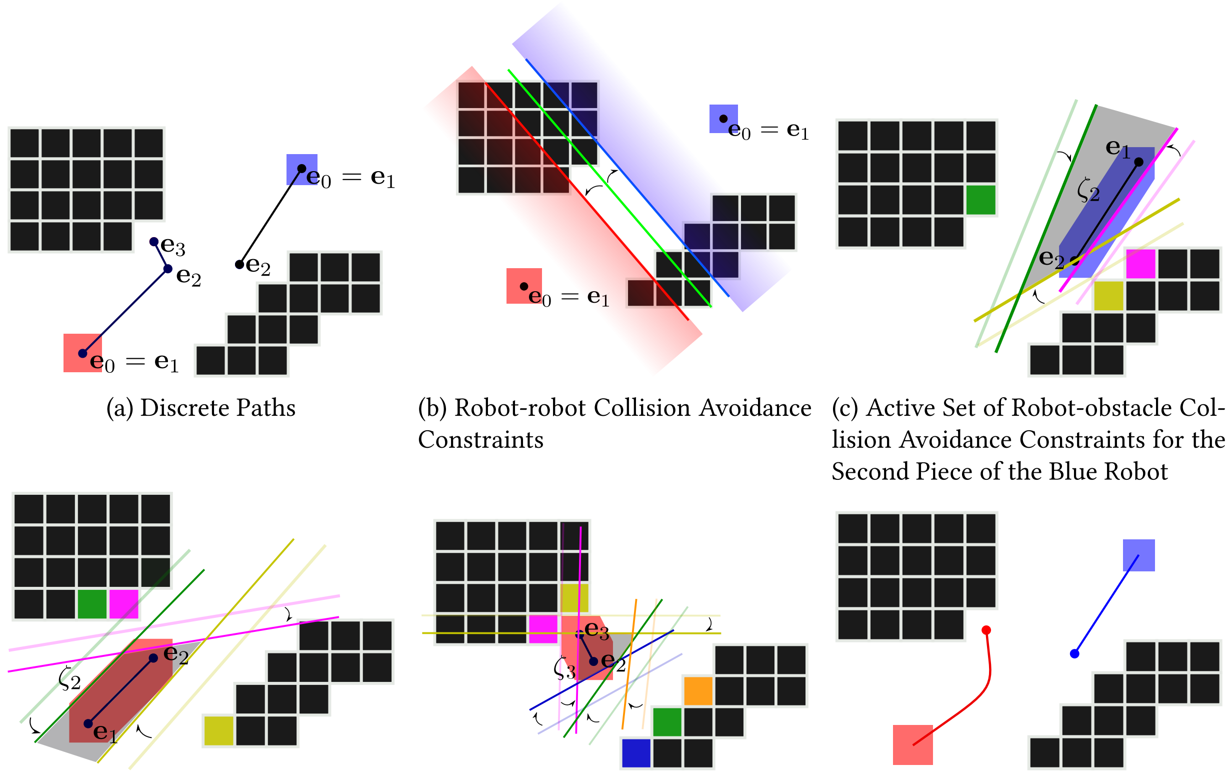
Buffering is achieved by changing the offset of the hyperplane. $\mathcal{R}(\mathbf{x})$ is the shape of the robot when placed at $\mathbf{x}$, defined as $\mathcal{R}(\mathbf{x}) = \{\mathbf{x}\} \oplus \mathcal{R}_0$, where $\mathcal{R}_0$ is the shape of the robot when placed the origin and $\oplus$ is the Minkowski sum operator. Given a hyperplane with normal $\mathcal{H}_{\mathbf{n}}$ and offset $\mathcal{H}_a$, we find $\mathcal{H}_{a'}$ that ensures $\mathcal{R}(\mathbf{x})$ is on the negative side of the hyperplane with normal $\mathcal{H}_{\mathbf{n}}$ and offset $\mathcal{H}_a$ whenever $\mathbf{x}$ is on the negative side of the hyperplane with normal $\mathcal{H}_{\mathbf{n}}$ and offset $\mathcal{H}_{a'}$, and vice versa, by setting $\mathcal{H}_{a'} = \mathcal{H}_a + \max_{\mathbf{y}\in\mathcal{R}_0} \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y}$. The following shows that whenever $\mathcal{R}(\mathbf{x})$ is on the negative side of the hyperplane $(\mathcal{H}_{\mathbf{n}}, \mathcal{H}_a)$, $\mathbf{x}$ is on the negative side of the hyperplane $(\mathcal{H}_{\mathbf{n}}, \mathcal{H}_{a'})$. The converse can be shown by following the steps backwards.

$$
\begin{aligned}
&\forall \mathbf{y} \in \mathcal{R}(\mathbf{x})\ \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y} + \mathcal{H}_a \leq 0 \\
&\implies \max_{\mathbf{y}\in\mathcal{R}(\mathbf{x})} \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y} + \mathcal{H}_a \leq 0 \\
&\implies \max_{\mathbf{y}\in\{\mathbf{x}\}\oplus\mathcal{R}_0} \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y} + \mathcal{H}_a \leq 0 \\
&\implies \max_{\mathbf{y}\in\mathcal{R}_0} \mathcal{H}_{\mathbf{n}} \cdot (\mathbf{y} + \mathbf{x}) + \mathcal{H}_a \leq 0 \\
&\implies \mathcal{H}_{\mathbf{n}} \cdot \mathbf{x} + \mathcal{H}_a + \max_{\mathbf{y}\in\mathcal{R}_0} \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y} \leq 0
\end{aligned}
$$

Since the duration of the first piece was set to the safety duration $s \geq \delta t$ in discrete planning, the robot stays within its buffered SVM cell for at least $\delta t$. Moreover, since planning is synchronized across all robots, the pairwise SVM hyperplanes they compute will match, thus the buffered SVM cells of robots are disjoint. This ensures robot-robot collision avoidance until the next planning iteration.

Computed SVMs and BSVMs are shown in Figure 4.5b for a two robot case.

To ensure that the number of constraints of the optimization problem does not grow indefinitely, SVM hyperplanes are only computed against those robots that are at most $\tilde{r}$ away from the ego robot. This does not result in unsafe trajectories so long as $\tilde{r}$ is more than the total maximum distance that can be traversed by two robots while following the first pieces of their trajectories, for which an upper bound is $\max_{i,j\in\{1,\ldots,N\}}(\gamma_i^1 s_i + \gamma_j^1 s_j)$, where $s_i$ and $s_j$ are the durations of the first pieces of the trajectories of robots $i$ and $j$ respectively.



(a) Discrete Paths

(b) Robot-robot Collision Avoidance Constraints

(c) Active Set of Robot-obstacle Collision Avoidance Constraints for the Second Piece of the Blue Robot

(d) Active Set of Robot-obstacle Collision Avoidance Constraints for the Second Piece of the Red Robot

(e) Active Set of Robot-obstacle Collision Avoidance Constraints for the Third Piece of the Red Robot

(f) Computed Trajectories

Figure 4.5: *Trajectory Optimization.* a) Discrete segments of two robots computed at the discrete planning stage. b) Robot-robot collision avoidance constraints are computed using BSVMs. The green hyperplane is the SVM hyperplane between two robots. Each robot shifts the SVM hyperplane to account for its geometry, and constrains the first piece of the trajectory with the resulting BSVM. c-d-e) Active set of robot-obstacle collision avoidance constraints for three different pieces (one belonging to the blue robot, two belonging to the red robot). In each figure, the region swept by the robot while traversing the segment is shown in robot's color. SVM hyperplanes between the swept region and the obstacles are given as light-colored lines. SVM hyperplanes are buffered to account for the robot's collision shape and shown as dark-colored lines (BSVMs). The shift operations are shown as arrows. Obstacles and constraints generated from them are colored using the same colors. For each piece, the feasible region that is induced by the robot-obstacle collision avoidance constraints is colored in gray. f) Trajectories computed by the trajectory optimization stage are given.

*3) Robot-obstacle collision avoidance constraints*: Buffered SVM hyperplanes are used for robot-obstacle collision avoidance as well. Let $\zeta_i \subset \mathbb{R}^d$ be the region swept by the ego robot while traversing the $i^{th}$ segment from $\mathbf{e}_{i-1}$ to $\mathbf{e}_i$. We compute the SVM hyperplane between $\zeta_i$ and each object in $\mathcal{O}$, buffer it as explained before to account for robot's collision shape, and constrain the $i^{th}$ piece by the resulting buffered SVM (Loop at line 8, Algorithm 3). This ensures that trajectory pieces do not cause collisions with obstacles.

The use of SVMs for collision avoidance against obstacles is a choice of convenience, as we already use them for robot-robot collision avoidance. For robot-obstacle collision avoidance, one can use any separating hyperplane between $\zeta_i$ and objects in $\mathcal{O}$, while in the case of robot-robot collision avoidance, pairs of robots must compute matching hyperplanes using the same algorithm.

Elements of the active set of SVM and BSVM hyperplanes for robot-obstacle collision avoidance are shown in an example scenario in Figures 4.5c to 4.5e for three different pieces.

Similar to robot-robot collision avoidance, to ensure that the number of constraints of the optimization problem does not grow indefinitely, we only compute SVM hyperplanes between $\zeta_i$ and obstacles that are not more than $\tilde{o}$ away from $\zeta_i$.

Let $\Upsilon_i$ be the set of buffered SVM hyperplanes that constrain the $i^{th}$ piece $\forall i \in \{1, \ldots, L\}$. $\Upsilon_1$ contains both robot-robot and robot-obstacle collision avoidance hyperplanes while $\Upsilon_j$ $\forall j \in \{2, \ldots, L\}$ contain only robot-obstacle collision avoidance hyperplanes. This is because the first piece is the only piece that we constrain with robot-robot collision avoidance hyperplanes because it is the only piece that will be executed until the next planning iteration.

*4) Continuity constraints*: We add two types of continuity constraints: i) continuity constraints between planning iterations, and ii) continuity constraints between trajectory pieces (Line 15, Algorithm 3).

To enforce continuity between planning iterations, we add constraints that enforce

$$\frac{d^j \mathbf{f}(0)}{dt^j} = \frac{d^j \mathbf{p}}{dt^j} \; \forall j \in \{0, \ldots, c\}$$

where $c$ is the task input denoting the continuity degree up to which the resulting trajectory must be continuous and $\mathbf{p}$ is the current position.

To enforce continuity between trajectory pieces, we add constraints that enforce

$$\frac{d^j \mathbf{f}_i(T_i)}{dt^j} = \frac{d^j \mathbf{f}_{i+1}(0)}{dt^j}$$
$$\forall i \in \{1, \ldots, L-1\} \; \forall j \in \{0, \ldots, c\}$$

where $\mathbf{f}_i(t)$ is the $i^{th}$ piece of the trajectory.

Remark 1 discusses that the SVM problems generated during trajectory optimization are feasible, i.e., the trajectory optimization stage will always succeed constructing the QP.

**Remark 1.** *All SVM problems generated for robot-robot and robot-obstacle collision avoidance from the discrete path outputted by the discrete planning stage are feasible.*

*Reasoning.* The discrete planning stage outputs a discrete path such that a robot with collision shape $\mathcal{R}$ following the path does not collide with any obstacles in $\mathcal{O}$ or any other robots in $\mathcal{S}$. It also ensures that $\mathbf{p} = \mathbf{e}_0$ since the search starts from the robot's current position $\mathbf{p}$. Hence, $\mathcal{R}(\mathbf{p})$ does not intersect with any $\mathcal{S}_j \in \mathcal{S}$. Since each robot is assumed to be convex, there exists at least one hyperplane that separates $\mathcal{R}(\mathbf{p})$ from $\mathcal{S}_j$ for each $j$ by the separating hyperplane theorem. SVM is an optimal separating hyperplane according to a cost function. Therefore, SVM problems between $\mathcal{R}(\mathbf{p})$ and $\mathcal{S}_j \in \mathcal{S}$ for robot-robot collision avoidance are feasible.

Since the robot moving along the discrete segments is collision free, $\zeta_i$ is collision free for all $i$. Also, since it is a sweep of a convex shape along a line segment, $\zeta_i$ is convex as shown in Lemma 5 in Appendix A. Similar to the previous argument, since $\zeta_i$ is collision-free and convex and all obstacles in the environment are convex, each SVM problem between $\zeta_i$ and $\mathcal{Q} \in \mathcal{O}$ for robot-obstacle collision avoidance is feasible by the separating hyperplane theorem.

Workspace, robot-robot collision avoidance, robot-obstacle collision avoidance, and position continuity constraints are kinematic constraints. Higher-order continuity constraints are dynamic constraints. Remark 2 discusses the ensured feasibility of the kinematic constraints. The feasibility of the dynamic constraints cannot be ensured for arbitrary degrees of continuity.

**Remark 2.** *Kinematic constraints of the optimization problem generated from a discrete path output from the discrete planning stage are feasible for the same path when the degree of Bézier curves $h \geq 1$.*

*Reasoning.* Any Bézier curve with degree $h \geq 1$ can represent a discrete segment by setting half of the points to the start of the segment and other half to the end of the segment. Hence, we will only show that a discrete path output from discrete planning stage by itself satisfies the kinematic constraints generated. Remember that discrete path output from the discrete planning stage has the property $\mathbf{p} = \mathbf{e}_0 = \mathbf{e}_1$.

Each robot-robot SVM problem is feasible (see Remark 1). Let $\mathcal{H}_\mathbf{n}$ be the normal and $\mathcal{H}_a$ be the offset of any of the robot-robot SVMs. It is shifted by setting the offset to $\mathcal{H}_{a'} = \mathcal{H}_a + \max_{\mathbf{y} \in \mathcal{R}_0} \mathcal{H}_\mathbf{n} \cdot \mathbf{y}$. The point $\mathbf{p}$ satisfies $\mathcal{H}_\mathbf{n} \cdot \mathbf{p} + \mathcal{H}_{a'} \leq 0$ because $\mathcal{R}(\mathbf{p})$ is on the negative side of the SVM. The robot-robot BSVM hyperplanes are used to constrain the first piece of the trajectory, and setting the first piece as a $0$-length segment with $\mathbf{p} = \mathbf{e}_0 = \mathbf{e}_1$ satisfies the robot-robot collision avoidance constraints.

Each robot-obstacle SVM problem is feasible (see Remark 1). Let $\mathcal{H}_\mathbf{n}$ be the normal and $\mathcal{H}_a$ be the offset of any of the robot-obstacle SVMs that is between $\zeta_i$ and an obstacle. It is shifted by setting the offset to $\mathcal{H}_{a'} = \mathcal{H}_a + \max_{\mathbf{y} \in \mathcal{R}_0} \mathcal{H}_\mathbf{n} \cdot \mathbf{y}$. All points on the line segment $\mathbf{p}_i(t) = \mathbf{e}_{i-1} + t(\mathbf{e}_i - \mathbf{e}_{i-1}), t \in [0, 1]$ from

$\mathbf{e}_{i-1}$ to $\mathbf{e}_i$ satisfy the BSVM constraint because $\mathcal{R}(\mathbf{p}_i(t))$ is on the negative side of the SVM hyperplane $\forall t \in [0, 1]$. Since each SVM hyperplane between $\zeta_i$ and obstacles is only used to constrain the $i^{th}$ piece, constraints generated by it are feasible for the segment from $\mathbf{e}_{i-1}$ to $\mathbf{e}_i$.

The feasibility of the workspace constraints are trivial since the robot moving along discrete segments is contained in the workspace, and we shift bounding hyperplanes of the workspace in the same way as SVM hyperplanes. Hence, the discrete path satisfies the workspace constraints.

Initial point position continuity of the robot is satisfied by the given discrete segments since $\mathbf{p} = \mathbf{e}_0$. Position continuity between segments are trivially satisfied by the given discrete segments, since the discrete path is position continuous by its definition.

#### 4.5.3.2 Cost Function

The cost function of the optimization problem has 3 terms: i) energy usage, ii) deviation from the discrete path, and iii) preferred distance to objects.

We use the sum of integrated squared derivative magnitudes as a metric for energy usage (Line 16, Algorithm 3), similar to the works of Richter, Bry, and Roy [94] and Hönig et al. [39]. Parameters $\boldsymbol{\lambda} = \{\lambda_j\}$ are the weights for integrated squared derivative magnitudes, where $\lambda_j$ is the weight for $j^{th}$ degree of derivative. The energy term $\mathcal{J}_{energy}(\mathcal{P})$ is given as

$$\mathcal{J}_{energy}(\mathcal{P}) = \sum_{\lambda_j \in \boldsymbol{\lambda}} \lambda_j \int_0^T \left\| \frac{d^j \mathbf{f}(t)}{dt^j} \right\|_2^2 dt.$$

We use squared Euclidean distances between trajectory piece endpoints and discrete segment endpoints as a metric for deviation from the discrete path (Line 17, Algorithm 3). Remember that each Bézier curve piece $i$ ends at its last control point $\mathbf{P}_{i,h}$. Parameters $\boldsymbol{\theta} = \{\theta_i\} \in \mathbb{R}^L$ are the weights for each segment endpoint. The deviation term $\mathcal{J}_{dev}(\mathcal{P})$ is given as

$$\mathcal{J}_{dev}(\mathcal{P}) = \sum_{i \in \{1,\ldots,L\}} \theta_i \left\| \mathbf{P}_{i,h} - \mathbf{e}_i \right\|_2^2 .$$

The last term of the cost function models the preferred distance to objects. We use this term to discourage robots from getting closer to other objects in the environment; this increases the numerical stability of the algorithm by driving robots away from tight states.

We shift each hyperplane in $\Upsilon_1$ (i.e., buffered SVM hyperplanes constraining the first piece) by the preferred distance to objects, $\tilde{p}$, to create the preferred hyperplanes $\overline{\Upsilon}_1$ (Line 18, Algorithm 3). Since each robot replans with the period $\delta t$, we add a cost term that drives the robot closer to the preferred hyperplanes at the replanning period (Line 19, Algorithm 3). We take the sum of squared distances between $\mathbf{f}(\delta t)$ and hyperplanes in $\overline{\Upsilon}_1$:

$$\mathcal{J}_{pref}(\mathcal{P}) = \alpha \sum_{\mathcal{H} \in \overline{\Upsilon}_1} \left( \mathcal{H}_{\mathbf{n}}^\top \mathbf{f}(\delta t) + \mathcal{H}_a \right)^2$$

where $\mathcal{H}_{\mathbf{n}}$ is the normal and $\mathcal{H}_a$ is the offset of hyperplane $\mathcal{H}$, and $\alpha$ is the weight of $\mathcal{J}_{pref}$ term. Notice that this term is defined over the control points of first piece since $\delta t \leq s = T_1$, supporting the utilization of $\Upsilon_1$ only.

The overall trajectory optimization problem is:

$$\begin{aligned}
&\min_{\mathcal{P}} \mathcal{J}_{energy}(\mathcal{P}) + \mathcal{J}_{dev}(\mathcal{P}) + \mathcal{J}_{pref}(\mathcal{P}) \text{ s.t.} \\
&\mathcal{H}_{\mathbf{n}}^{\top}\mathbf{P}_{i,j} + \mathcal{H}_a \leq 0 \qquad \forall i \in \{1, \ldots, L\} \\
&\qquad\qquad\qquad\qquad\quad \forall j \in \{0, \ldots, h\} \\
&\qquad\qquad\qquad\qquad\quad \forall \mathcal{H} \in \Upsilon_i \cup \Upsilon_{\mathcal{W}} \\
&\frac{d^j \mathbf{f}(0)}{dt^j} = \frac{d^j \mathbf{p}}{dt^j} \qquad \forall j \in \{0, \ldots, c\} \\
&\frac{d^j \mathbf{f}_i(T_i)}{dt^j} = \frac{d^j \mathbf{f}_{i+1}(0)}{dt^j} \ \forall i \in \{1, \ldots, L-1\} \\
&\qquad\qquad\qquad\qquad\quad \forall j \in \{0, \ldots, c\}.
\end{aligned}$$

Notice that we formulate continuity and the safety of the trajectory using hard constraints. This ensures that the resulting trajectory is kinematically safe and continuous up to degree $c$ if the optimization succeeds for all robots and planning is synchronized.

### 4.5.4 Temporal Rescaling

At the temporal rescaling stage, we check whether the dynamic limits of the robot are violated. If the resulting trajectory is valid, i.e. derivative magnitudes of the trajectory are less than or equal to the maximum derivative magnitudes $\gamma^k \ \forall k \in \{1, \ldots, K\}$, the trajectory is returned as the output of RLSS. If not, temporal rescaling is applied to the trajectory similar to Hönig et al. [39] and Park et al. [78] by increasing the durations of the pieces so that the trajectory is valid. We scale the durations of pieces by multiplying them with a constant parameter greater than $1$ and re-run the optimization until the dynamic limits are obeyed or a predefined number of rescalings, which is a parameter, are applied. If the dynamic limits are not obeyed after a predefined number of rescalings, the planning iteration fails.

We choose to enforce dynamic feasibility outside of the trajectory optimization problem as a post processing step because i) the output of the trajectory optimization stage is often dynamically feasible, hence rescaling is rarely needed, and ii) adding piece durations as variables to the optimization problem would make it a non quadratic program, potentially decreasing performance.

## 4.6 Evaluation

Here, using synchronized simulations, we first evaluate our algorithm's performance when different parameters are used in Section 4.6.1. Second, we conduct an ablation study to show the effects of two important steps, namely the prepend operation of the discrete planning stage and the preferred distance cost term of the trajectory optimization stage, to the performance of the algorithm in Section 4.6.2. Third, we compare our algorithm to two state-of-the-art baseline planners in Section 4.6.3. Finally, we show our algorithm's applicability to real robots by using it on quadrotors and differential drive robots in Section 4.6.4.

We conduct our simulations on a laptop computer with Intel i7-8565U @ 1.80 GHz running Ubuntu 20.04. We implement our algorithm for a single core because of implementation simplicity and fairness to the baseline planners, which are not parallelized. The memory usage of each simulation is 30 MB on average for our algorithm.

In synchronized simulations, we compute trajectories for each robot using the same snapshot of the environment, move robots perfectly according to computed trajectories for the re-planning period, and replan. The effects of planning iterations taking longer than the re-planning period are not modeled in the synchronized simulations. We show that all algorithms (RLSS and the baselines) can work in $1\,\text{Hz} - 10\,\text{Hz}$ on the hardware we use, and assert that more powerful computers can be used to shorten planning times. In addition, parallelization of the A* search on GPUs is possible. For example, Zhou and Zeng [137] show the possibility of 6 − 7 x speedup in A* search for pathfinding problems on GPUs. Moreover, parallelization of quadratic program solving is possible through i) running multiple competing solvers in multiple cores and returning the answer from the first one that solves the problem, or ii) parallelizing individual solvers. For instance, IBM ILOG CPLEX Optimizer[¶] and Gurobi Optimizer[‖] support running multiple competing solvers concurrently. IBM ILOG CPLEX Optimizer supports a parallel barrier optimizer[**], which parallelizes a barrier algorithm for solving QPs. Gondzio and Grothey [36] propose a parallel interior point method solver that exploits nested block structure of problems. Performance improvements of these approaches are problem dependent, and we do not investigate how much the performance of our trajectory optimization stage could be improved with such methods.

In all experiments, 32 robots are placed in a circle formation of radius $20\,\mathrm{m}$ in 3D, and the task is to swap all robots to the antipodal points on the circle. The workspace $\mathcal{W}$ is set to an axis aligned bounding box from $\begin{bmatrix}-25\,\mathrm{m} & -25\,\mathrm{m} & 0\,\mathrm{m}\end{bmatrix}^\top$ to $\begin{bmatrix}25\,\mathrm{m} & 25\,\mathrm{m} & 5\,\mathrm{m}\end{bmatrix}^\top$. Robot collision shapes are modeled as axis aligned cubes with $0.2\,\mathrm{m}$ edge lengths. The desired planning horizon $\tau$ is set to $5\,\mathrm{s}$. The safety distance $D$ of goal selection is set to $0.2\,\mathrm{m}$. Robots have velocity limit $\gamma^1 = 3.67\,\frac{\mathrm{m}}{\mathrm{s}}$, and accelaration limit $\gamma^2 = 4.88\,\frac{\mathrm{m}}{\mathrm{s}^2}$, which are chosen arbitrarily. The safety duration $s$ is set to $0.11\,\mathrm{s}$ and re-planning period $\delta t$ is set to $0.1\,\mathrm{s}$. We set integrated derivative cost weights $\lambda_1 = 2.0$ for velocity and $\lambda_2 = 2.8$ for acceleration. We set endpoint cost weights to $\theta_1 = 0$, $\theta_2 = 150$, $\theta_3 = 240$, $\theta_i = 300\ \forall i \geq 4$. Setting $\theta_1 = 0$ allows the optimization to stretch the first 0-length segment freely, and setting other $\theta$s incrementally increases the importance of tail segments. We set the preferred distance to objects to $\tilde{p} = 0.6\,\mathrm{m}$ and the preferred distance cost weight $\alpha = 0.3$. We use the octree data structure from the octomap library [41] to represent the environment. Each leaf-level occupied axis aligned box of the octree is used as a separate obstacle in all algorithms. Octrees allow fast axis aligned box queries which return obstacles that intersects with a given axis aligned box, an operation we use extensively in our implementation. For example, we use axis aligned box queries in discrete planning as broadphase collision checkers to find the obstacles that are close to the volume swept by the ego robot while traversing a given segment, and check collisions between the robot

[¶] https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizers-concurrent-optimizer-in-parallel

[‖] https://www.gurobi.com/documentation/9.5/refman/concurrent_environments.html

[**] https://www.ibm.com/docs/en/icos/22.1.0?topic=optimizers-using-parallel-barrier-optimizer

traversing the segment and only the obstacles returned from the query. Also, to generate robot-obstacle collision avoidance constraints, we execute axis aligned box queries around the segments to retrieve nearby obstacles. We generate BSVM constraints only against nearby obstacles that are no more than $\tilde{o}$ away from the robot traversing the segment. Other popular approaches for environment representations include i) Euclidean signed distance fields (ESDFs) [74], which support fast distance queries to nearest obstacles, ii) 3D circular buffers [118], which aim to limit memory usage of maps and supports fast occupancy checks, and iii) Gaussian mixture models [72], which continuously represent occupancy instead of discretizing the environment as the former approaches do. None of these representations are as suitable as octrees for RLSS since they do not allow fast querying of obstacles in the vicinity of segments. We use the IBM ILOG CPLEX Optimizer[††] to solve our optimization problems. In all experiments, robots that are closer than $0.25\,\mathrm{m}$ to their goal positions are considered as goal reaching robots. If a robot that has not reached its goal does not change its position more than $1\,\mathrm{cm}$ in the last $1\,\mathrm{s}$ of the simulation, it is considered as a deadlocked robot. At any point of the simulation, if each robot is either at the goal or deadlocked, we conclude the simulation.

### 4.6.1 Effects of Selected Parameters

We evaluate the performance of our algorithm when $4$ important parameters are changed: step size $\sigma$ of the search grid, degree $h$ of Bézier curves, obstacle check distance $\tilde{o}$, and robot check distance $\tilde{r}$. Step size $\sigma$ of the search grid is the parameter that affects discrete planning performance most because it determines the amount of movement at each step during the $A^*$ search. The degree $h$ of Bézier curves is important in trajectory optimization because it determines the number of decision variables. The obstacle check distance $\tilde{o}$ and robot check distance $\tilde{r}$ determine the number of collision avoidance constraints in the optimization problem.

[††] https://www.ibm.com/analytics/cplex-optimizer

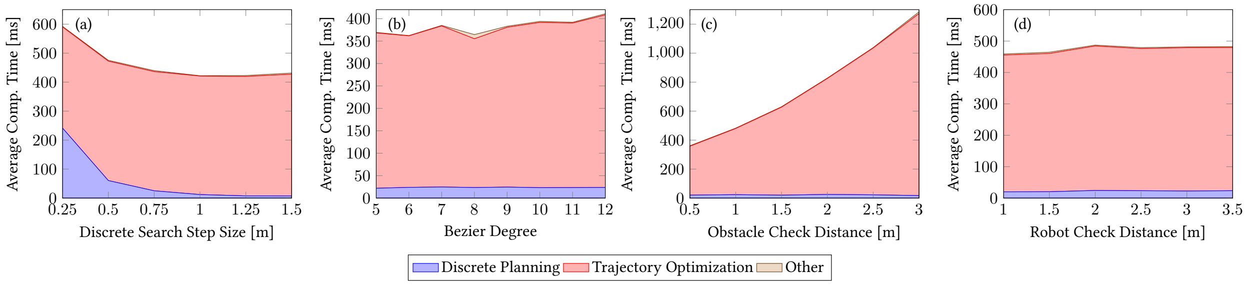


Figure 4.6: Average computation time per stage are given when different parameters are used. Goal selection and temporal rescaling steps are given as "other" since they take a small amount time compared to other two stages. All experiments are done in a 3D random forest environment with 10 % occupancy.

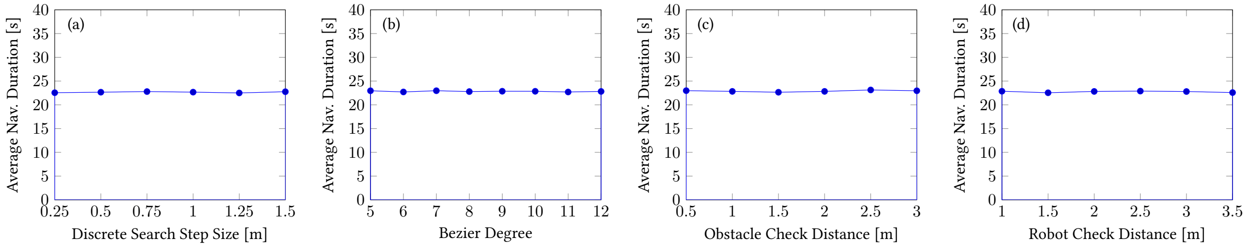


Figure 4.7: Average navigation duration of robots from their start positions to their goal positions are given when the selected parameters are changed. In all cases, average navigation duration is not affected by the changes in the selected parameters in the chosen ranges.

In all of the parameter evaluations, we use a random 3D forest environment with octree resolution $0.5\,\mathrm{m}$ in which $10\,\%$ of the environment is occupied. There are 2332 leaf-level boxes in octree, translating to 2332 obstacles in total. We set the desired trajectory of each robot to the straight line segment that connects the robot's start and goal positions. We set the duration of the segment to the length of the line segment divided by the maximum velocity of the robot. We enforce continuity up to velocity, hence set $c = 1$.

In our experiments, our algorithm does not result in any collisions or deadlocks.

We report average computation time per iteration (Figure 4.6) and average navigation duration of robots (Figure 4.7). Average navigation duration is the summed total navigation time for all robots divided by the number of robots.

#### 4.6.1.1 Step Size of Discrete Search

We evaluate our algorithm's performance when step size $\sigma$ is changed. We set $\sigma$ to values between $0.25\,\mathrm{m}$ to $1.5\,\mathrm{m}$ with $0.25\,\mathrm{m}$ increments. We set obstacle check distance $\tilde{o} = 1.0\,\mathrm{m}$, robot check distance $\tilde{r} = 2.0\,\mathrm{m}$, and degree of Bézier curves $h = 12$ in all cases, which are determined by premiliminary experiments and the results of the experiments in Sections 4.6.1.2 to 4.6.1.4. The results are summarized in Figure 4.6a and Figure 4.7a.

As the step size gets smaller, discrete search takes more time; but the algorithm can still work in about $2\,\mathrm{Hz}$ even when $\sigma = 0.25\,\mathrm{m}$. The average navigation duration of robots are close to $22.5\,\mathrm{s}$ in each case, suggesting the robustness of the algorithm to the changes in this parameter. At $\sigma \geq 0.75$, the time discrete planning takes is less than $6\,\%$ of the time trajectory optimization takes.

We also run the algorithm with $\sigma = 3\,\mathrm{m}$, $\sigma = 6\,\mathrm{m}$, and $\sigma = 12\,\mathrm{m}$, which decreases the flexibility of the discrete search considerably. In all of those cases, discrete search results in fluctuations, and some robots get stuck in livelocks, in which they move between same set of positions without reaching to their goal positions.

#### 4.6.1.2 Degree of Bézier Curves

Next, we evaluate our algorithm's performance when the degree $h$ of Bézier curves is changed. We set $h$ to values in $\{5, \ldots, 12\}$. We set step size $\sigma = 0.77\,\mathrm{m}$, obstacle check distance $\tilde{o} = 1.0\,\mathrm{m}$, and robot check distance $\tilde{r} = 2.0\,\mathrm{m}$, which are determined by premiliminary experiments and the results of the experiments in Sections 4.6.1.1, 4.6.1.3 and 4.6.1.4. The results are summarized in Figure 4.6b and Figure 4.7b.

Even if the degree of the Bézier curves determine the number of decision variables of the trajectory optimization, the computation time increase of the trajectory optimization stage is not more than $10\,\%$ between degree 5 Bézier curves and degree 12 Bézier curves. Also, average navigation duration of robots are close to $22.5\,\mathrm{m}$ in each case, suggesting the robustness of the algorithm to the changes in this parameter as well.

#### 4.6.1.3 Obstacle Check Distance

Next, we evaluate our algorithm's performance when the obstacle check distance $\tilde{o}$ is changed. We set $\tilde{o}$ to values between $0.5\,\mathrm{m}$ and $3\,\mathrm{m}$ with $0.5\,\mathrm{m}$ increments. Since the replanning period $\delta t = 0.1\,\mathrm{s}$, and maximum velocity $\gamma^1 = 3.67\frac{m}{s}$, the maximum amount of distance that can be traversed by a robot until the next planning iteration is $0.367\,\mathrm{m}$. The obstacle check distance must be more than this value for safety. We set step size $\sigma = 0.77\,\mathrm{m}$, robot check distance $\tilde{r} = 2.0\,\mathrm{m}$, and degree of Bézier curves $h = 12$, which are determined by premiliminary experiments and the results of the experiments in Sections 4.6.1.1, 4.6.1.2 and 4.6.1.4. The results are summarized in Figure 4.6c and Figure 4.7c.

The obstacle check distance is the most important parameter that determines the speed of trajectory optimization, and hence the planning pipeline. As $\tilde{o}$ increases, the number of SVM computations and the number of constraints in the optimization problem increases, which results in increased computation time. Average navigation durations of the robots are close to $22.5\,\mathrm{m}$ in all cases, suggesting the robustness of the algorithm to this parameter. We explain the reason of this robustness as follows. All obstacles are considered during discrete search and $\tilde{o}$ determines the obstacles that are considered during trajectory optimization. Therefore, the path suggested by the discrete search is already very good, and obstacle avoidance behavior of trajectory optimization is only important when the discrete path is close to obstacles. In those cases, all obstacle check distances capture the obstacles in the vicinity of the path. Therefore, the quality of the planned trajectories does not increase as $\tilde{o}$ increases.

#### 4.6.1.4 Robot Check Distance

Last, we evaluate our algorithm's performance when the robot check distance $\tilde{r}$ is changed. We set $\tilde{r}$ to values between $1\,\mathrm{m}$ and $3.5\,\mathrm{m}$ with $0.5\,\mathrm{m}$ increments. Robot check distance must be at least twice the amount of distance that can be traversed by the robot in one planning iteration, i.e. $0.734\,\mathrm{m}$, because two robots may be travelling towards each other with their maximum speed in the worst case. We set step size $\sigma = 0.77\,\mathrm{m}$, obstacle check distance $\tilde{o} = 1.0\,\mathrm{m}$, and degree of Bézier curves $h = 12$, which are determined by premiliminary experiments and the results of the experiments in Sections 4.6.1.1 to 4.6.1.3.

The speed of the algorithm is not affected by the robot check distance considerably, because there are 32 robots in the environment, and from the perspective of the ego robot, there are at most 31 constraints generated from other robots. Since there are more than 2000 obstacles in the environment, effects of the obstacle check distance are more drastic than robot check distance. Similar to other cases, average navigation duration of the robots is not affected by the choice of $\tilde{r}$ because constraints generated by the robots far away do not actually constrain the trajectory of the ego robot since the ego robot cannot move faster than its maximum speed and hence the first piece of the trajectory is never affected by those constraints.

Overall, RLSS does not result in collisions or deadlocks when parameters are not set to extreme values. In addition, changes in parameters, outside of extreme ranges, do not result in significant changes in the average navigation durations. These suggest that RLSS does not need extensive parameter tuning.

### 4.6.2 Ablation Study

We investigate the effects of i) the prepend operation of the discrete planning stage (Line 4, Algorithm 2) and ii) the preferred distance cost term $\mathcal{J}_{pref}$ of the trajectory optimization stage to the performance of the algorithm. The prepend operation enables the kinematic feasibility of the generated optimization problem as shown in Remark 2. The preferred distance cost term increases the numerical stability of the algorithm by encouraging robots to create a gap between themselves and other objects.

Table 4.2: The results of the ablation study. We compare four different versions of RLSS. We ablate i) the prepend operation of the discrete planning stage and ii) the preferred distance cost term of the trajectory optimization stage. The details of the metrics are given in Section 4.6.2. The reported values are means and standard deviations (given in parentheses) of 10 experiments in random maze-like environments. Both prepend operation and preferred distance cost term are important for the effectiveness of the algorithm.

| | Fail Rate | # Coll. | Avg. Nav. Dur [s] |
|---|---|---|---|
| RLSS | **0.89 (1.45) / 7904 (297)** | **0 (0)** | **24.68 (0.93)** |
| RLSS w/o pref. dist. | 4.78 (9.81) / 8194 (489) | 0.67 (1) | 25.58 (1.53) |
| RLSS w/o prepend | 1556 (120) / 12220 (538) | 6.78 (3.73) | 38.17 (1.68) |
| RLSS with neither | 1569 (101) / 14778 (412) | 8.67 (2.87) | 46.16 (1.29) |

We consider four versions our algorithm: RLSS, RLSS without the prepend operation (RLSS w/o prepend), RLSS without the preferred distance cost term (RLSS w/o pref. dist.) and RLSS with neither. We set step size $\sigma = 0.77\,\mathrm{m}$, obstacle check distance $\tilde{o} = 1.0\,\mathrm{m}$, robot check distance $\tilde{r} = 2.0\,\mathrm{m}$, degree of Bézier curves $h = 12$, and $c = 1$ (continuity up to velocity). Robots navigate in 3D maze like environments. The desired trajectories are set to straight line segments connecting robot start positions to goal positions and the durations of the segments are set to the length of the line segments divided by the maximum speed of the robots. During simulation, robots continue using their existing plans when planning fails. Also, colliding robots continue navigating and are not removed from the experiment.

We generate ten 3D-maze like environments and list the average and standard deviation values for our metrics in Table 4.2. We report the failure rate of all algorithms in the form of the ratio of number of failures to the number of planning iterations (*Fail Rate* column), the number of robots that are involved in at least one collision during navigation (*# Coll.* column) and the average navigation duration of all robots (*Avg. Nav. Dur.* column).

The failure rate of RLSS is $0.01\%$ on average. RLSS w/o pref. dist. fails $0.06\%$ of the time. RLSS w/o prepend fails $12.73\%$ of the time. This is drastically more than RLSS w/o pref. dist because our prepend operation ensures the kinematic feasibility of the optimization problem while our preferred distance cost term tackles numerical instabilities only. RLSS with neither results in a failure rate of $10.62\%$. Interestingly, RLSS with the preferred distance cost term but without the prepend operation (RLSS w/o prepend) results in a higher failure rate than RLSS with neither. We do not investigate the root cause of this since failure rates in both cases are a lot higher than of RLSS.

The effects of failures are seen in the next two metrics. RLSS results in no collisions. The number of colliding robots increase to $0.67$ in RLSS w/o pref. dist, $6.78$ in RLSS w/o prepend and $8.67$ in RLSS with neither on average. The average navigation duration of robots is lowest in RLSS. It increases by $3.65\%$ in RLSS w/o pref. dis, $54.66\%$ in RLSS w/o prepend and $87.03\%$ in RLSS with neither compared to RLSS.

These results show that the prepend operation is more important than the preferred distance cost term for the success of the algorithm. Nevertheless, RLSS needs both to be safe and effective.

### 4.6.3 Comparisons with Baseline Planners

We compare the performance of our planner to two baseline planners that do not require communication in 3D experiments. We set step size $\sigma = 0.77\,\mathrm{m}$, obstacle check distance $\tilde{o} = 1.0\,\mathrm{m}$, robot check distance $\tilde{r} = 2.0\,\mathrm{m}$, and degree of Bézier curves $h = 12$ in RLSS in all cases.

#### 4.6.3.1 Extended Buffered Voronoi Cell (eBVC) Planner

The first baseling planner is a MPC-style planner based on buffered Voronoi cells introduced by Zhou et al. [135], which we call BVC. In the BVC approach, each robot computes its Voronoi cell within the Voronoi tesselation of the environment. This is done by using the position information of other robots. Robots buffer their Voronoi cells to account for their collision shapes and plan their trajectories within their corresponding buffered Voronoi cells. Similar to RLSS, BVC does not require any communication between robots, requires perfect sensing of robot positions in the environment, and the resulting trajectories are safe only when planning is synchronized between robots. They also require that each robot stays within its buffered Voronoi cell until the next planning iterations. Unlike RLSS, BVC is based on buffered Voronoi cells and it cannot work with arbitrary convex objects. Instead, robots are modeled as hyperspheres in BVC.

The original formulation presented in the BVC article only allows position state for the robots, which cannot model a rich set of dynamics, including double, triple, or higher order integrators. We extend their formulation to all discrete linear time invariant systems with position output. We define the systems with three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. Since the output of the system is the position of the robot, it cannot depend on the current input, hence $\mathbf{D} = \mathbf{0}$. This allows us to formulate the problem for a richer set of dynamics and have constraints on higher order derivatives than robot velocity.

BVC as published also does not consider obstacles in the environment. We extend their formulation to static obstacles by modeling obstacles as robots. Since obstacles are static, they stay within their buffered Voronoi cells at all times. Since we model obstacles as robots, extended BVC does not require the ability of distinguishing robots from obstacles.

Our extended BVC formulation, which we call eBVC, is as follows:

$$\min_{\mathbf{u}_0,\ldots,\mathbf{u}_{M-1}} \sum_{i=1}^{M} \lambda_i \left\|\mathbf{p}_i - \mathbf{d}(\tilde{T} + M\Delta t)\right\|_2^2 + \sum_{i=0}^{M-1} \theta_i \left\|\mathbf{u}_i\right\|_2^2 \, s.t.$$

$$\mathbf{x}_{i+1} = \mathbf{A}\mathbf{x}_i + \mathbf{B}\mathbf{u}_i \;\; \forall i \in \{0, \ldots, M-1\}$$

$$\mathbf{p}_i = \mathbf{C}\mathbf{x}_i \;\; \forall i \in \{0, \ldots, M\}$$

$$\mathbf{p}_i \in \mathcal{V} \;\; \forall i \in \{0, \ldots, M\}$$

$$\mathbf{p}_i \in \mathcal{W} \;\; \forall i \in \{0, \ldots, M\}$$

$$\mathbf{u}_{min} \preceq \mathbf{u}_i \preceq \mathbf{u}_{max} \;\; \forall i \in \{0, \ldots, M-1\}$$

$$\mathbf{x}_{min} \preceq \mathbf{x}_i \preceq \mathbf{x}_{max} \;\; \forall i \in \{0, \ldots, M\}$$

where $\tilde{T}$ is the current timestamp, $\mathbf{d}(t)$ is the desired trajectory for the robot, $\Delta t$ is the discretization timestep of the system, $M$ is the number of steps to plan for, $\mathbf{u}_i$ is the input to apply from timestep $i$ to $i+1$, $\mathbf{x}_i$ is the state at timestep $i$, $\mathbf{p}_i$ is the position at timestep $i$, $\mathcal{V}$ is the buffered Voronoi cell of the robot, $\mathbf{u}_{min}$ and $\mathbf{u}_{max}$ are the limits for the inputs, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the limits for the states (which can be used to bound velocity in a double integrator system, or velocity and acceleration in a triple integrator system for example), and $\mathcal{W}$ is the workspace of the robot. $M\Delta t$ is the planning horizon. A robot plans toward the position $\mathbf{d}(\tilde{T} + M\Delta t)$, which is the position of the robot after the planning horizon if it could follow the desired trajectory perfectly. $\mathbf{x}_0$ is the current state of the robot. The first term of the cost function penalizes deviation from the goal position for the final and each intermediate position with different weights $\lambda_i$. The second term of the cost function is the input cost that penalizes input magnitudes with different weights $\theta_i$. We apply the first input of the solution for duration $\Delta t$, and replan at the next timestep.

We use our own implementation of eBVC as explained above during the comparisons.

#### 4.6.3.2 Relative Safe Flight Corridor (RSFC) Planner

The second planner we compare against is presented by Park and Kim [77], in which piecewise Bézier curves are computed, executed for a short duration, and replanning is done at the next iteration similar to our work. It utilizes the fact that the difference of two Bézier curves is another Bézier curve by constraining these relative Bézier curves to be inside safe regions (relative safe flight corridors, or RSFCs) defined according to robot collision shapes. We call this algorithm RSFC for short. RSFC does not require any communication between robots. It utilizes both positions and velocities of other objects in the environment, hence requires more sensing information than our algorithm. Velocities are used to predict the trajectories of other robots as piecewise Bézier curves. While it can handle dynamic obstacles as well, we use it in static environments in our comparisons, since RLSS does not handle dynamic obstacles explicitly.

We use the authors' implementation of RSFC during our comparisons.

Table 4.3: The results of the comparisons of RLSS, eBVC, and RSFC are summarized. Each experiment differ in the map used during navigation, prior map used during desired trajectory computation, and the required degree of continuity. RLSS results in longer navigation durations than eBVC and RSFC on average; but eBVC suffers from deadlocks in environments with obstacles and RSFC results in collisions in environments with obstacles.

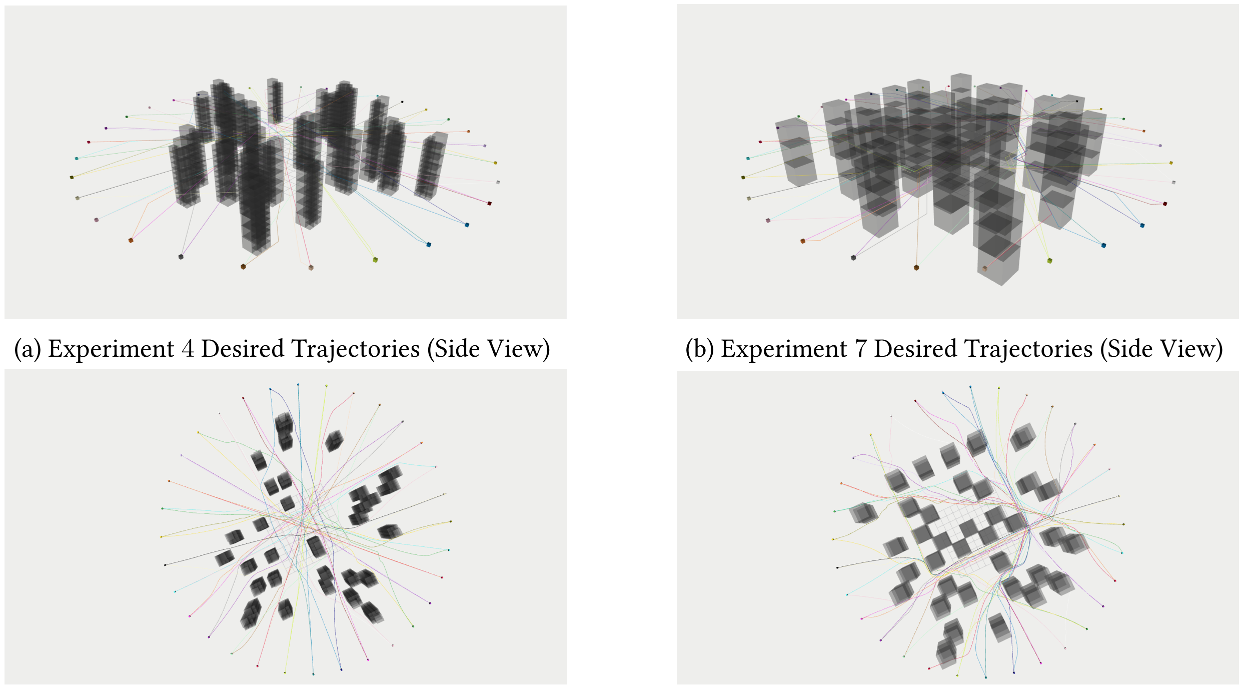
| Experiment | Map | Prior Map | Continuity | Algorithm | # Deadlocks | # Coll. Robots | Avg. Collision Dur. [s] | Avg. Nav. Dur. [s] | Avg. Comp. Time [ms] |
|---|---|---|---|---|---|---|---|---|---|
| 1 | empty | empty | velocity | RLSS | 0 | 0 | NA | 22.37 | 107 |
| | | | | eBVC | 0 | 0 | NA | **18.50** | **106** |
| 2 | | | acceleration | RLSS | 0 | 0 | NA | 22.12 | 110 |
| | | | | eBVC | 0 | 0 | NA | 20.65 | 161 |
| | | | | RSFC | 0 | 0 | NA | **19.83** | **47** |
| 3 | forest | forest | velocity | RLSS | **0** | **0** | **NA** | 23.11 | **182** |
| | | | | eBVC | 10 | 7 | 0.53 | **18.94** | 387 |
| 4 | | | acceleration | RLSS | **0** | **0** | **NA** | 23.06 | **147** |
| | | | | eBVC | 8 | 13 | 0.83 | **21.24** | 763 |
| 5 | forest | empty | velocity | RLSS | **0** | **0** | **NA** | 22.62 | **192** |
| | | | | eBVC | 8 | 24 | 1.35 | **18.54** | 186 |
| 6 | | | acceleration | RLSS | **0** | **0** | **NA** | 22.72 | 244 |
| | | | | eBVC | 11 | 15 | 0.71 | **21.28** | 970 |
| | | | | RSFC | **0** | 6 | 0.42 | 21.82 | **201** |
| 7 | maze | maze | velocity | RLSS | **0** | **0** | **NA** | 25.09 | 160 |
| | | | | eBVC | 21 | 9 | 0.67 | **20.12** | **145** |
| 8 | | | acceleration | RLSS | **0** | **0** | **NA** | 25.32 | **170** |
| | | | | eBVC | 17 | 13 | 1.49 | **23.80** | 264 |
| 9 | maze | empty | velocity | RLSS | **0** | **0** | **NA** | **27.98** | 410 |
| | | | | eBVC | 32 | 0 | NA | NA | **176** |
| 10 | | | acceleration | RLSS | **0** | **0** | **NA** | 32.04 | 386 |
| | | | | eBVC | 30 | 2 | 3.26 | **21.75** | 407 |
| | | | | RSFC | **0** | 22 | 0.34 | 28.23 | **80** |

#### 4.6.3.3 Experiments & Results

We compare RLSS against eBVC and RSFC in 10 different experiments differing in required degree of continuity, desired trajectories, and map of the environment. In all experiments, 32 robots are placed in a circle formation with radius 20 m in 3D. The task is to swap the positions of robots to the antipodal points on the circle. The results of the experiments are summarized in Table 4.3.

There are 3 maps we use: empty, forest (Figure 4.8a), and maze (Figure 4.8b), listed in the map column of Table 4.3. In the empty map, there are no obstacles in the environment. The forest map is a random forest with 10 % occupancy; it has a radius of 15 m and each tree is a cylinder with radius 0.5 m. The maze map is a maze-like environment with choke regions.

We compute the desired trajectories of the robots by running a single-agent shortest path using the discrete planning stage of RLSS. We run single-agent shortest path on a prior map, which is set to either a full map of the environment or to an empty map; this is listed in the prior map column of Table 4.3.



(a) Experiment 4 Desired Trajectories (Side View)

(b) Experiment 7 Desired Trajectories (Side View)

(c) Experiment 4 Executed Trajectories (RLSS / Top View)

(d) Experiment 7 Executed Trajectories (RLSS / Top View)

Figure 4.8: Desired trajectories and the used forest map of experiment 4 is given in (a). Same forest map is used in each forest experiment. Desired trajectories and the used maze map of experiment 7 is given in (b). Same maze map is used in each maze experiment. (c) shows the executed trajectories of robots running RLSS from top in experiment 4. (d) shows the executed trajectories of robots running RLSS from top in experiment 7.

When the prior map is empty, single-agent shortest paths are straight line segments connecting robot start positions to robot goal positions.

We set two different continuity requirements: velocity and acceleration, listed in the continuity column of Table 4.3. When velocity continuity is required, the system of eBVC is a double integrator with position output. When acceleration continuity is required, the system of eBVC is a triple integrator with position output.

We compare RSFC to RLSS and eBVC only with acceleration continuity requirement because the authors' implementation hard codes the acceleration continuity requirement, while in theory it can work with any degree. Also, we compare RSFC only in the case of an empty prior map in order not to change RSFC's source code, while in theory it can be guided with arbitrary trajectories.

We plan for $5\,\mathrm{s}$ long trajectories in every $0.1\,\mathrm{s}$ with all algorithms. In eBVC, we plan for $M = 50$ steps with discretization timestep $\Delta t = 0.1\,\mathrm{s}$. We set state and input upper and lower bounds in eBVC in order to obey the dynamic limits of the robots. We set distance to goal weights $\lambda_1 = 120$, $\lambda_i = 20\ \forall i \geq 2$ in eBVC, putting more importance on the position of the robot after $1$ timestep. We set $\theta_i = 1\ \forall i$ in eBVC.

Both eBVC and RSFC require spherical obstacles. We use the smallest spheres containing each leaf-level box of the octree structure as obstacles in eBVC and RSFC. Similar to RLSS, we use robot and obstacle check distance to limit the number of obstacles considered at each iteration. We set both obstacle and robot check distance $\tilde{o} = \tilde{r} = 2.0\,\mathrm{m}$ in eBVC, and set $\tilde{o} = \tilde{r} = 5.0\,\mathrm{m}$ in RSFC, since smaller values in RSFC result in a high number of collisions and higher values for the parameters do not improve the success of the algorithms. Robots are modeled as spheres in eBVC and RSFC as well. We set robot shapes to spheres with radius $0.173\,\mathrm{m}$, which are the smallest spheres containing the actual robot shapes. We count collisions only when contained leaf-level octree boxes and contained robot shapes intersect. This gives both eBVC and RSFC buffer zones for collisions.

We report the number of deadlocking robots (*# Deadlocks* column in Table 4.3), number of robots that are involved in at least one collision (*# Coll. Robots* column in Table 4.3), collision duration of robots that are involved in collisions averaged over robots (*Avg. Collision Dur.* column in Table 4.3), average navigation duration of non-deadlocking robots from their start positions to goal positions (*Avg. Nav. Dur.* column in Table 4.3), and computation time per iteration averaged over each planning iteration of each robot (*Avg. Comp. Time* column in Table 4.3). We continue the navigation of colliding robots and do not remove them from the experiment.

Executed trajectories of robots running RLSS are shown in Figure 4.8c for experiment 4, and in Figure 4.8d for experiment 7 as examples.

RLSS does not result in any deadlocks or collisions in all cases. eBVC has a significant number of deadlocks and RSFC results in collisions in experiments with obstacles.

When there are no obstacles in the environment, e.g., in experiments 1 and 2, no algorithm results in deadlocks or collisions. When velocity continuity is required, e.g., in experiment 1, the average navigation duration of robots running eBVC is 18 % lower than those that run RLSS. Both eBVC and RLSS run close to 9 Hz on average. When acceleration continuity is required, e.g., in experiment 2, the average navigation duration of robots running RSFC is 10 % lower than those that run RLSS; and the average navigation duration for eBVC is 7 % lower than that for RLSS. RLSS runs at about 9 Hz on average, eBVC runs close to 6 Hz on average, and RSFC runs close to 21 Hz on average. When there are no obstacles in the environment, the discrete search of RLSS results in unnecessary avoidance movements, which is the main reason for average navigation duration differences.

When there are obstacles in the environment, performance of both eBVC and RSFC degrades in terms of the number of deadlocks and collisions.

In experiment 3, even if the full prior map of the environment is given during initial discrete search with only velocity continuity, 10 out of 32 robots deadlock, and 7 out of the remaining 22 get involved in at least one collision when eBVC is used. When acceleration continuity is required (experiment 4), 8 robots deadlock and 13 other robots get involved in collisions, resulting in only 11 robot reaching their goal volumes without collisions in eBVC. RLSS both works faster than eBVC and results in no deadlocks or collisions in those cases.

When the prior map is not known in the forest environment (experiments 5 and 6), eBVC results in a lot of deadlocks and collisions. All robots in experiment 5 either deadlock or collide when eBVC is used. In experiment 6, RSFC does a lot better than eBVC. RSFC results in no deadlocks while eBVC results in 11 deadlocks. 15 out of remaining 21 robots get involved in at least one collision when eBVC is used with an average collision duration of 0.71 s. 6 out of 32 robots running RSFC collide at least once with average collision duration of 0.42 s. RLSS does not result in any deadlocks or collisions.

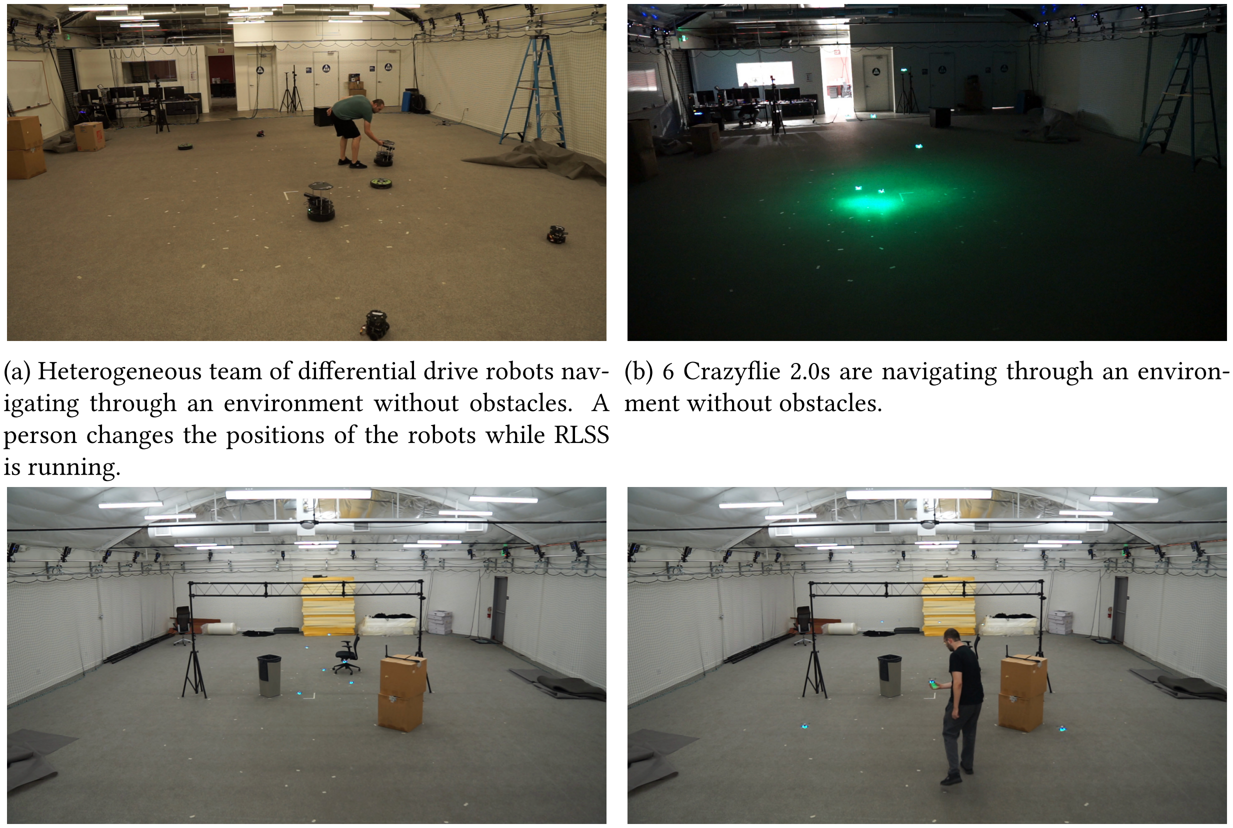
When the environment is a complicated maze, the performance of both eBVC and RSFC degrades more. With full prior map and velocity continuity (experiment 7), 21 out of 32 robots deadlock, 9 out of remaining 11 are involved in collisions, leaving only 2 reaching to their goal volumes without an incident when eBVC is used. With full prior map and acceleration continuity (experiment 8), 17 out of 32 robots deadlock, 13 out of remaining 15 are involved in collisions, again leaving only 2 that reach their goal volumes without incident when eBVC is used. RLSS does not result in any deadlocks or collisions in these scenarios. When the prior map is empty, i.e. the desired trajectories are straight line segments, the performance of eBVC degrades even more. All robots deadlock in the case with velocity continuity (experiment 9), while 30 out of 32 robots deadlock and the rest get involved in collisions in the case with acceleration continuity (experiment 10). No robot running RSFC deadlocks but 22 out of 32 get involved in collisions at least once with 0.34 s average collision duration in experiment 10 with acceleration continuity. RLSS does not result in any deadlocks or collisions in those cases.

Overall, robots running RLSS have higher navigation durations than those that use eBVC or RSFC in all experiments. While the higher navigation duration is not an important metric when other algorithms cause deadlocks or collisions, it is an important metric when they do not (experiments 1 and 2 with no obstacles in particular). eBVC does not have an integrated discrete planner, which is the reason for its good performance in terms of average navigation durations. When robots are close to each other, RLSS uses the free space less effectively since discrete planning has a step size $0.77\,\mathrm{m}$. eBVC does not rely on discrete planning and hence avoids other robots by executing small direction changes. RSFC utilizes velocities of other robots on top of positions, which allows it to estimate the intents of robots more effectively, resulting in a better usage of the free space, and hence results in better navigation durations on average. Since RLSS does not utilize communication and uses only positions of other robots, it cannot deduce the intents of other robots. This results in fluctuations of plans between planning iterations, which increases the average navigation durations of robots. Fluctuations of plans increase when the environment is dense as seen in the supplemental video[‡‡]. If the environment becomes overly constraining, e.g. tens of robots trying to pass through a narrow tube, these fluctuations may turn into livelocks.

However, when obstacles are introduced in the environment, the performance of RLSS is better than other algorithms. eBVC suffers greatly from deadlocks. RSFC does not result in deadlocks but results in collisions, even while using more information than RLSS (velocity and position instead of position only).

**A note about the statistical significance of the results:** In each experiment, we run each algorithm on single randomly generated forest-like or maze-like environment. To show that the results are consistent for environment types, we generate 10 forest-like environments with the same parameters for experiment 3 and run RLSS and eBVC. RLSS does not result in deadlocks or collisions in any of the cases. The average navigation duration of robots averaged over 10 environments is $19.15\,\mathrm{s}$ with standard deviation $0.24\,\mathrm{s}$ for eBVC and $23.59\,\mathrm{s}$ with standard deviation $0.25\,\mathrm{s}$ for RLSS. The ratio between average navigation durations when eBVC or RLSS is used is consistent with the reported values given in  Table 4.3 for experiment 3.

[‡‡] https://youtu.be/Jrdvf2qyzrg



(a) Heterogeneous team of differential drive robots navigating through an environment without obstacles. A person changes the positions of the robots while RLSS is running.

(b) 6 Crazyflie 2.0s are navigating through an environment without obstacles.

(c) 6 Crazyflie 2.0s are navigating through an environment with obstacles.

(d) 6 Crazyflie 2.0s are navigating through an environment with obstacles. A person changes the positions of the robots while RLSS is running.

Figure 4.9: Physical robot experiments using Turtlebot2s, Turtlebot3s, iRobot Create2s, and Crazyflie 2.0s. RLSS works in real-time under external disturbances.

### 4.6.4 Physical Robots

We implement and run RLSS on physical robots using iRobot Create2s, Turtlebot2s and Turtlebot 3s in 2D; and Crazyflie 2.0s in 3D. We use a VICON motion tracking system for localization. Robots do not sense, but receive the position of others using the VICON system. iRobot Create2s, Turtlebot3s, and Turtlebot2s are equipped with ODROID XU4 and ODROID C1+ single board computers running ROS Kinetic on the Ubuntu 16.04 operating system. In all cases the algorithm is run on a centralized base station computer using separate processes for each robot. Therefore, unlike simulations, planning is not synchronized between robots on real robot implementations. RLSS does not result in any deadlocks in collisions in these asynchronized deployments as well. Commands are sent to 2D robots over a WiFi network, and to Crazyflie 2.0s directly over their custom radio.

We conduct external disturbance experiments with 2 iRobot Create2s, 3 Turtlebot3s, and 2 Turtlebot2s (Figure 4.9a). A human changes the positions of some robots by moving them arbitrarily during execution several times. In all cases, robots replan in real-time and avoid each other successfully.

We demonstrate the algorithm in 3D using 6 Crazyflie 2.0s. We conduct an experiment without obstacles in which Crazyflies swap positions with straight lines as desired trajectories (Figure 4.9b). In another experiment, we show that Crazyflies can navigate through an environment with obstacles (Figures 4.9c and 4.9d). In each case, we externally disturb the Crazyflies and show that they can replan in real-time.

The recordings for our physical robot experiments are included in the supplemental video[§§].

## 4.7 Conclusion

In this chapter, we present RLSS, a real-time decentralized long horizon trajectory planning algorithm for the navigation of multiple robots in shared environments with obstacles that provides guarantees on collision avoidance if the resulting problems are feasible. The generated optimization problem to compute a smooth trajectory is kinematically feasible. It does not require any communication between robots, requires only position sensing of obstacles and robots in the environment, and robots to be able to distinguish other robots from obstacles. With its comparatively minimal sensing requirements and no reliance on communication, it presents a new baseline for algorithms that require communication and sensing/prediction of higher order state components of other robots. The algorithm considers the dynamic limits of the robots explicitly and enforces safety using hard constraints.

[§§] Since we define robots' goals as single points, i.e. sets of measure zero, in physical experiments, robots keep missing their goals slightly. This results in a spinning behavior in 2D since robots continuously fix their positions by replanning.

We show in synchronized simulation that RLSS performs better than two state-of-the-art planning algorithms (eBVC and RSFC), one of which requires velocity sensing on top of position sensing, in environments with obstacles in terms of number of deadlocks and number of colliding robots. In our experiments, RLSS does not result in any deadlocks or collisions, while eBVC suffers from deadlocks and RSFC results in collisions (while RLSS provides theoretical guarantees on collision avoidance when it succeeds, it does not provide theoretical guarantees on deadlock avoidance). When there are no obstacles in the environment, RSFC and eBVC outperform RLSS in terms of average navigation duration by $7\,\%$ to $20\,\%$.

# Chapter 5

# Decentralized Real-time Multi-robot Trajectory Planning for Asynchronous Teammate Avoidance

The assumption of the synchronization between robots in the previous chapter may result in unsafe behavior because of asynchronous planning inherent in decentralized deployments.

In this chapter, we present a novel overconstraining and constraint-discarding method for real-time, decentralized, multi-robot trajectory planning that ensures collision avoidance under asynchronous decision making. Our approach utilizes communication between robots. The communication medium is best-effort: messages may be dropped, re-ordered or delayed. Robots conservatively constrain themselves against others assuming they may be working with outdated information, and discard constraints when they receive update messages from others. Our method can augment existing synchronized decentralized receding horizon planning algorithms that utilize separating hyperplanes for collision avoidance, e.g., RLSS (Chapter 4) or BVC [135], thereby making them applicable to asynchronous setups. As an example, we extend an existing model predictive control based, synchronized, decentralized multi-robot planner, i.e., BVC [135], using our method. We show our method's effectiveness under asynchronous planning and imperfect communication by comparing our extension to the base version. Our extension does not result in any collisions or synchronization-induced deadlocks to which the base version is prone.

This chapter is based on Baskın Şenbaşlar and Gaurav S Sukhatme. "Asynchronous Real-time Decentralized Multi-Robot Trajectory Planning". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2022, pp. 9972–9979. DOI: 10.1109/IROS47612.2022.9981760.

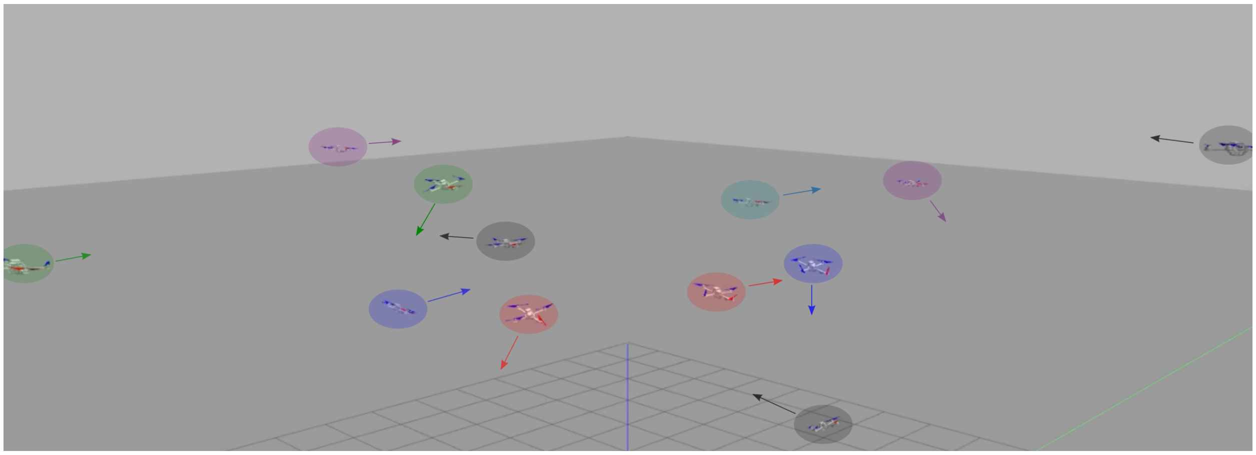


Figure 5.1: Twelve quadrotors navigating in close proximity to each other. Multi-robot trajectory planning is a central problem in close proximity scenarios. Our approach ensures collision and synchronization-induced deadlock avoidance between robots when planning is decentralized and asynchronous.

## 5.1 Introduction

One often-overlooked problem in decentralized multi-robot planning algorithms is the inherent asynchronicity that stems from the planning system. First, since each robot plans for itself, synchronization of when planning starts is not practical. Since planning start time points are not synchronized, robots use different snapshots of the environment while planning, which causes disagreements between them on the definitions of *safety*. Second, planning end time points between robots are not synchronized, which means that different robots start executing plans at different times, causing disagreements during plan switching, leading to disagreements on the definitions of *safety* as well. Additionally, there is asynchronicity introduced by the communication network – when robots use a communication channel to coordinate their actions, their messages may get delayed, dropped or re-ordered.

We propose a novel approach which ensures that robots do not collide with each other when planning is asynchronous and the communication medium is an imperfect best-effort network. Our contributions are as follows:

- We define **mutually computable separating hyperplane trajectories** that form the basis of our approach.

- Using these trajectories, we propose **a novel overconstraining and constraint-discarding system for decentralized multi-robot asynchronous planning** that ensures safety in the face of message delays, drops, and re-orderings.

- To illustrate the utility of our approach, we present an **implementation of AsyncBVC (Asynchronous Buffered Voronoi Cell)**, an extension of BVC [135] based on our system. In simulations, we show that AsyncBVC outperforms BVC in terms of number of synchronization-induced deadlocks and number of collisions under asynchronous planning in obstacle free environments. We also show the performance of AsyncBVC under message delays, drops and re-orderings.

Our method can be applied to several synchronized real-time decentralized multi-robot planning algorithms that enforce robot-robot collision avoidance using separating hyperplanes between robot geometries (examples include BVC [135], RTE [100] as well as RLSS (Chapter 4)), in order to extend them to asynchronous planning systems. In this chapter, we show how to do this in practice with BVC, and in Chapter 6, we show a way to do this in a RLSS-like planner.

## 5.2 Problem

Consider a team of $N$ robots tasked with navigating from their start positions $\mathbf{s}_i \in \mathbb{R}^d$ to their corresponding goal positions $\mathbf{g}_i \in \mathbb{R}^d$ without collisions where $i \in \{1, \ldots, N\}$ is the index of a robot. The robots may potentially all have different shapes and sizes. We model robots as non-rotating rigid bodies. Let $\mathcal{R}_i : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}^d)$ be the collision shape function of robot $i$, such that $\mathcal{R}_i(\mathbf{p})$ is the subset of $\mathbb{R}^d$ occupied by robot $i$ when it is placed at position $\mathbf{p}$. Here, $d \in \{2, 3\}$ is the dimension of the Euclidean space that the robot operates in and $\mathcal{P}(\mathbb{R}^d)$ is the power set of $\mathbb{R}^d$. We define $\mathcal{R}_i(\mathbf{p})$ as the Minkowski sum $\mathcal{R}_i^{\mathbf{0}} \oplus \{\mathbf{p}\}$, where $\mathcal{R}_i^{\mathbf{0}}$ is the subset of $\mathbb{R}^d$ occupied by robot $i$ when it is placed at the origin $\mathbf{0}$. For simplicity of exposition, we assume that the environment is obstacle-free.

The general problem is to compute trajectories $\mathbf{f}_i(t) : [0, T] \to \mathbb{R}^d$ for each robot $i$ along with the team navigation duration $T$ such that

- $\mathbf{f}_i(t)$ is dynamically feasible according to $i^{th}$ robot's dynamics,
- $\mathcal{R}_i(\mathbf{f}_i(t)) \cap \mathcal{R}_j(\mathbf{f}_j(t)) = \emptyset \; \forall t \in [0, T] \; \forall i \neq j$, i.e. robots do not collide with each other,
- $\frac{d^k \mathbf{f}_i(0)}{dt^k} = \frac{d^k \mathbf{f}_i(T)}{dt^k} = \mathbf{0} \; \forall k \geq 1 \; \forall i \in \{1, \dots, N\}$, i.e. robots are stationary at the start and the end of the computed trajectories,
- $\mathbf{f}_i(0) = \mathbf{s}_i, \mathbf{f}_i(T) = \mathbf{g}_i \; \forall i \in \{1, \dots, N\}$, i.e. the computed trajectory for each robot commences at its start position and ends at the prescribed goal position.

We focus on a subset of decentralized receding horizon planning style approaches to solve the general problem above. In decentralized receding horizon planning, robots plan long trajectories in real-time for themselves using on-board capabilities. They execute the planned trajectories for a short duration, and re-plan. Our focus is on algorithms that utilize separating hyperplanes between robots as collision avoidance constraints.

Robots may communicate with each other using a best-effort communication medium, but each robot plans a trajectory *only* for itself. The delay of the communication medium need not be bounded and can theoretically grow to infinity. The delay may freely vary from message to message. Messages sent through the medium may get lost or reordered.

Planning start and end times across robots need not be synchronized and planning algorithms are not necessarily proven to be failure-free.

## 5.3 Preliminaries

**Definition 9.** ***Commutative Deterministic Separating Hyperplane Computation Algorithm***. *A commutative deterministic separating hyperplane computation algorithm $\Omega$ accepts two linearly separable sets $\mathcal{A}$*

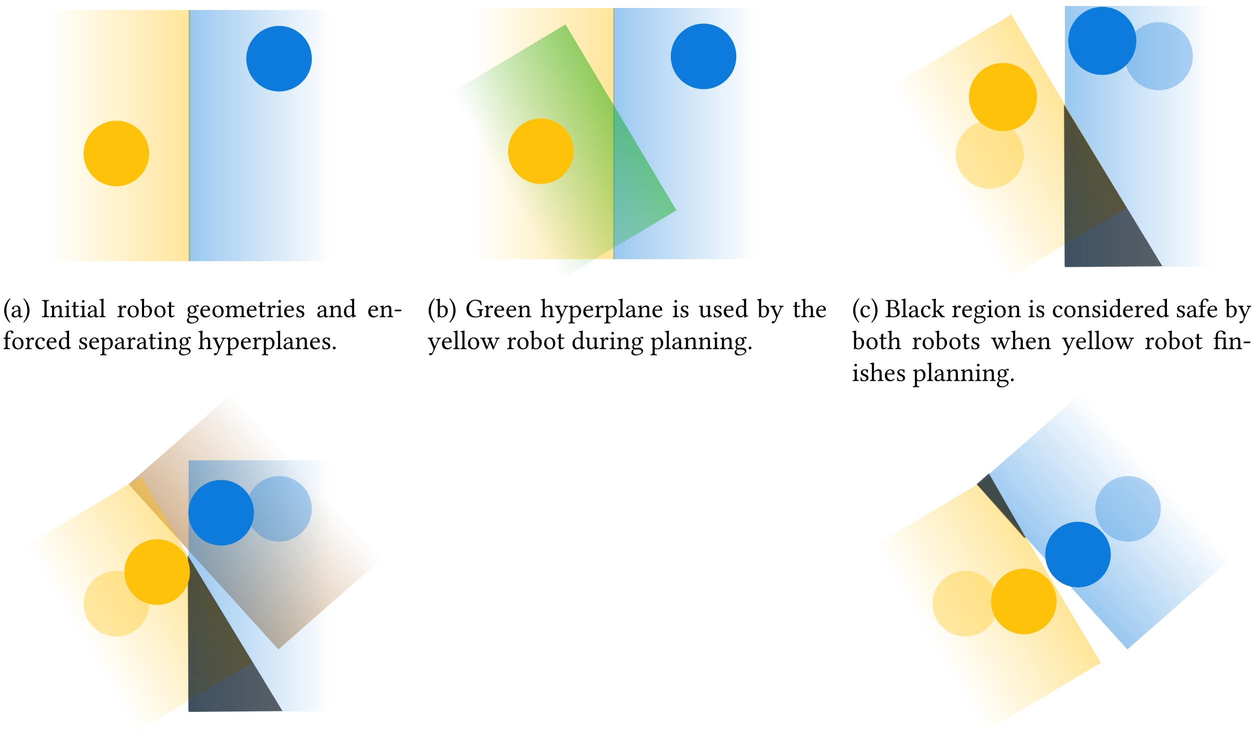
*and $\mathcal{B}$ as arguments, and computes a separating hyperplane between $\mathcal{A}$ and $\mathcal{B}$ such that the computed hyperplanes do not depend on the order of the arguments, i.e. $\Omega(\mathcal{A}, \mathcal{B}) = \Omega(\mathcal{B}, \mathcal{A})$, and the computed hyperplane is same for each call with the same arguments, i.e. there is no randomization.*

**Definition 10.** ***Mutually Computable Separating Hyperplane**. Given two linearly separable sets $\mathcal{A}$ and $\mathcal{B}$ and an hyperplane $\mathcal{H}$ that separates them, $\mathcal{H}$ is called a mutually computable separating hyperplane for $\mathcal{A}$ and $\mathcal{B}$ if and only if there exists a commutative deterministic separating hyperplane computation algorithm $\Omega$ such that $\Omega(\mathcal{A}, \mathcal{B}) = \mathcal{H}$.*

## 5.4 Motivation

Commutative deterministic separating hyperplane computation algorithms are used by several state-of-the-art decentralized real-time receding horizon planning algorithms for collision avoidance. At each planning iteration, robots sense each others' geometries (positions and shapes). Each robot computes separating hyperplanes between itself and other robots using a shared commutative deterministic separating hyperplane computation algorithm. Robots constrain their movements to the regions bounded by the computed separating hyperplanes during planning. Since the commutative deterministic separating hyperplane computation algorithm is shared among robots, each pair of robots compute exactly the same hyperplane using only geometry sensing: each robot in the pair constrains itself against the other using this hyperplane. Therefore, regions used to constrain robot movements are disjoint between robots.

Zhou et al. [135] and Şenbaşlar, Hönig, and Ayanian [100] model robots as hyperspheres and Voronoi diagrams are utilized for computing separating hyperplanes. Voronoi diagrams between hyperspherical shapes are unique, hence the algorithm is deterministic. For hyperspherical disjoint sets, Voronoi hyperplane computation algorithm is commutative, hence robots can compute the same hyperplane between each other using only geometry sensing. In RLSS (Chapter 4) robots are modeled as convex shapes and a hard-margin support vector machine (SVM) is used for computing separating hyperplanes. The hard-margin SVM problem is convex with a unique solution. Therefore it is commutative and deterministic.



(a) Initial robot geometries and enforced separating hyperplanes.

(b) Green hyperplane is used by the yellow robot during planning.

(c) Black region is considered safe by both robots when yellow robot finishes planning.

(d) Brown hyperplane is used by the blue robot during planning.

(e) Black region is considered safe by both robots when blue robot finishes planning.

Figure 5.2: **Hyperplane mismatch leads to unsafe behavior.** Blue and yellow robots utilize Voronoi hyperplanes to create safe regions within which to plan. However, when planning start and end times between robots are not synchronized, the Voronoi hyperplanes computed do not match. Figure 5.2a shows robots geometries (shapes and positions) and enforced Voronoi half-spaces, which are computed at the previous planning iteration, for each robot just before the yellow robot starts planning (assuming perfect synchronization before the current planning iteration). The green half-space in Figure 5.2b is the Voronoi half-space that the yellow robot computes and utilizes during planning. While the yellow robot is planning its own trajectory, robots keep moving according to their previous plans. When the yellow robot finishes planning as shown in Figure 5.2c, the safe regions of the blue and yellow robots intersect (intersection shown in black). Both robots are allowed to navigate through the black region, which would result in collisions if they did so. Robots keep moving before the blue robot starts planning. The brown half-space in Figure 5.2d is the Voronoi half-space the blue robot utilizes during planning. While the blue robot is planning, robots continue moving. When the blue robot finishes planning (Figure 5.2e), the safe regions of the robots intersect at another location.

When robots are in close proximity to each other, the success of these approaches depends on two important assumptions that are hard to realize in practice: i) pairs of robots must use the same inputs during separating hyperplane computation - this holds only if the assumption of synchronization of planning start times between robots holds, and ii) robots start executing the plans at the same time, which assumes synchronization of planning end times between robots. In physical deployments, such synchronization is not realistic. This creates a mismatch between the separating hyperplanes computed and/or used by robots as shown in Figure 5.2. The mismatch of hyperplanes introduces safety regions that intersect with each other, which results in unsafe behavior in tight scenarios.

We introduce mutually computable separating hyperplane *trajectories* as a solution to this problem. Instead of constraining a robot by a single hyperplane for each other robot, we constrain it with a carefully pruned trajectory of hyperplanes.

## 5.5 Approach

### 5.5.1 Assumptions

We assume that robot clocks are synchronized prior to navigation (using for example, the Network Time Protocol (NTP) [67]). This ensures that the time points used by the robots share the same frame of reference. Second, we assume that robots can sense each others' geometries and can identify each other instantly. In reality, assuming visual sensing, there is a time delay after light hits the camera until the robot detects the geometry of another robot and identifies it. We omit this delay. Frame-by-frame tracking and identification of robots can be done in $30\,\text{Hz}$ with state-of-the art object detectors [12]. Third, we assume that robots share a commutative deterministic separating hyperplane computation algorithm $\Omega$, which is given to them prior to navigation.

### 5.5.2 Mutually Computable Separating Hyperplane Trajectories

Without the loss of generality, assume that the navigation starts at time point $0$. Let $\tilde{T}$ be the current time point. Let $\mathbf{f}_i(t) : [0, \tilde{T}] \to \mathbb{R}^d$ and $\mathbf{f}_j(t) : [0, \tilde{T}] \to \mathbb{R}^d$ be the trajectories that robot $i$ and robot

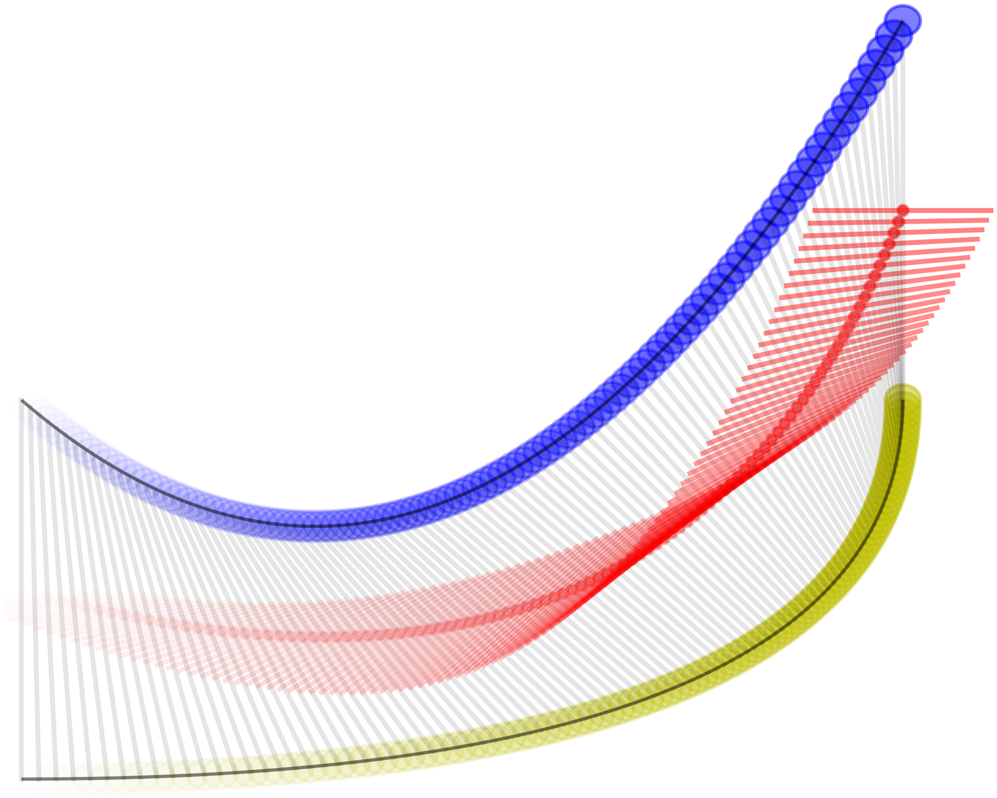


Figure 5.3: **Hyperplane trajectory.** Sampled trajectories of blue and yellow robots are given. The red sampled trajectory is the trajectory of middle points of robots at each sampling step. Red hyperplanes constitute the sampled mutually computable Voronoi hyperplane trajectory between robots.

$j$ traversed until the current time point respectively. The mutually computable separating hyperplane trajectory $\mathcal{H}_{i,j}(t) : [0, \tilde{T}] \rightarrow \mathcal{H}^d$ between robot $i$ and robot $j$ induced by $\Omega$ is defined as

$$\mathcal{H}_{i,j}(t) = \Omega(\mathcal{R}_i(\mathbf{f}_i(t)), \mathcal{R}_j(\mathbf{f}_j(t))) \; \forall t \in [0, \tilde{T}]$$

where $\mathcal{H}^d$ is the set of all hyperplanes in $\mathbb{R}^d$.

Notice that $\mathcal{H}_{i,j}$ can be computed without communication by both robot $i$ and robot $j$ independently as it requires geometry sensing only. Also, $\mathcal{H}_{j,i} = \mathcal{H}_{i,j}$ since $\Omega$ is commutative. An example sampled mutually computable hyperplane trajectory is shown in Figure 5.3 when the commutative deterministic hyperplane computation algorithm is the Voronoi hyperplane computation algorithm.

### 5.5.3 Overconstraining & Constraint-discarding using Mutually Computable Separating Hyperplane Trajectories and Inter-Robot Communication

Our approach is based on the following idea: If the trajectories of each pair $(i, j)$ of robots are constrained by a shared hyperplane, collision avoidance between them can be ensured. In order to ensure that their trajectories are constrained by a shared hyperplane at all times, we constrain them with all hyperplanes in tail portions of the $\mathcal{H}_{i,j}$ in a specific way.

Each robot $i$ stores a *tail time point variable* $T_{i,j} \leq \tilde{T}$ for every other robot $j$ denoting the time point after which the hyperplane trajectory should be used to constrain the trajectory of robot $i$ against robot $j$ where $\tilde{T}$ is the current time point. Specifically, at any instant $\tilde{T}$, if robot $i$ starts planning, it constrains its full trajectory with all hyperplanes $\mathcal{H}_{i,j}(t) \; \forall t \in [T_{i,j}, \tilde{T}]$ to avoid collision with robot $j$. Initially, we set $T_{i,j} = 0 \; \forall i \neq j$. Hence, robots use full mutually computable separating hyperplane trajectories to avoid collisions with each other. This ensures that robot $i$ and robot $j$ will not collide with each other for the initial values of $T_{i,j}$ since they are stationary until the end of their first planning iterations and they share $\mathcal{H}_{i,j}(0)$ afterwards.

**Remark 3.** *For a given robot $i$, its planning start time points are strictly increasing.*

**Lemma 4.** *Assume that robot $j$ successfully planned using information at time $\tau$. The following facts hold for its constraints against every other robot $i$:*

1. *Robot $j$ is constrained by the hyperplane $\mathcal{H}_{j,i}(\tau)$ (which is equal to $\mathcal{H}_{i,j}(\tau)$ since $\Omega$ is commutative).*
2. *In the future, there can be no case in which robot $j$ uses a hyperplane from timespan $[0, \tau)$ to constrain itself against robot $i$ in which it does not also use the hyperplane at time $\tau$.*

*Proof.* Since $T_{j,i}$ is less than or equal to the current time point $\tilde{T}$ at all times and $\tau = \tilde{T}$ at the start of the planning, robot $j$ uses $\mathcal{H}_{j,i}(\tau)$ as a constraint. By Remark 3, planning start time points of robot $j$ strictly increase. Therefore in the future, any planning start time point $\tau'$ will be greater than $\tau$. If robot $j$ uses any hyperplane from timespan $[0, \tau)$ during planning in the future, it means that $T_{j,i} < \tau$. Since it uses hyperplanes in timespan $[T_{j,i}, \tau']$, it has to use the hyperplane at time $\tau$ as well. □

Lemma 4 suggests that if a robot $j$ plans successfully using the information at time point $\tau$, there is no reason for another robot $i$ to use hyperplanes against robot $j$ before time point $\tau$, because robot $j$ is currently constrained by the hyperplane $\mathcal{H}_{i,j}(\tau)$ at time $\tau$ and there will not be any case in the future at which it is constrained by a hyperplane in timespan $[0, \tau)$ in which it is not constrained by the hyperplane at time point $\tau$. Therefore, every other robot $i$ can *prune* its hyperplane trajectory $\mathcal{H}_{i,j}$ against robot $j$ by setting $T_{i,j} = \tau$. Updating $T_{i,j}$ with this rule ensures that trajectories of each pair $(i, j)$ of robots share a constraining hyperplane at all times because i) $\mathcal{H}_{i,j}(0)$ is shared by robots $i$ and $j$ initally, and ii) pruning is done in a way that makes sure that there is at least one shared hyperplane at all times. Sharing a constraining hyperplane at all times guarantees collision avoidance.

Conveying the information to every other robot $i$ that robot $j$ planned successfully at time point $\tau$ is done through the communication medium. Whenever robot $j$ successfully plans a trajectory, it broadcasts the planning success signal $(j, \tau)$, stating that it planned successfully using the information at time point

$\tau$, and there is no reason for other robots to constrain themselves against robot $j$ using hyperplanes in timespan $[0, \tau)$. Until the message arrives, every other robot $i$ *overconstrains* itself against robot $j$ with hyperplanes in $[T_{i,j}, \tau) \cup [\tau, \tilde{T}]$, but *discards constraints* generated from hyperplanes in timespan $[T_{i,j}, \tau)$ when message arrives by setting $T_{i,j} = \tau$. When message re-ordering in the communication medium is possible, $T_{i,j} = \max(T_{i,j}, \tau)$ is used.

If a planning success signal from robot $j$ gets lost in the communication medium, other robots $i$ do not update their $T_{i,j}$ values and keep overconstraining themselves against robot $j$ until a signal from robot $j$ arrives. This allows robots to assume the worst about robot $j$ (i.e., it planned successfully at $T_{i,j}$ and it is not known if it planned successfully again in timespan $(T_{i,j}, \tilde{T}]$) until a message arrives. $T_{i,j}$ represents the last known time point to robot $i$ at which robot $j$ has planned a trajectory successfully.

Note that, if an upper bound $U$ exists on the time difference between two successful planning iterations for all robots, constraints can be discarded even without communication. Here, $T_{i,j}$ is the time point at which robot $j$ has planned (or after which robot $j$ must have planned) according to robot $i$. Robot $i$ simply makes sure that $\tilde{T} - T_{i,j} \leq U$ by setting $T_{i,j} = \max(T_{i,j}, \tilde{T} - U)$ at each iteration for asynchronous safety because robot $j$ plans successfully at least once in every time window of duration $U$.

## 5.6 Experiments

### 5.6.1 Buffered Voronoi Cell (BVC) Planner

We validate our approach using a decentralized model predictive control-based planner that uses Voronoi diagrams for collision avoidance (BVC) [135]. To recap, BVC models robots as hyperspheres parameterized by robot radii. At each planning iteration, Voronoi hyperplanes between robots are computed, buffered to account for robot radii, and used to constrain robot positions. The original formulation of the BVC planner uses discrete single integrator dynamics; we use our extension of BVC (Section 4.6.3.1) to discrete linear dynamics with position outputs. Each robot $i$ solves the following convex quadratic program during planning, making sure that the output (position) is contained in the buffered Voronoi cell of the robot.

$$\min_{\mathbf{u}_0,\ldots,\mathbf{u}_{M-1}} \sum_{k=1}^{M} \lambda_k \left\|\mathbf{p}_k - \mathbf{g}_i\right\|_2^2 + \sum_{k=0}^{M-1} \theta_k \left\|\mathbf{u}_k\right\|_2^2 \, s.t.$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \;\; \forall k \in \{0,\ldots,M-1\}$$

$$\mathbf{p}_k = \mathbf{C}\mathbf{x}_k \;\; \forall k \in \{0,\ldots,M\}$$

$$\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{V}_i \;\; \forall k \in \{0,\ldots,M\}$$

$$\mathbf{u}_{min} \preceq \mathbf{u}_k \preceq \mathbf{u}_{max} \;\; \forall k \in \{0,\ldots,M-1\}$$

$$\mathbf{x}_{min} \preceq \mathbf{x}_k \preceq \mathbf{x}_{max} \;\; \forall k \in \{0,\ldots,M\}$$

where $\mathbf{g}_i$, $\mathcal{V}_i$ and $\mathcal{R}_i$ are the goal position, Voronoi cell, and hyperspherical collision shape function of robot $i$, $M$ is the number of steps to plan for (or the planning horizon), $\mathbf{u}_k$ is the input to apply from timestep $k$ to timestep $k{+}1$, $\mathbf{x}_k$ is the state of robot at timestep $k$, $\mathbf{p}_k$ is the position of robot at timestep $k$, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are matrices that describe the dynamics of the robot, $\mathbf{u}_{min}$ and $\mathbf{u}_{max}$ are the limits for inputs, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the limits for states. The cost of the optimization problem is a weighted combination of the distance of intermediate and final robot positions to the goal position and the input magnitudes. $\lambda_k$s and $\theta_k$s are weight parameters of the cost function. The constraint $\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{V}_i$ is enforced by buffering the Voronoi hyperplanes with the robot radius, and constraining the $\mathbf{p}_k$ with the buffered Voronoi hyperplanes.

### 5.6.2 Asynchronous Buffered Voronoi Cell (AsyncBVC) Planner

In order to ensure safety of BVC under asynchronous planning, we integrate mutually computable separating hyperplane trajectories to the BVC planner by replacing the Voronoi cells with the Voronoi hyperplane trajectories.

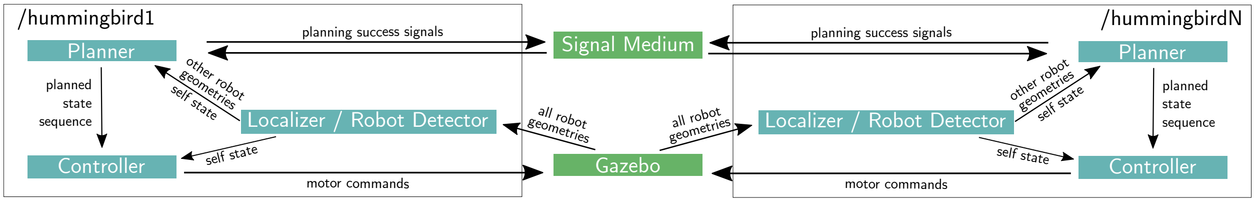


Figure 5.4: **System design.** Each robot has 3 components: a planner, a controller and a localizer/robot detector. The planner plans for a sequences of states, which gets fed into the controller. The controller sends motor commands to the simulator (Gazebo) to track the planned sequence. The localizer/robot detector connects directly to the simulator and feeds other robot geometries and ego robot state to the planner and the controller. A signal medium process acts as a message broker between planners of different robots. We introduce communication delays and message drops using the signal medium process.

Each robot uses Voronoi hyperplane computation algorithm as the commutative deterministic hyperplane computation algorithm $\Omega$. They construct mutually computable separating hyperplane trajectories, or Voronoi hyperplane trajectories specifically, as described in Section 5.5.2 and keep track of tail time point variables as described in Section 5.5.3. We call this algorithm asynchronous buffered Voronoi cell planner, or AsyncBVC for short.

Each robot $i$ solves the following optimization problem:

$$\min_{\mathbf{u}_0,\dots,\mathbf{u}_{M-1}} \sum_{k=1}^{M} \lambda_k \left\|\mathbf{p}_k - \mathbf{g}_i\right\|_2^2 + \sum_{k=0}^{M-1} \theta_k \left\|\mathbf{u}_k\right\|_2^2 \; s.t.$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \;\; \forall k \in \{0,\dots,M-1\}$$

$$\mathbf{p}_k = \mathbf{C}\mathbf{x}_k \;\; \forall k \in \{0,\dots,M\}$$

$$\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{H}^-_{i,j}(t) \;\; \forall k \in [0,\dots,M] \, \forall j \neq i \, \forall t \in [T_{i,j}, \tilde{T}]$$

$$\mathbf{u}_{min} \preceq \mathbf{u}_k \preceq \mathbf{u}_{max} \;\; \forall k \in \{0,\dots,M-1\}$$

$$\mathbf{x}_{min} \preceq \mathbf{x}_k \preceq \mathbf{x}_{max} \;\; \forall k \in \{0,\dots,M\}$$

where $\mathcal{H}^-_{i,j}(t)$ is the robot $i$'s side of the hyperplane $\mathcal{H}_{i,j}(t)$ against robot $j$ and $\tilde{T}$ is the current time point. Similarly to BVC, the constraint $\mathcal{R}_i(\mathbf{p}_k) \in \mathcal{H}^-_{i,j}(t)$ is enforced by buffering the hyperplane with the robot radius, and constraining the $\mathbf{p}_k$ with the buffered hyperplane.

This is a convex quadratic problem with linear constraints. However, since time is continuous, there are infinitely many hyperplanes in the trajectory $\mathcal{H}_{i,j}$, which results in infinitely many constraints in the optimization problem. We solve this problem by using a sequence of hyperplanes between robot $i$ and robot $j$ using a fine discretization of the full trajectory. Whenever robot $i$ detects the geometry of and identifies another robot $j$, we compute a new separating hyperplane between them using $\Omega$ and add it to the separating hyperplane list of robot $i$ against robot $j$. This is an approximation we employ for computational tractability. Whenever robot $i$ receives a planning success signal $(j, \tau)$ from robot $j$, we discard hyperplanes that are generated before $\tau$ from the sequence of robot $i$ against robot $j$.

If the optimization fails for a robot $i$, it keeps using the previous plan. When robot $i$'s planner fails, it does not broadcast a planning success signal, and other robots keep constraining themselves against robot $i$ using the constraints starting from and including the time point robot $i$ last succeeded. Hence, the collisions are avoided even under a planner failure, since at any point of time, plans of any pair of robots are constrained by at least one common separating hyperplane.

### 5.6.3 Experiment Design

We compare the behavior of BVC and AsyncBVC when message drops, delays, re-orderings and time point mismatches are introduced using simulations. We use the RotorS MAV Gazebo simulator [32] running on a desktop computer with 16 core Intel i9-9900 CPU @ 3.10GHz, Ubuntu 20.04 operating system and ROS Noetic. We plan in 3D using AscTec Hummingbird quadrotors already integrated into the RotorS simulator.

In all experiments, $4$ robots are placed in a square formation where they are at $[-10\,0\,5.0]^T$, $[10\,0\,5.0]^T$, $[0\,\,-10\,5.0]^T$, $[0\,10\,5.0]^T$ initially. Robots at the opposite ends of the square swap positions.

The general design of our simulation system is shown in Figure 5.4. Each robot's planning pipeline has 3 main components: a planner, a controller and a localizer/robot detector. The planner is set to either BVC or AsyncBVC. In all experiments, we use spheres with radii $r = 0.4\,\mathrm{m}$ to model robots. The real radius of AscTec Hummingbirds is $0.27\,\mathrm{m}$. We utilize extra $0.13\,\mathrm{m}$ as the buffer zone to compensate for the controller trajectory tracking errors. In all experiments, we use discretized double integrator dynamics during planning for the robots hence control actions $\mathbf{u}_k$ are acceleration commands, and states $\mathbf{x}_k$ are stacked position and velocity vectors in both BVC and AsyncBVC. We set input upper and lower bounds to $\mathbf{u}_{max} = -\mathbf{u}_{min} = [5\,\mathrm{m}/s^2\ 5\,\mathrm{m}/s^2\ 5\,\mathrm{m}/s^2]^T$, state upper bounds to $\mathbf{x}_{max} = [\infty\ \infty\ \infty\ 2\,\mathrm{m}/s\ 2\,\mathrm{m}/s\ 2\,\mathrm{m}/s]^T$ and state lower bounds to $\mathbf{x}_{min} = [-\infty\ -\infty\ r\ -2\,\mathrm{m}/s\ -2\,\mathrm{m}/s\ -2\,\mathrm{m}/s]^T$ i.e. the maximum acceleration along any axis is $5\,\mathrm{m/s^2}$, maximum velocity along any axis is $2\,\mathrm{m/s}$, and all positions except those with $z$ coordinates less than the robot radii $r$ are allowed. We plan for 20 steps with discretization timestep of $0.2\,\mathrm{s}$ (i.e. $4\,\mathrm{s}$ long trajectories). We set $\lambda_k = 2$ for all $k$ and $\theta_k = 1$ for all $k$. The planner sends planned state sequences to the controller for execution. We use one of the controllers integrated into RotorS simulator which is based on [54]. It sends motor commands to the simulator process to track the planned state sequence. The localizer/robot detector component simply receives perfect state and shape information from Gazebo simulator and feeds them to the planner and controller. The frequency of robot detection is 30 Hz, which results in discretization step length of $33\,\mathrm{ms}$ for the separating hyperplane trajectories. Planners on different robots send planning success signals to each other through the signal medium process, which acts as a message broker between planners simulating a communication restrictive environment. BVC does not need communication between planners, therefore signal medium is not utilized when the planner is BVC. When the planner is set to AsyncBVC, it uses the signal medium to broadcast planning success signals described in Section 5.5.3. We introduce artificial communication delays, message drops and message re-orderings to the system using the signal medium process. We model communication delays using exponential probability density functions of the form

$$P_{delay}(x;\alpha) = \begin{cases} \alpha e^{-\alpha x} & x \geq 0 \\ 0 & x < 0 \end{cases},$$

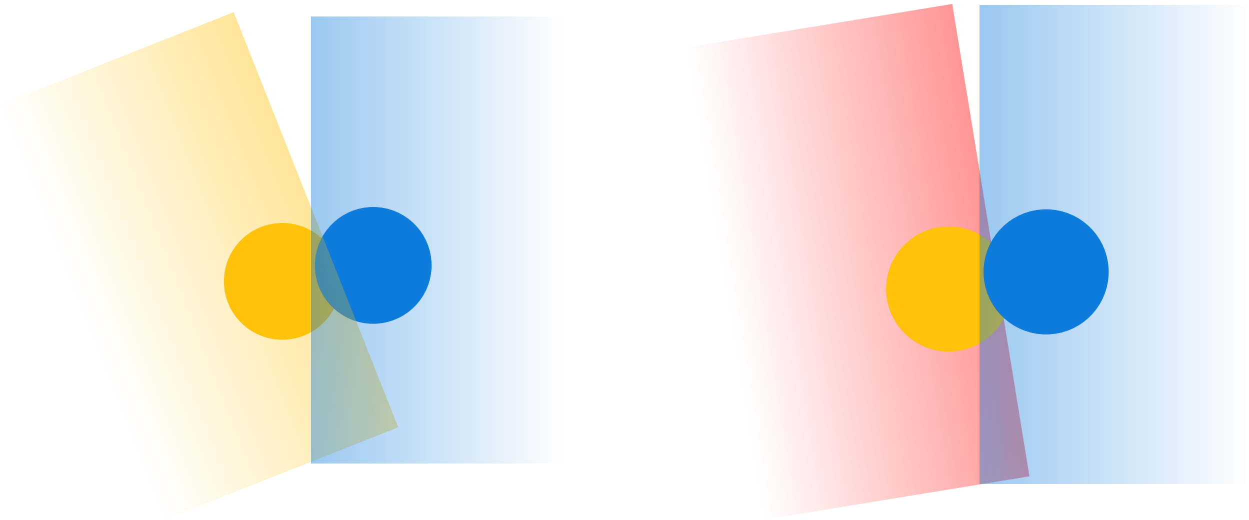


Figure 5.5: **Synchronization-induced deadlock due to separating hyperplane mismatch.** (left) Robots go into a collision state because of a separating hyperplane mismatch and (right) planning fails for the yellow robot because it initially violates the new red hyperplane.

where $1/\alpha \geq 0$ is the mean delay. The most probable delay in this distribution is $0$, and the probability decreases as the delay increases. Message re-orderings are naturally generated by different per-message delays generated from the given delay distribution.

We model message drop probability with a Bernoulli distribution of the form

$$P_{drop}(x;\beta) = \begin{cases} \beta & x \text{ is drop} \\ 1-\beta & x \text{ is no-drop.} \end{cases}$$

### 5.6.4 Evaluation Metrics

We evaluate the performance of algorithms using $4$ metrics: number of robots that are involved in at least one collision during navigation (# Coll.), number of robots that reach their goal positions (# Goal Reaching), number of robots that get stuck in synchronization-induced deadlocks (# Deadlocks), and the maximum navigation time among robots from their start to goal position (Makespan).

Synchronization-induced deadlocks occur because of separating hyperplane mismatches between robots when they are close to each other as shown in Figure 5.5. Since we model robots with spheres larger than

Table 5.1: **Effects of planning start/end time mismatch.** Metrics are averaged over $5$ runs in each scenario.

| | Freq.<br>Max. Start Mismatch | $2\,\mathrm{Hz}$<br>$0.5\,\mathrm{s}$ | $1\,\mathrm{Hz}$<br>$1\,\mathrm{s}$ | $0.5\,\mathrm{Hz}$<br>$2\,\mathrm{s}$ |
|---|---|---|---|---|
| BVC | # Coll. | 0 | 2 | 2.4 |
| | # Deadlocks | 0 | 0.4 | 2.2 |
| | # Goal Reaching | 4 | 3.6 | 1.8 |
| AsyncBVC | # Coll. | 0 | **0** | **0** |
| | # Deadlocks | 0 | **0** | **0** |
| | # Goal Reaching | 4 | **4** | **4** |

the actual robot dimensions (which we call collision shapes), planners plan for avoiding the spheres containing the actual robots. When a separating hyperplane mismatch occurs when robots are close to each other, their collision shapes may intersect with each other even if the real robots do not collide. This creates a situation at which planners cannot find collision free solutions because the robot is initially in a collision state. This results in a deadlock because the planner fails continuously. We call this a synchronization-induced deadlock.

In the simulations, robots that are involved in collisions can continue navigation because the controller can recover from them as the speed of the robots is not high.

### 5.6.5 Effects of Planning Start and End Time Mismatch

First, we check the behavior of BVC and AsyncBVC when the planning start and end times do not match between robots. We change the planning frequency $f$ to create this mismatch. If planning frequency of robots is set to $f$, a planning iteration occurs every $1/f$ seconds. This creates a maximum of $1/f$ seconds mismatch between planning start times of a pair robots. The planning end time is the sum of planning start time and planning duration. There is no bound on planning duration, but it does not take longer than $100\,\mathrm{ms}$ in our experiments, thereby creating a mismatch between planning end times no more than $1/f + 0.1$ seconds. Note that we do not use this value as an upper bound between the time points of two successful planning iterations because there is no guarantee that the planner always succeeds. We thus set the upper bound to $\infty$. We set mean delay time $1/\alpha = 0$ and message drop probability $\beta = 0$ in all experiments, meaning that the network has no delay and does not drop messages, to show the affects of only planning start/end time mismatches.

The results of our comparisons are listed in Table 5.1. We run each experiment $5$ times and report the average of the metrics across runs. We do not report the makespan of navigation between algorithms because the difference is not significant when the communication is instantaneous and failure-free.

In experiment $1$, we set planning frequency to $2$ Hz, which creates a maximum planning start time mismatch of $0.5$ s. In this case, neither BVC nor AsyncBVC results in any synchronization-induced deadlocks or collisions and all robots reach their goal positions. In experiment $2$, we decrease planning frequency to $1$ Hz, which creates a maximum planning start time mismatch of $1$ s. In this case, $2$ out of $4$ robots using BVC collide at least once on average, $0.4$ out of $4$ robots deadlock on average, and $3.6$ out of $4$ robots reach their goals on average. AsyncBVC does not result in any collisions or deadlocks. In experiment $3$, we decrease planning frequency further to $0.5$ Hz, which creates a maximum planning start time mismatch of $2$ s. BVC suffers from this mismatch siginficantly. On average $2.4$ out of $4$ robots collide at least once, and $2.2$ out of $4$ get stuck in synchronization-induced deadlocks, leaving $1.8$ robots on average reaching their goals. AsyncBVC does not result in any collisions or deadlocks.

### 5.6.6 Effects of Best-Effort Communication Medium

Next, we investigate the effects of communication delays and message drops on the performance of AsyncBVC. We do not investigate the behavior of BVC in this set of experiments because BVC does not utilize communication. We set planning frequency $f = 1$ Hz in all experiments which creates a maximum planning start time mismatch of $1$ s. As before, we set upper bound of time difference between successful planning iterations to $\infty$.

Table 5.2: **Effects of communication delays and message drops to AsyncBVC.** Metrics are averaged over 5 runs in each scenario. $1/\alpha$ is the mean delay and $\beta$ is the message drop probability.

| # | $1/\alpha$ | $\beta$ | # Coll. | # Deadlocks | # Goal Reaching | Makespan |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 4 | 20.82 s |
| 2 | 1 s | 0 | 0 | 0 | 4 | 25.57 s |
| 3 | 1 s | 0.1 | 0 | 0 | 4 | 26.68 s |
| 4 | 2 s | 0.1 | 0 | 0 | 4 | 28.31 s |
| 5 | 2 s | 0.2 | 0 | 0 | 4 | 28.92 s |
| 6 | 10 s | 0.2 | 0 | 0 | 4 | 40.88 s |
| 7 | 10 s | 0.5 | 0 | 0 | 4 | 48.04 s |
| 8 | 10 s | 0.75 | 0 | 0 | 4 | 92.26 s |
| 9 | — | 1.0 | 0 | 4 | 0 | $\infty$ |

The results of our experiments are summarized in Table 5.2. As before, we run each experiments 5 times and report the average of the metrics across runs. In experiment 1, we show the performance of AsyncBVC when there is no communication delay, i.e. mean delay is 0, and no message drops, i.e. message drop probability is 0. In this case, no robots collide with each other, and no robots get stuck in deadlocks. The makespan of the navigation, i.e. the maximum navigation duration of all robots, is 20.82 s. From experiment 2 to experiment 8, we increase mean delay from 0 to 10 s and message drop probability from 0 to 0.75. In experiment 8, for example, 3 out of 4 messages get lost in the medium, and 1 that does not get lost arrives 10 s late on average. In all experiments from 2 to 8, no robots collide with each other, and no robots deadlock. Makespan increases from 25.57 s (experiment 2) to 92.26 s (experiment 8).

When the message drop probability is set to 1.0 in experiment 9, i.e. when all messages are dropped, all robots get deadlocked, because they are overly constrained by the full separating hyperplane trajectories. This problem can be solved if there is an upper bound on time difference between two successful planning iterations for the robots in the team as explained in Section 5.5.3.

## 5.7 Conclusion

In this chapter, we present a novel overconstraining and constraint-discarding method for asynchronous real-time decentralized trajectory planning algorithms in multi-robot teams. Each robot overly constrains itself against other robots using mutually computable separating hyperplane trajectories until it receives a planning success signal from them. A portion of the constraints are discarded after a success signal is received from another robot. Our approach is a principled way for dealing with asynchronous planning and imperfect communication in real-time decentralized multi-robot systems.

Based on these ideas, we extend the BVC planner [135], which assumes synchronization in planning, to adapt it to an asynchronous setting. We call our extension AsyncBVC. We compare AsyncBVC with BVC and show that AsyncBVC does not result in any collisions or synchronization-induced deadlocks, while BVC is prone to these phenomena. We also demonstrate the encouraging behavior of AsyncBVC under message drops, message re-orderings and message delays.

# Chapter 6

# Decentralized Real-time Probabilistic Trajectory Planning for Static Obstacle, Interactive Dynamic Obstacle, and Asynchronous Teammate Avoidance

In this chapter, we propose i) a novel prediction representation for interactive behaviors of dynamic obstacles and ii) a decentralized, real-time trajectory planning algorithm for a multi-robot team allowing inter-robot collision and static and interactive dynamic obstacle avoidance. Our planner simulates the behavior of dynamic obstacles during decision-making, explicitly accounting for interactivity. In addition, it explicitly accounts for the perception inaccuracy of static obstacles and the prediction inaccuracy of dynamic obstacles. We handle asynchronous planning between teammates and message delays with unknown bounds, message drops, and re-orderings by using the constraint generation method introduced in Chapter 5. We evaluate our algorithm extensively in simulations under different environments and configurations using 79400 randomly generated runs and compare it against three state-of-the-art baselines using 2100 randomly generated runs, showing that it achieves up to 1.68x success rate using as low as 0.28x time in single-robot experiments, and up to 2.15x success rate using as low as 0.36x time in multi-robot experiments compared to the best baseline. We implement our algorithm on physical quadrotors to

This chapter is based on Baskın Şenbaşlar and Gaurav S Sukhatme. "DREAM: Decentralized Real-time Asynchronous Probabilistic Trajectory Planning for Collision-free Multi-Robot Navigation in Cluttered Environments". In: *arXiv preprint arXiv:2307.15887* (2023). Additional experiments are added to justify the choice of several parameters.

show its applicability to real-world navigation tasks. Supplemental video for this chapter can be found at https://youtu.be/XDcMdMzDBp8.

## 6.1 Introduction

Here, we present DREAM: a decentralized real-time asynchronous probabilistic trajectory planning algorithm for multi-robot teams, each of which using its onboard capabilities to navigate in environments with static and interactive dynamic obstacles as well as teammates (Figure 6.1).

Each robot uses onboard sensing to perceive its environment and classifies objects into three sets: static obstacles, dynamic obstacles, and teammates. It produces a probabilistic representation of static obstacles, in which each static obstacle has an existence probability. Each robot uses an onboard prediction system to predict the behavior of dynamic obstacles and assigns realization probabilities to each behavior. The perception system provides the current shapes of the teammates, i.e., we require geometry-only sensing for teammates and do not require estimation/communication of higher-order derivatives, e.g., velocities or accelerations. Each robot computes discretized separating hyperplane trajectories (DSHTs) introduced in Chapter 5 between itself and teammates, and uses DSHTs during decision making for inter-teammate collision avoidance, allowing safe operation under asynchronous planning and imperfect communication.

Using these uncertain representations of static and dynamic obstacles and DSHTs for teammates, each robot generates dynamically feasible polynomial trajectories in real-time by primarily minimizing probabilities of collisions against static and dynamic obstacles and DSHT violations, while minimizing distance, duration, rotation, and energy usage as secondary objectives using our planner. A DSHT hyperplane is *violated* when the robot is not fully contained in the safe side of it. During decision making, we consider interactive behaviors of dynamic obstacles in response to the actions of the planning robot. The planner runs in a receding horizon fashion, in which the planned trajectory is executed for a short duration and a

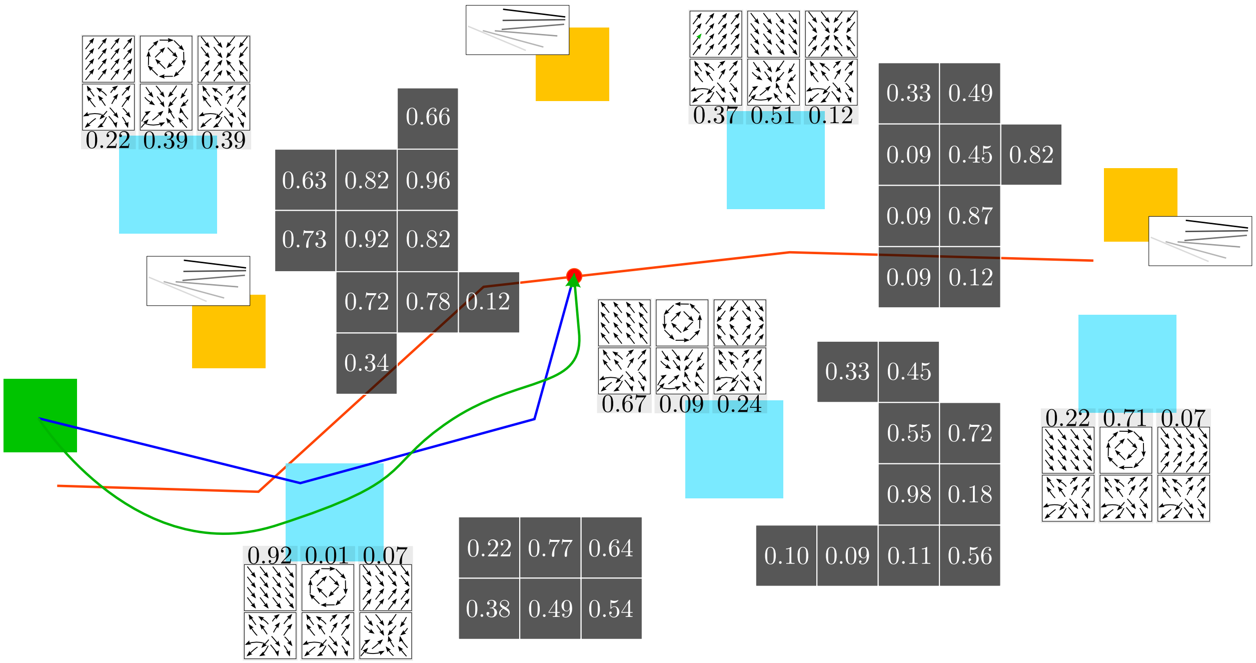


Figure 6.1: Static obstacles (gray) are modeled using their shapes and existence probabilities. Dynamic obstacles (cyan) are modeled using their shapes, current positions, and a probability distribution over their behavior models, each of which comprises a movement and an interaction model. Teammates (orange) are modeled using discretized separating hyperplane trajectories (DSHTs). The planner selects a goal position on a desired trajectory (red), plans a spatiotemporal discrete path (blue) to the goal position minimizing collision probabilities against static and dynamic obstacles and DSHT violations, and solves a quadratic program to fit a smooth trajectory (green) to the discrete plan while preserving the collision probabilities computed and DSHT hyperplanes not violated during search.

new trajectory is planned from scratch. The planner can be guided with desired trajectories, therefore it can be used in conjunction with offline planners that perform longer horizon decision making.

DREAM utilizes a three-stage widely used pipeline [114, 50, 58, 94, 17, 33], differing in specific operations at each stage:

1. **Goal Selection:** Choose a goal position on the desired trajectory to plan to as well as the time point the goal position should be (or should have been) reached at,

2. **Discrete Search:** Find a discrete spatiotemporal path to the goal position that minimizes collision probabilities against static and dynamic obstacles, DSHT violations, and total duration, distance, and the number of rotations,

3. **Trajectory Optimization:** Solve a quadratic program (QP) to safely fit a dynamically feasible trajectory to the discrete plan while preserving i) the collision probabilities computed and ii) DSHT elements not violated during search.

The contributions of this chapter are as follows:

- We define a representation for interactive behaviors of dynamic obstacles that can be used within a planner. We propose three model-based prediction algorithms to predict interactive behavior models of dynamic obstacles.

- We propose a decentralized real-time trajectory planning algorithm for multi-robot navigation in cluttered environments that produces dynamically feasible trajectories avoiding static and (interactive) dynamic obstacles and teammates that plan asynchronously. Our algorithm handles message delays, drops, and re-orderings between teammates. It explicitly accounts for sensing uncertainty with static obstacles and prediction uncertainty with dynamic obstacles.

- We evaluate our algorithm extensively in simulations to show its performance under different environments and configurations using 79400 randomly generated runs. We compare its performance to three state-of-the-art multi-robot navigation decision making algorithms using 2100 randomly generated runs, and show that our algorithm achieves up to 1.68x success rate using as low as 0.28x time in the single-robot case, and 2.15x success rate using as low as 0.36x time in multi-robot scenarios compared to the best baseline. We implement our algorithm for physical quadrotors, and show its feasibility in the real world.

State-of-the-art approaches that predict future trajectories of dynamic obstacles given past observations, potentially in a multi-modal way, use relatively computationally heavy approaches, making them hard to re-query to model interactivity between the robot and dynamic obstacles during decision making. In this chapter, we propose *policies* that are fast to query as prediction outputs instead of future trajectories. Policies model intentions of the dynamic obstacles (movement models) as well as the interaction between dynamic obstacles and the robot (interaction models) as vector fields of velocities.

## 6.2 Problem Definition

Consider a team of $\#^R$ robots. Let $\mathcal{R}_i^{robot} : \mathbb{R}^d \rightarrow P(\mathbb{R}^d)$ be the convex set valued collision shape function of robot $i$, where $i \in \{1, \ldots, \#^R\}$ and $\mathcal{R}_i^{robot}(\mathbf{p})$ is the space occupied by the robot when placed at position $\mathbf{p}$. Here, $d \in \{2, 3\}$ is the ambient dimension that the robots operate in and $P(\mathbb{R}^d)$ is the power set of $\mathbb{R}^d$. We assume that the robots are rigid, and the collision shape functions are defined as $\mathcal{R}_i^{robot}(\mathbf{p}) = \mathcal{R}_{i,\mathbf{0}}^{robot} \oplus \{\mathbf{p}\}$ where $\mathcal{R}_{i,\mathbf{0}}^{robot}$ is the shape of robot $i$ when placed at the origin $\mathbf{0}$ and $\oplus$ is the Minkowski sum operator.

We assume that the robots are differentially flat as defined in Section 3.4, i.e., their states and inputs can be expressed in terms of their output trajectories and finite derivatives of them, and the output trajectory is the Euclidean trajectory that the robot follows. Each robot $i$ requires output trajectory continuity up to degree $c_i$, and has maximum derivative magnitudes $\gamma_i^k$ for derivative degrees $k \in \{1, \ldots, K_i\}$.

Each robot $i$ detects objects and classifies them into three sets: static obstacles $\mathcal{O}_i$, dynamic obstacles $\mathcal{D}_i$, and teammates $\mathcal{C}_i$. Static obstacles do not move. Dynamic obstacles move with or without interaction with the robot. Teammates are objects that navigate using our planner, i.e., the other robots.

Each static obstacle $j \in \mathcal{O}_i$ has a convex shape $\mathcal{Q}_{i,j} \subset \mathbb{R}^d$, and has an existance probability $p^{stat}_{i,j} \in [0, 1]$. Many existing data structures including occupancy grids [38] and octrees [41] support storing obstacles in this form. Each perceived teammate $j \in \mathcal{C}_i$ has a convex shape $\mathcal{S}_{i,j}$ sensed by robot $i$.

Each dynamic obstacle $j \in \mathcal{D}_i$ is modeled using i) its current position $\mathbf{p}^{dyn}_{i,j}$, ii) its convex set valued collision shape function $\mathcal{R}^{dyn}_{i,j} : \mathbb{R}^d \to P(\mathbb{R}^d)$ where $\mathcal{R}^{dyn}_{i,j}(\mathbf{p}) = \mathcal{R}_{i,j,\mathbf{0}} \oplus \{\mathbf{p}\}$ and $\mathcal{R}_{i,j,\mathbf{0}}$ is the shape of obstacle $j$ when placed at the origin, and iii) a probability distribution over its $\#^B_{i,j}$ predicted behavior models $\mathcal{B}_{i,j,k}$, $k \in \{1, \ldots, \#^B_{i,j}\}$, where each behavior model is a 2-tuple $\mathcal{B}_{i,j,k} = (\mathcal{M}_{i,j,k}, \mathcal{I}_{i,j,k})$ such that $\mathcal{M}_{i,j,k}$ is the movement and $\mathcal{I}_{i,j,k}$ is the interaction model of the dynamic obstacle. $p^{dyn}_{i,j,k}$ is the probability that dynamic obstacle $j$ moves according to behavior model $\mathcal{B}_{i,j,k}$ such that $\sum_{k=1}^{\#^B_{i,j}} p^{dyn}_{i,j,k} \leq 1$ for all $j \in \mathcal{D}_i$.

A movement model $\mathcal{M} : \mathbb{R}^d \to \mathbb{R}^d$ is a function from dynamic obstacle's position to its desired velocity, describing its intention. An interaction model $\mathcal{I} : \mathbb{R}^{4d} \to \mathbb{R}^d$ is a function describing robot-dynamic obstacle interaction of the form $\mathbf{v}^{dyn} = \mathcal{I}(\mathbf{p}^{dyn}, \tilde{\mathbf{v}}^{dyn}, \mathbf{p}^{robot}, \mathbf{v}^{robot})$. Its arguments are 4 vectors: position $\mathbf{p}^{dyn}$ of the dynamic obstacle, desired velocity $\tilde{\mathbf{v}}^{dyn}$ of the dynamic obstacle (which is obtained from the movement model, i.e., $\tilde{\mathbf{v}}^{dyn} = M(\mathbf{p}^{dyn})$), and position $\mathbf{p}^{robot}$ and velocity $\mathbf{v}^{robot}$ of robot. It outputs the velocity $\mathbf{v}^{dyn}$ of the dynamic obstacle. Notice that interaction models do not model interactions between multiple dynamic obstacles or interactions with multiple teammates, i.e., the velocity $\mathbf{v}^{dyn}$ a dynamic obstacle executes does not depend on the position or velocity of other dynamic obstacles or the other teammates from the perspective of a single teammate. This is an accurate representation in sparse environments where moving objects are not in close proximity of each other, but an inaccurate assumption in dense environments. We choose to model interactions this way for computational efficiency as well as non-reliance on perfect communication: modeling interactions between multiple dynamic dynamic obstacles would result in a combinatorial explosion of possible dynamic obstacle behaviors since we support multiple hypotheses for each dynamic obstacle, and modeling interactions of dynamic obstacles with multiple teammates would require joint planning for all robots, requiring perfect communication[*]. While using only position and velocity to model robot-dynamic obstacle interaction is an approximation of the reality, we choose this model because of its simplicity. This simplification allows us to use interaction models to update the behavior of dynamic obstacles during discrete search efficiently[†].

Each robot $i$ has a state estimator that estimates its output derivatives up to derivative degree $c_i$, where degree $0$ corresponds to position, degree $1$ corresponds to velocity, and so on. If state estimation accuracy is low, the trajectories computed by the planner can be used to compute the expected derivatives in an open-loop fashion assuming perfect execution. The $k^{th}$ derivative of robot $i$'s current position is denoted with $\mathbf{p}_{i,k}^{self}$ where $k \in \{0, \ldots, c_i\}$.

Each robot $i$ is tasked with following a desired trajectory $\mathbf{d}_i(t) : [0, T_i] \rightarrow \mathbb{R}^d$ with duration $T_i$ without colliding with obstacles or other robots. The desired trajectory $\mathbf{d}_i(t)$ can be computed by a global planner using potentially incomplete prior knowledge about obstacles. It does not need to be collision-free with respect to static or dynamic obstacles. If no such global planner exists, it can be set to a straight line from a start position to a goal position.

## 6.3 Approach

In order to follow the desired trajectories $\mathbf{d}_i$ as closely as possible while avoiding collisions, we propose a decentralized real-time planner executed in a receding horizon fashion.

[*] One could also define a single joint interaction model for all dynamic obstacles and do non-probabilistic decision making against dynamic obstacles if inter-dynamic obstacle interactions exists and single dynamic obstacle models are insufficient at describing the dynamic obstacle behaviors.

[†] During planning, we evaluate movement and interaction models sequentially to compute the velocity of the dynamic obstacles. One could also combine movement and interaction models, and have a single function to describe the dynamic obstacle behavior for the purposes of our planner. We choose to model them separately in order to allow separate prediction of these models.

### 6.3.1 Discretized Separating Hyperplane Trajectories (DSHTs)

We utilize DSHTs (Chapter 5) as constraints, which allows us to enforce safety when planning is asynchronous, i.e., robots start and end planning at different time points, and the communication medium is imperfect. We briefly reiterate the theory behind DSHTs next.

Let $\Omega$ be a commutative deterministic separating hyperplane computation algorithm: it computes a separating hyperplane between two linearly separable sets and each call to it with the same pair of arguments results in the same hyperplane. We use hard-margin support vector machines (SVM) as $\Omega$.

Let $\mathbf{f}_i(t) : [0, T_{cur}] \rightarrow \mathbb{R}^d$ and $\mathbf{f}_j(t) : [0, T_{cur}] \rightarrow \mathbb{R}^d$ be the trajectories robots $i$ and $j$ executed from navigation start time $0$ to current time $T_{cur}$, respectively. Separating hyperplane trajectory $\mathcal{H}_{i,j} : [0, T_{cur}] \rightarrow \mathcal{H}^d$ between robots $i$ and $j$ induced by $\Omega$ is defined as $\mathcal{H}_{i,j}(t) = \Omega(\mathcal{R}_i^{robot}(\mathbf{f}_i(t)), \mathcal{R}_j^{robot}(\mathbf{f}_j(t)))$ where $\mathcal{H}^d$ is the set of all hyperplanes in $\mathbb{R}^d$.

Each robot $i$ stores a tail time point variable $T_{i,j}^{tail} \leq T_{cur}$ for each other robot $j$ denoting the time point after which the hyperplanes in $\mathcal{H}_{i,j}$ should be used to constrain robot $i$'s plan against robot $j$. If robot $i$ starts planning at $T_{cur}$, it uses all hyperplanes $\mathcal{H}_{i,j}(t)$ where $t \in [T_{i,j}^{tail}, T_{cur}]$ to constrain itself against robot $j$ by enforcing its trajectory to be in the safe side of each hyperplane. When robot $i$ successfully finishes a planning iteration that started at $T_{i,start}$, meaning that it is now constrained by hyperplanes from $T_{i,j}^{tail}$ to $T_{i,start}$ on $\mathcal{H}_{i,j}$ against each other robot $j$, it broadcasts its identity $i$ and $T_{i,start}$. Robots $j$ receiving the message update their tail time points against robot $i$ by setting $T_{j,i}^{tail} = T_{i,start}$, discarding constraints, and those that do not receive it do not update their tail points, over-constraining themselves against robot $i$. As shown in Chapter 5, this constraint discarding and over-constraining mechanism ensures that active trajectories of each pair of robots share a constraining hyperplane at all times under asynchronous planning, message delays, drops and re-orderings.

Let $\mathcal{H}_{i,j}^{active} = \{\mathcal{H}_{i,j}(t) \mid t \in [T_{i,j}^{tail}, T_{cur}]\}$ be the active set of separating hyperplanes of robot $i$ against robot $j$. There are infinitely many hyperplanes in $\mathcal{H}_{i,j}^{active}$ when $T_{i,j}^{tail} < T_{cur}$. We sample hyperplanes in

$\mathcal{H}_{i,j}^{active}$ using a sampling step in the time domain shared among all teammates. Let $\tilde{\mathcal{H}}_{i,j}^{active}$, which is the active DSHT of robot $i$ against robot $j$, be the finite sampling of $\mathcal{H}_{i,j}^{active}$, and $\tilde{\mathcal{H}}_{i}^{active} = \{\mathcal{H} \in \tilde{\mathcal{H}}_{i,j}^{active} \mid j \in \{1, \dots, \#^R\} \setminus \{i\}\}$ be the set of all hyperplanes that should constraint robot $i$ on a planning iteration that starts at time $T_{cur}$. We use hyperplanes $\tilde{\mathcal{H}}_{i}^{active}$ during planning to enforce safety against robot teammates.

It is assumed that perception, prediction, and state estimation systems are executed independently from the planner and produce the information described in Section 6.2. DSHT computation is done asynchronously and independently from the planner, maintaining tail time points $T_{i,j}$ and providing $\tilde{\mathcal{H}}_{i}^{active}$ to the planner. The inputs from these systems to the planner in robot $i$ are:

- **Static obstacles**: Convex shapes $\mathcal{Q}_{i,j}$ with their existence probabilities such that $p_{i,j}^{stat}$ is the probability that obstacle $j \in \mathcal{O}_i$ exists.
- **Dynamic obstacles**: Set $\mathcal{D}_i$ of dynamic obstacles where each dynamic obstacle $j \in \mathcal{D}_i$ has the current position $\mathbf{p}_{i,j}^{dyn}$, collision shape function $\mathcal{R}_{i,j}^{dyn}$, and behavior models $\mathcal{B}_{i,j,k}$ with corresponding realization probabilities $p_{i,j,k}^{dyn}$ where $k \in \{1, \dots, \#_{i,j}^{B}\}$.
- **Active DSHTs**: Set $\tilde{H}_i^{active}$ of separating hyperplanes against all other robots.
- **Self state**: The state $\{\mathbf{p}_{i,0}^{self}, \dots, \mathbf{p}_{i,c_i}^{self}\}$ of the robot.

There are three stages of our algorithm: i) goal selection, which selects a goal position on the desired trajectory to plan to, ii) discrete search, which computes a spatiotemporal discrete path to the goal position, minimizing collision probabilities against two classes of obstacles, DSHT violations, distance, duration, and rotations using a multi-objective search method, and iii) trajectory optimization, which safely computes a dynamically feasible trajectory by smoothing the discrete path while preserving the collision probabilities computed and DSHT hyperplanes not violated during the search. The planner might fail during trajectory optimization, the reasons for which are described in Section 6.3.4. If planning fails, the robot continues using its previous plan.

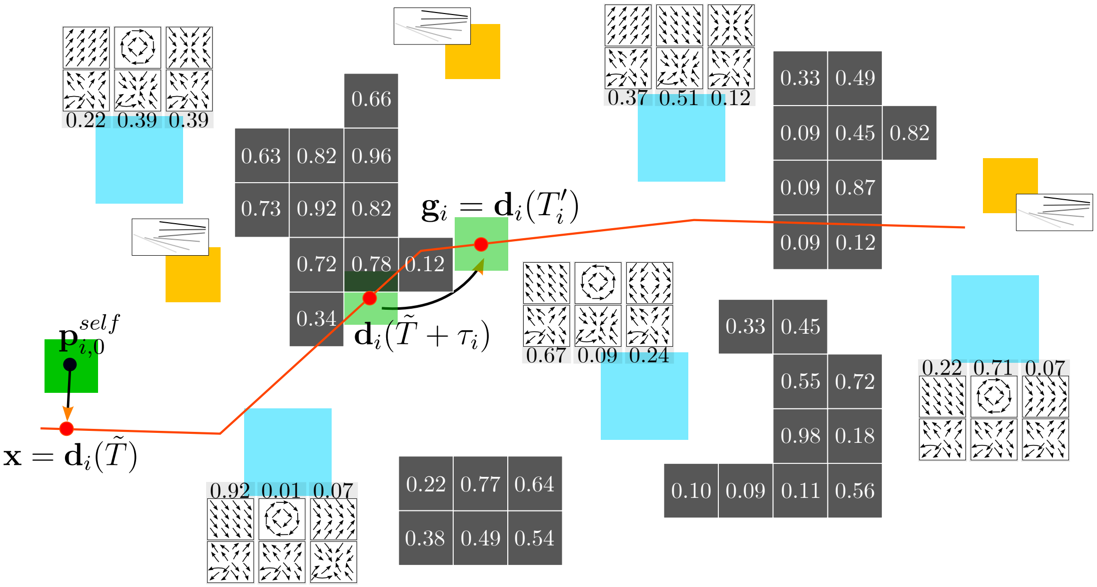


Figure 6.2: **Goal selection.** The goal selection stage selects the goal position $\mathbf{g}_i$ to plan to on the desired trajectory $\mathbf{d}_i$ (red) and the time point $T'_i$ it should be (or should have been) reached at. It finds the closest point $\mathbf{x}$ on $\mathbf{d}_i$ to the current robot position $\mathbf{p}^{self}_{i,0}$ and its time point $\tilde{T}$, and finds the smallest time point $T'_i$ that is greater than the time point that is one desired time horizon away from $\tilde{T}$, i.e., $\tilde{T} + \tau_i$, at which the robot is collision free against all static obstacles with existence probability greater than $p^{min}_i$.

### 6.3.2 Goal Selection

In the goal selection stage (Figure 6.2), each robot $i$ chooses a goal position $\mathbf{g}_i$ on the desired trajectory $\mathbf{d}_i$ and the time point $T'_i$ at which $\mathbf{g}_i$ should be (or should have been) reached at.

This stage has two parameters: the desired time horizon $\tau_i$ and the static obstacle existence probability threshold $p^{min}_i$.

First, the goal selection finds the closest point $\mathbf{x}$ on the desired trajectory $\mathbf{d}_i$ to the robot's current position $\mathbf{p}^{self}_{i,0}$, by discretizing $\mathbf{d}_i$. Let $\tilde{T}$ be the time point of $\mathbf{x}$ on $\mathbf{d}_i$, i.e., $\mathbf{x} = \mathbf{d}_i(\tilde{T})$. Then, goal selection finds the smallest time point $T'_i \in [\min(\tilde{T} + \tau_i, T_i), T_i]$ on $\mathbf{d}_i$ such that the robot is collision-free against static obstacles with existence probabilities at least $p^{min}_i$ when placed on $\mathbf{d}_i(T'_i)$ using collision checks with small increments in time. The goal position $\mathbf{g}_i$ is set to $\mathbf{d}_i(T'_i)$ and the time point it should be (or should have been) reached at is $T'_i$. We assume that the robot placed at $\mathbf{d}_i(T_i)$ is collision-free against static obstacles, hence such a $T'_i$ always exists.

The selected goal position $\mathbf{g}_i$ and the time point $T'_i$ are used during the discrete search stage, which uses a goal-directed search algorithm. Note that goal selection chooses the goal position on the desired trajectory without considering its reachability; the actual trajectory the robot follows is planned by the rest of the algorithm.

### 6.3.3 Discrete Search

In the discrete search stage, we plan a path to the goal position $\mathbf{g}_i$ using cost algebraic A* search [28]. Cost algebraic A* is a generalization of standard A* to a richer set of cost systems, namely cost algebras. We compute six cost terms during search, define the cost of an action as the vector of the computed cost terms, and order the cost vectors lexicographically, forming a cost algebra. Cost algebraic A* finds an optimal action sequence according to the lexicographical ordering of our cost vectors.

Here, we summarize the formalism of cost algebras from the original paper [28]. The reader is advised to refer to the original paper for a detailed and complete description of concepts.

**Definition 11.** *Let $A$ be a set and $\times : A \times A \to A$ be a binary operator. A monoid is a tuple $(A, \times, \mathbf{0})$ if the identity element $\mathbf{0} \in A$ exists, $\times$ is associative and $A$ is closed under $\times$.*

**Definition 12.** *Let $A$ be a set. A relation $\preceq \subseteq A \times A$ is a total order if it is reflexive, anti-symmetric, transitive, and total. The least operation $\sqcup$ gives the least element of the set according to a total order, i.e., $\sqcup A = c \in A$ such that $c \preceq a \; \forall a \in A$, and the greatest operation $\sqcap$ gives the greatest element of the set according to the total order, i.e. $\sqcap A = c \in A$ such that $a \preceq c \; \forall a \in A$.*

**Definition 13.** *A set $A$ is isotone if $a \preceq b$ implies both $a \times c \preceq b \times c$ and $c \times a \preceq c \times b$ for all $a, b, c \in A$. $a \prec b$ is defined as $a \preceq b \wedge a \neq b$. A set $A$ is strictly isotone if $a \prec b$ implies both $a \times c \prec b \times c$ and $c \times a \prec c \times b$ for all $a, b, c \in A, c \neq \mathbf{1}$ where $\mathbf{1} = \sqcap A$.*

**Definition 14.** *A cost algebra is a 6-tuple $(A, \sqcup, \times, \preceq, \mathbf{1}, \mathbf{0})$ such that $(A, \times, \mathbf{0})$ is a monoid, $\preceq$ is a total order, $\sqcup$ is the least operation induced by $\preceq$, $\mathbf{1} = \sqcap A$, and $\mathbf{0} = \sqcup A$, i.e. the identity element is the least element.*

Intuitively, $A$ is the set of cost values, $\sqcup$ is the operation used to select the best among the values, $\times$ is the operation to cumulate the cost values, $\preceq$ is the operator to compare the cost values, $\mathbf{1}$ is the greatest and $\mathbf{0}$ is the least cost value as well as the identity cost value under $\times$.

To support multiple objectives during search, the prioritized Cartesian product of cost algebras is defined as follows.

**Definition 15.** *The prioritized Cartesian product of cost algebras $C_1 = (A_1, \sqcup_1, \times_1, \preceq_1, \mathbf{1}_1, \mathbf{0}_1)$ and $C_2 = (A_2, \sqcup_2, \times_2, \preceq_2, \mathbf{1}_2, \mathbf{0}_2)$, denoted by $C_1 \times_p C_2$ is a tuple $(A_1 \times A_2, \sqcup, \times, \preceq, (\mathbf{1}_1, \mathbf{1}_2), (\mathbf{0}_1, \mathbf{0}_2))$ where $(a_1, a_2) \times (b_1, b_2) = (a_1 \times_1 b_1, a_2 \times_2 b_2)$, $(a_1, a_2) \preceq (b_1, b_2)$ iff $a_1 \prec_1 b_1 \vee (a_1 = b_1 \wedge a_2 \preceq_2 b_2)$, and $\sqcup$ is induced by $\preceq$.*

Note that, $\preceq$ in Definition 15 induces lexicographical ordering among cost algebras $C_1$ and $C_2$.

**Proposition 1.** *If $C_1$ and $C_2$ are cost algebras, and $C_1$ is strictly isotone, then $C_1 \times_p C_2$ is also a cost algebra. If, in addition, $C_2$ is strictly isotone, $C_1 \times_p C_2$ is also strictly isotone.*

*Proof.* Given in [28]. □

Proposition 1 allows one to take Cartesian product of any number of strictly isotone cost algebras and end up with a strictly isotone cost algebra.

Given a cost algebra $C = (A, \sqcup, \times, \preceq, \mathbf{1}, \mathbf{0})$, cost algebraic A* finds a lowest cost path according to $\sqcup$ between two nodes in a graph where edge costs are elements of set $A$, which are ordered according to $\preceq$ and combined with $\times$ where the lowest cost value is $\mathbf{0}$ and the largest cost value is $\mathbf{1}$. Cost algebraic A* uses a heuristic for each node of the graph, and cost algebraic A* with re-openings finds cost optimal paths only if the heuristic is admissible. An admissible heuristic for a node is a cost $h \in A$, which underestimates the cost of the lowest cost path from the node to the goal node according to $\preceq$.

We conduct a multi-objective search, in which each individual cost term is a strictly isotone cost algebra, and optimize over their Cartesian product. The individual cost terms are defined over two cost algebras, namely $(\mathbb{R}_{\geq 0} \cup \{\infty\}, min, +, \leq, \infty, 0)$, i.e. non-negative real number costs with standard addition and comparison, and $(\mathbb{N} \cup \{\infty\}, min, +, \leq, \infty, 0)$, natural numbers with standard addition and comparison, both of which are strictly isotone. Therefore, any number of their Cartesian products are also cost algebras by Proposition 1.

We explain the discrete search for an arbitrary robot $i$ in the team, each robot runs the same algorithm. The planning horizon of the search is $\tau'_i = \max(\tilde{\tau}_i, T'_i - T_{cur}, \alpha_i \frac{\left\|\mathbf{p}^{self}_{i,0} - \mathbf{g}_i\right\|_2}{\tilde{\gamma}^1_i})$ where $\tilde{\tau}_i$ is the minimum search horizon and $\tilde{\gamma}^1_i$ is the *maximum speed for search* parameter. In other words, the planning horizon is set to the maximum of minimum search horizon, time difference between the goal time point and the current time point, and a multiple of the minimum required time to reach to the goal position $\mathbf{g}_i$ from current position $\mathbf{p}^{self}_{i,0}$ applying maximum speed $\tilde{\gamma}^1_i$ where multiplier $\alpha_i \geq 1$. The planning horizon $\tau'_i$ is used as a suggestion in the search, and is exceeded if necessary as explained later in this section.

**States.** The states $x$ in our search formulation have six components: i) $x.\mathbf{p} \in \mathbb{R}^d$ is the position of the state, ii) $x.\mathbf{\Delta} \in \{-1, 0, 1\}^d \setminus \{\mathbf{0}\}$ is the direction of the state on a grid oriented along robot's current velocity $\mathbf{p}^{self}_{i,1}$ with a rotation matrix $R_{rot} \in SO(d)$ such that $R_{rot}(1, 0, \dots, 0)^\top = \frac{\mathbf{p}^{self}_{i,1}}{\left\|\mathbf{p}^{self}_{i,1}\right\|_2}$, iii) $x.t \in [0, \infty)$ is the time of the state, iv) $x.\mathcal{O} \subseteq \mathcal{O}_i$ is the set of static obstacles that collide with the robot $i$ following the path from start state to $x$, v) $x.\mathcal{D}$ is the set of dynamic obstacle behavior model–position pairs $(\mathcal{B}_{i,j,k}, \mathbf{p}^{dyn}_{i,j,k})$ such that dynamic obstacle $j$ moving according to $\mathcal{B}_{i,j,k}$ does not collide the robot $i$ following the path from start state to $x$, and the dynamic obstacle ends up at position $\mathbf{p}^{dyn}_{i,j,k}$, and vi) $x.\mathcal{H} \subseteq \tilde{\mathcal{H}}^{active}_i$ is the set of active DSHT hyperplanes that the robot $i$ violates following the path from start state to $x$.

The start state of the search is $x^1$ with components $x^1.\mathbf{p} = \mathbf{p}^{self}_{i,0}$, $x^1.\mathbf{\Delta} = (1, 0, \ldots, 0)^\top$, $x^1.t = 0$, $x^1.\mathcal{O}$ are set of all obstacles that intersect with $\mathcal{R}^{robot}_i(\mathbf{p}^{self}_{i,0})$, $x^1.\mathcal{D}$ contains behavior model–position pairs $(\mathcal{B}_{i,j,k}, \mathbf{p}^{dyn}_{i,j})$ of dynamic obstacles $j$ that do not initially collide with robot, i.e. $\mathcal{R}^{robot}_i(\mathbf{p}^{self}_{i,0}) \cap \mathcal{R}^{dyn}_{i,j}(\mathbf{p}^{dyn}_{i,j}) = \emptyset$, one for each $k \in \{1, \ldots, \#^B_{i,j}\}$, and $x^1.\mathcal{H}$ contains all hyperplanes in $\tilde{\mathcal{H}}^{active}_i$ that the robot $i$ violates initially at $\mathbf{p}^{self}_{i,0}$. The goal states are all states $x^g$ with position $x^g.\mathbf{p} = \mathbf{g}_i$.

**Actions.** There are three action types in our search. Let $x$ be the current state and $x^+$ be the state after applying an action.

- FORWARD$(s, t)$ moves the current state $x$ to $x^+$ by applying constant speed $s$ along current direction $x.\mathbf{\Delta}$ for time $t$. The state components change as follows.
  - $x^+.\mathbf{p} = x.\mathbf{p} + R_{rot}\frac{x.\mathbf{\Delta}}{\|x.\mathbf{\Delta}\|_2} st$
  - $x^+.\mathbf{\Delta} = x.\mathbf{\Delta}$
  - $x^+.t = x.t + t$.
  - We compute static obstacles $\mathcal{O}^+$ colliding with the robot with shape $\mathcal{R}^{robot}_i$ travelling from $x.\mathbf{p}$ to $x^+.\mathbf{p}$ and set $x^+.\mathcal{O} = x.\mathcal{O} \cup \mathcal{O}^+$.
  - Let $(\mathcal{B}_{i,j,k}, \mathbf{p}^{dyn}_{i,j,k}) \in x.\mathcal{D}$ be a dynamic obstacle behavior model–position pair that does not collide with the state sequence from the start state to $x$. Note that robot applies velocity $\mathbf{v} = \frac{x^+.\mathbf{p} - x.\mathbf{p}}{x^+.t - x.t}$ from state $x$ to $x^+$. We get the desired velocity $\tilde{\mathbf{v}}^{dyn}_{i,j,k}$ of the dynamic obstacle at time $x.t$ using its movement model: $\tilde{\mathbf{v}}^{dyn}_{i,j,k} = \mathcal{M}_{i,j,k}(\mathbf{p}^{dyn}_{i,j,k})$. The velocity $\mathbf{v}^{dyn}_{i,j,k}$ of the dynamic obstacle can be computed using the interaction model: $\mathbf{v}^{dyn}_{i,j,k} = \mathcal{I}_{i,j,k}(\mathbf{p}^{dyn}_{i,j,k}, \tilde{\mathbf{v}}^{dyn}_{i,j,k}, x.\mathbf{p}, \mathbf{v})$. We check whether the dynamic obstacle shape $\mathcal{R}^{dyn}_{i,j}$ swept between $\mathbf{p}^{dyn}_{i,j,k}$ and $\mathbf{p}^{dyn}_{i,j,k} + \mathbf{v}^{dyn}_{i,j,k}t$ collides with robot shape $\mathcal{R}^{robot}_i$ swept between $x.\mathbf{p}$ and $x^+.\mathbf{p}$. If not, we add not colliding dynamic obstacle behavior model by $x^+.\mathcal{D} = x^+.\mathcal{D} \cup \{(\mathcal{B}_{i,j,k}, \mathbf{p}^{dyn}_{i,j,k} + \mathbf{v}^{dyn}_{i,j,k}t)\}$. Otherwise, we discard the behavior model.

  - We compute the hyperplanes $\mathcal{H}^+ \subseteq \tilde{\mathcal{H}}_i^{active}$ the robot $i$ violates at $x^+.\mathbf{p}$, and set $x^+.\mathcal{H} = x.\mathcal{H} \cup \mathcal{H}^+$.

- ROTATE($\mathbf{\Delta}'$) changes the current state $x$ to $x^+$ by changing its direction to $\mathbf{\Delta}'$. It is only available if $x.\mathbf{\Delta} \neq \mathbf{\Delta}'$. The rotate action is added to penalize turns during search as discussed in the description of costs. The state components remains the same except $x^+.\mathbf{\Delta}$ is set to $\mathbf{\Delta}'$.

- REACHGOAL changes the current state $x$ to $x^+$ by connecting $x.\mathbf{p}$ to the goal position $\mathbf{g}_i$. The remaining search horizon for the robot to reach its goal position is given by $\tau_i' - x.t$. Recall that the maximum speed of the robot during search is $\tilde{\gamma}_i^1$, hence the robot needs at least $\frac{\|\mathbf{g}_i - x.\mathbf{p}\|_2}{\tilde{\gamma}_i^1}$ seconds to reach the goal position from state $x$. We set the duration of this REACHGOAL action to the maximum of these two values: $\max(\tau_i' - x.t, \frac{\|\mathbf{g}_i - x.\mathbf{p}\|_2}{\tilde{\gamma}_i^1})$. Therefore, the search horizon $\tau_i'$ is merely a suggestion during search, and is exceeded whenever it is not dynamically feasible to reach the goal position within the search horizon. The state components change as follows.

  - $x^+.\mathbf{p} = \mathbf{g}_i$
  - $x^+.\mathbf{\Delta} = x.\mathbf{\Delta}$
  - $x^+.t = x.t + \max(\tau_i' - x.t, \frac{\|\mathbf{g}_i - x.\mathbf{p}\|_2}{\tilde{\gamma}_i^1})$
  - $x^+.\mathcal{O}$, $x^+.\mathcal{D}$, and $x^+.\mathcal{H}$ are computed in the same way as FORWARD.

Note that we run interaction models only when robot applies a time changing action (FORWARD or REACHGOAL), which is an approximation of reality because dynamic objects can potentially change their velocities between robot actions. We also conduct *conservative collision checks* against dynamic obstacles because we do not include the time domain to the collision check. This conservatism allows us to preserve collision probability upper bounds against dynamic obstacles during trajectory optimization as discussed in Section 6.3.4.

We compute the probability of not colliding with static obstacles and a lower bound on the probability of not colliding with dynamic obstacles for each state of the search tree recursively. We interleave the computation of sets $x.\mathcal{O}$ and $x.\mathcal{D}$ with the probability computation.

#### 6.3.3.1 Computing the Probability of Not Colliding With Static Obstacles

Let $x^{1:n} = x^1, \ldots, x^n$ be a state sequence in the search tree. Let $\mathcal{C}_s(x^{l:m})$ be the proposition that is true if and only if the robot following timed path $(x^l.\mathbf{p}, x^l.t), \ldots, (x^m.\mathbf{p}, x^m.t)$ collides with any of the static obstacles in $\mathcal{O}_i$, where $m \geq l$. The event of not colliding with any of the static obstacles while following a prefix of state sequence $x^{1:n}$ admits a recursive definition: $\neg\, \mathcal{C}_s(x^{1:l}) = \neg\, \mathcal{C}_s(x^{1:m}) \bigwedge \neg\, \mathcal{C}_s(x^{m:l})\ \forall l \in \{1, \ldots, n\}\ \forall m \in \{1, \ldots, l\}$.

We compute the probability of not colliding any of the static obstacles for each prefix of the state sequence $x^{1:n}$ during search, and store this probability as a metadata of each state. Probability $p(\neg\, \mathcal{C}_s(x^{1:l}))$ of not colliding with static obstacles $\mathcal{O}_i$ while traversing the state sequence $x^{1:l}$ is given by:

$$\begin{aligned} p(\neg\, \mathcal{C}_s(x^{1:l})) &= p(\neg\, \mathcal{C}_s(x^{1:l-1}) \wedge \neg\, \mathcal{C}_s(x^{l-1:l})) \\ &= p(\neg\, \mathcal{C}_s(x^{1:l-1}))p(\neg\, \mathcal{C}_s(x^{l-1:l}) \mid \neg\, \mathcal{C}_s(x^{1:l-1})) \end{aligned}$$

The first term $p(\neg\, \mathcal{C}_s(x^{1:l-1}))$ is the recursive term that can be obtained from the parent state during search.

The second term $p(\neg\, \mathcal{C}_s(x^{l-1:l})|\neg\, \mathcal{C}_s(x^{1:l-1}))$ is the conditional term that we compute during state expansion. Let $\mathcal{O}_i^{l:m} \subseteq \mathcal{O}_i$ be the set of static obstacles that collides with robot $i$ traversing $x^{l:m}$ where $l \leq m$. Given that the robot has not collided while traversing $x^{1:l-1}$ means that no static obstacle that collides with the robot while traversing $x^{1:l-1}$ exists. Therefore, we compute the conditional probability as the probability that none of the obstacles in $\mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}$ exists as ones in $\mathcal{O}_i^{l-1:l} \cap \mathcal{O}_i^{1:l-1}$ do not exist as presumption. We assume that static obstacles' non-existence events are independent. Let $E(j)$ be the event that static obstacle $j \in \mathcal{O}_i$ exists. We have

$$\begin{aligned}
p(\neg\, \mathcal{C}_s(x^{l-1:l}) \mid \neg\, \mathcal{C}_s(x^{1:l-1})) &= p\left(\bigwedge_{j \in \mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}} \neg\, E(j)\right) \\
&= \prod_{j \in \mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}} p(\neg E(j)) \\
&= \prod_{j \in \mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}} (1 - p_{i,j}^{stat})
\end{aligned}$$

The key operation for computing the conditional is computing the set $\mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}$. We do this in a time efficient way by using the fact that $x^l.\mathcal{O} = \mathcal{O}_i^{1:l}$. During node expansion, we compute $\mathcal{O}_i^{l-1:l}$ by querying the static obstacles for collisions against the region swept by $\mathcal{R}_i^{robot}$ from position $x^{l-1}.\mathbf{p}$ to $x^l.\mathbf{p}$. We obtain $\mathcal{O}_i^{1:l-1}$ from the parent state's $x^{l-1}.\mathcal{O}$. The probability of not colliding is computed according to obstacles in $\mathcal{O}_i^{l-1:l} \setminus \mathcal{O}_i^{1:l-1}$ and $x^l.\mathcal{O}$ is set to $\mathcal{O}_i^{1:l} = \mathcal{O}_i^{1:l-1} \cup \mathcal{O}_i^{l-1:l}$ as described before.

The recursive term $p(\neg\, \mathcal{C}_s(x^{1:1}))$ is initialized for the start state $x^1$ by computing the non-existence probability of obstacles in $x^1.\mathcal{O}$, i.e., $p(\neg\, \mathcal{C}_s(x^{1:1})) = \prod_{j \in x^1.\mathcal{O}} (1 - p_{i,j}^{stat})$.

#### 6.3.3.2 Computing a Lower Bound on the Probability of not Colliding with Dynamic Obstacles

A lower bound on the probability of not colliding computation with dynamic obstacles are done using not colliding dynamic obstacle behavior model–position pairs computed during state expansion.

Let $C_d(x^{l:m})$ be the proposition, conditioned on the full state sequence $x^{1:n}$, that is true if and only if the robot following the $(x^l.\mathbf{p}, x^l.t), \ldots, (x^m.\mathbf{p}, x^m.t)$ portion of $x^{1:n}$ collides with any of the dynamic obstacles in $\mathcal{D}_i$ where $l \leq m$. Similar to the static obstacles, the event of not colliding with any of the dynamic obstacles while following a prefix of the path $x^{1:n}$ is recursive: $\neg\, C_d(x^{1:l}) = \neg\, C_d(x^{1:m}) \bigwedge \neg\, C_d(x^{m:l})\; \forall l \in \{1, \ldots, n\}\; \forall m \in \{1, \ldots, l\}$.

The formulation of the probability of not colliding with dynamic obstacles is identical to that of the static obstacles:

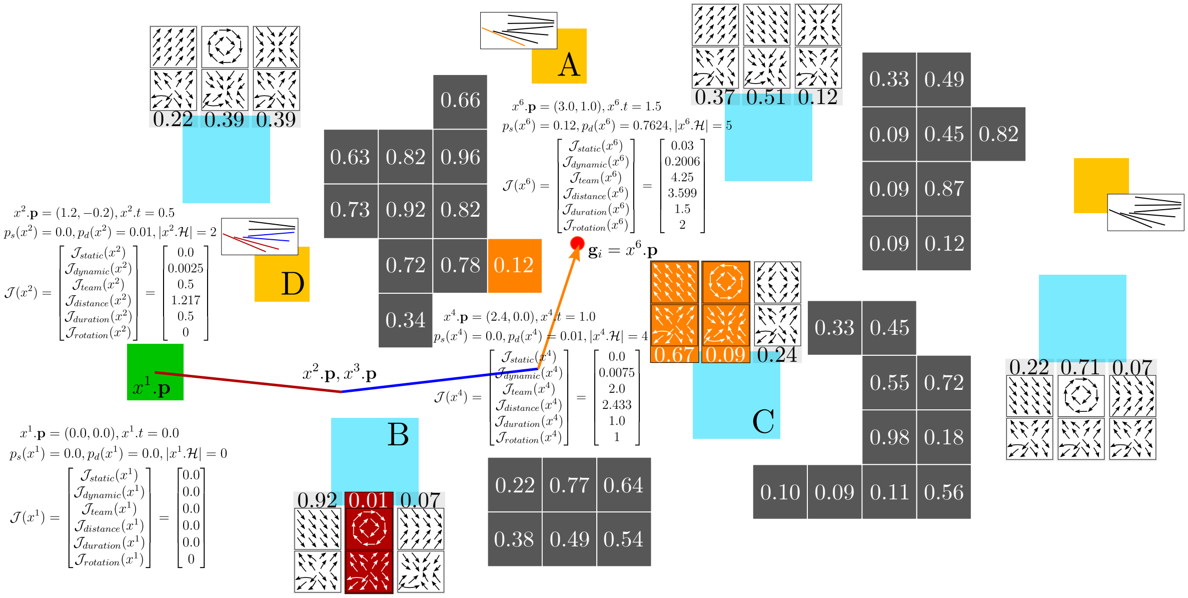
$$
\begin{aligned}
p(\neg\, \mathcal{C}_d(x^{1:l})) &= p(\neg\, \mathcal{C}_d(x^{1:l-1}) \wedge \neg\, \mathcal{C}_d(x^{l-1:l})) \\
&= p(\neg\, \mathcal{C}_d(x^{1:l-1}))p(\neg\, \mathcal{C}_d(x^{l-1:l}) \mid \neg\, \mathcal{C}_d(x^{1:l-1}))
\end{aligned}
$$

The first term $p(\neg\, \mathcal{C}_d(x^{1:l-1}))$ is the recursive term that can be can be obtained from the parent state during search.

The second term $p(\neg\, \mathcal{C}_d(x^{l-1:l}) \mid \neg\, \mathcal{C}_d(x^{1:l-1}))$ is the conditional term that is computed during state expansion. Let $C_{d,j}(x^{l:m})$ be the proposition, conditioned on the full path $x^{1:n}$, that is true if and only if the robot following the $(x^l.\mathbf{p}, x^l.t), \ldots, (x^m.\mathbf{p}, x^m.t)$ portion of $x^{1:n}$ collides with dynamic obstacle $j \in \mathcal{D}_i$ where $l \leq m$. We assume independence between not colliding with different dynamic obstacles, hence, the conditional term simplifies as follows.

$$
\begin{aligned}
p(\neg\, \mathcal{C}_d(x^{l-1:l}) \mid \neg\, \mathcal{C}_d(x^{1:l-1})) &= p\left(\bigwedge_{j\in\mathcal{D}_i} \neg\, \mathcal{C}_{d,j}(x^{l-1:l}) \mid \bigwedge_{j\in\mathcal{D}_i} \neg\, \mathcal{C}_{d,j}(x^{1:l-1})\right) \\
&= \prod_{j\in\mathcal{D}_i} p(\neg\, C_{d,j}(x^{l-1:l})) \mid \neg\, C_{d,j}(x^{1:l-1}))
\end{aligned}
$$

The computation of the terms $p(\neg\, C_{d,j}(x^{l-1:l}) \mid \neg\, C_{d,j}(x^{1:l-1}))$ for each obstacle $j \in \mathcal{D}_i$ is done by using $x^{l-1}.\mathcal{D}$ and $x^l.\mathcal{D}$. Given that robot following states $x^{1:l-1}$ has not collided with dynamic obstacle $j$ means that no behavior model of $j$ that resulted in a collision while traversing $x^{1:l-1}$ is realized. We store all not colliding dynamic obstacles behavior models in $x^{l-1}.\mathcal{D}$. Within these, all dynamic obstacle modes that does not collide with the robot while traversing from $x^{l-1}$ to $x^l$ are stored in $x^l.\mathcal{D}$. Let $x^l.\mathcal{D}_j$ be the set of all behavior model indices of dynamic obstacle $j \in \mathcal{D}_i$ that has not collided with $x^{1:l}$. Probability



0.22 0.39 0.39
A
0.66
$x^6.\mathbf{p} = (3.0, 1.0), x^6.t = 1.5$
$p_s(x^6) = 0.12, p_d(x^6) = 0.7624, |x^6.\mathcal{H}| = 5$
$\mathcal{J}(x^6) = \begin{bmatrix} \mathcal{J}_{static}(x^6) \\ \mathcal{J}_{dynamic}(x^6) \\ \mathcal{J}_{team}(x^6) \\ \mathcal{J}_{distance}(x^6) \\ \mathcal{J}_{duration}(x^6) \\ \mathcal{J}_{rotation}(x^6) \end{bmatrix} = \begin{bmatrix} 0.03 \\ 0.2006 \\ 4.25 \\ 3.599 \\ 1.5 \\ 2 \end{bmatrix}$
0.37 0.51 0.12
0.33 0.49
0.63 0.82 0.96
0.09 0.45 0.82
0.73 0.92 0.82
0.09 0.87
$x^2.\mathbf{p} = (1.2, -0.2), x^2.t = 0.5$
$p_s(x^2) = 0.0, p_d(x^2) = 0.01, |x^2.\mathcal{H}| = 2$
$\mathcal{J}(x^2) = \begin{bmatrix} \mathcal{J}_{static}(x^2) \\ \mathcal{J}_{dynamic}(x^2) \\ \mathcal{J}_{team}(x^2) \\ \mathcal{J}_{distance}(x^2) \\ \mathcal{J}_{duration}(x^2) \\ \mathcal{J}_{rotation}(x^2) \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0025 \\ 0.5 \\ 1.217 \\ 0.5 \\ 0 \end{bmatrix}$
D
0.72 0.78 0.12
$\mathbf{g}_i = x^6.\mathbf{p}$
0.09 0.12
0.34
$x^4.\mathbf{p} = (2.4, 0.0), x^4.t = 1.0$
$p_s(x^4) = 0.0, p_d(x^4) = 0.01, |x^4.\mathcal{H}| = 4$
$\mathcal{J}(x^4) = \begin{bmatrix} \mathcal{J}_{static}(x^4) \\ \mathcal{J}_{dynamic}(x^4) \\ \mathcal{J}_{team}(x^4) \\ \mathcal{J}_{distance}(x^4) \\ \mathcal{J}_{duration}(x^4) \\ \mathcal{J}_{rotation}(x^4) \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0075 \\ 2.0 \\ 2.433 \\ 1.0 \\ 1 \end{bmatrix}$
0.67 0.09 0.24
0.33 0.45
$x^1.\mathbf{p}$
$x^2.\mathbf{p}, x^3.\mathbf{p}$
0.22 0.71 0.07
0.55 0.72
C
B
0.98 0.18
$x^1.\mathbf{p} = (0.0, 0.0), x^1.t = 0.0$
$p_s(x^1) = 0.0, p_d(x^1) = 0.0, |x^1.\mathcal{H}| = 0$
$\mathcal{J}(x^1) = \begin{bmatrix} \mathcal{J}_{static}(x^1) \\ \mathcal{J}_{dynamic}(x^1) \\ \mathcal{J}_{team}(x^1) \\ \mathcal{J}_{distance}(x^1) \\ \mathcal{J}_{duration}(x^1) \\ \mathcal{J}_{rotation}(x^1) \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0 \end{bmatrix}$
0.22 0.77 0.64
0.10 0.09 0.11 0.56
0.92 0.01 0.07
0.38 0.49 0.54

Figure 6.3: **Discrete search.** A sample discrete state sequence, the associated meta data, and computed cost terms are shown. The computed state sequence has six states: $x^{1:6}$. $x^3$ and $x^5$ are expanded with ROTATE actions from $x^2$ and $x^4$ respectively, therefore, their information is not shown here to reduce the clutter. The robot initially does not collide with any static or dynamic obstacle and does not violate any DSHT hyperplane at $x^1$. While traversing the first segment from $x^1$ to $x^2$ (red), it collides with dynamic obstacle B's second behavior model and violates two hyperplanes in the DSHT against teammate D. While traversing the second segment from $x^3$ to $x^4$ (blue), it violates 2 more hyperplanes from the DSHT against teammate D. While traversing the last segment from $x^5$ to $x^6$ (orange), it collides with the static obstacle with existence probability 0.12, first two behavior models of dynamic obstacle C and violates a hyperplane in the DSHT against teammate A.

that robot does not collide with dynamic obstacle $j$ while traversing from $x^{l-1}$ to $x^l$ given that it has not collided with it while travelling from $x^1$ to $x^{l-1}$ is given as

$$p(\neg\, C_{d,j}(x^{l-1:l}) \mid \neg\, C_{d,j}(x^{1:l-1})) = \frac{\sum_{k \in x^l.\mathcal{D}_j} p^{dyn}_{i,j,k}}{\sum_{k \in x^{l-1}.\mathcal{D}_j} p^{dyn}_{i,j,k}}.$$

The computed probabilities for not colliding are lower bounds because *collision checks against dynamic obstacles are done conservatively*, i.e., time domain is not considered during sweep to sweep collision checks. Conservative collision checks never miss collisions, but may over-report them.

**Costs.** Let $p_s(x^l) = 1 - p(\neg\, C_s(x^{1:l}))$ be the probability of collision with any of the static obstacles and $p_d(x^l) = 1 - p(\neg\, C_d(x^{1:l}))$ be an upper bound for the probability of collision with any of the dynamic obstacles while traversing state sequence $x^{1:l}$. We define $P_s(t) : [0, x^n.t] \to [0, 1]$ of state sequence $x^{1:n}$ as the linear interpolation of $p_s$:

$$P_s(t) = \begin{cases} \frac{x^2.t-t}{x^2.t-x^1.t} p_s(x^1) + \frac{t-x^1.t}{x^2.t-x^1.t} p_s(x^2) & x^1.t \leq t < x^2.t \\ \dots & \\ \frac{x^n.t-t}{x^n.t-x^{n-1}.t} p_s(x^{n-1}) + \frac{t-x^{n-1}.t}{x^n.t-x^{n-1}.t} p_s(x^n) & x^{n-1}.t \leq t \leq x^n.t \end{cases}$$

We define $P_d(t) : [0, x^n.t] \to [0, 1]$ of a state sequence $x^{1:n}$ in a similar way using $p_d$. We define $P_c(t) : [0, x^n.t] \to [0, \infty)$ as the linear interpolation of the number of violated hyperplanes in active DSHTs of the state sequence $x^{1:n}$, i.e., the points $(x^1.t, |x^1.\mathcal{H}|), \dots, (x^n.t, |x^n.\mathcal{H}|)$.

We associate six different cost terms to each state $x^l$ in state sequence $x^{1:n}$: i) $\mathcal{J}_{static}(x^l) \in [0, \infty)$ is the cumulative static obstacle collision probability defined as $\mathcal{J}_{static}(x^l) = \int_0^{x^l.t} P_s(\tau) d\tau$, ii) $\mathcal{J}_{dynamic}(x^l) \in [0, \infty)$ is the cumulative dynamic obstacle collision probability defined as $\mathcal{J}_{dynamic}(x^l) = \int_0^{x^l.t} P_d(\tau) d\tau$, iii) $\mathcal{J}_{team}(x^l) \in [0, \infty)$ is the cumulative number of violated active DSHT hyperplanes defined as $\mathcal{J}_{team}(x^l) = \int_0^{\min(x^l.t, T_i^{team})} P_c(\tau) d\tau$, in which violation cost accumulation is cut off at $T_i^{team}$ parameter, iv) $\mathcal{J}_{distance}(x^l) \in [0, \infty)$ is the distance travelled from start state $x^1$ to state $x^l$, v) $\mathcal{J}_{duration}(x^l) \in [0, \infty)$ is the time elapsed from start state $x^1$ to state $x^l$, and vi) $\mathcal{J}_{rotation}(x^l) \in \mathbb{N}$ is the number of rotations from start state $x^1$ to state $x^l$.

We cut off violation cost accumulation of DSHTs because of the conservative nature of using separating hyperplanes for teammate safety: they divide the space into two disjoint sets linearly, without considering robots' intents. The robots need to be safe until the next successful planning iteration because of the receding horizon planning, and overly constraining a large portion of the plan at each planning iteration with conservative constraints decreases agility. We investigate the effects of $T_i^{team}$ on navigation performance in Section 6.4.2.13.

We compute the cost terms of the new state $x^+$ after applying actions to current state $x$ as follows.

- $\mathcal{J}_{static}(x^+) = \mathcal{J}_{static}(x) + \int_{x.t}^{x^+.t} P_s(\tau)\tau$
- $\mathcal{J}_{dynamic}(x^+) = \mathcal{J}_{dynamic}(x) + \int_{x.t}^{x^+.t} P_d(\tau)\tau$
- $\mathcal{J}_{team}(x^+) = \mathcal{J}_{team}(x) + \int_{\min(x.t, T_i^{team})}^{\min(x^+.t, T_i^{team})} P_c(\tau)d\tau$
- $\mathcal{J}_{distance}(x^+) = \mathcal{J}_{distance}(x) + \|x^+.\mathbf{p} - x.\mathbf{p}\|_2$
- $\mathcal{J}_{duration}(x^+) = \mathcal{J}_{duration}(x) + (x^+.t - x.t)$
- $\mathcal{J}_{rotation}(x^+) = \mathcal{J}_{rotation}(x) + \mathbb{1}_{\neq}(x.\mathbf{\Delta}, x^+.\mathbf{\Delta})$

where $\mathbb{1}_{\neq}$ is the indicator function with value $1$ if its arguments are unequal, and $0$ otherwise.

Lower cost (with respect to standard comparison operator $\leq$) is better in all cost terms. All cost terms have the minimum cost of $0$ and upper bound cost of $\infty$. All cost terms are additive using standard addition operator $+$. $\mathcal{J}_{static}$, $\mathcal{J}_{dynamic}$, $\mathcal{J}_{team}$, $\mathcal{J}_{distance}$, and $\mathcal{J}_{duration}$ are cost algebras ($\mathbb{R}_{\geq 0} \cup \{\infty\}$, min, $+$, $\leq$, $\infty$, 0) and $\mathcal{J}_{rotation}$ is cost algebra ($\mathbb{N} \cup \{\infty\}$, min, $+$, $\leq$, $\infty$, 0), both of which are strictly isotone. Therefore, their Cartesian product is also a cost algebra, which is what we optimize over. The cost $\mathcal{J}(x)$ of each state $x$ is:

$$\mathcal{J}(x) = \begin{bmatrix} \mathcal{J}_{static}(x) \\ \mathcal{J}_{dynamic}(x) \\ \mathcal{J}_{team}(x) \\ \mathcal{J}_{distance}(x) \\ \mathcal{J}_{duration}(x) \\ \mathcal{J}_{rotation}(x) \end{bmatrix}.$$

We order cost terms lexicographically as required by Cartesian product cost algebras. A sample state sequence and computed costs are shown in Figure 6.3.

This induces an ordering between cost terms: we first minimize cumulative static obstacle collision probability, and among the states that minimizes that, we minimize cumulative dynamic obstacle collision probability, and so on. Hence, safety is the primary; distance, duration, and rotation optimality are the secondary concerns. Out of safety against static and dynamic obstacles and teammates, we prioritize static obstacles over dynamic obstacles, because static obstacles can be considered a special type of dynamic ones, i.e., with **0** velocity, and hence, prioritizing dynamic obstacles would make the static obstacle avoidance cost unnecessary. This ordering allows us to optimize the special case first, and then attempt the harder one. The reason we prioritize dynamic obstacles over teammates is the conservative nature of using DSHTs for teammates. Violating a separating hyperplane does not necessarily result in a collision because each hyperplane divides the space into two between robots, and the robots occupy a very small portion of their side of each hyperplane.

The heuristic $H(x)$ we use for each state $x$ during search is as follows.

$$H(x) = \begin{bmatrix} H_{static}(x) \\ H_{dynamic}(x) \\ H_{team}(x) \\ H_{distance}(x) \\ H_{duration}(x) \\ H_{rotation}(x) \end{bmatrix} = \begin{bmatrix} P_s(x.t)H_{duration}(x) \\ P_d(x.t)H_{duration}(x) \\ P_c(x.t)\max(0, \min(H_{duration}(x), T_i^{team} - x.t)) \\ \|x.\mathbf{p} - \mathbf{g}_i\|_2 \\ \max(\tau_i' - x.t, \frac{H_{distance}(x)}{\tilde{\gamma_1}}) \\ 0 \end{bmatrix}$$

We first compute $H_{distance}(x)$, which we use in computation of $H_{duration}(x)$. Then, we use $H_{duration}(x)$ during the computation of $H_{static}(x)$, $H_{dynamic}(x)$, and $H_{team}(x)$.

**Proposition 2.** *All individual heuristics are admissible.*

*Proof.* **Admissibility of $\boldsymbol{H_{distance}}$:** $H_{distance}(x)$ is the Euclidean distance from $x.\mathbf{p}$ to $\mathbf{g}_i$, and never overestimates the true distance.

**Admissibility of $\boldsymbol{H_{duration}}$:** The goal position $\mathbf{g}_i$ can be any position in $\mathbb{R}^d$, which is an uncountable set. The FORWARD and ROTATE actions can only move robot to a discrete set of positions, which is countable, as any discrete subset of an Euclidean space is countable. Therefore, the probability that robot reaches $\mathbf{g}_i$ by only executing FORWARD and ROTATE actions is zero. The robot cannot execute any action after REACHGOAL action in an optimal path to a goal state, because the REACHGOAL action already ends in the goal position and any subsequent actions would only increase the total cost. Hence, the last action in an optimal path to a goal state should be REACHGOAL. There are two cases to consider.

If the last action while arriving at state $x$ is REACHGOAL, $x.t \geq \tau'_i$ holds (as REACHGOAL enforces this, see the descriptions of actions). Since $x.\mathbf{p} = \mathbf{g}_i$, $H_{distance}(x) = 0$. Therefore, $H_{duration}(x) = \max(\tau'_i - x.t, \frac{H_{distance}(x)}{\tilde{\gamma}^1_i}) = 0$, which is trivially admissible, as $0$ is the lowest cost.

If the last action while arriving at state $x$ is not REACHGOAL, the search should execute REACHGOAL action to reach to the goal position in the future, which enforces that goal position will not be reached before $\tau'_i$. Also, since the maximum speed that can be executed during search is $\tilde{\gamma}^1_i$, robot needs at least $\frac{H_{distance}(x)}{\tilde{\gamma}^1_i}$ seconds to reach to the goal position as $H_{distance}(x)$ is an admissible heuristic for distance to goal position. Hence, $H_{duration}(x) = \max(\tau'_i - x.t, \frac{H_{distance}(x)}{\tilde{\gamma}^1_i})$ is admissible.

**Admissibility of $\boldsymbol{H_{static}}$ and $\boldsymbol{H_{dynamic}}$:** We prove the admissibility of $H_{static}$. Proof of admissibility of $H_{dynamic}$ follows identical steps. $P_s$ is a nondecreasing nonnegative function as it is the accumulation of linear interpolation of probabilities, which are defined over $[0, 1]$. Therefore, $P_s(x.t) \leq P_s(t)$ for $t \geq x.t$ in an optimal path to a goal state traversing $x$. The robot needs at least $H_{duration}(x)$ seconds to reach a goal state from $x$, since $H_{duration}$ is an admissible heuristic. Let $T^g(x) \geq H_{duration}(x)$ be the actual duration needed to reach to a goal state from $x$ on an optimal path. The actual cumulative static obstacle collision probability to a goal on an optimal path from $x$ is $\int_{x.t}^{x.t+T^g(x)} P_s(\tau)d\tau$. We have

$$\begin{aligned} H_{static}(x) &= P_s(x.t)H_{duration}(x) \\ &= \int_{x.t}^{x.t+H_{duration}(x)} P_s(x.t)d\tau \\ &\leq \int_{x.t}^{x.t+T^g(x)} P_s(x.t)d\tau \\ &\leq \int_{x.t}^{x.t+T^g(x)} P_s(\tau)d\tau. \end{aligned}$$

In other words, $H_{static}(x)$ does not overestimate the true static obstacle cumulative collision probability from $x$ to a goal state.

**Admissibility of $\boldsymbol{H_{team}}$:** Let $x^{1:n}$ be an optimal state sequence from start state $x^1$ to a goal state $x^n$ traversing $x$. If $x.t \geq T_i^{team}$, $P_c(t)$ will not be accumulated in the future because of the cut off. If $x.t \leq T_i^{team}$, $P_c(t)$ will be accumulated for duration at least $\min(H_{duration}(x), T_i^{team} - x.t)$ because $H_{duration}$ never overestimates the true duration to a goal and accumulation is cut off at $T_i^{team}$. Therefore, $P_c(t)$ will be accumulated for at least $\max(0, \min(H_{duration}(x), T_i^{team} - x.t))$ after state $x$. Let $T^c(x) \geq \max(0, \min(H_{duration}(x), T_i^{team} - x.t))$ be the actual duration $P_c(t)$ will be accumulated after state $x$. The actual cumulative number of violated active DSHT hyperplanes is given by $\int_{x.t}^{x.t+T^c(x)} P_c(\tau)d\tau$.

$|x^l.\mathcal{H}| \geq |x^{l-1}.\mathcal{H}|$ for all $l \in \{2, \ldots, n\}$ because if a hyperplane is violated while traversing $x^1, \ldots, x^{l-1}$, it is also violated while traversing $x^1, \ldots, x^l$. Therefore, linear interpolation $P_c(t)$ of number of violated hyperplanes is a nondecreasing function, i.e., $P_c(x.t) \leq P_c(t) \; \forall t \in [x.t, x^n.t]$. In addition, $P_c(t)$ is a nonnegative function as it is a linear interpolation of set cardinalities. Hence, we have

$$\begin{aligned} H_{team}(x) &= P_c(x.t) \max(0, \min(H_{duration}(x), T_i^{team} - x.t)) \\ &= \int_{x.t}^{x.t + \max(0, \min(H_{duration}(x), T_i^{team} - x.t))} P_c(x.t) d\tau \\ &\leq \int_{x.t}^{x.t + T^c(x)} P_c(x.t) d\tau \\ &\leq \int_{x.t}^{x.t + T^c(x)} P_c(\tau) d\tau. \end{aligned}$$

In other words, $H_{team}(x)$ never overestimates true accumulated $P_c(t)$ in an optimal path to the goal state $x^n$ from $x$.

**Admissibility of $\boldsymbol{H_{rotation}}$:** $H_{rotation} = 0$ is trivially admissible because 0 is the lowest cost.

□

As each individual cost term is admissible, their Cartesian product is also admissible. Hence, cost algebraic A* with re-openings minimizes $\mathcal{J}$ with the given heuristics $H$.

**Time limited best effort search.** Finding the optimal state sequence induced by the costs $\mathcal{J}$ to the goal position $\mathbf{g}_i$ can take a lot of time. Because of this, each robot $i$ limits the duration of the search using a maximum search duration parameter $T_i^{search}$. When the search gets cut off because of the time limit, we return the lowest cost state sequence to a goal state so far. During node expansion, A* applies all actions to the state with the lowest cost. One of those actions is always REACHGOAL. Therefore, we connect all expanded states to the goal position using REACHGOAL action. Hence, when the search is cut off, there are many candidate plans to the goal position, which are already sorted according to their costs by A*.

We remove the states generated by ROTATE actions from the search result and provide the resulting sequence to the trajectory optimization stage. Note that ROTATE changes only the direction $x.\boldsymbol{\Delta}$.

The trajectory optimization stage does not use $x.\mathbf{\Delta}$, therefore we remove repeated states for the input of trajectory optimization..

### 6.3.4 Trajectory Optimization

Let $x^1, \ldots, x^N$ be the state sequence provided to trajectory optimization. In the trajectory optimization stage, each robot $i$ fits a Bézier curve $\mathbf{f}_{i,l}(t) : [0, T_{i,l}] \rightarrow \mathbb{R}^d$ of degree $h_{i,l}$ (which are parameters) where $T_{i,l} = x^{l+1}.t - x^l.t$ to each segment from $(x^l.t, x^l.\mathbf{p})$ to $(x^{l+1}.t, x^{l+1}.\mathbf{p})$ for all $l \in \{1, \ldots, N-1\}$ to compute a piecewise trajectory $\mathbf{f}_i(t) : [0, x^N.t] \rightarrow \mathbb{R}^d$ where each piece is the fitted Bézier curve, i.e.

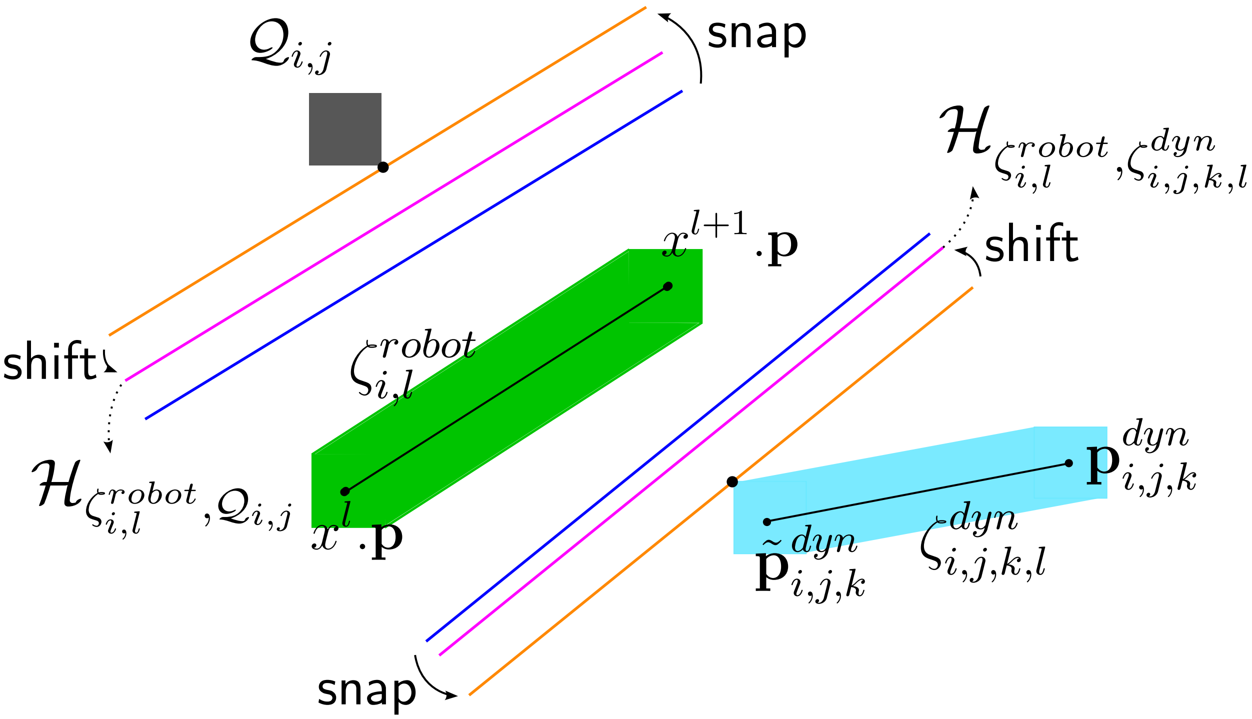
$$\mathbf{f}_i(t) = \begin{cases} \mathbf{f}_{i,1}(t - x^1.t) & x^1.t = 0 \leq t < x^2.t \\ \ldots & \\ \mathbf{f}_{i,N-1}(t - x^{N-1}.t) & x^{N-1}.t \leq t \leq x^N.t \end{cases}$$

Let $\mathbf{P}_{i,l,0}, \ldots, \mathbf{P}_{i,l,h_{i,l}}$ be the control points of Bézier curve $\mathbf{f}_{i,l}(t)$ of degree $h_{i,l}$. Let $\mathcal{P}_i = \{\mathbf{P}_{i,l,k} \mid l \in \{1, \ldots, N-1\}, k \in \{0, \ldots, h_{i,l}\}\}$ be the set of the control points of all pieces of robot $i$.

During the trajectory optimization stage, we construct a QP where decision variables are control points $\mathcal{P}_i$ of the trajectory.

We preserve the cumulative collision probabilities $P_s$ and $P_d$ and the cumulative number of violated active DSHT hyperplanes $P_c$ of the state sequence $x^{1:N}$, by ensuring that robot following $\mathbf{f}_{i,l}(t)$ i) avoids the same static obstacles and dynamic obstacle behavior models robot travelling from $x^1$ to $x^{l+1}$ avoids and ii) does not violate any active DSHT hyperplane that the robot travelling from $x^1$ to $x^{l+1}$ does also not violate for all $l \in \{1, \ldots, N-1\}$. The constraints generated for these are always feasible.

In order to encourage dynamic obstacles to determine their behavior using the interaction model in the same way they determine it in response to $x^{1:N}$, we add cost terms that matches the position and the



$\mathcal{Q}_{i,j}$
snap
$\mathcal{H}_{\zeta_{i,l}^{robot},\zeta_{i,j,k,l}^{dyn}}$
$x^{l+1}.\mathbf{p}$
shift
$\zeta_{i,l}^{robot}$
shift
$\mathbf{p}_{i,j,k}^{dyn}$
$\mathcal{H}_{\zeta_{i,l}^{robot},\mathcal{Q}_{i,j}}$
$x^{l}.\mathbf{p}$
$\tilde{\mathbf{p}}_{i,j,k}^{dyn}$
$\zeta_{i,j,k,l}^{dyn}$
snap

Figure 6.4: **Static and dynamic obstacle collision constraints.** Given the gray static obstacle $j \in \mathcal{O}_i$ with shape $\mathcal{Q}_{i,j}$ and the green sweep $\zeta_{i,l}^{robot}$ of $\mathcal{R}_i^{robot}$ from $x^l.\mathbf{p}$ to $x^{l+1}.\mathbf{p}$, we compute the blue support vector machine hyperplane between them. We compute the orange separating hyperplane by snapping it to $\mathcal{Q}_{i,j}$. The robot should stay in the safe side of the orange hyperplane. We shift orange hyperplane to account for robot's collision shape $\mathcal{R}_i^{robot}$ and compute the magenta hyperplane $\mathcal{H}_{\zeta_{i,l}^{robot},\mathcal{Q}_{i,j}}$. The Bézier curve $\mathbf{f}_{i,l}(t)$ is constrained by $\mathcal{H}_{\zeta_{i,l}^{robot},\mathcal{Q}_{i,j}}$ to avoid $\mathcal{Q}_{i,j}$. To avoid the dynamic obstacle $j \in \mathcal{D}_i$ moving from $\tilde{\mathbf{p}}_{i,j,k}^{dyn}$ to $\mathbf{p}_{i,j,k}^{dyn}$, the support vector machine hyperplane between the region $\zeta_{i,j,k,l}^{dyn}$ swept by $\mathcal{R}_{i,j}^{dyn}$ and $\zeta_{i,l}^{robot}$ is computed. The same snap and shift operations are conducted to compute the magenta hyperplane $\mathcal{H}_{\zeta_{i,l}^{robot},\zeta_{i,j,k,l}^{dyn}}$, constraining $\mathbf{f}_{i,l}(t)$.

velocity at the start of each Bézier piece $\mathbf{f}_{i,l}(t)$ to the position and the velocity of the robot at $x^l$ for each $l \in \{1, \ldots, N-1\}$.

#### 6.3.4.1 Constraints

There are five types of constraints we impose on the trajectory, all of which are linear in control points $\mathcal{P}_i$.

**Static obstacle avoidance constraints.** Let $j \in \mathcal{O}_i \setminus x^{l+1}.\mathcal{O}$ be a static obstacle that robot $i$ travelling from $x^1$ to $x^{l+1}$ avoids for an $l \in \{1, \ldots, N-1\}$. Let $\zeta_{i,l}^{robot}$ be the space swept by the robot travelling the straight line from $x^l.\mathbf{p}$ to $x^{l+1}.\mathbf{p}$. Since the shape of the robot is convex and it is swept along a straight line segment, $\zeta_{i,l}^{robot}$ is also convex (Section A). Static obstacle $j$ is also convex by definition. Since robot avoids $j$, $\mathcal{Q}_{i,j} \cap \zeta_{i,l}^{robot} = \emptyset$. Hence, they are linearly separable by the separating hyperplane theorem (Theorem 1). We compute the support vector machine (SVM) hyperplane between $\zeta_{i,l}^{robot}$ and $\mathcal{Q}_{i,j}$, snap it to $\mathcal{Q}_{i,j}$ by shifting it along its normal so that it touches $\mathcal{Q}_{i,j}$, and shift it back to account for robot's collision shape $\mathcal{R}_i^{robot}$ similarly to RLSS (Chapter 4) (Figure 6.4). Let $\mathcal{H}_{\zeta_{i,l}^{robot},\mathcal{Q}_{i,j}}$ be this hyperplane. We constrain $\mathbf{f}_{i,l}$ with $\mathcal{H}_{\zeta_{i,l}^{robot},\mathcal{Q}_{i,j}}$ for it to avoid static obstacle $j$, which is a feasible linear constraint as shown in Remark 2.

These constraints enforce that robot traversing $\mathbf{f}_{i,l}(t)$ avoids the same obstacles robot traversing from $x^1$ to $x^{l+1}$ avoids, not growing the set $x^{l+1}.\mathcal{O}$ between $[x^l.t, x^{l+1}.t]$ $\forall l \in \{1, \dots, N-1\}$, and hence preserving $P_s(t)$ $\forall t \in [0, x^N.t]$.

**Dynamic obstacle avoidance constraints.** Let $(\mathcal{B}_{i,j,k}, \mathbf{p}_{i,j,k}^{dyn}) \in x^{l+1}.\mathcal{D}$ be a dynamic obstacle behavior model–position pair that does not collide with robot travelling from $x^1$ to $x^{l+1}$ for an $l \in \{1, \dots, N-1\}$. $\mathcal{B}_{i,j,k}$ should be in $x^l.\mathcal{D}$ as well, because the behavior models in $x^{l+1}.\mathcal{D}$ are a subset of behavior models in $x^l.\mathcal{D}$ by definition. Let $\tilde{\mathbf{p}}_{i,j,k}^{dyn}$ be the position of the dynamic obstacle $j$ moving according to behavior model $\mathcal{B}_{i,j,k}$ at state $x^l$. Let $\zeta_{i,j,k,l}^{dyn}$ be the region swept by the dynamic object $j$ from $\tilde{\mathbf{p}}_{i,j,k}^{dyn}$ to $\mathbf{p}_{i,j,k}^{dyn}$. During collision check of state expansion from $x^l$ to $x^{l+1}$, we check whether $\zeta_{i,j,k,l}^{dyn}$ intersects with $\zeta_{i,l}^{robot}$ and add the model to $x^{l+1}.\mathcal{D}$ if they do not. Since these sweeps are convex sets (because they are sweeps of convex sets along straight line segments), they are linearly separable (Theorem 1). We compute the SVM hyperplane between them, snap it to the region swept by dynamic obstacle and shift it back to account for the robot shape $\mathcal{R}_i^{robot}$ (Figure 6.4). Let $\mathcal{H}_{\zeta_{i,l}^{robot},\zeta_{i,j,k,l}^{dyn}}$ be this hyperplane. We constrain $\mathbf{f}_{i,l}$ with $\mathcal{H}_{\zeta_{i,l}^{robot},\zeta_{i,j,k,l}^{dyn}}$, which is a feasible linear constraint as shown in Remark 2.

These constraints enforce that robot traversing $\mathbf{f}_{i,l}(t)$ avoids same dynamic obstacle behavior models robot travelling from $x^1$ to $x^{l+1}$ avoids, not shrinking the set $x^{l+1}.\mathcal{D}$ $\forall l \in \{1, \dots, N-1\}$, and hence preserving $P_d(t)$ $\forall t \in [0, x^N.t]$.

The reason we perform *conservative collision checks* for dynamic obstacle avoidance during discrete search is to use the separating hyperplane theorem. Without the conservative collision check, there is no proof of linear separability, and SVM computation might fail.

**Teammate avoidance constraints.** Let $\mathcal{H} \in \tilde{\mathcal{H}}_i^{active} \setminus x^{l+1}.\mathcal{H}$ be an active DSHT hyperplane that is not violated while traversing states from $x^1$ to $x^{l+1}$. If $x^l.t < T_i^{team}$, i.e., the segment from $x^l$ to $x^{l+1}$ is within the teammate safety enforcement period, we constrain $\mathbf{f}_{i,l}$ with $\mathcal{H}$ by shifting it to account for robot's collision shape, and enforcing $\mathbf{f}_{i,l}$ to be in the safe side of the shifted hyperplane, which is a feasible constraint(Remark 2). Otherwise, we do not constrain the piece $\mathbf{f}_{i,l}$ with active DSHT hyperplanes.

Within the safety enforcement period $T_i^{team}$, any $\mathbf{f}_{i,l}$ does not violate any active DSHT hyperplane that is not violated while traversing the state sequence $x^{1:l+1}$, preserving the cardinality of sets $x^{l+1}.\mathcal{H}$, and hence $P_c(t)$.

**Continuity constraints.** We enforce continuity up to the desired degree $c_i$ between pieces by

$$\frac{d^k \mathbf{f}_{i,l}(T_{i,l})}{dt^k} = \frac{d^k \mathbf{f}_{i,l+1}(0)}{dt^k} \; \forall l \in \{1, \dots, N-2\}$$
$$\forall k \in \{0, \dots, c_i\}.$$

We enforce continuity up to desired degree $c_i$ between planning iterations by

$$\frac{d^k \mathbf{f}_i(0)}{dt^k} = \mathbf{p}_{i,k}^{self} \; \forall k \in \{0, \dots, c_i\}.$$

**Dynamic limit constraints.** Derivative of a Bézier curve is another Bézier curve with a lower degree, control points of which are linearly related to the control points of the original curve [29]. Let $\mathcal{P}_i^k = \mathbf{D}^k(\mathcal{P}_i)$ be the control points of the $k^{th}$ derivative of $\mathbf{f}_i$, where $\mathbf{D}^k$ is the linear transformation relating $\mathcal{P}_i$ to $\mathcal{P}_i^k$. We enforce dynamic constraints uncoupled among dimensions by limiting maximum $k^{th}$ derivative magnitude in each dimension by $\frac{\gamma_i^k}{\sqrt{d}}$ so that they are linear. Utilizing the convex hull property of Bézier curves, we enforce

$$-\frac{\gamma_i^k}{\sqrt{d}} \preceq \mathbf{P} \preceq \frac{\gamma_i^k}{\sqrt{d}} \; \forall \mathbf{P} \in \mathcal{P}_i^k$$

which limits $k^{th}$ derivative magnitude with $\gamma_i^k$ along the trajectory $\mathbf{f}_i$ where $\preceq$ between a vector and a scalar is true if and only if all elements of the vector are less than or equal to the scalar.

While collision avoidance constraints are always feasible, we do not have a general proof of feasibility for continuity and dynamic limit constraints, which may cause planner to fail. If the planner fails, the robot continues using the last successfully planned trajectory.

Imposing collision avoidance as described results in a high number of hyperplane constrains, resulting in non-real-time execution. In order to decrease the number of collision avoidance constraints while also still preserving $P_s$, and $P_d$, we utilize an ordered constraint generation method, which decreases the number of constraints for static and dynamic obstacles from thousands to less than ten for each piece in obstacle rich scenarios. The details of this method are described in Appendix B.

#### 6.3.4.2 Objective Function

We use a linear combination of three cost terms as our objective function, all of which are quadratic in control points $\mathcal{P}_i$.

**Energy term.** We use sum of integrated squared derivative magnitudes as a metric for energy usage similar to [39, 94, 100]. The energy usage cost term $\mathcal{J}_{energy}(\mathcal{P}_i)$ is

$$\mathcal{J}_{energy}(\mathcal{P}_i) = \sum_{\lambda_{i,k} \in \boldsymbol{\lambda}_i} \lambda_{i,k} \int_0^{x^N.t} \left\| \frac{d^k \mathbf{f}_i(t)}{dt^k} \right\|_2^2 dt$$

where $\lambda_{i,k}$s are parameters.

**Position matching term.** We add a position matching term $J_{position}(\mathcal{P}_i)$ that penalizes distance between piece endpoints and state sequence positions $x^2.\mathbf{p}, \ldots, x^N.\mathbf{p}$.

$$\mathcal{J}_{position}(\mathcal{P}_i) = \sum_{l \in \{1,\ldots,N-1\}} \theta_{i,l} \left\| \mathbf{f}_{i,l}(T_{i,l}) - x^{l+1}.\mathbf{p} \right\|_2^2$$

where $\theta_{i,l}$s are weight parameters.

**Velocity matching term.** We add a velocity matching term $J_{velocity}$ that penalizes divergence from the velocities of the state sequence $x^{1:N}$ at piece start points.

$$\mathcal{J}_{velocity}(\mathcal{P}_i) = \sum_{l \in \{1,\ldots,N-1\}} \beta_{i,l} \left\| \frac{d\mathbf{f}_{i,l}(0)}{dt} - \frac{x^{l+1}.\mathbf{p} - x^l.\mathbf{p}}{x^{l+1}.t - x^l.t} \right\|_2^2$$

where $\beta_{i,l}$s are weight parameters.

Position and velocity matching terms encourage matching the positions and velocities of the state sequence $x^{1:N}$. This causes dynamic obstacles to make similar interaction decisions against the robot following trajectory $\mathbf{f}_i(t)$ to they do to the robot following the state sequence $x^{1:N}$. One could also add constraints to the optimization problem to exactly match positions and velocities. Adding position and velocity matching terms as constraints resulted in a high rate of optimization infeasibilities in our experiments. Therefore, we choose to add them to the cost function of the optimization term in the final algorithm.

## 6.4 Evaluation

We implement our algorithm in C++. We use CPLEX 12.10 to solve the quadratic programs generated during trajectory optimization stage, including the SVM problems. We evaluate our planner's behavior in simulations in 3D. All simulation experiments are conducted in a computer with Intel(R) i7-8700K CPU @3.70GHz, running Ubuntu 20.04 as the operating system. The planning pipeline is executed in a single core of the CPU in each planning iteration of each robot. We compare our algorithm's performance with three state-of-the-art decentralized navigation decision making algorithms, namely SBC [121], RLSS [99], and RMADER [50], in both single-robot and multi-robot scenarios in simulations and show that our algorithm achieves considerably higher success rate compared to the baselines. We implement our algorithm for physical quadrotors and show its feasibility in real world in single and multi robot experiments. The supplemental video includes recordings from both i) simulations, including some not covered in this paper, and ii) physical robot experiments.

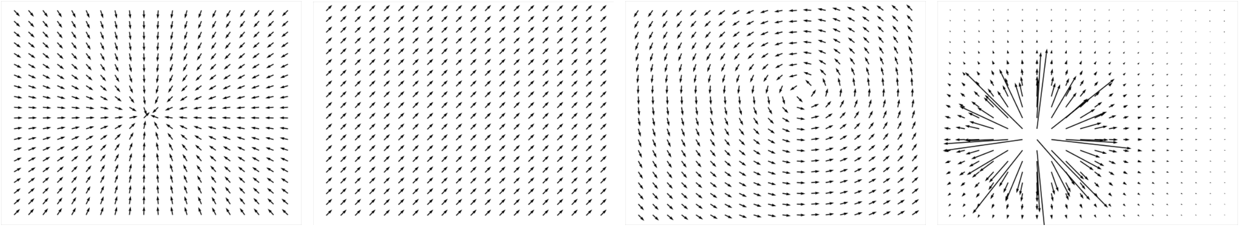
### 6.4.1 Simulation Evaluation Setup

#### 6.4.1.1 Obstacle Sensing

We use octrees [41] to represent the static obstacles. Each axis aligned box with its stored existence probability is used as a static obstacle. We model static obstacle sensing imperfections using three operations applied to the octree representation of the environment in static obstacle sensing uncertainty experiments (Section 6.4.2.11):

- **increaseUncertainty:** Increases the uncertainty of existing obstacles by moving their existence probabilities closer to $0.5$, by sampling a probability between the existence probability $p$ of an obstacle and $0.5$ uniformly.
- **leakObstacles($\boldsymbol{p_{leak}}$):** Leaks each obstacle to a neighbouring region with probability $p_{leak}$.
- **deleteObstacles:** Deletes obstacles randomly according to their non-existence probabilities.

We model dynamic obstacle shapes $\mathcal{R}^{dyn}_{i,j}$ as axis aligned boxes. For each robot $i$, to simulate imperfect sensing of $\mathcal{R}^{dyn}_{i,j}$, we inflate or deflate it in each axis randomly according to a one dimensional $0$ mean



(a) Goal attractive movement (b) Constant velocity movement (c) Rotating movement (d) Repulsive interaction

Figure 6.5: The movement and interaction models we define for dynamic obstacles. Each model has associated parameters described in Section 6.4.1.2.

Gaussian noise with standard deviation $\sigma_i$ in experiments with dynamic obstacle sensing uncertainty (Section 6.4.2.10)[‡]. Each robot $i$ is assumed to be noisily sensing the position and velocity of each dynamic obstacle $j \in \mathcal{D}_i$ according to a $2d$ dimensional $0$ mean Gaussian noise with positive definite covariance $\Sigma_i \in \mathbb{R}^{2d \times 2d}$. The first $d$ terms of the noise are applied to the real position and the second $d$ terms of the noise are applied to the real velocity of the obstacle to compute sensed the position and velocity at each simulation step.

#### 6.4.1.2 Predicting Behavior Models of Dynamic Obstacles

We introduce three simple model-based online dynamic obstacle behavior model prediction methods to use during evaluation[§].

Let $\mathbf{p}^{dyn}$ be the position of a dynamic obstacle. We define three movement models:

- **Goal attractive movement model $\mathcal{M}_g(\mathbf{p}^{dyn}|\hat{\mathbf{g}}, \hat{s})$ (Figure 6.5a):** Attracts the dynamic obstacle to the goal position $\hat{\mathbf{g}}$ with desired speed $\hat{s}$. The desired velocity $\tilde{\mathbf{v}}^{dyn}$ of the dynamic obstacle is computed as $\tilde{\mathbf{v}}^{dyn} = \mathcal{M}_g(\mathbf{p}^{dyn}|\hat{\mathbf{g}}, \hat{s}) = \frac{\hat{\mathbf{g}}-\mathbf{p}^{dyn}}{\left\|\hat{\mathbf{g}}-\mathbf{p}^{dyn}\right\|_2}\hat{s}$.

[‡] Note that, we do not explicitly account for the dynamic obstacle shape sensing uncertainty during planning, yet we still show our algorithm's performance under such uncertainty.

[§] More sophisticated behavior prediction methods can be developed and integrated with our planner, which might potentially use domain knowledge about the environment objects exists or handle position and velocity sensing uncertainties explicitly.

- **Constant velocity movement model $\boldsymbol{\mathcal{M}_c}(\mathbf{p}^{\boldsymbol{dyn}}|\hat{\mathbf{v}})$ (Figure 6.5b):** Moves the dynamic obstacle with constant velocity $\hat{\mathbf{v}}$. The desired velocity $\tilde{\mathbf{v}}^{dyn}$ of the dynamic obstacle is computed as $\tilde{\mathbf{v}}^{dyn} = \mathcal{M}_c(\mathbf{p}^{dyn}|\hat{\mathbf{v}}) = \hat{\mathbf{v}}$.

- **Rotating movement model $\boldsymbol{\mathcal{M}_r}(\mathbf{p}^{\boldsymbol{dyn}}|\hat{\mathbf{c}}, \hat{s})$ (Figure 6.5c):** Rotates the robot around the rotation center $\hat{\mathbf{c}}$ with desired speed $\hat{s}$. The desired velocity $\tilde{\mathbf{v}}^{dyn}$ of the dynamic obstacle is computed as $\tilde{\mathbf{v}}^{dyn} = \mathcal{M}_r(\mathbf{p}^{dyn}|\hat{\mathbf{c}}, \hat{s}) = \frac{\mathbf{r}}{\|\mathbf{r}\|_2}\hat{s}$ where $\mathbf{r} \perp (\mathbf{p}^{dyn} - \hat{\mathbf{c}})$[¶].

Let $\mathbf{p}^{robot}$ be the current position and $\mathbf{v}^{robot}$ be the current velocity of a robot. We define one interaction model:

- **Repulsive interaction model $\boldsymbol{\mathcal{I}_r}(\mathbf{p}^{\boldsymbol{dyn}}, \tilde{\mathbf{v}}^{\boldsymbol{dyn}}, \mathbf{p}^{\boldsymbol{robot}}, \mathbf{v}^{\boldsymbol{robot}}|\hat{\boldsymbol{f}})$ (Figure 6.5d):** Causes dynamic obstacle to repulse away from the robot with repulsion strength $\hat{f}$. The velocity of the dynamic obstacle is computed as $\mathbf{v}^{dyn} = \mathcal{I}_r(\mathbf{p}^{dyn}, \tilde{\mathbf{v}}^{dyn}, \mathbf{p}^{robot}, \mathbf{v}^{robot}|\hat{f}) = \tilde{\mathbf{v}}^{dyn} + \frac{(\mathbf{p}^{dyn}-\mathbf{p}^{robot})\hat{f}}{\|\mathbf{p}^{dyn}-\mathbf{p}^{robot}\|_2^3}$. The dynamic obstacle gets repulsed away from the robot linearly proportional to repulsion strength $\hat{f}$, and quadratically inversely proportional to the distance to the robot[‖].

We propose three online prediction methods to predict the behavior models of dynamic obstacles from the sensed position and velocity histories of dynamic obstacles and the robots, one for each combination of movement and interaction models. Each robot runs the prediction algorithms for each dynamic obstacle. Let $\mathbf{p}^{robot}_{hist}$ be the position and $\mathbf{v}^{robot}_{hist}$ be the velocity history of the robot collected synchronously with $\mathbf{p}^{dyn}_{hist}$ and $\mathbf{v}^{dyn}_{hist}$ for the dynamic obstacle.

[¶] During prediction, we assume that we have access to the algorithm computing $\mathbf{r}$ from $\mathbf{p}^{dyn}$ and $\hat{\mathbf{c}}$ as there are infinitely many vectors perpendicular to $\mathbf{p}^{dyn} - \mathbf{c}$ when $d \geq 3$.

[‖] Note that the interaction model we use does not utilize the velocity $\mathbf{v}^{robot}$ of the robot, while our planner allows it. We choose to use this interaction model for easier online prediction of model parameters as our paper is not focused on prediction algorithms.

**Goal attractive repulsive predictor** Assuming the dynamic obstacle moves according to goal attractive movement model $\mathcal{M}_g(\mathbf{p}^{dyn}|\hat{\mathbf{g}}, \hat{s})$ and repulsive interaction model $\mathcal{I}_r(\mathbf{p}^{dyn}, \tilde{\mathbf{v}}^{dyn}, \mathbf{p}^{robot}, \mathbf{v}^{robot}|\hat{f})$, we estimate parameters $\hat{\mathbf{g}}$, $\hat{s}$ and $\hat{f}$.

We solve two consecutive quadratic programs (QP): i) one for goal $\hat{\mathbf{g}}$ estimation, ii) one for desired speed $\hat{s}$ and repulsion strength $\hat{f}$ estimation[**].

**Goal estimation.** Let $\mathbf{p}^{dyn}_{hist,k}$ and $\mathbf{v}^{dyn}_{hist,k}$ be the $k$th elements of $\mathbf{p}^{dyn}_{hist}$ and $\mathbf{v}^{dyn}_{hist}$ respectively. $\mathbf{p}^{dyn}_{hist,k} + t_k\mathbf{v}^{dyn}_{hist,k}, t_k \geq 0$ is the ray the dynamic obstacle would have followed if it did not change its velocity after $k$th sample. We estimate the goal position $\hat{\mathbf{g}}$ of the dynamic obstacle by computing the point whose average squared distance to these rays is minimal:

$$\min_{\hat{\mathbf{g}}, t_1, \ldots, t_K} \frac{1}{K} \sum_{k=1}^{K} \left\| \mathbf{p}^{dyn}_{hist,k} + t_k \mathbf{v}^{dyn}_{hist,k} - \hat{\mathbf{g}} \right\|_2^2, \text{s.t.}$$

$$t_k \geq 0 \; \forall k \in \{1, \ldots, K\}$$

where $K$ is the number of recorded position/velocity pairs.

**Desired speed and repulsion strength estimation.** Assuming the dynamic obstacle moves according to goal attractive repulsive behavior model, its estimated velocity at step $k$ is:

$$\begin{aligned}\hat{\mathbf{v}}^{dyn}_k &= \mathcal{I}_r(\mathbf{p}^{dyn}_{hist,k}, \mathcal{M}_g(\mathbf{p}^{dyn}_{hist,k}|\hat{\mathbf{g}}, \hat{s}), \mathbf{p}^{robot}_{hist,k}, \mathbf{v}^{robot}_{hist,k}|\hat{f}) \\ &= \frac{\hat{\mathbf{g}} - \mathbf{p}^{dyn}_{hist,k}}{\left\|\hat{\mathbf{g}} - \mathbf{p}^{dyn}_{hist,k}\right\|_2} \hat{s} + \frac{\left(\mathbf{p}^{dyn}_{hist,k} - \mathbf{p}^{robot}_{hist,k}\right) \hat{f}}{\left\|\mathbf{p}^{dyn}_{hist,k} - \mathbf{p}^{robot}_{hist,k}\right\|_2^3}\end{aligned}$$

[**] While joint estimation of $\hat{\mathbf{g}}$, $\hat{s}$ and $\hat{f}$ would be more accurate, we choose to estimate the parameters in two steps so that the individual problems are QPs, and can be solved quickly.

We minimize the average squared distance between estimated and sensed dynamic obstacle velocities to estimate $\hat{s}$ and $\hat{f}$:

$$\min_{\hat{s},\hat{f}} \frac{1}{K} \sum_{k=1}^{K} \left\| \hat{\mathbf{v}}_k^{dyn} - \mathbf{v}_{hist,k}^{dyn} \right\|_2^2 .$$

**Constant velocity repulsive predictor** Assuming the dynamic obstacle moves according to the constant velocity movement model $\mathcal{M}_c(\mathbf{p}^{dyn}|\hat{\mathbf{v}})$ and repulsive interaction model $\mathcal{I}_r(\mathbf{p}^{dyn}, \tilde{\mathbf{v}}^{dyn}, \mathbf{p}^{robot}, \mathbf{v}^{robot}|\hat{f})$, we estimate the parameters $\hat{\mathbf{v}}$ and $\hat{f}$.

We solve a single QP to estimate both parameters. Assuming the dynamic obstacle moves according to constant velocity repulsive behavior model, its estimated velocity at step $k$ is

$$\begin{aligned} \hat{\mathbf{v}}_k^{dyn} &= \mathcal{I}_r(\mathbf{p}_{hist,k}^{dyn}, \mathcal{M}_c(\mathbf{p}_{hist,k}^{dyn}|\hat{\mathbf{v}}), \mathbf{p}_{hist,k}^{robot}, \mathbf{v}_{hist,k}^{robot}) \\ &= \hat{\mathbf{v}} + \frac{\left(\mathbf{p}_{hist,k}^{dyn} - \mathbf{p}_{hist,k}^{robot}\right)\hat{f}}{\left\|\mathbf{p}_{hist,k}^{dyn} - \mathbf{p}_{hist,k}^{robot}\right\|_2^3} \end{aligned}$$

We minimize the average squared distance between estimated and sensed velocities to estimate $\hat{\mathbf{v}}$ and $\hat{f}$:

$$\min_{\hat{\mathbf{v}},\hat{f}} \frac{1}{K} \sum_{k=1}^{K} \left\| \hat{\mathbf{v}}_k^{dyn} - \mathbf{v}_{hist,k}^{dyn} \right\|_2^2 .$$

**Rotating repulsive predictor** Assuming the dynamic obstacle moves according to the rotating movement model $\mathcal{M}_r(\mathbf{p}^{dyn}|\hat{\mathbf{c}}, \hat{s})$ and repulsive interaction model $\mathcal{I}_r(\mathbf{p}^{dyn}, \tilde{\mathbf{v}}^{dyn}, \mathbf{p}^{robot}, \mathbf{v}^{robot}|\hat{f})$, we estimate the parameters $\hat{\mathbf{c}}$, $\hat{s}$ and $\hat{f}$.

We solve two consecutive optimization problems: i) one for rotation center $\hat{\mathbf{c}}$ estimation, ii) one for desired speed $\hat{s}$ and repulsion strength $\hat{f}$ estimation[††].

[††] We separate estimation to two steps because of a similar reason with the goal attractive repulsive predictor. The first problem is a linear program (LP) and the second one is a QP, both of which allowing fast online solutions.

**Rotation center estimation.** We estimate $\hat{\mathbf{c}}$ by minimizing averaged dot product between sensed velocities and the vectors pointing from rotation center to the sensed positions. The reasoning is that if the obstacle is rotating around a point $\hat{\mathbf{c}}$, its velocity should always be perpendicular to the vector connecting $\hat{\mathbf{c}}$ to its position. The LP we solve is:

$$\min_{\hat{\mathbf{c}}} \frac{1}{K} \sum_{i=1}^{K} |\mathbf{v}_{hist,k}^{dyn} \cdot (\mathbf{p}_{hist,k}^{dyn} - \hat{\mathbf{c}})|.$$

**Desired speed and repulsion strength estimation.** Assuming the dynamic obstacle moves according to rotating repulsive behavior model, its estimated velocity at step $k$ is

$$\begin{aligned}\hat{\mathbf{v}}_k^{dyn} &= \mathcal{I}_r(\mathbf{p}_{hist,k}^{dyn}, \mathcal{M}_r(\mathbf{p}_{hist,k}^{dyn} | \hat{\mathbf{c}}, \hat{s}), \mathbf{p}_{hist,k}^{robot}, \mathbf{v}_{hist,k}^{robot}) \\ &= \frac{\mathbf{r}_k^{dyn}}{\left\|\mathbf{r}_k^{dyn}\right\|_2} \hat{s} + \frac{\left(\mathbf{p}_{hist,k}^{dyn} - \mathbf{p}_{hist,k}^{robot}\right) \hat{f}}{\left\|\mathbf{p}_{hist,k}^{dyn} - \mathbf{p}_{hist,k}^{robot}\right\|_2^3}\end{aligned}$$

where $\mathbf{r}_k^{dyn}$ is a perpendicular vector to $\mathbf{p}_{hist,k}^{dyn} - \hat{\mathbf{c}}$, computed with the known algorithm. We minimize the average squared distance between estimated and sensed velocities to estimate $\hat{s}$ and $\hat{f}$:

$$\min_{\hat{s}, \hat{f}} \frac{1}{K} \sum_{k=1}^{K} \left\| \hat{\mathbf{v}}_k^{dyn} - \mathbf{v}_{hist,k}^{dyn} \right\|_2^2.$$

**Assigning probabilities to behavior models** Each robot runs the three predictors for each dynamic obstacle. For each predicted behavior model $(\mathcal{M}_j, \mathcal{I}_j)$, $j \in \{1, 2, 3\}$, we compute the average estimation error $E_j$ as the average $L^2$ norm between the predicted and the actual velocities:

$$\frac{1}{K} \sum_{k=1}^{K} \|\mathbf{v}_{hist,k}^{dyn} - \mathcal{I}_j(\mathbf{p}_{hist,k}^{dyn}, \mathcal{M}_j(\mathbf{p}_{hist,k}^{dyn}), \mathbf{p}_{hist,k}^{robot}, \mathbf{v}_{hist,k}^{robot})\|_2$$

We compute the softmax of errors $E_j$ with base $b$ where $0 < b < 1$, and use them as probabilities, i.e., $p_j^{dyn} = \frac{b^{E_j}}{\sum_{k=1}^{3} b^{E_k}}$.

#### 6.4.1.3 Metrics

We run each single robot simulation experiment 1000 times and each multi robot simulation experiment 100 times in randomized environments in performance evaluation of our algorithm (Section 6.4.2). In baseline comparisons, we run each single-robot simulation experiment 250 times and each multi-robot simulation experiment 100 times (Section 6.4.3). In each experiment, the robots are tasked with navigating from their start to goal positions through an environment with static and dynamic obstacles. There are nine metrics that we report for each experiment, averaged over all robots in all runs.

- **Success rate:** Ratio of robots that navigate from their start positions to their goal positions successfully without any collisions.
- **Collision rate:** Ratio of robots that collide with a static or a dynamic obstacle or a teammate at least once.
- **Deadlock rate:** Ratio of robots that deadlock, i.e., do not reach its goal position.
- **Static obstacle collision rate:** Ratio of robots that collide with a static obstacle at least once.
- **Dynamic obstacle collision rate:** Ratio of robots that collide with a dynamic obstacle at least once.
- **Teammate collision rate:** Ratio of robots that collide with a teammate at least once.
- **Average navigation duration:** Average time it takes for robot to navigate from its start position to its goal position across no-deadlock no-collision robots.
- **Planning fail rate:** Ratio of failing planning iterations over all planning iterations of all robots in all runs.

- **Average planning duration:** Average planning duration over all planning iterations of all robots in all runs.

#### 6.4.1.4 Fixed Parameters and Run Randomization

Here, we describe fixed parameters across all experiments and the parameters that are randomized in all experiments. Fixed parameters are shared by each robot $i$, and randomized parameters are randomized the same way for all robots $i$.

**Fixed parameters.** We set $p_i^{min} = 0.1$, $\tilde{\gamma}_i^1 = 5.0\,\frac{\mathrm{m}}{\mathrm{s}}$, $\tilde{\tau}_i = 2.0\,\mathrm{s}$, $\alpha_i = 1.5$, $h_{i,l} = 13$ for all $l$, $\theta_{i,l}$ and $\beta_{i,l}$ $10, 20, 30$ for the first three pieces, and $40$ for the remaining pieces, $\lambda_{i,1} = 2.8$, $\lambda_{i,2} = 4.2$, $\lambda_{i,4} = 0.2$, and $\lambda_{i,l} = 0$ for all other degrees. The FORWARD actions of search are ($2.0\,\frac{\mathrm{m}}{\mathrm{s}}$, $0.5\,\mathrm{s}$), ($3.5\,\frac{\mathrm{m}}{\mathrm{s}}$, $0.5\,\mathrm{s}$), and ($4.5\,\frac{\mathrm{m}}{\mathrm{s}}$, $0.5\,\mathrm{s}$).

In all runs of all experiments, robots navigate in random forest environments, i.e., static obstacles are tree-like objects. The forest has $15\,\mathrm{m}$ radius, and trees are $6\,\mathrm{m}$ high and have a radius of $0.5\,\mathrm{m}$. The forest is centered around the origin. The octree structure has a resolution of $0.5\,\mathrm{m}$. The density $\rho$ of the forest, i.e., ratio of occupied cells in the octree within the forest. It is set differently in each experiment.

**Run randomization.** We randomize the following parameters in all runs of each experiment in the same way.

*Dynamic obstacle randomization.* We randomize the axis aligned box collision shape of each dynamic obstacle by randomizing its size in each dimension uniformly in $[1\,\mathrm{m}, 4\,\mathrm{m}]$. The dynamic obstacle's initial position is uniformly sampled in the box $A$ with minimum corner $\begin{bmatrix}-12\,\mathrm{m} & -12\,\mathrm{m} & -2\,\mathrm{m}\end{bmatrix}^\top$ and maximum corner $\begin{bmatrix}12\,\mathrm{m} & 12\,\mathrm{m} & 6\,\mathrm{m}\end{bmatrix}^\top$. We sample the movement model of the obstacle among goal attractive, constant velocity, and rotating models. If goal attractive movement model is sampled, we sample its goal position $\hat{\mathbf{g}}$ uniformly in the box $A$. If rotating model is sampled, we sample the rotation center $\hat{\mathbf{c}}$ in the box with minimum corner $\begin{bmatrix}-0.5\,\mathrm{m} & -0.5\,\mathrm{m} & 0.0\,\mathrm{m}\end{bmatrix}^\top$ and maximum corner $\begin{bmatrix}0.5\,\mathrm{m} & 0.5\,\mathrm{m} & 6.0\,\mathrm{m}\end{bmatrix}^\top$. The desired speed $\hat{s}$ of the obstacles is sampled differently in each experiment. If constant velocity model is sampled, the velocity $\hat{\mathbf{v}}$ is set by uniformly sampling a unit direction vector and multiplying it with $\hat{s}$. The interaction model is always the repulsive model. The repulsion strength $\hat{f}$ is set/randomized differently in each experiment.[‡‡] Each dynamic obstacle changes its velocity every decision making period, which is sampled uniformly from $[0.1\,\mathrm{s}, 0.5\,\mathrm{s}]$. Each dynamic obstacle runs its movement model to get its desired velocity, and runs its interaction model for each robot, and executes the average velocity, i.e., dynamic obstacles interact with all robots, while an individual robot models only the interactions with itself. The number of dynamic obstacles $\#^D$ is set differently in each experiment.

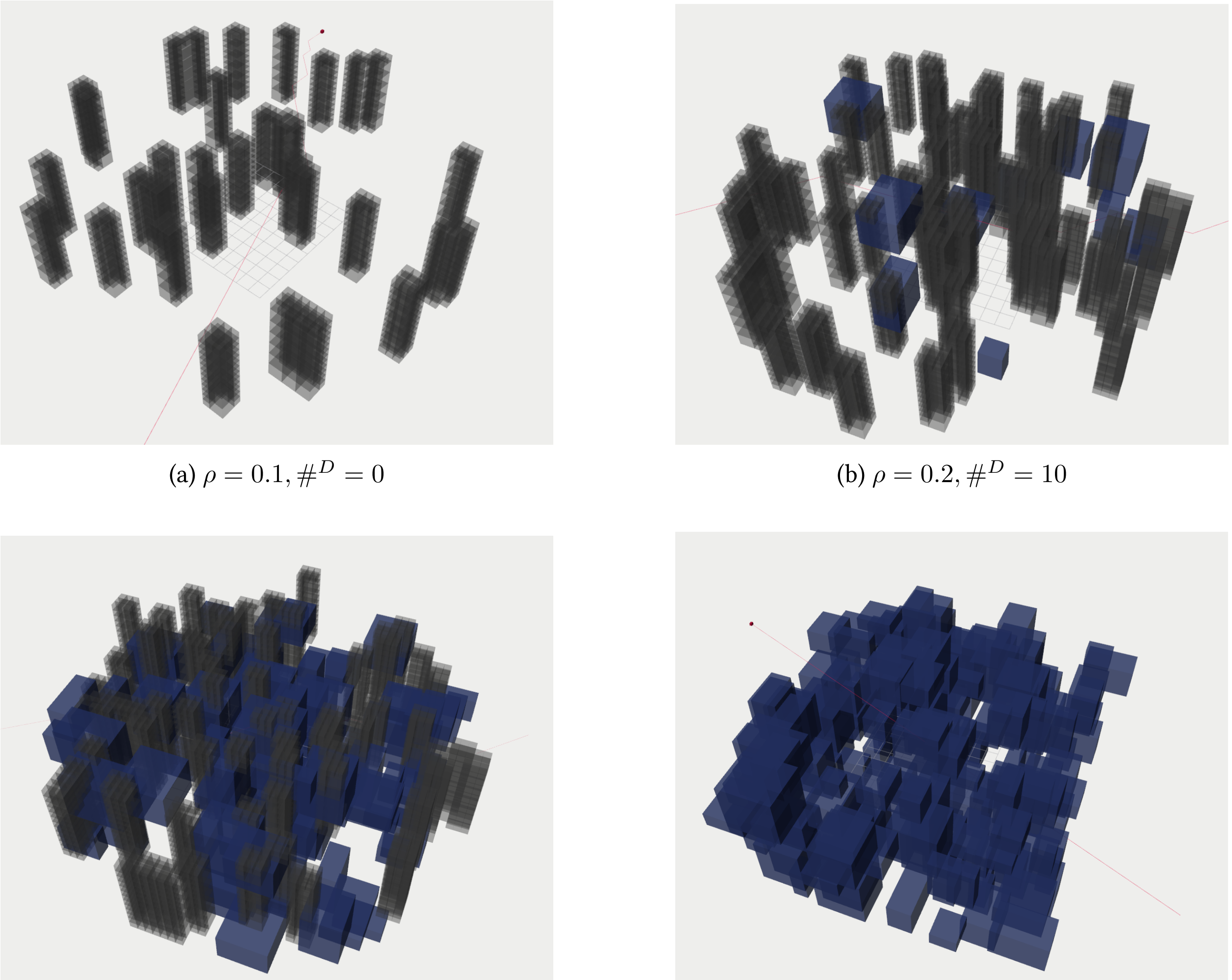
*Robot randomization.* We randomize the axis aligned box collision shape of each robot by randomizing its size in each dimension uniformly in $[0.2\,\mathrm{m}, 0.3\,\mathrm{m}]$. We sample the replanning period of each robot uniformly in $[0.2\,\mathrm{s}, 0.4\,\mathrm{s}]$. First robot's start positions is selected randomly around the forest on a circle with radius $21.5\,\mathrm{m}$ at height $2.5\,\mathrm{m}$. Other robots are placed on the circle with equal arc length distance between them. The goal positions are set to the antipodal points of the start positions on the circle. The time limit of the search $T_i^{search}$, the desired time horizon $\tau_i$, number of robots $\#^R$, continuity degree $c_i$, maximum velocity $\gamma_i^1$ and maximum acceleration $\gamma_i^2$ are set differently in each experiment.

Sample environments with varying static obstacle densities and number of dynamic obstacles are shown in Figure 6.6.

#### 6.4.1.5 Simulating Communication Imperfections

Robots communicate with each other in order to update tail time points, whenever they successfully plan. We simulate imperfections in the communication medium. We model message delays as an exponential

[‡‡] Note that, in reality, there is no necessity that the dynamic obstacles move according to the behavior models that we define. The reason we choose to move dynamic obstacles according to these behavior models is so that at least one of our predictors assume the correct model for the obstacle. The prediction is still probabilistic, in the sense that we generate three hypotheses for each dynamic obstacle and assign probabilities.



(a) $\rho = 0.1, \#^D = 0$

(b) $\rho = 0.2, \#^D = 10$

(c) $\rho = 0.2, \#^D = 100$

(d) $\rho = 0.0, \#^D = 200$

Figure 6.6: Sample environments generated during run randomization with varying static obstacle density $\rho$ and number of dynamic obstacles $\#^D$. Gray objects are static obstacles representing the forest. Blue objects are dynamic obstacles.

distribution with mean $\delta$ seconds and message drops with a Bernoulli distribution with drop probability $\kappa$ similar to Chapter 5.

$$P_{delay}(x;\delta) = \begin{cases} \delta e^{-\delta x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$P_{drop}(x;\kappa) = \begin{cases} \kappa & x \text{ is drop} \\ 1-\kappa & x \text{ is no-drop.} \end{cases}$$

Each message a robot broadcasts is dropped with probability $\kappa$, and applied a delay sampled from the delay distribution if it is not dropped. Message re-orderings are naturally introduced by random delays.

### 6.4.2 Performance under Different Configurations and Environments in Simulations

We evaluate the performance of our algorithm when it is used in different environments and configurations. In addition, we show its performance under different configurations.

#### 6.4.2.1 Desired Trajectory Quality

To evaluate the effects of desired trajectory quality to navigation performance, we compare two different approaches to compute desired trajectories in single-robot experiments. In the first, which we call the *without prior* strategy, we set the desired trajectory of the robot to the straight line segment connecting its start position to its goal position. In the second, which we call the *with prior* strategy, we set the desired trajectory of the robot to the shortest path between its start position to its goal position avoiding static obstacles.

During these experiments, we set $T_i^{search} = 75\,\mathrm{ms}$. The desired planning horizon $\tau_i$ is set to $2.5\,\mathrm{s}$. We set continuity degree $c_i = 2$, maximum velocity $\gamma_i^1 = 10\,\frac{\mathrm{m}}{\mathrm{s}}$ and maximum acceleration $\gamma_i^2 = 15\,\frac{\mathrm{m}}{\mathrm{s}^2}$.

Table 6.1: Effects of desired trajectory quality

| **Exp. #** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|
| **Prior** | w/o | with | w/o | with | w/o | with |
| $\boldsymbol{\rho}$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.3 | 0.3 |
| $\boldsymbol{\#^D}$ | 0 | 0 | 25 | 25 | 50 | 50 |
| **succ. rate** | 0.945 | 0.994 | 0.885 | 0.952 | 0.596 | 0.866 |
| **coll. rate** | 0.016 | 0.000 | 0.094 | 0.045 | 0.319 | 0.126 |
| **deadl. rate** | 0.040 | 0.006 | 0.028 | 0.004 | 0.139 | 0.011 |
| **s. coll. rate** | 0.016 | 0.000 | 0.043 | 0.001 | 0.207 | 0.015 |
| **d. coll. rate** | 0.000 | 0.000 | 0.062 | 0.044 | 0.217 | 0.118 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 27.27 | 28.42 | 27.76 | 28.51 | 31.19 | 30.86 |
| **pl. fail rate** | 0.060 | 0.043 | 0.066 | 0.052 | 0.093 | 0.076 |
| **avg. pl. dur. [ms]** | 88.02 | 41.14 | 106.27 | 61.42 | 170.56 | 90.41 |

The desired speed $\hat{s}$ of dynamic obstacles is uniformly sampled in interval $[0.5\,\frac{\text{m}}{\text{s}}, 1.0\,\frac{\text{m}}{\text{s}}]$, and repulsion strength $\hat{f}$ uniformly sampled in interval $[0.2, 0.5]$ for each obstacle.

The duration of the desired trajectory is set assuming that robot desires to follow it with $\frac{1}{3}$ with its maximum speed $\tilde{\gamma}_i^1$ for search.

We control the density $\rho$ of the forest, the number $\#^D$ of dynamic obstacles, and whether the desired trajectory is computed using with or without the knowledge of static obstacles a priori, and report the metrics.

The results are summarized in Table 6.1. With prior strategy results in higher success rates, lower collision rates, deadlock rate, planning failure rate and planning duration in all pairs of cases (cyan vs. red values). Average navigation duration of without prior strategy is slightly lower than that of the with prior strategy (orange vs. magenta values), which we attribute to the lower success rate of the without prior strategy, in which, hard navigation tasks are failed and the average navigation duration is computed over easier tasks. These suggests the necessity of providing good desired trajectories.

In all following experiments, we use the *with prior* strategy, and provide the desired trajectory by computing the shortest path avoiding only static obstacles and setting its duration assuming that robot desires to follow it with $\frac{1}{3}$ of its maximum speed $\tilde{\gamma}_i^1$ for search.

Table 6.2: Effects of enforced continuity degree

| **Exp. #** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|
| $c_i$ | 0 | 1 | 2 | 3 | 4 | 5 |
| **succ. rate** | 0.980 | 0.986 | 0.971 | 0.930 | 0.893 | 0.830 |
| **coll. rate** | 0.020 | 0.014 | 0.023 | 0.054 | 0.059 | 0.057 |
| **deadl. rate** | 0.000 | 0.000 | 0.006 | 0.018 | 0.053 | 0.119 |
| **s. coll. rate** | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 |
| **d. coll. rate** | 0.019 | 0.014 | 0.023 | 0.054 | 0.059 | 0.056 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 30.87 | 29.23 | 28.57 | 28.50 | 28.31 | 28.29 |
| **pl. fail rate** | 0.001 | 0.012 | 0.041 | 0.093 | 0.121 | 0.154 |
| **avg. pl. dur. [ms]** | 49.83 | 51.79 | 53.36 | 55.05 | 53.77 | 54.82 |

#### 6.4.2.2 Enforced Degree of Continuity

The degree of continuity $c_i$ we enforce in the trajectory optimization stage determines the smoothness of the resulting trajectory. The search step enforces only position continuity and degree of continuity greater than that is enforced solely by the trajectory optimizer.

To evaluate the effects of enforced degree of continuity to navigation performance, we compare the navigation metrics when different degrees are enforced in single-robot experiments.

During these experiments, we do not set any maximum derivative magnitudes to evaluate the effects of only the degree of continuity. We set $\tau_i = 2.5\,\mathrm{s}$, $T_i^{search} = 75\,\mathrm{ms}$, $\rho = 0.2$, and $\#^D = 25$. $\hat{s}$ is sampled in interval $[0.5\,\frac{\mathrm{m}}{\mathrm{s}}, 1.0\,\frac{\mathrm{m}}{\mathrm{s}}]$, and repulsion strength $\hat{f}$ is sampled in interval $[0.2, 0.5]$ uniformly for each obstacle.

We control the enforced degree of continuity $c_i$, and report the metrics.

The results are summarized in Table 6.2. In general, success rates decrease and deadlock and collision rates increase as the enforced degree of continuity increases. Interestingly, success rate from degree 0 to degree 1 increases (red values in **succ. rate** row), which we adhere to the non-smooth changes in robot's position when $c_i = 0$, causing hard to predict interactions between dynamic obstacles and the robot. Navigation duration is not significantly affected by the enforced degree of continuity. Planning failure rate increases with the enforced degree of continuity as seen in cyan values in **pl. fail rate** row, which is the main cause of collision rate increase. Average planning duration is not significantly affected by the enforced degree of continuity, because the number of continuity constraints are insignificant compared to safety constraints.

In the remaining experiments, we set $c_i = 2$ for all robots $i$, i.e., enforce continuity up to acceleration, unless explicitly stated otherwise.

#### 6.4.2.3 Dynamic Limits

The discrete search stage limits maximum speed of the resulting discrete plan by enforcing maximum speed $\tilde{\gamma}_i^1$ (which is set to $5.0\,\frac{\mathrm{m}}{\mathrm{s}}$ during the experiments). While this encourages limiting the maximum speed of the resulting trajectory, trajectory optimization stage actually enforces the dynamic limits.

To evaluate the effects of different dynamic limits $\gamma_i^k$, we compare the navigation metrics when different dynamic limits are enforced in single-robot experiments.

During these experiments, we set $T_i^{search} = 75\,\mathrm{ms}$, $\rho = 0.2$, and $\#^D = 25$. The desired speed $\hat{s}$ of dynamic obstacles is sampled in interval $[0.5\,\frac{\mathrm{m}}{\mathrm{s}}, 1.0\,\frac{\mathrm{m}}{\mathrm{s}}]$, and repulsion strength $\hat{f}$ is sampled in interval $[0.2, 0.5]$ uniformly for each obstacle.

We control the maximum velocity $\gamma_i^1$ and maximum acceleration $\gamma_i^2$, and report the metrics. We do not decrease $\gamma_i^1$ below $10\,\frac{\mathrm{m}}{\mathrm{s}}$, because search has a speed limit of $\tilde{\gamma}_i^1 = 5\,\frac{\mathrm{m}}{\mathrm{s}}$, and setting $\gamma_i^1 = 10\,\frac{\mathrm{m}}{\mathrm{s}}$ enforces $\frac{10}{\sqrt{3}}\frac{m}{s} \approx 5.77\,\frac{\mathrm{m}}{\mathrm{s}}$ speed limit in all dimensions. If a lower speed limit is required, maximum speed limit $\tilde{\gamma}_i^1$ of search should also be decreased.

The results are given in Table 6.3. Unsurprisingly, collision, deadlock and planning failure rates increase and success rate decreases as the dynamic limits get more constraining. Since the velocity is limited during the discrete search step explicitly, decreasing $\gamma_i^1$ has a relatively smaller effect on metrics (cyan values in **succ. rate** row) compared to decreasing $\gamma_i^2$ (red values in **succ. rate** row).

Table 6.3: Effects of dynamic limits

| **Exp. #** | **13** | **14** | **15** | **16** | **17** | **18** |
|---|---|---|---|---|---|---|
| $\gamma_i^1[\frac{m}{s}]$ | $\infty$ | 15 | 10 | 10 | 10 | 10 |
| $\gamma_i^2[\frac{m}{s^s}]$ | $\infty$ | 20 | 20 | 15 | 10 | 7 |
| **succ. rate** | 0.977 | 0.961 | 0.961 | 0.955 | 0.897 | 0.844 |
| **coll. rate** | 0.022 | 0.035 | 0.037 | 0.037 | 0.101 | 0.148 |
| **deadl. rate** | 0.001 | 0.004 | 0.002 | 0.008 | 0.002 | 0.019 |
| **s. coll. rate** | 0.002 | 0.002 | 0.000 | 0.000 | 0.007 | 0.013 |
| **d. coll. rate** | 0.021 | 0.033 | 0.037 | 0.037 | 0.095 | 0.144 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 28.55 | 28.57 | 28.51 | 28.56 | 28.49 | 28.47 |
| **pl. fail rate** | 0.040 | 0.046 | 0.045 | 0.053 | 0.069 | 0.093 |
| **avg. pl. dur. [ms]** | 54.66 | 63.14 | 62.44 | 61.90 | 59.84 | 60.34 |

Table 6.4: Effects of repulsion strength

| **Exp. #** | **19** | **20** | **21** | **22** | **23** | **24** |
|---|---|---|---|---|---|---|
| $\hat{f}$ | $-0.5$ | 0 | 0.5 | 1.5 | 3.0 | 6.0 |
| **succ. rate** | 0.818 | 0.897 | 0.915 | 0.936 | 0.970 | 0.991 |
| **coll. rate** | 0.182 | 0.102 | 0.084 | 0.064 | 0.030 | 0.009 |
| **deadl. rate** | 0.001 | 0.004 | 0.002 | 0.000 | 0.000 | 0.000 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.182 | 0.102 | 0.084 | 0.064 | 0.030 | 0.009 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 26.71 | 26.72 | 26.71 | 26.71 | 26.69 | 26.68 |
| **pl. fail rate** | 0.029 | 0.030 | 0.026 | 0.022 | 0.019 | 0.014 |
| **avg. pl. dur. [ms]** | 49.72 | 50.63 | 50.55 | 46.98 | 43.17 | 37.76 |

In the remaining experiments, we set $\gamma_i^1 = 10\,\frac{\mathrm{m}}{\mathrm{s}}$ and $\gamma_i^2 = 15\,\frac{\mathrm{m}}{\mathrm{s}^2}$ for all robots $i$, unless explicity stated otherwise.

#### 6.4.2.4 Repulsive Dynamic Obstacle Interactivity

The evaluate the behavior our planner with different levels of repulsive interactivity of dynamic obstacles, we compare navigation metrics when dynamic obstacles use different repulsion strengths $\hat{f}$ in single-robot experiments. During these experiments, we set $T_i^{search} = 75\,\mathrm{ms}$, $\tau_i = 2.5\,\mathrm{s}$, $\rho_i = 0$, and $\#^D = 50$.

We control the repulsion strength $\hat{f}$ and report the metrics.

The results are summarized in Table 6.4. In general, as the repulsive interactivity increases, the collision and deadlock rates decrease, and the success rate increases. In addition, the average planning duration

Table 6.5: Effects of dynamic obstacle desired speed

| **Exp. #** | **25** | **26** | **27** | **28** | **29** | **30** |
|---|---|---|---|---|---|---|
| $\hat{s}[\frac{m}{s}]$ | 0.5 | 0.75 | 1.0 | 1.5 | 2.5 | 5.0 |
| **succ. rate** | 0.937 | 0.914 | 0.900 | 0.864 | 0.717 | 0.272 |
| **coll. rate** | 0.062 | 0.086 | 0.099 | 0.136 | 0.283 | 0.728 |
| **deadl. rate** | 0.002 | 0.002 | 0.002 | 0.001 | 0.000 | 0.000 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.062 | 0.086 | 0.099 | 0.136 | 0.283 | 0.728 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 26.68 | 26.74 | 26.72 | 26.67 | 26.68 | 27.36 |
| **pl. fail rate** | 0.026 | 0.026 | 0.028 | 0.026 | 0.031 | 0.038 |
| **avg. pl. dur. [ms]** | 48.76 | 51.03 | 48.94 | 36.10 | 30.83 | 25.20 |

decreases as the repulsive interactivity increases, because the problem gets easier for the robot if dynamic obstacles take some responsibility of collision avoidance even with a simple repulsion rule. In experiment 7, repulsion strength is set to $-0.5$ (cyan value), causing dynamic obstacles to get attracted to the robot, i.e., they move towards the robot. Even in that case, the robot can achieve a high success rate, i.e., 0.818, as it models dynamic obstacle interactivity.

In the remaining experiments, we sample $\hat{f}$ in $[0.2, 0.5]$, unless explicitly states otherwise.

#### 6.4.2.5 Dynamic Obstacle Desired Speed

To evaluate the behavior of our planner in environments with dynamic obstacles having different desired speeds, we control the desired speed $\hat{s}$ and report the metrics in single-robot experiments.

During these experiments, we set $\rho_i = 0$, $\#^D = 50$, $T_i^{search} = 75\,\text{ms}$, $\tau_i = 2.5\,\text{s}$.

The results are given in Table 6.5. The collision rates increase and the success rate decreases as the desired speed of dynamic obstacles increase. Deadlock rates are close to 0 in all cases (cyan values in **deadl. rate** row), because there are no static obstacles in the environment and dynamic obstacles eventually move away from the robot. The average planning duration for the robot decreases as the desired speed increases as it can be seen in red values in the **avg. pl. dur.** row. We attribute this to the dynamic obstacles with constant velocity. As the desired speed of the dynamic obstacles increase, constant velocity obstacles leave

Table 6.6: Effects of number of dynamic obstacles

| **Exp. #** | **31** | **32** | **33** | **34** | **35** | **36** |
|---|---|---|---|---|---|---|
| $\#^D$ | 10 | 25 | 50 | 75 | 100 | 200 |
| **succ. rate** | 0.997 | 0.973 | 0.897 | 0.838 | 0.747 | 0.402 |
| **coll. rate** | 0.003 | 0.027 | 0.102 | 0.161 | 0.252 | 0.598 |
| **deadl. rate** | 0.000 | 0.000 | 0.004 | 0.007 | 0.017 | 0.087 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.003 | 0.027 | 0.102 | 0.161 | 0.252 | 0.598 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 26.69 | 26.69 | 26.72 | 26.72 | 26.76 | 28.05 |
| **pl. fail rate** | 0.005 | 0.013 | 0.027 | 0.037 | 0.049 | 0.098 |
| **avg. pl. dur. [ms]** | 17.88 | 30.72 | 50.59 | 65.90 | 76.68 | 104.70 |

the environment faster, causing the robot to avoid lesser number of obstacles, albeit faster obstacles. The planning failure rate also increases with the increase in dynamic obstacle desired speeds as it can be seen in orange values in **pl. fail rate** row, because the robot has to take abrupt actions more frequently to avoid faster obstacles, causing the pipeline to fail because of dynamic limits of the robot.

In the remaining experiments we sample desired speed $\hat{s}$ uniformly in interval $[0.5\,\frac{\mathrm{m}}{\mathrm{s}}, 1.0\,\frac{\mathrm{m}}{\mathrm{s}}]$.

#### 6.4.2.6 Number of Dynamic Obstacles

To evaluate the behavior of our planner in environments with different number of dynamic obstacles, we control the number of dynamic obstacles $\#^D$, and report the metrics in single-robot experiments.

During these experiments, we set $T_i^{search} = 75\,\mathrm{ms}$, $\tau_i = 2.5\,\mathrm{s}$, and $\rho = 0$.

The results are given in Table 6.6. Expectedly, as the number of dynamic obstacles increase, collision, deadlock, and planning failure rates increase, and the success rate decreases. Average navigation duration is not affected until the number of dynamic obstacles is increased from 100 to 200 as it can be seen in red values in **avg. nav. dur** row. Even then, increase in navigation duration is small.

Table 6.7: Effects of static obstacle density

| **Exp. #** | **37** | **38** | **39** | **40** | **41** | **42** |
|---|---|---|---|---|---|---|
| $\rho$ | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 |
| **succ. rate** | 0.998 | 0.998 | 0.999 | 0.985 | 0.984 | 0.991 |
| **coll. rate** | 0.000 | 0.000 | 0.001 | 0.004 | 0.005 | 0.000 |
| **deadl. rate** | 0.002 | 0.002 | 0.000 | 0.012 | 0.013 | 0.009 |
| **s. coll. rate** | 0.000 | 0.000 | 0.001 | 0.004 | 0.005 | 0.000 |
| **d. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 27.52 | 28.46 | 30.28 | 33.42 | 35.13 | 35.35 |
| **pl. fail rate** | 0.012 | 0.039 | 0.068 | 0.063 | 0.026 | 0.013 |
| **avg. pl. dur. [ms]** | 34.33 | 41.73 | 46.73 | 50.12 | 42.74 | 38.98 |

#### 6.4.2.7 Static Obstacle Density

We control the static obstacle density $\rho$ and report the metrics to evaluate the behavior of our planner in environments with different static obstacle densities in single-robot experiments.

We set $\tau_i = 2.5\,\text{s}$, $T_i^{search} = 75\,\text{ms}$. We set number of dynamic obstacles $\#^D = 0$ to evaluate the affects of static obstacles only. Note that, during these experiments, desired trajectory is set using the *with prior* strategy, meaning that the desired trajectory avoids all static obstacles. We do not mock sensing imperfections of static obstacles.

The results are given in Table 6.7. The effect of $\rho$ to success, collision, and deadlock rates is small. This stems from the fact that desired trajectory is already good, and our planner has to track it as close as possible. Average navigation duration increases with $\rho$ because the environment gets more complicated, causing the robot to traverse longer paths. From $\rho = 0.1$ to $\rho = 0.4$, planning failure rate and planning duration increase, while from $\rho = 0.4$ to $\rho = 0.6$, they decrease as it can be seen in cyan and red values in **pl. fail rate** and **avg. pl. dur** rows. The reason is the fact that as the environment density increases, it is less likely that the shortest path from the start position to the goal position goes through the forest, as there is no path through the forest. Instead, desired trajectory avoids the forest all together. This causes planning to become easier.

Table 6.8: Effects of the time limit of search with dynamic limits

| **Exp. #** | **43** | **44** | **45** | **46** | **47** | **48** |
|---|---|---|---|---|---|---|
| $T_i^{search}[ms]$ | 75 | 150 | 300 | 600 | 1000 | 2000 |
| **succ. rate** | 0.655 | 0.731 | 0.708 | 0.730 | 0.682 | 0.700 |
| **coll. rate** | 0.343 | 0.268 | 0.289 | 0.269 | 0.315 | 0.296 |
| **deadl. rate** | 0.013 | 0.013 | 0.014 | 0.018 | 0.026 | 0.022 |
| **s. coll. rate** | 0.031 | 0.016 | 0.005 | 0.003 | 0.001 | 0.000 |
| **d. coll. rate** | 0.339 | 0.268 | 0.288 | 0.269 | 0.315 | 0.296 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 29.15 | 29.11 | 29.05 | 28.81 | 420.97 | 28.83 |
| **pl. fail rate** | 0.078 | 0.082 | 0.089 | 0.107 | 0.119 | 0.130 |
| **avg. pl. dur. [ms]** | 108.51 | 145.55 | 205.40 | 300.02 | 420.97 | 688.66 |

#### 6.4.2.8 Time Limit of Search

The time limit $T_i^{search}$ of the discrete search determines the number of states discovered and expanded during the cost algebraic A* search, directly affecting the quality of the resulting plan. To evaluate the affects of $T_i^{search}$, we run our algorithm with different values of it and report the metrics in single-robot experiments. During evaluations, we set $T_i^{search}$ to values from $75\,\mathrm{ms}$ to $2000\,\mathrm{ms}$. Note that, for high values of $T_i^{search}$, the planning is not real-time anymore. However, we still evaluate the hypothetical performance of our algorithm by freezing the simulation until the planning is done.

During these experiments we set $\tau_i = 2.5\,\mathrm{s}$, $\rho = 0.2$, and $\#^D = 100$.

First set of results are summarized in Table 6.8. Surprisingly, as $T_i^{search}$ increases, the collision and deadlock rates increase and the success rate decreases as it can be seen by comparing cyan values to red values, except for the change from $75\,\mathrm{ms}$ to $150\,\mathrm{ms}$. This is surprising because, discrete search theoretically produces no worse plans than before as the time allocated for it increases. The reason of this stems from the dynamic limits of the robot. We enforce continuity up to acceleration, maximum velocity $10\,\frac{\mathrm{m}}{\mathrm{s}}$ and maximum acceleration $15\,\frac{\mathrm{m}}{\mathrm{s}^2}$. Enforcing these during trajectory optimization becomes harder as the discrete plan gets more and more complicated, having more rotations and speed changes. Increasing the time allocated to discrete search makes it more likely to produce complicated plans. Increase in planning failure rates support this argument, as it can be seen in orange values in **pl. fail rate** column.

Table 6.9: Effects of the time limit of search without dynamic limits

| **Exp. #** | **49** | **50** | **51** | **52** | **53** | **54** |
|---|---|---|---|---|---|---|
| $T_i^{search}[ms]$ | 75 | 150 | 300 | 600 | 1000 | 2000 |
| **succ. rate** | 0.828 | 0.892 | 0.922 | 0.934 | 0.926 | 0.933 |
| **coll. rate** | 0.170 | 0.107 | 0.076 | 0.061 | 0.074 | 0.066 |
| **deadl. rate** | 0.003 | 0.003 | 0.002 | 0.005 | 0.000 | 0.001 |
| **s. coll. rate** | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.169 | 0.107 | 0.076 | 0.061 | 0.074 | 0.066 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 29.80 | 29.90 | 29.69 | 29.50 | 29.47 | 29.54 |
| **pl. fail rate** | 0.015 | 0.017 | 0.024 | 0.035 | 0.043 | 0.049 |
| **avg. pl. dur. [ms]** | 97.27 | 134.80 | 192.80 | 301.26 | 437.43 | 742.61 |

To further support this argument, we run another set of experiments in which we set $c_i = 1$, $\gamma_i^1 = \infty$, and $\gamma_i^2 = \infty$. The results are given in Table 6.9. This time, deadlock and collision rates decrease and the success rate increases as $T_i^{search}$ increases as it can be seen in cyan values. Planning failure rate also increases (red values in **pl. fail rate** row) but its effects are not detrimental to collision and deadlock rates like before.

In the remaining experiments, we set $T_i^{search} = 75\,\text{ms}$.

#### 6.4.2.9 Desired Planning Horizon

The desired planning horizon $\tau_i$ is used by the goal selection stage. The goal selection stage selects the position on the desired trajectory that is one planning horizon away from the closest point on the desired trajectory to robot's current position as the goal position if robot is collision-free when placed on it. As $\tau_i$ increases, our planner tends to plan longer and longer trajectories.

We evaluate the effects of $\tau_i$ to navigation performance by changing its value and reporting navigation metrics in single-robot experiments. In these experiments, we set $\rho = 0.2$, and $\#^D = 25$.

The results are summarized in Table 6.10. When $\tau_i$ is set to $1.0\,\text{s}$, the robot tends to plan shorter trajectories, limiting its longer horizon decision making ability. Increasing $\tau_i$ to $2.5\,\text{s}$, decreases collision and deadlock rates and increases the success rate, as well as the average navigation duration compared to

Table 6.10: Effects of the desired planning horizon

| **Exp. #** | **55** | **56** | **57** | **58** | **59** | **60** |
|---|---|---|---|---|---|---|
| $\tau_i[s]$ | 1.0 | 2.5 | 5.0 | 7.5 | 10.0 | 20.0 |
| **succ. rate** | 0.943 | 0.948 | 0.935 | 0.924 | 0.906 | 0.883 |
| **coll. rate** | 0.052 | 0.046 | 0.061 | 0.066 | 0.074 | 0.093 |
| **deadl. rate** | 0.008 | 0.006 | 0.004 | 0.011 | 0.022 | 0.031 |
| **s. coll. rate** | 0.001 | 0.002 | 0.004 | 0.009 | 0.015 | 0.049 |
| **d. coll. rate** | 0.052 | 0.045 | 0.058 | 0.058 | 0.064 | 0.058 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 29.52 | 28.49 | 28.42 | 28.33 | 28.32 | 28.61 |
| **pl. fail rate** | 0.053 | 0.052 | 0.069 | 0.071 | 0.063 | 0.050 |
| **avg. pl. dur. [ms]** | 58.71 | 62.55 | 80.71 | 119.52 | 152.78 | 195.13 |

$\tau_i = 1.0\,\mathrm{s}$, which can be seen by comparing the cyan and red values. However, increasing $\tau_i$ more causes an increase in collision and deadlock rates and a decrease in the success rate, which can be seen in orange values. The reason is that increasing $\tau_i$ causes distance between the goal position and robot's current position to increase. As the distance to the goal position increases, the discrete search step needs more time to produce high quality discrete plans. Since $T_i^{search}$ is fixed during these experiments, increasing $\tau_i$ causes robot to collide or deadlock more.

In the remaining experiments, we set $\tau_i = 2.5\,\mathrm{s}$.

#### 6.4.2.10 Dynamic Obstacle Sensing Uncertainty

The dynamic obstacle sensing uncertainty is modeled by i) applying a zero mean Gaussian with covariance $\Sigma_i$ to the sensed positions and velocities of dynamic obstacles and ii) randomly inflating dynamic obstacle shapes by a zero mean Gaussian with standard deviation $\sigma_i$. These create three inaccuracies reflected to our planner: i) dynamic obstacle shapes provided to our planner become wrong, ii) prediction inaccuracy increases, and iii) current positions $\mathbf{p}_{i,j}^{dyn}$ of dynamic obstacles provided to the planner become wrong. Our planner models uncertainty across behavior models by using realization probabilities, but does not explicitly account for current position or shape uncertainty.

Table 6.11: Effects of dynamic obstacle sensing uncertainty

| **Exp. #** | **61** | **62** | **63** | **64** | **65** | **66** |
|---|---|---|---|---|---|---|
| $\Sigma_i[\times I_{2d}]$ | 0.0 | 0.1 | 0.1 | 0.2 | 0.2 | 0.5 |
| $\sigma$ | 0.0 | 0.0 | 0.1 | 0.1 | 0.2 | 0.5 |
| **succ. rate** | 0.905 | 0.879 | 0.832 | 0.797 | 0.661 | 0.410 |
| **coll. rate** | 0.094 | 0.119 | 0.168 | 0.203 | 0.339 | 0.590 |
| **deadl. rate** | 0.001 | 0.003 | 0.002 | 0.001 | 0.004 | 0.009 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.094 | 0.119 | 0.168 | 0.203 | 0.339 | 0.590 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 26.71 | 26.70 | 26.70 | 26.71 | 26.69 | 26.72 |
| **pl. fail rate** | 0.026 | 0.030 | 0.028 | 0.029 | 0.031 | 0.037 |
| **avg. pl. dur. [ms]** | 59.76 | 55.53 | 54.54 | 54.83 | 54.46 | 56.15 |

Table 6.12: Effects of dynamic obstacle inflation under dynamic obstacle sensing uncertainty of $\Sigma_i = 0.2I_{2d}, \sigma_i = 0.2$.

| **Exp. #** | **67** | **68** | **69** | **70** | **71** | **72** |
|---|---|---|---|---|---|---|
| inflation $[m]$ | 0.2 | 0.5 | 1.0 | 1.5 | 2.0 | 4.0 |
| **succ. rate** | 0.779 | 0.818 | 0.747 | 0.617 | 0.417 | 0.057 |
| **coll. rate** | 0.220 | 0.182 | 0.250 | 0.382 | 0.581 | 0.943 |
| **deadl. rate** | 0.003 | 0.001 | 0.005 | 0.007 | 0.003 | 0.000 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.220 | 0.182 | 0.250 | 0.382 | 0.581 | 0.943 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 26.70 | 26.72 | 26.93 | 27.16 | 27.79 | 26.81 |
| **pl. fail rate** | 0.032 | 0.035 | 0.047 | 0.058 | 0.067 | 0.021 |
| **avg. pl. dur. [ms]** | 57.80 | 63.014 | 70.92 | 74.63 | 72.66 | 23.13 |

To evaluate our planner's performance under different levels of dynamic obstacle sensing uncertainty, we control $\Sigma_i$ and $\sigma_i$ and report the metrics in single-robot experiments. During these experiments, we set $\rho = 0$ to evaluate the effects of dynamic obstacles only. We set $\#^D = 50$.

The results are given in Table 6.11. $\Sigma_i$ is set to a constant multiple of identity matrix $I_{2d}$ of size $2d \times 2d$ in each experiments. Expectedly, as the uncertainty increases, success rate decreases. Increase in collision and deadlock rates are also seen. Similarly, planning failure rate tends to increase as well.

One common approach to tackle unmodeled uncertainty for obstacle avoidance is artificially inflating shapes of obstacles. To show the effectiveness of this approach, we set $\Sigma_i = 0.2I_{2d}$ and $\sigma_i = 0.2$, and inflate the shapes of obstacles with different amounts during planning. The results of these experiments

Table 6.13: Effects of static obstacle sensing uncertainty

| **Exp. #** | **73** | **74** | **75** | **76** | **77** | **78** |
|---|---|---|---|---|---|---|
| imperfections | $L(0.2)$ | $L(0.2)^2$ | $L(0.3)^2I$ | $L(0.3)^2ID$ | $L(0.3)^2IDL(0.2)$ | $L(0.3)^2IDL(0.5)$ |
| **succ. rate** | 0.994 | 0.992 | 0.986 | 0.956 | 0.971 | 0.949 |
| **coll. rate** | 0.001 | 0.003 | 0.005 | 0.037 | 0.020 | 0.039 |
| **deadl. rate** | 0.005 | 0.005 | 0.009 | 0.007 | 0.009 | 0.015 |
| **s. coll. rate** | 0.001 | 0.003 | 0.005 | 0.037 | 0.020 | 0.039 |
| **d. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **t. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **avg. nav. dur. [s]** | 27.49 | 27.53 | 27.73 | 27.49 | 27.55 | 28.18 |
| **pl. fail rate** | 0.029 | 0.041 | 0.053 | 0.038 | 0.047 | 0.063 |
| **avg. pl. dur. [ms]** | 36.16 | 41.76 | 53.39 | 40.58 | 46.41 | 67.65 |

are given in Table 6.12. Inflation clearly helps when done in reasonable amounts. When inflation is set to $0.2\,\mathrm{m}$, success rate increases from $0.661$ as reported in Table 6.11 (cyan value) to $0.779$. It further increases to to $0.818$ when inflation amount is set to $0.5\,\mathrm{m}$, which can be seen in cyan values in Table 6.12. However, as the inflation amount increases more, metrics start to degrade as the planner becomes overly conservative. Success rate decreases down to $0.057$ when the inflation amount is set to $4.0\,\mathrm{m}$ as it can be seen in the red value in Table 6.12.

#### 6.4.2.11 Static Obstacle Sensing Uncertainty

As we describe in Section 6.4.1.1, we use three operations to model sensing imperfections of static obstacles: i) increaseUncertainty, ii) leakObstacles($p_{leak}$), and iii) deleteObstacles. We evaluate the effects of static obstacle sensing uncertainty by applying a sequence of these operations to the octree representation of static obstacles, and provide the resulting octree to our planner in single-robot experiments. Application of leakObstacles increases the density of the map, but the resulting map contains the original obstacles. increaseUncertainty does not change the density of the map, but increases the uncertainty associated with the obstacles. deleteObstacles decreases the density of the map, but the resulting map may not contain the original obstacles, leading to unsafe behavior. In these experiments, we set $\rho = 0.1$, and $\#^D = 0$.

The results are given in Table 6.13. In imperfections row of the table, we use $L(p_{leak})$ as an abbreviation for leakObstacles($p_{leak}$), and $L(p_{leak})^n$ as an abbreviation for repeated application of leakObstacles to the octree. We use $I$ for increaseIncertainty, and $D$ for deleteObstacles. Leaking obstacles or increasing the uncertainty associated with them does not increase the collision and deadlock rates significantly as seen in the cyan values. The planning duration and failure rates increase as the number of obstacles increases. Deleting obstacles causes a sudden jump of the collision rate as it can be seen between cyan and red values because the planner does not know about existing obstacles. Leaking obstacles back with $p_{leak} = 0.2$ after deleting them decreases the collision rate back as it can be seen between red and orange values. However, leaking obstacles with high probability increases the collision rate back as it can be seen between orange and magenta values. This happens because the environments get significantly more complicated because the number of obstacles increase. Complication of the environments can also be seen by the increased deadlock rate.

#### 6.4.2.12 Teammate Safety Enforcement Strategies

During discrete search, we compute active DSHT violation cost term $\mathcal{J}_{team}$ for each state and minimize it as a part of our cost vector. An alternative way of ensuring teammate avoidance is using active DSHTs as constraints, and enforcing them by discarding any state that violates any of the hyperplanes during search. Using active DSHTs as constraints instead of a cost term *ensures* teammate avoidance as discussed in Chapter 5. However, we show that using DSHTs as constraints results in a conservative behavior when communication medium is imperfect, and using them as cost terms results in better performance in environments with dynamic obstacles or high message delays and drops.

In these experiments, we set static object density $\rho = 0$, and teammate safety enforcement duration $T_i^{team} = \infty$. The dynamic objects are not interactive, i.e., $\hat{f} = 0$. The number of robots is $\#^R = 16$. We

Table 6.14: Effects of teammate safety enforcement strategies under imperfect communication with or without dynamic objects

| **Exp. #** | $\delta[s]$ | $\kappa$ | $\#^D$ | Safety | **succ. rate** | **coll. rate** | **deadl. rate** | **s. coll. rate** | **d. coll. rate** | **t. coll. rate** | **avg. nav. dur. [s]** | **pl. fail rate** | **avg. pl. dur. [ms]** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **79** | 0.0 | 0.0 | 0 | Constraint | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 61.17 | 0.000 | 200.84 |
| | | | | Cost | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 60.45 | $< 0.001$ | 168.90 |
| **80** | 0.0 | 0.0 | 50 | Constraint | 0.791 | 0.209 | 0.001 | 0.000 | 0.209 | 0.000 | 44.12 | 0.007 | 260.44 |
| | | | | Cost | 0.952 | 0.048 | 0.000 | 0.000 | 0.045 | 0.004 | 45.35 | 0.005 | 164.06 |
| **81** | 0.25 | 0.1 | 0 | Constraint | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 75.46 | 0.000 | 258.24 |
| | | | | Cost | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 65.97 | $< 0.001$ | 249.36 |
| **82** | 0.25 | 0.1 | 50 | Constraint | 0.798 | 0.202 | 0.001 | 0.000 | 0.202 | 0.000 | 54.42 | 0.007 | 230.55 |
| | | | | Cost | 0.956 | 0.044 | 0.000 | 0.000 | 0.043 | 0.003 | 50.18 | 0.006 | 208.89 |
| **83** | 1.0 | 0.25 | 0 | Constraint | 0.983 | 0.000 | 0.017 | 0.000 | 0.000 | 0.000 | 105.0 | 0.000 | 486.86 |
| | | | | Cost | 0.995 | 0.000 | 0.005 | 0.000 | 0.000 | 0.000 | 82.49 | $< 0.001$ | 382.10 |
| **84** | 1.0 | 0.25 | 50 | Constraint | 0.709 | 0.291 | 0.000 | 0.000 | 0.291 | 0.000 | 72.36 | 0.005 | 360.38 |
| | | | | Cost | 0.923 | 0.076 | 0.001 | 0.000 | 0.072 | 0.006 | 59.21 | 0.008 | 297.25 |
| **85** | 5.0 | 0.75 | 0 | Constraint | 0.015 | 0.000 | 0.985 | 0.000 | 0.000 | 0.000 | 141.22 | 0.000 | 1951.44 |
| | | | | Cost | 0.490 | 0.000 | 0.510 | 0.000 | 0.000 | 0.000 | 128.97 | $< 0.001$ | 1042.66 |
| **86** | 5.0 | 0.75 | 50 | Constraint | 0.047 | 0.436 | 0.875 | 0.000 | 0.436 | 0.000 | 114.52 | 0.004 | 1735.21 |
| | | | | Cost | 0.702 | 0.289 | 0.013 | 0.000 | 0.250 | 0.067 | 79.06 | 0.037 | 768.56 |

control the average delay $\delta$, message drop probability $\kappa$, number of dynamic obstacles $\#^D$, and the safety enforcement strategy. The results are given in Table 6.14.

When the safety between teammates are enforced with constraints, robots never collide with each other as it can be seen in red values in **t. coll. rate** column. This is the case because teammate constraint generation method we use makes sure that trajectories of each pair of robots share a constraining hyperplane at all times under asynchronous planning and communication imperfections. When these constraints are used as hard constraints throughout the trajectory, robots provably never collide (Chapter 5). When teammate safety is enforced as a cost term, our algorithm can still achieve no collisions between teammates when there are no dynamic objects in the environment as it can be seen in teal values in **t. coll. rate** column. However, when dynamic objects are present, teammates may collide with each other as it can be seen in cyan values in **t. coll. rate**, since we prioritize dynamic obstacle avoidance to teammate avoidance during search.

As the average message delay and message drop rate increases, our algorithm becomes more and more conservative for teammate safety. When there are no dynamic obstacles in the environment or when the teammate safety is enforced as constraints, this causes conservative behavior, increasing average navigation duration as well as the deadlock rate as it can be seen in magenta values in **deadl. rate** and **avg. nav. dur** columns. In addition, sizes of $\tilde{\mathcal{H}}_i^{active}$ increases in these scenarios, causing the number of constraints to increase substiantially during trajectory optimization, slowing down the algorithm such that it is not real-time anymore as it can be seen in orange values in **avg. pl. dur.** column.

Even if enforcing teammate safety as constraints ensures collision avoidance between teammates, it results in a higher collision rate and a lower success rate compared to enforcing it with a cost term when dynamic objects are introduced to the environment as it can be seen in brown values in **succ. rate**, **coll. rate**, and **d. coll. rate.** columns. Enforcing teammate safety using constraints causes robots to be conservative against teammates to ensure safety under asynchronous planning and imperfect communication. Since dynamic objects do not explicitly cooperate with the teammates, this causes teammates to collide with dynamic objects because the feasible set of plans is considerably smaller. Also, enforcing teammate safety using a cost term takes a lower planning duration as it can be seen in **avg. pl. dur.** column and robots reach to their goal positions faster as it can be seen in **avg. nav. dur.** column.

Overall, using active DSHTs as a part of a cost term is a better choice as it results in a higher success rate, lower collision rate and lower navigation duration using less time.

#### 6.4.2.13 Teammate Safety Enforcement Duration

Enforcing teammate safety for the full trajectories causes planning to be not real-time and results in a high rate of deadlocks when communication imperfections are high. We investigate the effects of relaxing teammate constraints by controlling safety enforcement durations $T_i^{team}$. In these experiments, we set $\rho = 0$, $\#^D = 50$, $\hat{f} = 0$, $\delta = 0.25\,\text{s}$, $\kappa = 0.1$, and $\#^R = 16$. Hence, these experiments can be compared with experiment 82 in Table 6.14 when safety is enforced using a cost term. The results are given in Table 6.15.

The average navigation duration of the robots tends to increase as $T_i^{team}$ increases, because the planner becomes more and more conservative, as it can be seen in orange values. Setting $T_{team}^i = 1.0$ results in the best success rate, $0.964$, as seen in the red value. The average planning duration decreases by $\approx 50\%$

Table 6.15: Effects of teammate safety duration

| **Exp. #** | **87** | **88** | **89** | **90** | **91** | **92** |
|---|---|---|---|---|---|---|
| $T_{team}$ | 0.0 | 0.5 | 1.0 | 1.5 | 2.0 | 2.5 |
| **succ. rate** | 0.300 | 0.748 | 0.964 | 0.963 | 0.961 | 0.933 |
| **coll. rate** | 0.700 | 0.252 | 0.036 | 0.038 | 0.039 | 0.068 |
| **deadl. rate** | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 |
| **s. coll. rate** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **d. coll. rate** | 0.068 | 0.098 | 0.036 | 0.038 | 0.036 | 0.058 |
| **t. coll. rate** | 0.677 | 0.183 | 0.001 | 0.000 | 0.005 | 0.010 |
| **avg. nav. dur. [s]** | 27.57 | 30.55 | 37.60 | 44.71 | 51.01 | 49.38 |
| **pl. fail rate** | 0.011 | 0.025 | 0.008 | 0.005 | 0.005 | 0.006 |
| **avg. pl. dur. [ms]** | 46.58 | 77.87 | 115.30 | 153.47 | 195.93 | 205.35 |

when $T^i_{team}$ is set to $1.0$ compared to setting it $\infty$, greatly increasing the cases our algorithm is real-time as it can be seen in the magenta value compared to the red value in Table 6.14. The success rate of $0.964$ is higher than that of experiment $82$ in Table 6.14, which is $0.956$. In addition, the average navigation duration decreases to $37.60$ compared to $50.18$ in Table 6.14. Relaxing safety against teammates not only decreases planning and navigation durations, but improves the success rate of the algorithm.

In the remaining experiments, we set $T_i^{team} = 1.0$.

### 6.4.3 Baseline Comparisons in Simulations

We summarize the collision avoidance and deadlock-free navigation comparisons between state-of-the-art decentralized navigation decision making algorithms in Figure 6.7. The comparison graph suggests that SBC [121], RLSS (Chapter 4), RMADER [50], and TASC [117, 116] are the best performing algorithms. TASC and RMADER are shown to have a similar collision avoidance performance in [116]. Therefore, we compare our algorithm DREAM to SBC, RLSS, and RMADER, and establish that it results in a better performance than them. SBC is a short horizon algorithm: it computes next safe acceleration to execute to drive the robot to its goal position each iteration. RMADER, RLSS, and DREAM are medium horizon algorithms: they compute long trajectories, which they execute for a shorter duration in a receding horizon fashion.

All of our baselines drive robots to their given goal positions. We add support of desired trajectories by running our goal selection algorithm in every planning iteration, and providing the selected goal position as intermediate goal positions.

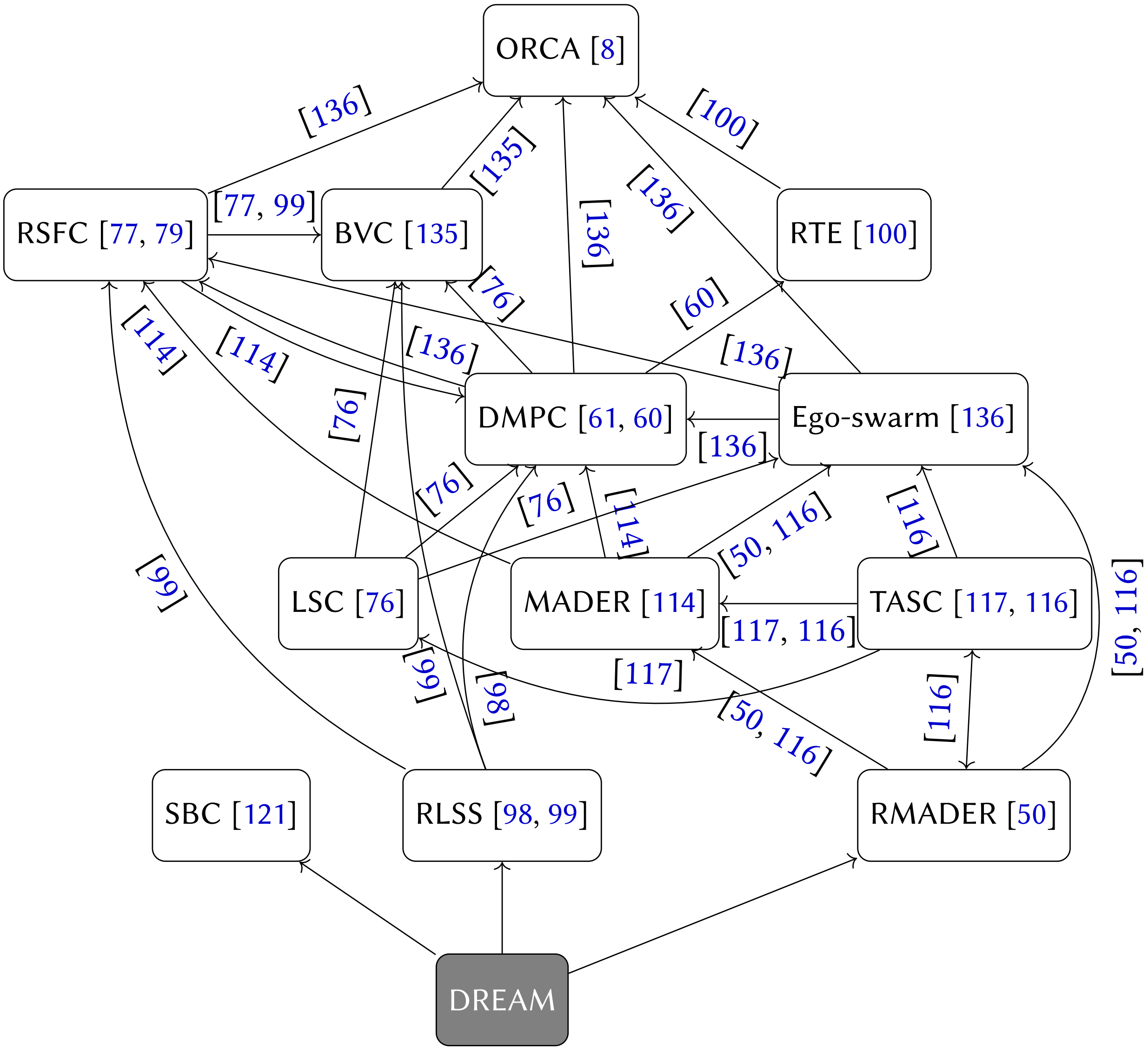


Figure 6.7: **Collision avoidance and deadlock-free navigation comparisons of state-of-the-art decentralized multi-robot navigation decision making algorithms.** A directed edge means that the source algorithm is shown to be better than the destination algorithm in experiments with multiple robots in some environments. TASC and RMADER are shown to have a similar collision avoidance performance in [116]. We compare our algorithm to SBC [121], RLSS [99, 98], and RMADER [50] and establish that it results in a better performance than the listed state-of-the-art baselines.

#### 6.4.3.1 Using RMADER in Comparisons

RMADER is a real-time decentralized trajectory planning algorithm for static and dynamic obstacle and multi-robot collision avoidance. It explicitly accounts for asynchronous planning between robots using communication, and accounts for communication delays with known bounds. It models dynamic obstacle movements using predicted trajectories. It does not explicitly model robot–dynamic obstacle interactions.

Dynamic obstacles move according to movement and interaction models in our formulation. We convert movement models to predicted trajectories by propagating dynamic obstacles' positions according to the desired velocities from the movement models. Since the interactive behavior of dynamic obstacles depend on the trajectory that the robot is going to follow, which is computed by the planner, their effect to the future trajectories are unknown prior to planning. Therefore, we do not use interactive obstacles during baseline comparisons.

We use the code of RMADER published by its authors [35] and integrate it to our simulation system. We set desired maximum planning time of RMADER to $500\,\mathrm{ms}$ in each planning iteration, which it exceeds if needed, even when the simulated replanning period is smaller. We freeze the environment until RMADER is done replanning to cancel the affects of exceeding the replanning period.

Our prediction system generates three behavior models for each dynamic obstacle. RMADER does not support multiple behavior hypotheses explicitly. Therefore, RMADER has the choice of avoiding most likely or all behavior models of dynamic obstacles, modelling each behavior model as a separate obstacle. We provide only the most likely behavior models to RMADER during evaluation, because avoiding all behavior models resulted in a highly conservative behavior.

#### 6.4.3.2 Using RLSS in Comparisons

RLSS is a real-time decentralized trajectory planning algorithm for static obstacle and multi-robot collision avoidance. It does not account for asynchronous planning between teammates. It does not utilize communication and depends position/shape sensing only. When using RLSS in comparisons, we model dynamic obstacles as robots, and provide their current positions and shapes to the planning algorithm. We use our own implementation of RLSS during our comparisons.

#### 6.4.3.3 Using SBC in Comparisons

SBC is a safety barrier certificates based safe controller for static and dynamic obstacle and multi-robot collision avoidance. SBC runs at a high frequency ($> 50\,\mathrm{Hz}$), therefore it does not need to account for asynchronous planning. It does not utilize communication and depends on position, velocity and shape sensing. When simulating SBC, we do not use the predicted behavior models and feed the current positions and velocities of the dynamic obstacles to the algorithm, which assumes that the dynamic obstacles executes the same velocity for the short future ($< 20\,\mathrm{ms}$).

SBC models shapes of obstacles and robots as hyperspheres. We provide shapes of objects to SBC as the smallest hyperspheres containing the axis aligned boxes. We sample robot sizes in interval $[0.1\,\mathrm{m}, 0.2\,\mathrm{m}]$ instead of $[0.2\,\mathrm{m}, 0.3\,\mathrm{m}]$ in SBC runs so that the robot can fit between static obstacles easily when its collision shape is modelled using bounding hyperspheres. (Since resolution of octrees we generate is $0.5\,\mathrm{m}$, the smallest possible gap between static obstacles is $0.5\,\mathrm{m}$.) We use our own implementation of SBC.

Table 6.16: Baseline comparisons in single robot scenarios

| **Exp. #** | $\rho$ | $\#^D$ | Alg. | **succ. rate** | **coll. rate** | **deadl. rate** | **s. coll. rate** | **d. coll. rate** | **t. coll. rate** | **avg. nav. dur. [s]** | **pl. fail rate** | **avg. pl. dur. [ms]** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **93** | 0.0 | 15 | SBC | 0.992 | 0.008 | 0.000 | 0.000 | 0.008 | 0.000 | 33.96 | 0.001 | 0.26 |
| | | | RMADER | 0.960 | 0.036 | 0.008 | 0.000 | 0.036 | 0.000 | 25.31 | 0.100 | 19.24 |
| | | | RLSS | 0.776 | 0.224 | 0.012 | 0.000 | 0.224 | 0.000 | 26.80 | 0.052 | 6.69 |
| | | | DREAM | 0.984 | 0.016 | 0.000 | 0.000 | 0.016 | 0.000 | 26.67 | 0.008 | 13.93 |
| **94** | 0.1 | 15 | SBC | 0.352 | 0.648 | 0.236 | 0.648 | 0.004 | 0.000 | 34.46 | 0.006 | 0.73 |
| | | | RMADER | 0.756 | 0.096 | 0.212 | 0.000 | 0.096 | 0.000 | 38.17 | 0.313 | 341.14 |
| | | | RLSS | 0.816 | 0.184 | 0.008 | 0.000 | 0.184 | 0.000 | 27.26 | 0.034 | 28.37 |
| | | | DREAM | 0.984 | 0.016 | 0.000 | 0.000 | 0.016 | 0.000 | 27.47 | 0.025 | 21.09 |
| **95** | 0.2 | 15 | SBC | 0.072 | 0.928 | 0.564 | 0.928 | 0.044 | 0.000 | 35.25 | 0.011 | 1.47 |
| | | | RMADER | 0.456 | 0.204 | 0.516 | 0.000 | 0.204 | 0.000 | 49.74 | 0.470 | 769.83 |
| | | | RLSS | 0.780 | 0.204 | 0.036 | 0.000 | 0.204 | 0.000 | 28.36 | 0.088 | 48.07 |
| | | | DREAM | 0.988 | 0.012 | 0.000 | 0.000 | 0.012 | 0.000 | 28.31 | 0.043 | 25.56 |
| **96** | 0.2 | 25 | SBC | 0.080 | 0.920 | 0.560 | 0.920 | 0.068 | 0.000 | 37.09 | 0.019 | 1.47 |
| | | | RMADER | 0.400 | 0.264 | 0.568 | 0.000 | 0.264 | 0.000 | 50.77 | 0.535 | 751.45 |
| | | | RLSS | 0.752 | 0.228 | 0.044 | 0.000 | 0.228 | 0.000 | 28.66 | 0.095 | 35.59 |
| | | | DREAM | 0.956 | 0.040 | 0.008 | 0.000 | 0.040 | 0.000 | 28.28 | 0.059 | 31.54 |
| **97** | 0.2 | 50 | SBC | 0.056 | 0.944 | 0.556 | 0.944 | 0.144 | 0.000 | 42.23 | 0.032 | 1.42 |
| | | | RMADER | 0.116 | 0.632 | 0.844 | 0.000 | 0.632 | 0.000 | 49.79 | 0.755 | 890.89 |
| | | | RLSS | 0.536 | 0.448 | 0.064 | 0.000 | 0.448 | 0.000 | 29.33 | 0.151 | 52.93 |
| | | | DREAM | 0.900 | 0.100 | 0.008 | 0.000 | 0.100 | 0.000 | 28.46 | 0.073 | 47.02 |
| **98** | 0.3 | 50 | SBC | 0.008 | 0.992 | 0.788 | 0.992 | 0.212 | 0.000 | 48.75 | 0.041 | 1.92 |
| | | | RMADER | 0.092 | 0.536 | 0.904 | 0.000 | 0.536 | 0.000 | 56.68 | 0.745 | 1217.24 |
| | | | RLSS | 0.536 | 0.436 | 0.160 | 0.000 | 0.436 | 0.000 | 31.28 | 0.237 | 171.81 |
| | | | DREAM | 0.884 | 0.116 | 0.000 | 0.000 | 0.116 | 0.000 | 30.25 | 0.080 | 48.10 |

#### 6.4.3.4 Single Robot Experiments

We compare our planner DREAM with the baselines in environments with different static obstacle densities $\rho$ and number of dynamic obstacles $\#^D$ in single-robot experiments. The results are summarized in Table 6.16.

SBC's performance decreases sharply when static obstacles are introduced to the environment as it can be seen in red values in **succ. rate** column. SBC mainly suffers from collisions with static obstacles compared to dynamic obstacles (cyan vs orange values). All medium horizon algorithms can avoid static obstacles perfectly (magenta values).

SBC and RMADER result in a high deadlock rate as it can be seen in green values. The reason SBC results in a high ratio of deadlocks is its short horizon decision making setup. Since it does not consider longer horizon effects of the generated actions, as the environment density increases, it tends to deadlock. RMADER results in a high ratio of planning failures as the density increases as it can be seen in brown values in **pl. fail rate** column, which causes it to consume the generated plans and not being able generate new ones, which results in deadlocks. RLSS has better deadlock avoidance compared to SBC and RMADER,

Table 6.17: Baseline comparisons in multi robot scenarios with $\delta = 1\,\mathrm{s}$ average delay and $\kappa = 0.25$ message drop probability for our algorithm and $\delta = 1\,\mathrm{s}$ maximum delay and no message drops for RMADER. SBC and RLSS do not depend on communication.

| **Exp. #** | $\rho$ | $\#^D$ | $\#^R$ | Alg. | **succ. rate** | **coll. rate** | **deadl. rate** | **s. coll. rate** | **d. coll. rate** | **t. coll. rate** | **avg. nav. dur. [s]** | **pl. fail rate** | **avg. pl. dur. [ms]** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **99** | 0.2 | 25 | 16 | SBC | 0.059 | 0.941 | 0.483 | 0.941 | 0.126 | 0.004 | 38.82 | 0.021 | 1.58 |
| | | | | RMADER | 0.116 | 0.551 | 0.870 | 0.000 | 0.551 | 0.001 | 60.90 | 0.932 | 813.71 |
| | | | | RLSS | 0.488 | 0.508 | 0.039 | 0.000 | 0.428 | 0.134 | 34.95 | 0.259 | 97.78 |
| | | | | DREAM | **0.960** | 0.036 | 0.005 | 0.003 | 0.035 | 0.000 | 40.77 | 0.014 | 123.68 |
| **100** | 0.2 | 25 | 32 | SBC | 0.061 | 0.938 | 0.450 | 0.937 | 0.121 | 0.003 | 39.13 | 0.018 | 1.62 |
| | | | | RMADER | 0.112 | 0.520 | 0.868 | $< 0.001$ | 0.514 | 0.008 | 62.82 | 0.923 | 912.37 |
| | | | | RLSS | 0.471 | 0.521 | 0.059 | 0.000 | 0.373 | 0.225 | 35.53 | 0.354 | 90.16 |
| | | | | DREAM | **0.841** | 0.097 | 0.069 | 0.006 | 0.074 | 0.026 | 60.31 | 0.014 | 185.87 |
| **101** | 0.3 | 25 | 16 | SBC | 0.006 | 0.994 | 0.691 | 0.994 | 0.184 | 0.001 | 42.68 | 0.028 | 1.78 |
| | | | | RMADER | 0.054 | 0.534 | 0.935 | 0.000 | 0.533 | 0.003 | 68.67 | 0.952 | 1457.58 |
| | | | | RLSS | 0.591 | 0.396 | 0.056 | 0.000 | 0.332 | 0.093 | 38.47 | 0.324 | 300.33 |
| | | | | DREAM | **0.941** | 0.051 | 0.011 | 0.005 | 0.044 | 0.003 | 42.53 | 0.020 | 120.33 |
| **102** | 0.3 | 25 | 32 | SBC | 0.006 | 0.993 | 0.669 | 0.993 | 0.191 | 0.006 | 40.40 | 0.031 | 1.79 |
| | | | | RMADER | 0.027 | 0.449 | 0.973 | 0.000 | 0.449 | 0.004 | 64.90 | 0.934 | 1675.96 |
| | | | | RLSS | 0.503 | 0.486 | 0.064 | 0.000 | 0.343 | 0.209 | 39.10 | 0.440 | 282.33 |
| | | | | DREAM | **0.782** | 0.108 | 0.127 | 0.014 | 0.081 | 0.025 | 58.04 | 0.019 | 178.71 |
| **103** | 0.3 | 50 | 16 | SBC | 0.006 | 0.994 | 0.636 | 0.993 | 0.318 | 0.003 | 47.60 | 0.046 | 1.96 |
| | | | | RMADER | 0.042 | 0.726 | 0.959 | 0.000 | 0.726 | 0.000 | 58.19 | 0.937 | 1396.72 |
| | | | | RLSS | 0.402 | 0.592 | 0.073 | 0.000 | 0.533 | 0.116 | 39.15 | 0.375 | 354.34 |
| | | | | DREAM | **0.864** | 0.128 | 0.012 | 0.006 | 0.122 | 0.008 | 43.30 | 0.023 | 128.87 |
| **104** | 0.3 | 50 | 32 | SBC | 0.006 | 0.993 | 0.646 | 0.992 | 0.348 | 0.011 | 44.82 | 0.050 | 2.01 |
| | | | | RMADER | 0.047 | 0.516 | 0.938 | 0.000 | 0.516 | 0.000 | 63.68 | 0.909 | 1086.02 |
| | | | | RLSS | 0.357 | 0.634 | 0.098 | 0.000 | 0.540 | 0.197 | 39.77 | 0.475 | 325.69 |
| | | | | DREAM | **0.712** | 0.216 | 0.093 | 0.018 | 0.183 | 0.046 | 59.46 | 0.028 | 189.06 |

but it too results in deadlocks as the environment density increases as it can be seen in purple value in **deadl. rate** column. DREAM results in little to no deadlocks (yellow values).

In terms of the success rate, DREAM considerably improves the state-of-the-art, resulting in $\approx 110x$ improvement over SBC, $\approx 9.6x$ improvement over RMADER, and $\approx 1.65x$ improvement over RLSS in the hardest scenario (violet values in **succ. rate** column).

#### 6.4.3.5 Multi Robot Experiments

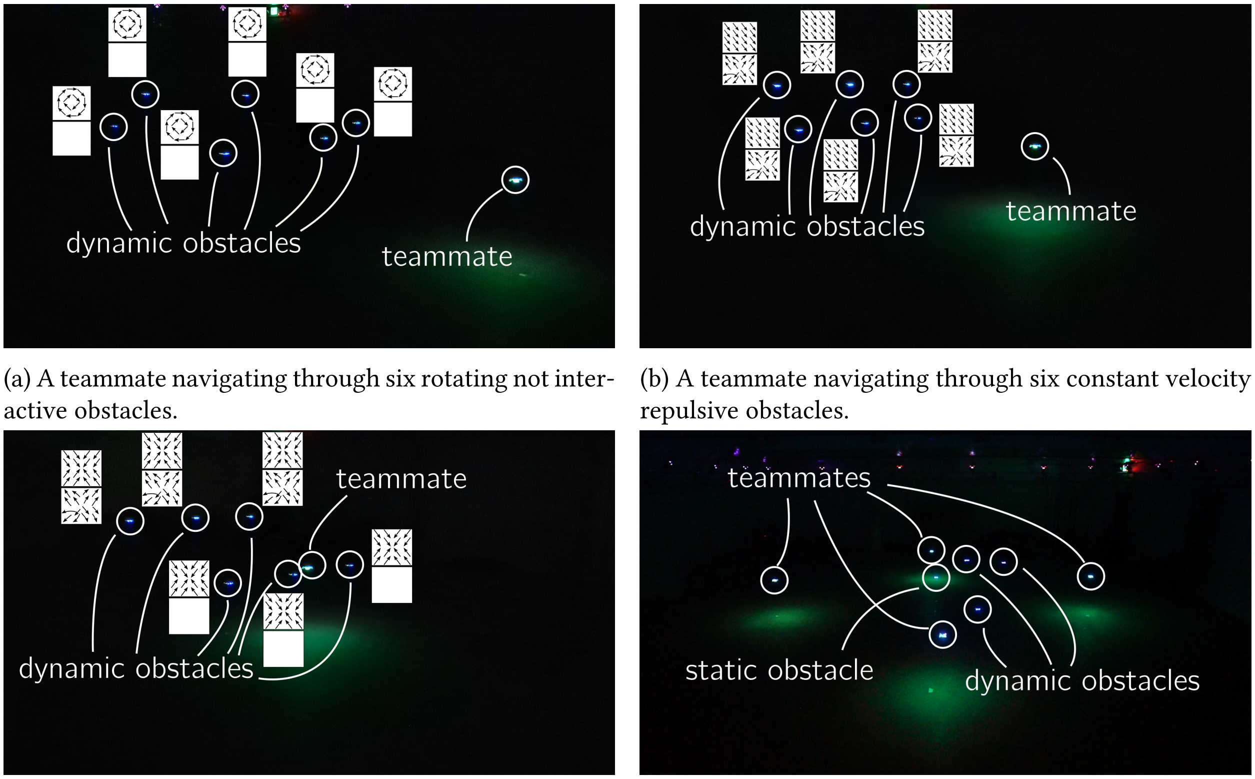
We compare our planner with the baselines in highly cluttered environments with different $\rho$, $\#^D$ and $\#^R$. During these experiments, we simulate communication imperfections. SBC and RLSS do not depend on communication and hence communication imperfections do not effect them. RMADER accounts for message delays with known bounds. DREAM accounts for message delays with unknown bounds as well as message drops.

For DREAM, we introduce mean communication delay $\delta = 1s$ and message drop probability $\kappa = 0.25$. Since RMADER does not account for message drops, we set $\kappa = 0.0$ for RMADER. In addition, in RMADER, we bound communication delays with the mean $\delta$ by generating a random sample from the distribution and setting it to $\delta$ if it is more than $\delta$. Therefore, DREAM runs in considerably more challenging environments during these experiments compared to RMADER. RMADER has a *delay check* phase to account for communication delays, which should run for at least the maximum communication delay. We set its duration to $1.1\,\mathrm{s}$. The environments used are more challenging compared to single-robot experiments, as not only $\rho$ and $\#^D$ are high, but also multiple teammates navigate under asynchronous decision making and considerable communication imperfections.

The results are summarized in Table 6.17. SBC is ineffective for navigating in environments with high clutter (red values in **succ. rate** column). Since the communication imperfections are high, RMADER results in conservative behavior, resulting in deadlocks. Given that it already results in considerable rate of deadlocks in single-robot scenarios, almost all robots using RMADER deadlock in the hardest scenarios (cyan values in **deadl. rate** column). The high planning failure rate of RMADER (magenta values in **pl. fail rate** column) is the main cause of the deadlocks: once plans are consumed and planning fails, it keeps failing until the end of the simulation.

Both RLSS and RMADER results in no collisions against static obstacles as they i) avoid static obstacles using hard constraints unlike DREAM, and ii) they enforce static obstacle avoidance for the full horizon unlike SBC (orange values in **s. coll. rate** column). DREAM results in the lowest dynamic obstacle avoidance rate (violet values in **d. coll. rate** column). RLSS results in high teammate collisions, because it is the only algorithm that is effected by asynchronous planning but does not account for it (brown values in **t. coll. rate** column).

In terms of success rate, DREAM considerably improves the state-of-the-art, resulting in $\approx 156.8x$ improvement over SBC, $\approx 28.96x$ improvement over RMADER, $\approx 2.15x$ improvement over RLSS (**bold** values in **succ. rate** column).



(a) A teammate navigating through six rotating not interactive obstacles.

(b) A teammate navigating through six constant velocity repulsive obstacles.

(c) A teammate navigating through three goal attractive repulsive and three goal attractive not interactive obstacles.

(d) Four teammates navigating through three rotating not interactive dynamic obstacles and a static obstacle.

Figure 6.8: **Pictures from physical robot experiments.** We implement our algorithm for physical quadrotor flight. We conduct single and multi robot experiments to show the real-world applicability of our algorithm in various environments. In our physical robot experiments, teammates navigate through the environments without collisions or deadlocks. The recordings of the physical robot experiments can be found in the supplemental video.

### 6.4.4 Physical Robot Experiments

We implement and run our planner for Crazyflie 2.1 quadrotors. We use quadrotors as i) dynamic obstacles moving according to goal attractive, rotating, or constant velocity movement models and repulsive interaction model, ii) static obstacles, and iii) teammates navigating to their goal positions using our planner. Each planning quadrotor runs the predictors in real-time to generate a probability distribution over behavior models of each dynamic obstacle. Then, it runs our planner to compute trajectories in real-time.

For obstacle and robot localization, we use VICON motion tracking system, and we manage the Crazyflies using Crazyswarm [86]. We use Robot Operating System (ROS) [89] as the underlying software system.

Predictors and our algorithm run on a separate process for each robot in a basestation computer with Intel(R) Xeon(R) CPU E5-2630 v4 @2.20GHz CPU, running Ubuntu 20.04 as the operating system.

Pictures from the physical experiments can be seen in Figure 6.8. The recordings from our physical experiments can be found in our supplemental video. Our physical experiments show the feasibility of running our planner in real-time in the real world.

## 6.5 Conclusion

In this chapter, we present DREAM – a decentralized multi-robot real-time trajectory planning algorithm for mobile robots navigating in environments with static and interactive dynamic obstacles. DREAM explicitly minimizes collision probabilities against static and dynamic obstacles and DSHT violations against teammates as well as distance, duration, and rotations using a multi-objective search method; and energy usage during optimization. The behavior of dynamic obstacles are modeled using two vector fields, namely movement and interaction models. DREAM simulates the behaviors of dynamic obstacles during decision making in response to the actions the planning robot is going to take using the interaction model. We present three online model based prediction algorithms to predict the behavior of dynamic obstacles and assign probabilities to them. We extensively evaluate DREAM in different environments and configurations and compare it with three state-of-the-art decentralized real-time multi-robot navigation decision making methods. DREAM considerably improves the state-of-the-art, achieving up to 1.68x success rate using as low as 0.28x time in single-robot, and up to 2.15x success rate using as low as 0.36x time in multi-robot experiments compared to the best baseline. We show its feasibility in the real-world by implementing and running it for quadrotors.

# Chapter 7

# Multi-Robot Aware Planning and Control Stack for Collision-free and Deadlock-free Navigation in Cluttered Environments

In Chapters 4 to 6, we focus on medium horizon decision making algorithms deployed in a decentralized fashion. In this chapter, we introduce MRNAV, a framework for planning and control to effectively navigate in environments with static and dynamic obstacles, which can optionally be interactive, as well as teammates. Our design utilizes short, medium, and long horizon decision making modules with qualitatively different properties, and defines the responsibilities of them. The decision making modules complement each other and provide the effective navigation capability. MRNAV is the first hierarchical approach combining these three levels of decision making modules and explicitly defining their responsibilities. We implement our design for simulated multi-quadrotor flight. In our evaluations, we show that all three modules are required for effective navigation in diverse situations. We show the long-term executability of our approach in an eight hour long wall time (six hour long simulation time) uninterrupted simulation without collisions or deadlocks.

This chapter is based on Baskın Şenbaşlar et al. “MRNAV: Multi-Robot Aware Planning and Control Stack for Collision and Deadlock-free Navigation in Cluttered Environments”. In: *arXiv preprint arXiv:2308.13499* (2023). Additional experiments are added to discuss the limits of the design.

## Supplemental Videos

**Experiment Recordings:** https://youtu.be/6WC0YCEctoE

**Long Term Execution:** https://youtu.be/LEMNKBRULg4

## 7.1 Introduction

MRNAV is a planning and control stack design for collision and deadlock-free navigation of multi-robot teams in cluttered environments, which might contain static and dynamic obstacles that can optionally be interactive. Hence, our system provides a solution for navigation in all types of environments. We discuss the design as well as deployment options for the components. MRNAV has a clear interface to interoperate with perception, localization, mapping, and prediction subsystems that might exist in a full robotic navigation stack.

The main novelty of our design is utilizing three different navigation decision making components, namely long, medium, and short horizon decision making modules, with qualitatively different collision and deadlock-free navigation capabilities and argue that effective multi-robot navigation requires a harmonious operation of them. These qualitatively different multi-robot navigation decision making approaches are not alternatives, but complements of each other. MRNAV is the first hierarchical approach combining these three qualitatively different decision making components.

Vital properties of MRNAV include assurance of dynamic feasibility of the actions generated, real-time executability, compliance with communication imperfections (including message drops, delays, and reorderings, and not depending on communication for collision-free operation), and reasonable liveness of the system, i.e., not being overly conservative for collision-free operation.

We provide a concrete implementation of our design for navigation of multiple quadrotors in highly cluttered environments with static and dynamic obstacles (Figure 7.1). We run experiments using simulated

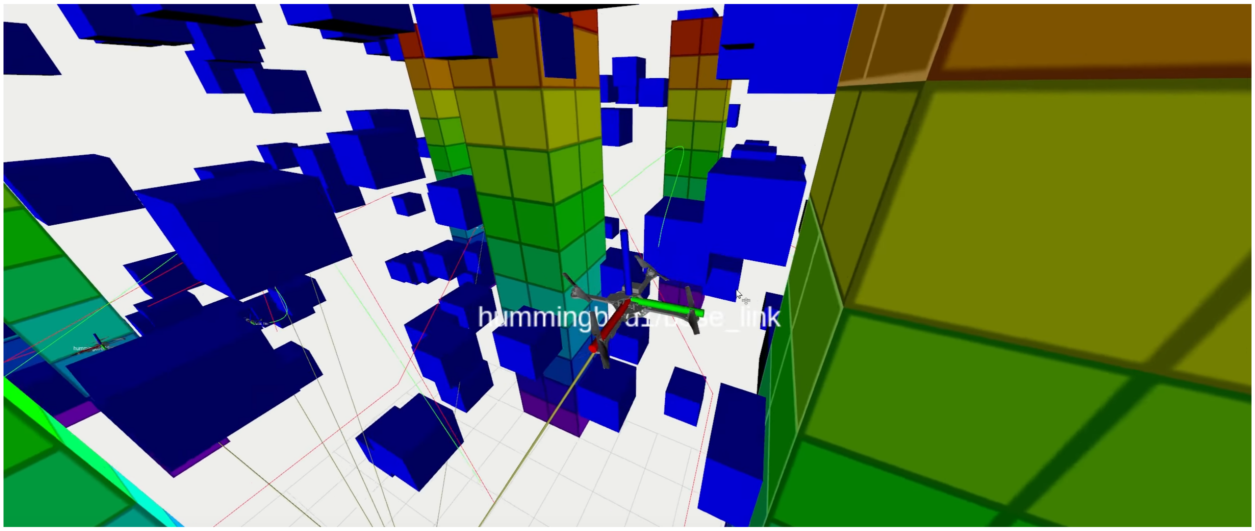


Figure 7.1: We implement MRNAV for collision-free multiple quadrotor flight in highly cluttered environments with static and dynamic obstacles.

quadrotors with motor speed control, and experimentally show that MRNAV's three modules are required for effective collision-free and deadlock-free navigation. In order to demonstrate the long term executability of our implementation, we run an eight hour wall time (six hour simulation time) long uninterrupted simulation with static and dynamic obstacles in which eight quadrotors navigate without collisions or deadlocks.

## 7.2 System Design & Implementation

Our system design includes long, medium, and short horizon decision making modules integrated together. We discuss major challenges of multi-robot navigation to justify our design, deployment, and the algorithmic choices we make later in Section 7.2.3 for flight of multiple quadrotors.

### 7.2.1 Major Design Challenges

#### 7.2.1.1 Limited On-board Compute, Memory and Storage

We assume that the on-board memory and storage capabilities of the robots are limited, which prohibits them from maintaining global environment representation, but they can maintain local environment representations. In addition, we assume that the on-board compute capabilities of the robots are also limited. These prohibit the robots from computing long horizon plans with completeness guarantees on-board.

#### 7.2.1.2 Imperfect Communication

We embrace that the communication systems are inherently imperfect, and sometimes communication can be lost completely. This necessitates on-board communication-resilient autonomy. Robots should be able to ensure collision-free operation without communicating with a centralized system or between each other. In addition to the collision-free operation, robots should be able to locally resolve deadlocks[*], because communication to the centralized system may not be maintained. Since the robots cannot rely on inter-robot communication for robot-robot collision avoidance, they need sensing and estimation capabilities to assess each other's states.

#### 7.2.1.3 Imperfect Prior Knowledge

In general, it is impossible to know all of the static or dynamic obstacles a priori, except for the use cases in which the environment is specifically designed or obstacle free. Generally, some static obstacles are known a priori. Since dynamic obstacles are transitionary, modelling and predicting them offline is unrealistic. This necessitates on-board decision making autonomy as well as local environment representation building. On-board collision-free navigation and local deadlock resolution capabilities allow robots to handle problems stemming from imperfect prior knowledge.

[*] While local deadlock resolution does not have an exact definition as global deadlock resolution, i.e., completeness, we use the term to refer to the ever-increasing ability of the robots to resolve deadlocks using their on-board decision making modules.

#### 7.2.1.4 Imperfect Reference Trajectory Tracking

The control systems of mobile robots are characteristically imperfect, in which local deviations from reference trajectories are expected, and explicitly accounted for with feedback control. Local deviations can stem from unmodeled physics, disturbances, state estimation errors, or imperfect system identification. When the robots deviate from their reference trajectories in an unforeseen amount, safety guarantees the trajectories satisfy may become invalid. Therefore, providing collision avoidance guarantees during reference trajectory computation is not enough for collision-free operation, and the control actions executed by the robot should re-evaluate safety at every control step.

#### 7.2.1.5 Minimal Sensing and Estimation Capabilities

Sensing and estimating the states of entities produces outputs with errors. We consider current positions and shapes of objects as first order information, such that robots can use inputs from their sensors to estimate these values directly. Higher order derivative estimates can be done by using a numerical estimator, which increases the error of the estimation. For the reliability of the safety properties, the robots should need as few higher order derivatives as possible to enforce safety. In general, position and shape sensing for static objects, and position, velocity and shape sensing for non-static objects is preferred, as these are minimal representations to describe the current state and the immediate future of these objects.

#### 7.2.1.6 High Reactivity

Behavior prediction becomes a central problem when there are dynamic obstacles. In general, these predictions are the best real-time estimations of the behaviors, and may fail to describe the medium to long horizon trajectories of the dynamic obstacles. When dynamic obstacles move unlike their predictions, the collision-free operation properties may be violated. In such situations, the robots should be able to react to unforeseen circumstances swiftly for collision-free operation.

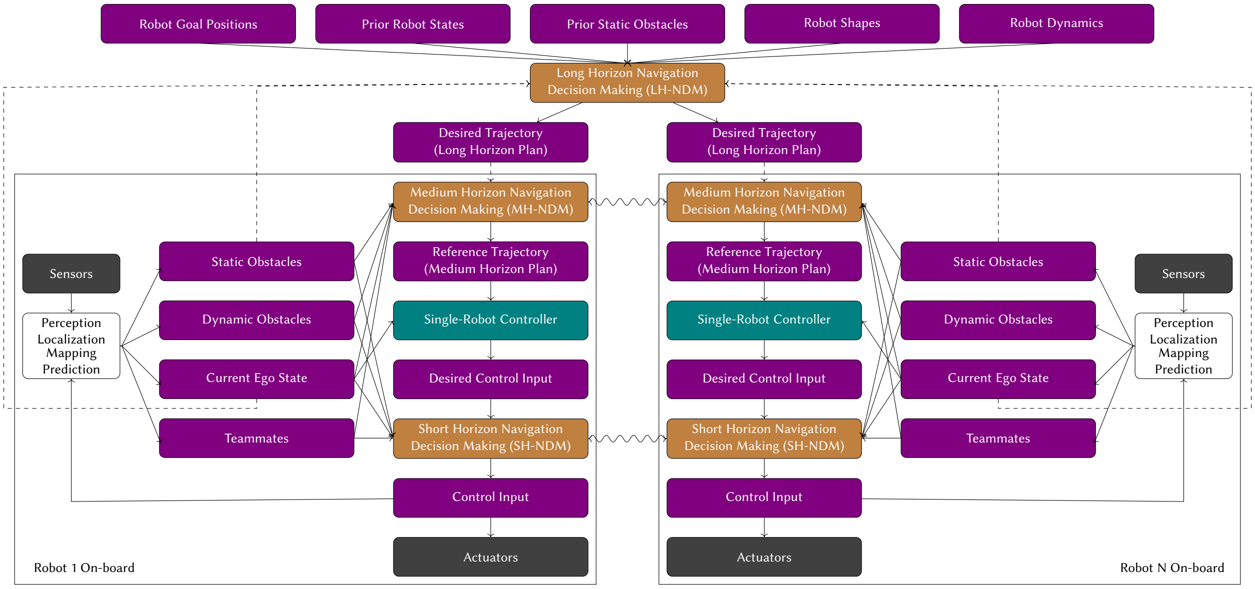

Robot Goal Positions
Prior Robot States
Prior Static Obstacles
Robot Shapes
Robot Dynamics
Long Horizon Navigation Decision Making (LH-NDM)
Desired Trajectory (Long Horizon Plan)
Desired Trajectory (Long Horizon Plan)
Medium Horizon Navigation Decision Making (MH-NDM)
Medium Horizon Navigation Decision Making (MH-NDM)
Sensors
Static Obstacles
Reference Trajectory (Medium Horizon Plan)
Reference Trajectory (Medium Horizon Plan)
Static Obstacles
Sensors
Perception Localization Mapping Prediction
Dynamic Obstacles
Single-Robot Controller
Single-Robot Controller
Dynamic Obstacles
Perception Localization Mapping Prediction
Current Ego State
Desired Control Input
Desired Control Input
Current Ego State
Teammates
Short Horizon Navigation Decision Making (SH-NDM)
Short Horizon Navigation Decision Making (SH-NDM)
Teammates
Control Input
Control Input
Actuators
Actuators
Robot 1 On-board
Robot N On-board

Figure 7.2: **System Design.** Our system utilizes long, medium, and short horizon decision making modules to allow collision and deadlock-free operation. The purple boxes are the messages passed between components, the brown boxes are decision making modules, and the teal box is the controller. The physical robot parts are denoted with gray boxes, and the abstract perception, localization, mapping, and prediction subsystems are shown as a single white box. Solid lines represent connections on the same machine, dashed lines represent connections over a communication medium, and cannot be expected to be always possible, and lastly, squiggly lines represent interactions between components either explicitly through communication, or implicitly using the fact that robots use the same algorithm. The long horizon decision making module is centrally deployed, and other components run on-board.

### 7.2.2 Design

We describe the abstract system design of MRNAV (Figure 7.2), its components, responsibilities assigned to the components, and their deployment options.

**Perception, Localization, Mapping, and Prediction Interfaces.** The robots' on-board perception, localization, mapping, and prediction systems consume sensor data and the control inputs applied to the actuators to i) build representations for static and dynamic obstacles and teammates, and ii) estimate the robot states. Generally, static obstacles are modeled using their positions and shapes, dynamic obstacles are modeled using their states, shapes, and possibly future behavior predictions, and teammates are modeled using their states, shapes, and possibly estimated or communicated future trajectories.

**Long Horizon Navigation Decision Making.** An LH-NDM module is deployed on a centralized computing system, which is responsible for generating *desired trajectories*. It sends the computed desired trajectories to the robots through a communication link, which the robots follow as closely as possible using their on-board decision making system, while ensuring that they remain collision-free, deviating from the desired trajectories as necessary, but navigating back to them whenever possible. This allows robots to delegate global navigation decisions to the LH-NDM module, and locally resolve collisions not accounted for by the desired trajectories in real-time. If reliable communication to the central entity is possible, robots send their local static obstacle representations to the centralized system, in order for it to update its global environment representation. Also, the robots send their estimated states to the centralized system, to allow it to track task progress and re-plan a new desired trajectory if i) a previous navigation task is completed, and/or ii) a deadlock cannot be resolved locally. The LH-NDM module can be given robot goal positions explicitly if the navigation system is used by an outside task planner, or it can compute goal positions itself if it does task planning as well. Desired trajectories can be made dynamically feasible if desired. The long horizon decision making module runs on-demand, and does not have any real-time requirements, allowing it to provide completeness and optimality guarantees if desired.

**Medium Horizon Navigation Decision Making.** Robots employ on-board MH-NDM modules for medium term collision avoidance and local deadlock resolution. These modules generate *reference trajectories* in real-time for robots' controllers' to track. Dynamic feasibility of the generated reference trajectories is desirable so that the downstream single-robot controller can track them without major trajectory tracking imperfections, which allows the system to realize the collision avoidance and deadlock resolution behavior of the MH-NDM. This module uses the desired trajectories as guidance, such that it generates reference trajectories that are close to the desired trajectories, but allows deviations from them as necessary. Since the robot can maintain only the local representation of the environment because of memory and storage limitations, and can reason about a limited horizon during decision making because of the real-time operation, the reference trajectories are computed in a receding horizon fashion. The MH-NDM modules across multiple robots can interact with each other either explicitly through communication, and/or implicitly using the fact that all teammates run the same algorithm to allow inter-robot collision avoidance and cooperation. The module should not depend on perfect instantaneous communication assumption, and be robust to communication imperfections.

**Short Horizon Navigation Decision Making.** The single-robot controller computes control actions to track the reference trajectories. However, collision avoidance behavior enforced by MH-NDM module is not enough for safe operation. The reasons are i) medium horizon planners ensure safe operation not for the full navigation task but for the medium term future and all state-of-the-art medium horizon planners fail occasionally because the solved problem has no feasibility guarantee, which may cause reference trajectories to be unsafe when they are used for a long duration, and ii) even if the reference trajectories are dynamically feasible, robots may diverge from them because of state estimation inaccuracies, unmodeled dynamics, disturbances, or imperfect system identification. In order to allow collision-free operation in these cases, we utilize the SH-NDM module. This module is a high frequency safety module that takes control inputs generated by the controller, which we call the *desired control inputs*, and makes them safe in a minimally invasive manner, computing safe control inputs that are as close as possible to the desired control inputs, which are subsequently sent to the actuators. SH-NDM modules do not reason about deadlock-free operation as that responsibility is shared between the LH-NDM and MH-NDM modules. Its main aim is to ensure that the control inputs sent to the actuators do not result in collisions. Similarly to the MH-NDM modules, the SH-NDM modules across multiple robots can interact with each other explicitly or implicitly, and they should not depend on perfect communication assumption to enforce safety.

The system (Figure 7.2) is designed in such a way that desired trajectories computed by the LH-NDM guide the MH-NDM modules, and the desired control inputs generated to track the reference trajectories guide the SH-NDM module. The inputs and outputs of on-board safety modules have the same structure, which allows removing MH-NDM or SH-NDM modules by connecting their inputs to their outputs, if the benefits they provide are not needed. In general, i) LH-NDM module should aim to provide desired trajectories that results in low spatiotemporal deviation of the reference trajectories computed by the MH-NDM from the desired ones, and ii) the MH-NDM module should aim to provide reference trajectories that results in low spatiotemporal deviation of the robot position from the reference trajectories after executing the actions generated by the SH-NDM module in order for effective navigation. Otherwise, robots will take a considerably longer time to reach their goal positions, as the lower level decision making modules will override higher level decisions in order to enforce safety.

### 7.2.3 Implementation for Quadrotor Flight

We implement our design for quadrotor flight in simulations using the RotorS simulator [32] and the Robot Operating System (ROS) [89] in C++. RotorS is a plugin for Gazebo [49], and we simulate physics using the Open Dynamics Engine (ODE)[†]. The quadrotors are controlled via speed commands for each of the four motors. The implementation details for the quadrotor flight are discuseed in Appendix C.

We use octrees [41] to represent static obstacles. We use position and velocities as the sensed states of the teammates. Dynamic obstacle positions, velocities, and shapes are sensed by the teammates. Each object's shape is modeled as an axis aligned box.

**Long Horizon Navigation Decision Making.** The LH-NDM module randomly generates goal positions for the robots, computes shortest paths from robots' current positions to the goal positions avoiding only the prior static obstacles, which we set to either the actual map of the environment or an empty map in the experiments, using A* search and assigns durations to each segment of the computed path. Our LH-NDM module does not enforce dynamic feasibility, because LH-NDM module does not enforce collision avoidance against dynamic obstacles or teammates, which requires necessary divergence from desired

[†] http://ode.org/

trajectories during execution. When a robot reaches to its goal position, the LH-NDM module randomly generates a new goal position, and computes a new desired trajectory.

**Medium Horizon Navigation Decision Making.** We use DREAM (Chapter 6) as the MH-NDM module. To recap, for static obstacle avoidance, it requires shapes of the obstacles and their existence probabilities. We provide the static obstacles from the octree to DREAM. We do not implement probabilistic sensing, hence, static obstacles have existence probability $1.0$. For dynamic obstacle avoidance, DREAM requires i) current shapes and positions of the dynamic obstacles, and ii) a probability distribution across possible behavior models. A behavior model $\mathcal{B} = (\mathcal{M}, \mathcal{I})$ is a tuple of movement model $\mathcal{M}$ and interaction model $\mathcal{I}$. A movement model $\mathcal{M} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is a function mapping the dynamic obstacle's position to its desired velocity, describing the intention of the obstacle. An interaction model $\mathcal{I} : \mathbb{R}^{12} \rightarrow \mathbb{R}^3$ is a function describing the interaction of the dynamic obstacle with teammates, mapping the dynamic obstacle's position, desired velocity, the ego robot's position and velocity to the obstacle's velocity. In the simulations, each dynamic obstacle is modeled as running its movement model to obtain its velocity, and running its interaction model against each teammate, and executing the average interaction model outputs as its velocity. The on-board systems do not have explicit access to the behavior models of the dynamic obstacles, and use on-board probabilistic behavior model estimation instead as explained in Section 6.4.1.2. The output of the behavior prediction is a set of behavior models $\mathcal{B}_{i,j}$ for each dynamic obstacle $i$ and probabilities $p_{i,j}$ assigned to them, reflecting that dynamic obstacle $i$ moves according to behavior model $\mathcal{B}_{i,j}$ with probability $p_{i,j}$. Necessarily, $\sum_j p_{i,j} \leq 1$ holds for all $i$. For teammate avoidance, DREAM relies on discretized separating hyperplane trajectories (DSHTs) (Chapter 5), which are the trajectories of separating hyperplanes between two moving bodies, discretized at a high frequency. DSHT based constraints allow the active trajectories of each pair of robots to share a separating hyperplane constraint at all times under asynchronous planning and communication imperfections as discussed in Section 5.5.3. Robots prune DSHTs whenever they know another robot has planned successfully. MH-NDM module computes hard-margin support vector machine hyperplanes between the ego robot and each other robot at a high frequency and updates the DSHTs in real-time. Whenever MH-NDM plans successfully, it broadcasts a message to other robots. If others receive this message, they prune the DSHT between the sender robot and themselves as explained in Chapters 5 and 6.

**Controller and Short Horizon Decision Making.** The RotorS simulator accepts angular motor speed inputs for each of the four propellers of the quadrotor. The forces and moments applied by the propellers are simulated based on the model of a single propeller near hovering [65, 45]. Our design (Figure 7.2) necessitates a SH-NDM module that accepts desired control inputs and outputs safe control inputs in a minimally invasive manner, considering static and dynamic obstacles as well as teammates. We integrate a quadrotor controller [54] that internally computes desired acceleration commands with an safety barrier certificates (SBC) based SH-NDM module for double integrator systems [121] allowing static and dynamic obstacle and cooperative teammate avoidance[‡]. By doing so, we diverge from the proposed design slightly, because we do not ensure the safety of the final motor speeds, but ensure that the intermediate accelerations computed by the controllers are safe.

The controller computes a desired body acceleration using a PD controller to track reference trajectory samples composed of desired position, velocity, and yaw angles. We set desired yaw angles to zero. Then, the controller computes the desired angular acceleration of the quadrotor in each of the three frame axes in order to produce the given desired acceleration and yaw. Afterwards, it computes the desired total body thrust and torques to achieve the desired acceleration and angular accelerations. Next, it computes the forces that need to be generated by the propellers in order to produce the desired total body thrust and torques. The conversion of the desired forces to motor speed commands are done similarly to the simulator.

[‡] Some alternatives are Wang, Ames, and Egerstedt [122]'s approach, which makes the reference trajectories safe utilizing SBCs, and Wu and Sreenath [126]'s approach, which computes forces and moments directly, but does not explicitly model multiple robots. We use Wang, Ames, and Egerstedt [121]'s approach because it explicitly accounts for multiple robots and has extensive evaluations

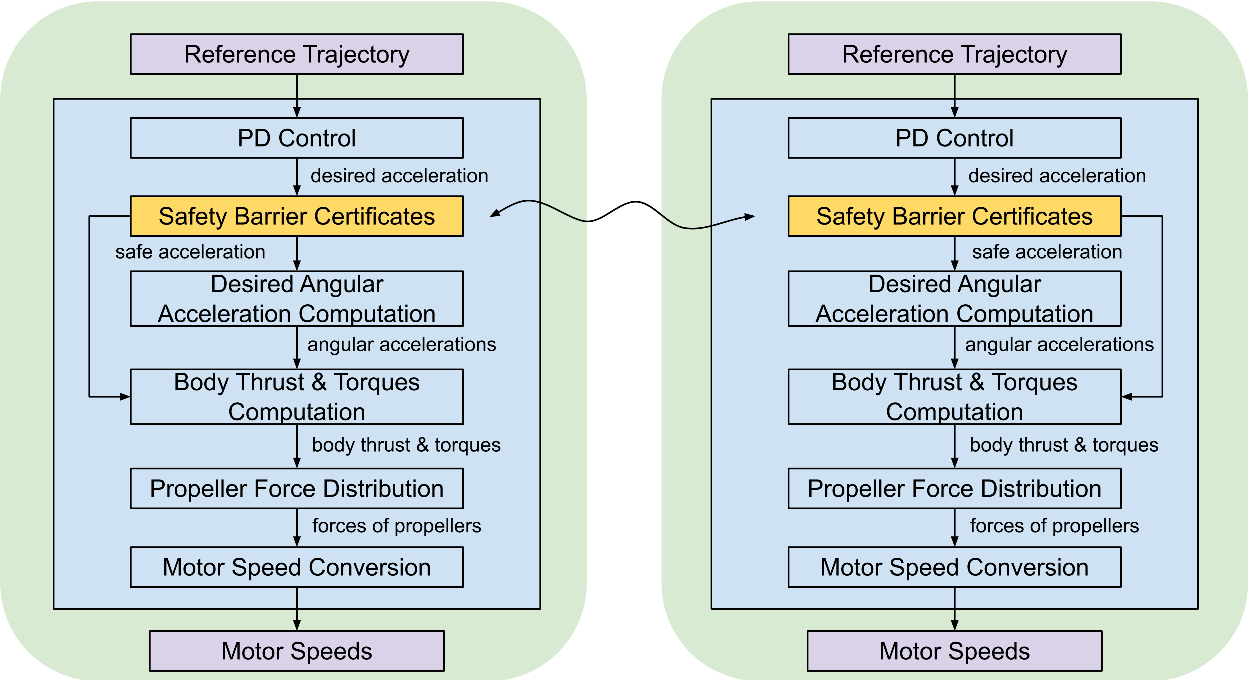


Figure 7.3: **Controller and SH-NDM Module Integration.** Safety barrier certificates based SH-NDM module [121] assumes double integrator dynamics, i.e., that robots can be controlled with acceleration commands. Since quadrotors cannot be directly controlled with acceleration commands, we augment a quadrotor controller [54] with SBC by inserting it after desired acceleration computation within the controller. The SH-NDM modules in different robots interact assuming that all robots run the same algorithm.

We insert the SH-NDM module [121] within the controller after desired body acceleration computation (Figure 7.3).

## 7.3 Evaluation

In order to show the efficacy of our design, i) we ablate long, medium, and short horizon decision making modules and show that all components are required for collision and deadlock-free navigation, and ii) we run our system in an uninterrupted eight hours wall time (six hours simulation time) long simulation with static and dynamic obstacles as well as multiple robots without collisions and deadlocks. We also discuss the limits of our design by increasing the number and speeds of dynamic obstacles until our design is not effective anymore.

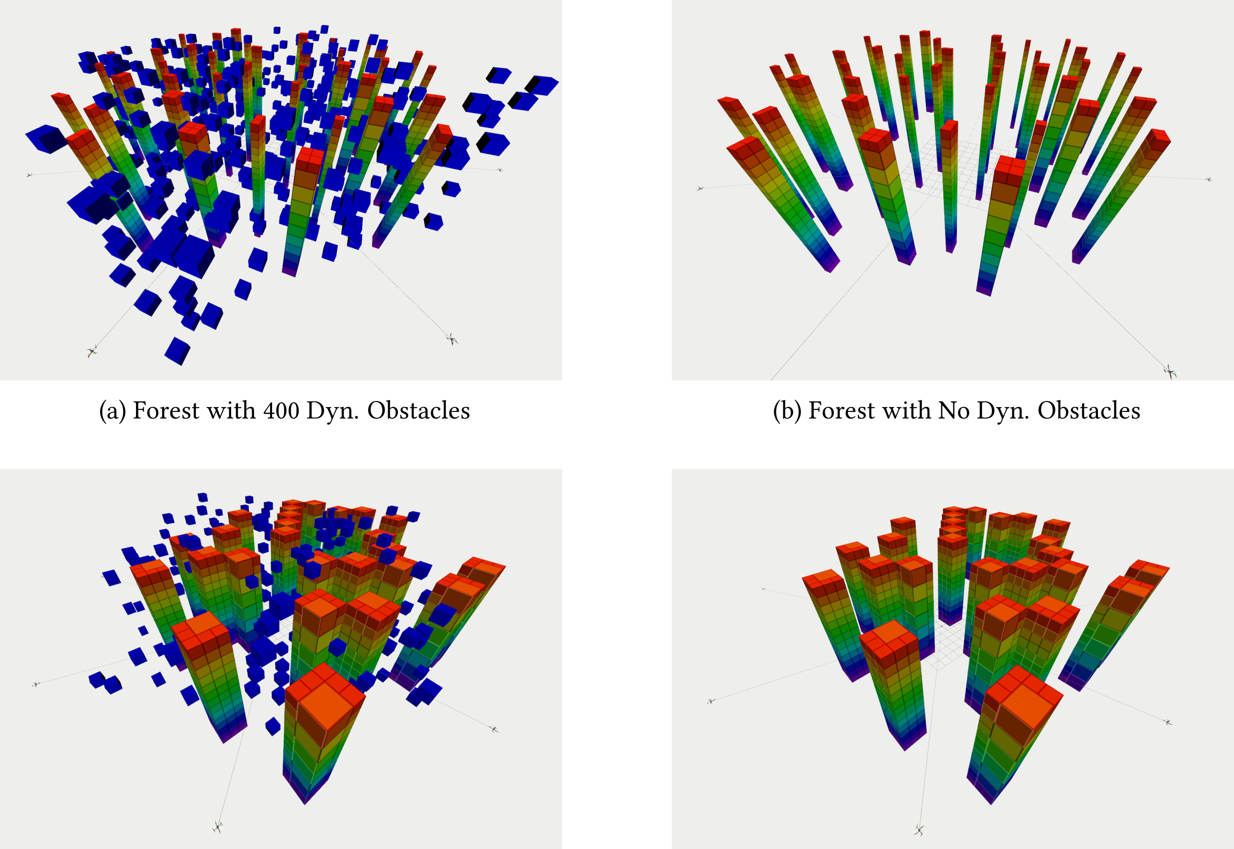
### 7.3.1 Ablation Study

In our ablation study, eight quadrotors navigate in a forest-like or a maze-like environment. The maze-like environment contains concave regions, in which the robots may stall and deadlock. Dynamic obstacles are randomly generated. Each dynamic obstacle has an axis aligned box shape with a diagonal length sampled uniformly in $[1.2\,\mathrm{m}, 1.6\,\mathrm{m}]$. Movement models generate desired velocities so that the dynamic obstacles rotate around the line going perpendicular to the x-y plane and goes through the origin. The norm of the desired velocities are sampled uniformly in $[0.25\,\frac{\mathrm{m}}{\mathrm{s}}, 0.35\,\frac{\mathrm{m}}{\mathrm{s}}]$ once, i.e., they move with constant speed. In the ablation study, we do not use interactive dynamic obstacles since the interactions would give some of the collision avoidance responsibility to the dynamic obstacles, which would make it hard to evaluate the contributions of decision making modules. All objects exist in a $32 \times 32 \times 12m^3$ box.

We run each experiment for $300\,\mathrm{s}$ in simulation time. Each randomly generated goal position by the LH-NDM module is called a task that should be navigated to. When we ablate the SH-NDM module, we set safe acceleration to the desired acceleration (Figure 7.3). When we ablate the MH-NDM module, we set reference trajectory to the desired trajectory. When we ablate the LH-NDM module, we provide an empty map as the prior static obstacles, which causes LH-NDM to produce straight lines to the goal positions as desired trajectories, otherwise we provide the real map to it.

We use four evaluation metrics: i) deadlock rate (**Deadl. R.**), the ratio of robots that deadlock at the end of the experiment, ii) collision rate (**Coll. R.**), the ratio of tasks during which the robot has collided with an object, iii) completion rate (**Comp. R.**), the ratio of tasks robots complete without a collision, and iv) average navigation duration (**Avg. N. Dur.**), the average duration for robots to complete no-collision tasks[§].

[§] Deadlock, collision, and completion rates are primary metrics to compare module combinations. The average navigation duration is a secondary metric and is only meaningful when there are no deadlocks or collisions.



(a) Forest with 400 Dyn. Obstacles

(b) Forest with No Dyn. Obstacles

(c) Maze with 200 Dyn. Obstacles

(d) Maze with No Dyn. Obstacles

Figure 7.4: The environments used in the ablation study. The dynamic obstacles are blue boxes, and static obstacles are the pillars.

Table 7.1: Forest environment with 400 dynamic obstacles

| Modules | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| SH | 0/8 | 5/132 | 127/132 | 13.19 |
| MH | 0/8 | 73/216 | 143/216 | 10.52 |
| LH | 0/8 | 246/313 | 67/246 | 4.34 |
| **MH+SH** | 0/8 | 0/148 | 148/148 | 15.45 |
| LH+SH | 2/8 | 4/159 | 155/159 | 11.93 |
| LH+MH | 0/8 | 102/220 | 118/220 | 10.13 |
| **LH+MH+SH** | 0/8 | 0/158 | 158/158 | 14.79 |

Table 7.2: Forest environment with no dynamic obstacles

| Modules | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| SH | 6/8 | 0/158 | 158/158 | 11.43 |
| MH | 0/8 | 4/167 | 163/167 | 13.93 |
| LH | 0/8 | 37/313 | 276/313 | 7.30 |
| **MH+SH** | 0/8 | 0/157 | 157/157 | 13.04 |
| **LH+SH** | 0/8 | 0/335 | 335/335 | 6.99 |
| LH+MH | 0/8 | 3/193 | 190/193 | 11.94 |
| **LH+MH+SH** | 0/8 | 0/185 | 185/185 | 12.95 |

There are four cases we use: i) forest-like environment with 400 dynamic obstacles (Figure 7.4a), or ii) no dynamic obstacles (Figure 7.4b), and iii) maze-like environment with 200 dynamic obstacles (Figure 7.4c), or iv) no dynamic obstacles (Figure 7.4d). The results of our experiments are summarized in Tables 7.1 to 7.4. Module combinations that result in zero deadlock and collision rates are highlighted in the tables. Recordings can be found in the supplemental videos.

**Enforcing safety of each executed action is essential.** The SH-NDM module enforces that each action executed is safe. We observe that this is an essential property for the effectiveness of the design. All successful combinations contain the SH-NDM module. It i) enables highly reactive collision avoidance behavior, ii) ensures safety when the MH-NDM fails, and iii) allows arbitrary divergence from the desired and reference trajectories without giving up safety.

**Dynamic obstacles create the need for local deadlock resolution.** When there are no dynamic obstacles, LH+SH module combination is successful in both forest and maze-like environments. Since there are no dynamic obstacles, desired trajectories generated by the LH-NDM module avoid all obstacles.

Table 7.3: Maze environment with 200 dynamic obstacle

| Modules | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| SH | 8/8 | 12/39 | 27/39 | 14.84 |
| MH | 2/8 | 49/107 | 58/107 | 12.27 |
| LH | 0/8 | 149/286 | 137/286 | 6.74 |
| MH+SH | 7/8 | 6/22 | 16/22 | 15.21 |
| LH+SH | 6/8 | 2/141 | 139/141 | 10.38 |
| LH+MH | 0/8 | 29/183 | 154/183 | 12.23 |
| **LH+MH+SH** | 0/8 | 0/142 | 142/142 | 16.20 |

Table 7.4: Maze environment with no dynamic obstacles

| Modules | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| SH | 8/8 | 0/20 | 20/20 | 6.19 |
| MH | 6/8 | 16/42 | 26/42 | 15.54 |
| LH | 0/8 | 43/287 | 244/287 | 7.81 |
| MH+SH | 8/8 | 0/4 | 4/4 | 16.46 |
| **LH+SH** | 0/8 | 0/274 | 274/274 | 8.49 |
| LH+MH | 0/8 | 6/145 | 139/145 | 15.76 |
| **LH+MH+SH** | 0/8 | 0/151 | 151/151 | 15.44 |

SH-NDM module is enough to locally resolve collisions between teammates, after which, the robot is able to navigate back to its desired trajectory.

When dynamic obstacles are added to the environment, LH+SH is not effective anymore because the desired trajectories do not avoid dynamic obstacles, and hence, dynamic obstacle avoidance responsibility is given to the SH-NDM module. This causes robots to diverge from their desired trajectories considerably, causing robots to not be able to navigate back to them using only SH-NDM, resulting in deadlocks. The LH+MH+SH combination is effective in such scenarios, because the MH-NDM allows local deadlock resolution.

**Medium horizon decision making increases navigation duration.** If the LH+SH module combination is successful, adding MH-NDM increases the average navigation duration (Tables 7.2 and 7.4), because, we impose lower dynamic limits on the reference trajectories than the actual dynamic limits of the quadrotors for the real-time executability during MH-NDM. Otherwise, the planner needs to generate

Table 7.5: Maze environment with varying number of dynamic obstacles

| # Dyn. | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| 200 | 0/8 | 0/142 | 142/142 | 16.20 |
| 300 | 0/8 | 1/146 | 145/146 | 15.88 |
| 400 | 0/8 | 4/132 | 128/132 | 16.97 |
| 500 | 2/8 | 7/95 | 88/95 | 20.54 |
| 600 | 3/8 | 7/98 | 91/98 | 17.89 |

spatially longer trajectories with the same temporal horizon, which increases the run-time of the algorithm to achieve a similar collision avoidance performance.

**Global deadlock resolution requires global reasoning.** In the maze environment, all successful combinations include LH whether regardless of dynamic obstacles. This is the case because the robot needs to reason about long trajectories to navigate in such environments. The system needs to provide global reasoning capability for global deadlock resolution.

**No approach is sufficient by itself and all is required for effectiveness in all cases.** None of LH, MH, or SH-NDM modules are sufficient for navigating collision and deadlock-free in neither forest nor maze-like environments by themselves. The LH+MH+SH combination is the only one that allows collision and deadlock-free navigation in all cases.

### 7.3.2 Long Term Execution

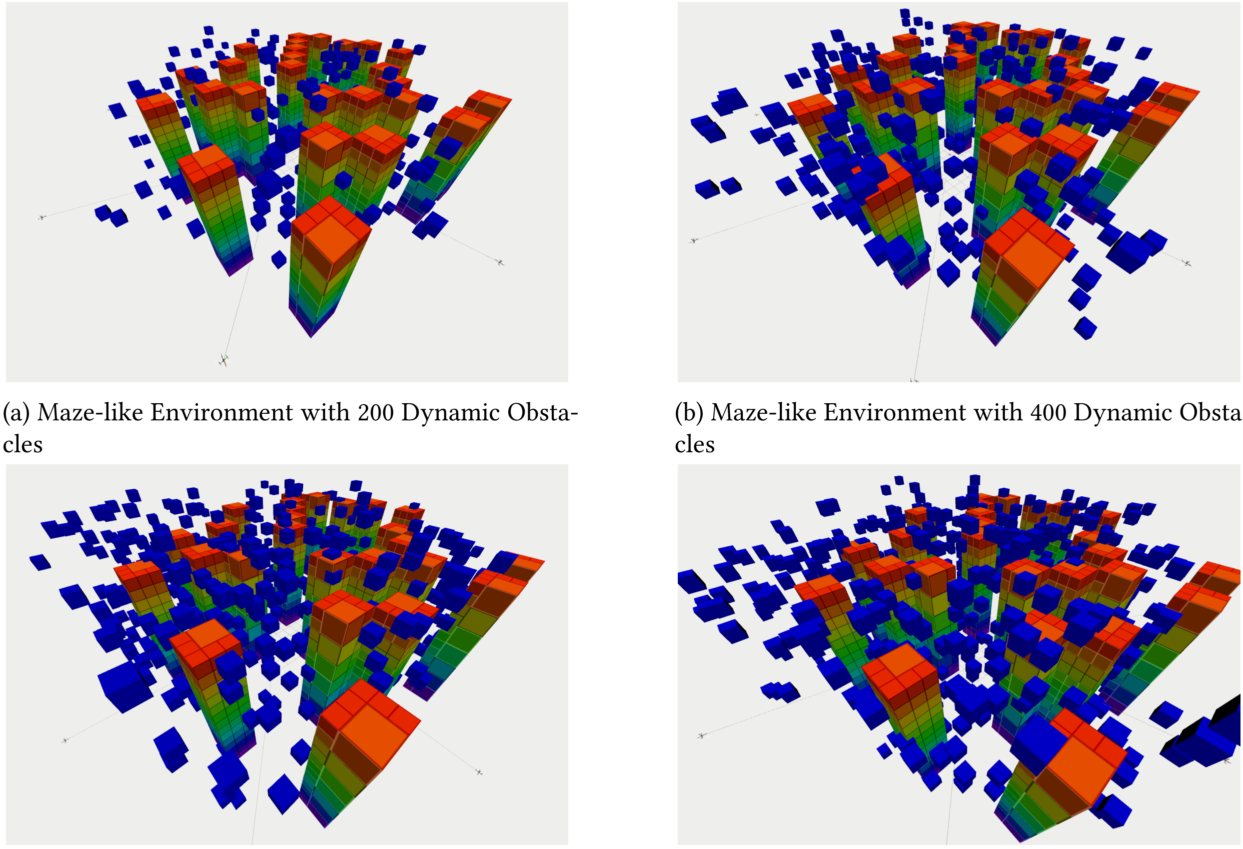
In order to show the efficacy of our design, we run our system with LH+MH+SH module combination in an eight hours long wall time (six hours long simulation time) simulation *without any collisions and deadlocks.* The environment is forest-like and contains 300 randomly generated dynamic obstacles in which eight quadrotors navigate. The recording be found in the long term execution supplemental video.

### 7.3.3 Increasing the Number and Speeds of Dynamic Obstacles

As the environment hardness increases, it becomes more difficult for robots to ensure collision and deadlock avoidance. To evaluate the performance of our approach as the environment hardness increases, we

Table 7.6: Maze environment with 200 dynamic obstacles with varying sampling ranges of desired speeds

| D. Spd. Rng. [$\frac{m}{s}$] | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| $[0.25, 0.35]$ | $0/8$ | $0/142$ | $142/142$ | 16.20 |
| $[0.5, 0.7]$ | $0/8$ | $1/148$ | $148/148$ | 15.16 |
| $[1.0, 1.4]$ | $0/8$ | $8/152$ | $144/152$ | 14.54 |
| $[2.0, 2.8]$ | $3/8$ | $19/71$ | $52/72$ | 17.14 |
| $[4.0, 5.6]$ | $7/8$ | $10/12$ | $2/12$ | 24.08 |



(a) Maze-like Environment with 200 Dynamic Obstacles

(b) Maze-like Environment with 400 Dynamic Obstacles

(c) Maze-like Environment with 500 Dynamic Obstacles

(d) Maze-like Environment with 600 Dynamic Obstacles

Figure 7.5: The environments used in the environment hardness study with number of dynamic obstacles. The dynamic obstacles are blue boxes, and static obstacles are the pillars.

use the maze environment used in the ablation study, and i) increase the number of dynamic obstacles and ii) increase the desired speeds of the dynamic obstacles. Similarly to the ablation study, we run the simulations for $300\,\mathrm{s}$. All of the parameters are same as the ablation study except for the number of dynamic obstacles and desired speeds. The robots navigate with LH+MH+SH module combination.

We increase the number of dynamic obstacles from 200 to 600 (Figure 7.5). The effects of the number of dynamic obstacles are given in Table 7.5. In general, as the number of dynamic obstacles increases, deadlock and collision rates increase and completion rate as well as number of completed tasks decrease.

In the maze environment with 200 dynamic obstacles, we increase the limits of desired speed sampling ranges. The effects of increasing desired speeds are given in Table 7.6. The performance of the navigation system decreases considerably as the speed sampling ranges increase. For instance, when the desired speeds of dynamic obstacles are sampled in range $[4.0\,\frac{\mathrm{m}}{\mathrm{s}}, 5.6\,\frac{\mathrm{m}}{\mathrm{s}}]$, 7 out of 8 robots deadlock, and only 2 tasks are completed without collisions.

We attribute the performance decrease with increased environment hardness to four reasons. First, capabilities of medium and short horizon decision making modules for deadlock and collision resolution limit the performance of the system. As the number or speeds of dynamic obstacles increase, the robots diverge more and more from their desired trajectories in order to avoid collisions, which causes robots to deadlock when they cannot locally resolve them. In addition, the short horizon decision making module assumes dynamic obstacles move with their current velocities for the immediate future, which is an approximation of the reality. As the environment gets more dense, this approximation results in infeasibilities in the short horizon decision making module. Second, dynamic limits of the quadrotors limits the capabilities of the robots fundamentally. When a dynamic obstacle moves towards a quadrotor at a speed such that quadrotor cannot avoid it in a dynamically feasible manner, collisions are inevitable. Third, the divergence from our design when integrating the short horizon decision making module to the system (Figure 7.3) is another point of approximation. Since the safe acceleration computed by the short horizon module is

Table 7.7: Maze environment with 200 dynamic obstacles with different levels of interactivity (repulsion strength)

| R. Strength | Deadl. R. | Coll. R. | Comp. R. | Avg. N. Dur. [s] |
|---|---|---|---|---|
| 0.0 | 0/8 | 8/152 | 144/152 | 14.54 |
| 0.25 | 0/8 | 5/120 | 115/120 | 15.95 |
| 0.5 | 0/8 | 3/128 | 125/128 | 15.58 |
| 0.75 | 0/8 | 4/138 | 134/138 | 16.26 |
| 1.5 | 0/8 | 1/147 | 146/147 | 15.79 |
| 3.0 | 0/8 | 0/152 | 152/152 | 15.11 |

used to compute the final control inputs, which is a process that attempts to produce the safe acceleration, safety enforced by the short horizon decision making module is approximately obeyed by the rest of the controller. As the number or speeds of the dynamic obstacles increase, robots' safe accelerations change more abruptly, causing the rest of the controller to have increased error. Forth, non-interactivity of the dynamic obstacles burdens the full collision avoidance responsibility to the quadrotors. As the number or speeds of the dynamic obstacles increase, it becomes more difficult for the quadrotors to avoid collisions. Sometimes, the quadrotors end up in states in which they are cornered by the dynamic obstacles, and there is no feasible way of avoiding collisions.

In order to show how interactivity of dynamic obstacles help, we set the interaction models of the dynamic obstacles to the repulsive models as explained in Section 6.4.1.2, which repulse the dynamic obstacles away from the robots. Repulsive interaction model has a repulsion strength parameter. As the repulsion strength increases the dynamic obstacles are repulsed more and more. We use the dynamic obstacle desired speed experiment with desired speed range $[1.0\,\frac{\text{m}}{\text{s}}, 1.4\,\frac{\text{m}}{\text{s}}]$ and gradually increase the repulsion strength of the dynamic obstacles. The results are shown in Table 7.7. As the repulsion strength increases, collision rate decreases to 0.

## 7.4 Conclusion

In this chapter, we propose MRNAV, a multi-robot aware planning and control stack design for collision and deadlock-free navigation in cluttered environments with static and dynamic obstacles. Our design utilizes three qualitatively different decision making modules. We assign responsibilities to the decision making modules, arguing that they complement each other. MRNAV is the first hierarchical approach combining these three qualilatively different decision making modules. We define the abstractions we require for the interfaces to the perception, localization, mapping, and prediction subsystems. We implement our design for simulated multi-quadrotor flight, and discuss our algorithm choices and integration of the modules to the simulation environment. In our evaluations, we show that all three decision making modules are required for effective navigation in diverse types of environments. We show the long-term executability of our approach in an eight hour uninterrupted run without collisions or deadlocks.

# Chapter 8

# Conclusions & Future Directions

## 8.1 Conclusions

Collision-free (multi-)robot navigation in cluttered environments is a fundamental enabler for many real-world applications. The environments may contain static obstacles as well as dynamic obstacles that can optionally be interactive. In this dissertation, we improve the state-of-the-art in decentralized real-time multi-robot planning in Chapters 4 to 6 and introduce the context such a decentralized planner can be used to enable effective navigation in Chapter 7.

The wide real-life applicability of the developed methods is the main central concern of ours. To that end, we i) embrace on-board compute, memory and storage limitations, ii) do not rely on communication for safe operation and explicitly account for communication imperfections, iii) develop algorithms that can work with imperfect a priori knowledge, iv) ensure dynamic feasibility of the plans and embrace controller imperfections, v) work with minimal sensing and estimation capabilities, and v) provide highly reactive collision avoidance behavior.

In Chapter 4, we introduce RLSS, a real-time decentralized multi-robot trajectory planning algorithm for static obstacle and teammate avoidance, that does not rely on communication and uses position/shape only sensing, forming the basis of the future algorithms. In Chapter 5, we introduce a constraint generation, overconstraining, and constraint-discarding scheme to allow inter-robot collision avoidance under asynchronous planning, which is inherent in decentralized approaches. In Chapter 6, we introduce DREAM, a real-time decentralized multi-robot trajectory planning algorithm for static and interactive dynamic obstacle as well as teammate avoidance under asynchronous planning, imperfect static obstacle sensing, imperfect dynamic obstacle prediction, and imperfect communication. DREAM draws inspiration from RLSS and our constraint generation method, and introduces new concepts for interactive dynamic obstacle avoidance. In Chapter 7, we introduce MRNAV, a full planning and control stack to allow multi-robot navigation in diverse types of environments. MRNAV combines three qualitatively different decision making approaches, i.e., long, medium, and short horizon, to allow highly successful collision-free and deadlock-free operation. We use DREAM as the medium horizon decision making module in MRNAV, and show that when it is combined with long and short horizon methods, the system is highly effective. We verify our algorithms in simulations with or without physics, and on real robots to show their success as well as real-word applicability.

## 8.2 Future Directions

The algorithms and approaches we introduce can be improved in several directions.

- **Worst-case dynamic obstacle behavior.** DREAM (Chapter 6) allows multiple behavior models for each dynamic obstacle. However, dynamic obstacles may move outside of these predictions. While artificial inflation of shapes can be used to tackled this inaccuracy, a more principled approach would be to utilize a reachability based approach during decision making. The output of the prediction system can be a set of behavior models as explained Chapter 6 as well as a reachability based residual model with appropriate realization probabilities. During decision making, probabilities can be computed using the regular behavior models as well as the residual model.

- **Uncertainty of the state transition.** DREAM (Chapter 6) does not model uncertainty associated with state transition of the ego robot. Using state transition uncertainties of velocity commands during discrete search would allow a more general framework to allow probabilistic collision avoidance. Preserving such collision probabilities during trajectory optimization is another avenue for future work.

- **Dynamic obstacle interactivity during short horizon decision making.** In the implementation of MRNAV (Chapter 7), we use Wang, Ames, and Egerstedt [121]'s safety barrier certificate based approach, which models dynamic obstacles as constant velocity objects. Explicitly modeling interactivity of dynamic obstacles under multiple behavior models during SH-NDM would increase the accuracy of the approach, and decrease the conservativeness that is required to account for unmodeled interactivity.

- **Full stack integration.** MRNAV (Chapter 7) is a planning and control stack, and we define the interfaces to other subsystems that might exist in a full navigation system. During experiments, we mock the behavior of these systems. Integration perception, prediction, mapping, and localization subsystems to MRNAV would allow a better understanding of its performance under real-world conditions.

- **Communicating plans explicitly.** In its latest form, DREAM (Chapter 6) utilizes communication only to prune discretized separating hyperplane trajectories. However, robots could use communication to share their plans as well as the durations up to which they will commit to them. Combining DSHTs for uncommitted time intervals with communicated plans for committed time intervals could help decrease the conservativeness of DSHTs.

# Bibliography

[1] Faten Aljalaud and Nathan Sturtevant. “Finding bounded suboptimal multi-agent path planning solutions using increasing cost tree search”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 4. 1. 2013, pp. 203–204.

[2] Jean-Marc Alkazzi, Anthony Rizk, Michel Salomon, and Abdallah Makhoul. “MAPFASTER: A Faster and Simpler take on Multi-Agent Path Finding Algorithm Selection”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10088–10093. DOI: 10.1109/IROS47612.2022.9981981.

[3] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. “Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots”. In: *Distributed Autonomous Robotic Systems: The 10th International Symposium*. 2013, pp. 203–216. DOI: 10.1007/978-3-642-32723-0.

[4] Georges S Aoude, Brandon D Luders, Joshua M Joseph, Nicholas Roy, and Jonathan P How. “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns”. In: *Autonomous Robots* 35.1 (2013), pp. 51–76.

[5] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. “Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 5. 2014, pp. 19–27.

[6] Federico Bartoli, Giuseppe Lisanti, Lamberto Ballan, and Alberto Del Bimbo. “Context-Aware Trajectory Prediction”. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. 2018, pp. 1941–1946. DOI: 10.1109/ICPR.2018.8545447.

[7] Sumeet Batra, Zhehui Huang, Aleksei Petrenko, Tushar Kumar, Artem Molchanov, and Gaurav S. Sukhatme. “Decentralized Control of Quadrotor Swarms with End-to-end Deep Reinforcement Learning”. In: *5th Conference on Robot Learning, CoRL 2021*. Proceedings of Machine Learning Research. 2021.

[8] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. “Reciprocal n-Body Collision Avoidance”. In: *Robotics Research*. 2011, pp. 3–19. ISBN: 978-3-642-19457-3.

[9] Subhrajit Bhattacharya, Vijay Kumar, and Maxim Likhachev. “Search-Based Path Planning with Homotopy Class Constraints”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI’10. Atlanta, Georgia, 2010, pp. 1230–1237.

[10] Franco Blanchini. “Set invariance in control”. In: *Automatica* 35.11 (1999), pp. 1747–1767.

[11] Liefeng Bo, Ling Wang, and Licheng Jiao. “Training Hard-Margin Support Vector Machines Using Greedy Stagewise Algorithm”. In: *IEEE Transactions on Neural Networks* 19.8 (2008), pp. 1446–1455. DOI: 10.1109/TNN.2008.2000576.

[12] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[13] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[14] Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. “Autonomous driving in urban environments: approaches, lessons and challenges”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 368.1928 (2010), pp. 4649–4672.

[15] G. Campion, G. Bastin, and B. Dandrea-Novel. “Structural properties and classification of kinematic and dynamic models of wheeled mobile robots”. In: *IEEE Transactions on Robotics and Automation* 12.1 (1996), pp. 47–62. DOI: 10.1109/70.481750.

[16] Gang Chen, Siyuan Wu, Moji Shi, Wei Dong, Hai Zhu, and Javier Alonso-Mora. “RAST: Risk-Aware Spatio-Temporal Safety Corridors for MAV Navigation in Dynamic Uncertain Environments”. In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 808–815. DOI: 10.1109/LRA.2022.3231832.

[17] Jing Chen, Tianbo Liu, and Shaojie Shen. “Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1476–1483. DOI: 10.1109/ICRA.2016.7487283.

[18] Jingkai Chen, Jiaoyang Li, Chuchu Fan, and Brian C Williams. “Scalable and safe multi-agent motion planning with nonlinear dynamics and bounded disturbances”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 2021, pp. 11237–11245.

[19] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[20] Yuxiang Cui, Longzhong Lin, Xiaolong Huang, Dongkun Zhang, Yunkai Wang, Wei Jing, Junbo Chen, Rong Xiong, and Yue Wang. “Learning Observation-Based Certifiable Safe Policy for Decentralized Multi-Robot Navigation”. In: *International Conference on Robotics and Automation*. 2022, pp. 5518–5524.

[21] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. “PRIMAL2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2666–2673.

[22] George B Dantzig. “Maximization of a linear function of variables subject to linear inequalities”. In: *Activity analysis of production and allocation* 13 (1951), pp. 339–347.

[23] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[24] Mark Debord, Wolfgang Hönig, and Nora Ayanian. “Trajectory planning for heterogeneous robot teams”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7924–7931.

[25] A. Desai and N. Michael. “Online Planning for Quadrotor Teams in 3-D Workspaces via Reachability Analysis On Invariant Geometric Trees”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 8769–8775. DOI: 10.1109/ICRA40945.2020.9197195.

[26] Arjav Desai, Matthew Collins, and Nathan Michael. “Efficient Kinodynamic Multi-Robot Replanning in Known Workspaces”. In: *International Conference on Robotics and Automation (ICRA)*. 2019, pp. 1021–1027. DOI: 10.1109/ICRA.2019.8793647.

[27] Kurt Dresner and Peter Stone. “A multiagent approach to autonomous intersection management”. In: *Journal of Artificial Intelligence Research* 31 (2008), pp. 591–656. DOI: 10.1613/jair.2502.

[28] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch-Lafuente. “Cost-algebraic heuristic search”. In: *AAAI*. 2005, pp. 1362–1367.

[29] Rida T. Farouki. “The Bernstein polynomial basis: A centennial retrospective”. In: *Computer Aided Geometric Design* 29.6 (2012), pp. 379–419. DOI: 10.1016/j.cagd.2012.03.001.

[30] Giuseppe Fedele, Luigi D’Alfonso, Francesco Chiaravalloti, and Gaetano D’Aquila. “Obstacles avoidance based on switching potential functions”. In: *Journal of Intelligent & Robotic Systems* 90.3 (2018), pp. 387–405.

[31] Paolo Fiorini and Zvi Shiller. “Motion Planning in Dynamic Environments Using Velocity Obstacles”. In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772. DOI: 10.1177/027836499801700706.

[32] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. “Robot Operating System (ROS): The Complete Reference (Volume 1)”. In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23.

[33] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. “Online Safe Trajectory Generation for Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 344–351. DOI: 10.1109/ICRA.2018.8462878.

[34] Shuzhi Sam Ge and Yun J Cui. “Dynamic motion planning for mobile robots using potential field method”. In: *Autonomous robots* 13.3 (2002), pp. 207–222.

[35] *GitHub - mit-acl/rmader: Decentralized Multiagent Trajectory Planner Robust to Communication Delay — github.com*. `https://github.com/mit-acl/rmader`. [Accessed 19-01-2023].

[36] Jacek Gondzio and Andreas Grothey. “Exploiting structure in parallel implementation of interior point methods for optimization”. In: *Computational Management Science* 6.2 (2009), pp. 135–160. DOI: `10.1007/s10287-008-0090-3`.

[37] Daniel Harabor and Alban Grastien. “Online graph pruning for pathfinding on grid maps”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. 1. 2011, pp. 1114–1119.

[38] Florian Homm, Nico Kaempchen, Jeff Ota, and Darius Burschka. “Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection”. In: *IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2010, pp. 1006–1013. DOI: `10.1109/IVS.2010.5548091`.

[39] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian. “Trajectory Planning for Quadrotor Swarms”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 856–869. DOI: `10.1109/TRO.2018.2853613`.

[40] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. “On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman’s Problem"”. In: *The International Journal of Robotics Research* 3.4 (1984), pp. 76–88. ISSN: 0278-3649. DOI: `10.1177/027836498400300405`.

[41] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous Robots* 34.3 (2013), pp. 189–206. DOI: `10.1007/s10514-012-9321-0`.

[42] Rafia Inam, Klaus Raizer, Alberto Hata, Ricardo Souza, Elena Forsman, Enyu Cao, and Shaolei Wang. “Risk Assessment for Human-Robot Collaboration in an automated warehouse scenario”. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2018, pp. 743–751. DOI: `10.1109/ETFA.2018.8502466`.

[43] Lucas Janson, Edward Schmerling, and Marco Pavone. “Monte Carlo motion planning for robot trajectory optimization under uncertainty”. In: *Robotics Research*. Springer, 2018, pp. 343–361.

[44] S. Jiang and K. Song. “Differential flatness-based motion control of a steer-and-drive omnidirectional mobile robot”. In: *IEEE International Conference on Mechatronics and Automation (ICMA)*. 2013, pp. 1167–1172. DOI: `10.1109/ICMA.2013.6618079`.

[45] Wayne Johnson. *Helicopter theory*. Courier Corporation, 2012.

[46] Sertac Karaman and Emilio Frazzoli. “Incremental sampling-based algorithms for optimal motion planning”. In: *Robotics Science and Systems VI* 104.2 (2010). DOI: `10.15607/RSS.2010.VI.034`.

[47] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[48] Sanmin Kim, Hyeongseok Jeon, Jun Won Choi, and Dongsuk Kum. “Diverse Multiple Trajectory Prediction Using a Two-Stage Prediction Network Trained With Lane Loss”. In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 2038–2045. DOI: 10.1109/LRA.2022.3231525.

[49] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[50] Kota Kondo, Jesus Tordesillas, Reinaldo Figueroa, Juan Rached, Joseph Merkel, Parker C. Lusk, and Jonathan P. How. “Robust MADER: Decentralized and Asynchronous Multiagent Trajectory Planner Robust to Communication Delay”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1687–1693. DOI: 10.1109/ICRA48891.2023.10161244.

[51] Justin Kottinger, Shaull Almagor, and Morteza Lahijanian. “Conflict-Based Search for Multi-Robot Motion Planning with Kinodynamic Constraints”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 13494–13499. DOI: 10.1109/IROS47612.2022.9982018.

[52] Edward Lam, Pierre Le Bodic, Daniel D. Harabor, and Peter J. Stuckey. “Branch-and-Cut-and-Price for Multi-Agent Pathfinding”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. 2019, pp. 1289–1296. DOI: 10.24963/ijcai.2019/179.

[53] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. “Desire: Distant future prediction in dynamic scenes with interacting agents”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 336–345.

[54] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. “Geometric tracking control of a quadrotor UAV on SE(3)”. In: *49th IEEE Conference on Decision and Control (CDC)*. 2010, pp. 5420–5425. DOI: 10.1109/CDC.2010.5717652.

[55] Bai Li, Shaoshan Liu, Jie Tang, Jean-Luc Gaudiot, Liangliang Zhang, and Qi Kong. “Autonomous Last-Mile Delivery Vehicles in Complex Traffic Environments”. In: *Computer* 53.11 (2020), pp. 26–35. DOI: 10.1109/MC.2020.2970924.

[56] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. “Eecbs: A bounded-suboptimal search for multi-agent path finding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 2021, pp. 12353–12362.

[57] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. “Graph neural networks for decentralized multi-robot path planning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 11785–11792. DOI: 10.1109/iros45743.2020.9341668.

[58] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J. Taylor, and Vijay Kumar. “Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1688–1695. DOI: 10.1109/LRA.2017.2663526.

[59] Brandon Luders, Mangal Kothari, and Jonathan How. “Chance constrained RRT for probabilistic robustness to environmental uncertainty”. In: *AIAA guidance, navigation, and control conference.* 2010, p. 8160.

[60] Carlos Luis, Marijan Vukosavljev, and Angela Schoellig. “Online Trajectory Generation With Distributed Model Predictive Control for Multi-Robot Motion Planning”. In: *IEEE Robotics and Automation Letters* PP (2020), pp. 1–1. DOI: 10.1109/LRA.2020.2964159.

[61] Carlos E. Luis and Angela P. Schoellig. “Trajectory Generation for Multiagent Point-To-Point Transitions via Distributed Model Predictive Control”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 375–382. DOI: 10.1109/LRA.2018.2890572.

[62] Ryan Luna and Kostas E Bekris. “Push and swap: Fast cooperative path-finding with completeness guarantees”. In: *IJCAI.* 2011, pp. 294–300.

[63] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. “Searching with Consistent Prioritization for Multi-Agent Path Finding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 7643–7650. DOI: 10.1609/AAAI.V33I01.33017643.

[64] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. “It is not the journey but the destination: Endpoint conditioned trajectory prediction”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16.* Springer. 2020, pp. 759–776.

[65] Philippe Martin and Erwan Salaün. “The true role of accelerometer feedback in quadrotor control”. In: *IEEE International Conference on Robotics and Automation.* 2010, pp. 1623–1629. DOI: 10.1109/ROBOT.2010.5509980.

[66] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *IEEE International Conference on Robotics and Automation (ICRA).* 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[67] D. Mills, J. Martin, J. Burbank, and W. Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification.* RFC 5905. June 2010, pp. 1–109. URL: https://datatracker.ietf.org/doc/html/rfc5905.

[68] R. M. Murray and S. S. Sastry. “Nonholonomic motion planning: steering using sinusoids”. In: *IEEE Transactions on Automatic Control* 38.5 (1993), pp. 700–716. DOI: 10.1109/9.277235.

[69] Richard M Murray, Muruhan Rathinam, and Willem Sluis. “Differential flatness of mechanical control systems: A catalog of prototype systems”. In: *ASME International Mechanical Engineering Congress and Exposition.* Citeseer. 1995.

[70] Siddharth H Nair, Eric H Tseng, and Francesco Borrelli. “Collision avoidance for dynamic obstacles with uncertain predictions using model predictive control”. In: *2022 IEEE 61st Conference on Decision and Control (CDC).* IEEE. 2022, pp. 5267–5272.

[71] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming.* SIAM, 1994.

[72] Cormac O'Meadhra, Wennie Tabib, and Nathan Michael. "Variable Resolution Occupancy Mapping Using Gaussian Mixture Models". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2015–2022. DOI: 10.1109/LRA.2018.2889348.

[73] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. "Continuous-time trajectory optimization for online UAV replanning". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 5332–5339. DOI: 10.1109/IROS.2016.7759784.

[74] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. DOI: 10.1109/IROS.2017.8202315.

[75] Panos M Pardalos and Stephen A Vavasis. "Quadratic programming with one negative eigenvalue is NP-hard". In: *Journal of Global optimization* 1.1 (1991), pp. 15–22. DOI: 10.1007/BF00120662.

[76] Jungwon Park, Dabin Kim, Gyeong Chan Kim, Dahyun Oh, and H. Jin Kim. "Online Distributed Trajectory Planning for Quadrotor Swarm With Feasibility Guarantee Using Linear Safe Corridor". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4869–4876. DOI: 10.1109/LRA.2022.3152702.

[77] Jungwon Park and H. Jin Kim. "Online Trajectory Planning for Multiple Quadrotors in Dynamic Environments Using Relative Safe Flight Corridor". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 659–666. DOI: 10.1109/LRA.2020.3047786.

[78] Jungwon Park, Junha Kim, Inkyu Jang, and H Jin Kim. "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 434–440. DOI: 10.1109/ICRA40945.2020.9197162.

[79] Jungwon Park, Junha Kim, Inkyu Jang, and H. Jin Kim. "Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee using Relative Bernstein Polynomial". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 434–440. DOI: 10.1109/ICRA40945.2020.9197162.

[80] Jungwon Park, Yunwoo Lee, Inkyu Jang, and H. Jin Kim. "DLSC: Distributed Multi-Agent Trajectory Planning in Maze-Like Dynamic Environments Using Linear Safe Corridor". In: *IEEE Transactions on Robotics* (2023), pp. 1–20. DOI: 10.1109/TRO.2023.3279903.

[81] Min Gyu Park and Min Cheol Lee. "A new technique to escape local minimum in artificial potential field based path planning". In: *KSME international journal* 17.12 (2003), pp. 1876–1885.

[82] Ryan Peterson, Ali Tevfik Buyukkocak, Derya Aksaray, and Yasin Yazıcıoğlu. "Distributed Safe Planning for Satisfying Minimal Temporal Relaxations of TWTL Specifications". In: *Robotics and Autonomous Systems* 142 (Aug. 1, 2021), p. 103801. ISSN: 0921-8890. DOI: 10.1016/j.robot.2021.103801.

[83] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996.

[84] Stephen Prajna and Ali Jadbabaie. “Safety verification of hybrid systems using barrier certificates”. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2004, pp. 477–492.

[85] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Vol. 6. Springer, 2002.

[86] James A Preiss, Wolfgang Honig, Gaurav S Sukhatme, and Nora Ayanian. “Crazyswarm: A large nano-quadcopter swarm”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3299–3304.

[87] Yao Qi, Binbing He, Rendong Wang, Le Wang, and Youchun Xu. “Hierarchical Motion Planning for Autonomous Vehicles in Unstructured Dynamic Environments”. In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 496–503. DOI: 10.1109/LRA.2022.3228159.

[88] Jorge Peña Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. “Collaborative Multi-Robot Search and Rescue: Planning, Coordination, Perception, and Active Vision”. In: *IEEE Access* 8 (2020), pp. 191617–191643. DOI: 10.1109/ACCESS.2020.3030190.

[89] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. “ROS: an open-source Robot Operating System”. In: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. 2009.

[90] Eike Rehder and Horst Kloeden. “Goal-Directed Pedestrian Prediction”. In: *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. 2015, pp. 139–147. DOI: 10.1109/ICCVW.2015.28.

[91] Eike Rehder, Florian Wirth, Martin Lauer, and Christoph Stiller. “Pedestrian prediction by planning using deep neural networks”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5903–5908.

[92] Jingyao Ren, Vikraman Sathiyanarayanan, Eric Ewing, Baskin Senbaslar, and Nora Ayanian. “MAPFAST: A deep algorithm selector for multi agent path finding using shortest path embeddings”. In: *arXiv preprint arXiv:2102.12461* (2021).

[93] Craig W Reynolds. “Flocks, herds and schools: A distributed behavioral model”. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[94] Charles Richter, Adam Bry, and Nicholas Roy. “Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments”. In: *International Symposium of Robotic Research (ISRR)*. Vol. 114. 2013, pp. 649–666. DOI: 10.1007/978-3-319-28872-7_37.

[95] Elon Rimon. *Exact robot navigation using artificial potential functions*. Yale University, 1990.

[96] Benjamin Riviere, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. “GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning”. In: *IEEE robotics and automation letters* 5.3 (2020), pp. 4249–4256.

[97] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, G. Wagner, T. K. Kumar, Sven Koenig, and H. Choset. "PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning". In: *IEEE Robotics and Automation Letters* 4 (2019), pp. 2378–2385. DOI: 10.1109/LRA.2019.2903261.

[98] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. *RLSS: Real-time Multi-Robot Trajectory Replanning using Linear Spatial Separations*. 2021. arXiv: 2103.07588 [cs.RO]. URL: https://arxiv.org/abs/2103.07588.

[99] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. "RLSS: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations". In: *Autonomous Robots* (2023), pp. 1–26. DOI: 10.1007/s10514-023-10104-w.

[100] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. "Robust Trajectory Execution for Multi-robot Teams Using Distributed Real-time Replanning". In: *Distributed Autonomous Robotic Systems (DARS)*. 2019, pp. 167–181. DOI: 10.1007/978-3-030-05816-6_12.

[101] Baskın Şenbaşlar, Pilar Luiz, Wolfgang Hönig, and Gaurav S. Sukhatme. "MRNAV: Multi-Robot Aware Planning and Control Stack for Collision and Deadlock-free Navigation in Cluttered Environments". In: *arXiv preprint arXiv:2308.13499* (2023).

[102] Baskın Şenbaşlar and Gaurav S Sukhatme. "Asynchronous Real-time Decentralized Multi-Robot Trajectory Planning". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 9972–9979. DOI: 10.1109/IROS47612.2022.9981760.

[103] Baskın Şenbaşlar and Gaurav S Sukhatme. "DREAM: Decentralized Real-time Asynchronous Probabilistic Trajectory Planning for Collision-free Multi-Robot Navigation in Cluttered Environments". In: *arXiv preprint arXiv:2307.15887* (2023).

[104] James A Sethian. "Fast marching methods". In: *SIAM review* 41.2 (1999), pp. 199–235.

[105] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66. DOI: 10.1016/j.artint.2014.11.006.

[106] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. "The increasing cost tree search for optimal multi-agent pathfinding". In: *Artificial intelligence* 195 (2013), pp. 470–495.

[107] Kiril Solovey, Oren Salzman, and Dan Halperin. "Finding a Needle in an Exponential Haystack: Discrete RRT for Exploration of Implicit Roadmaps in Multi-Robot Motion Planning". In: *The International Journal of Robotics Research* 107 (2013). DOI: 10.1007/978-3-319-16595-0_34.

[108] A. Stentz. "Optimal and efficient path planning for partially-known environments". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 3310–3317 vol.4. DOI: 10.1109/ROBOT.1994.351061.

[109] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 10. 2019, pp. 151–158.

[110] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. “Efficient SAT approach to multi-agent path finding under the sum of costs objective”. In: *Proceedings of the twenty-second european conference on artificial intelligence*. 2016, pp. 810–818.

[111] Sarah Tang and Vijay Kumar. “Safe and complete trajectory generation for robot teams with higher-order dynamics”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1894–1901. DOI: 10.1109/IROS.2016.7759300.

[112] Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro. “Learning decentralized controllers for robot swarms with graph neural networks”. In: *Conference on robot learning*. PMLR. 2020, pp. 671–682.

[113] Jesus Tordesillas and Jonathan P How. “MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves”. In: *arXiv preprint arXiv:2010.10726* (2020). DOI: 10.48550/arXiv.2010.10726.

[114] Jesus Tordesillas and Jonathan P. How. “MADER: Trajectory Planner in Multiagent and Dynamic Environments”. In: *IEEE Transactions on Robotics* 38.1 (2022), pp. 463–476. DOI: 10.1109/TRO.2021.3080235.

[115] Jesus Tordesillas, Brett T. Lopez, Michael Everett, and Jonathan P. How. “FASTER: Fast and Safe Trajectory Planner for Navigation in Unknown Environments”. In: *IEEE Transactions on Robotics* 38.2 (2022), pp. 922–938. DOI: 10.1109/TRO.2021.3100142.

[116] Charbel Toumieh. “Decentralized Multi-Agent Planning for Multirotors: a Fully online and Communication Latency Robust Approach”. In: *arXiv preprint arXiv:2304.09462* (2023).

[117] Charbel Toumieh and Alain Lambert. “Decentralized Multi-Agent Planning Using Model Predictive Control and Time-Aware Safe Corridors”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11110–11117. DOI: 10.1109/LRA.2022.3196777.

[118] Vladyslav Usenko, Lukas Von Stumberg, Andrej Pangercic, and Daniel Cremers. “Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 215–222. DOI: 10.1109/IROS.2017.8202160.

[119] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. “Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6932–6939. DOI: 10.1109/LRA.2020.3026638.

[120] Jiangxing Wang, Jiaoyang Li, Hang Ma, Sven Koenig, and S Kumar. “A new constraint satisfaction perspective on multi-agent path finding: Preliminary results”. In: *18th International Conference on Autonomous Agents and Multi-Agent Systems*. 2019.

[121] Li Wang, Aaron D Ames, and Magnus Egerstedt. “Safety barrier certificates for collisions-free multirobot systems”. In: *IEEE Transactions on Robotics* 33.3 (2017), pp. 661–674. DOI: 10.1109/TRO.2017.2659727.

[122] Li Wang, Aaron D. Ames, and Magnus Egerstedt. “Safe certificate-based maneuvers for teams of quadrotors using differential flatness”. In: *IEEE International Conference on Robotics and Automation*. 2017, pp. 3293–3298. DOI: 10.1109/ICRA.2017.7989375.

[123] Xinyi Wang, Lele Xi, Yizhou Chen, Shupeng Lai, Feng Lin, and Ben M. Chen. “Decentralized MPC-Based Trajectory Generation for Multiple Quadrotors in Cluttered Environments”. In: *Guidance, Navigation and Control* 01.02 (June 1, 2021), p. 2150007. ISSN: 2737-4807. DOI: 10.1142/S2737480721500072.

[124] C.W. Warren. “Fast path planning using modified A* method”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 662–667 vol.2. DOI: 10.1109/ROBOT.1993.291883.

[125] Jürgen Wiest, Matthias Höffken, Ulrich Kreßel, and Klaus Dietmayer. “Probabilistic trajectory prediction with Gaussian mixture models”. In: *2012 IEEE Intelligent Vehicles Symposium*. 2012, pp. 141–146. DOI: 10.1109/IVS.2012.6232277.

[126] Guofan Wu and Koushil Sreenath. “Safety-critical control of a 3d quadrotor with range-limited sensing”. In: *Dynamic Systems and Control Conference*. Vol. 50695. American Society of Mechanical Engineers. 2016, V001T05A006.

[127] Peter Wurman, Raffaello D’Andrea, and Mick Mountz. “Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses”. In: *AI Magazine* 29 (2008), pp. 9–20.

[128] Yinyu Ye and Edison Tse. “An extension of Karmarkar’s projective algorithm for convex quadratic programming”. In: *Mathematical Programming* 44.1-3 (1989), pp. 157–179. DOI: 10.1007/BF01587086.

[129] Yan Yongjie and Zhang Yan. “Collision avoidance planning in multi-robot based on improved artificial potential field and rules”. In: *IEEE International Conference on Robotics and Biomimetics*. 2009, pp. 1026–1031. DOI: 10.1109/ROBIO.2009.4913141.

[130] Jingjin Yu and Steven M LaValle. “Planning optimal paths for multiple robots on graphs”. In: *IEEE International Conference on Robotics and Automation*. 2013, pp. 3612–3617.

[131] Jingjin Yu and Steven M. LaValle. “Multi-agent Path Planning and Network Flow”. In: *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 157–173. DOI: 10.1007/978-3-642-36279-8_10.

[132] Jingjin Yu and Steven M. LaValle. “Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics”. In: *IEEE Transactions on Robotics* 32.5 (2016), pp. 1163–1177. DOI: 10.1109/TRO.2016.2593448.

[133] Jingjin Yu and Steven M. LaValle. “Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs”. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI’13. Bellevue, Washington, 2013, pp. 1443–1449. DOI: 10.1609/aaai.v27i1.8541.

[134] Min Zhang, Yi Shen, Qiang Wang, and Yibo Wang. “Dynamic artificial potential field based multi-robot formation control”. In: *2010 IEEE Instrumentation & Measurement Technology Conference Proceedings*. 2010, pp. 1530–1534. DOI: 10.1109/IMTC.2010.5488238.

[135] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, and Mac Schwager. "Fast, On-line Collision Avoidance for Dynamic Vehicles Using Buffered Voronoi Cells". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1047–1054. DOI: 10.1109/LRA.2017.2656241.

[136] Xin Zhou, Jiangchao Zhu, Hongyu Zhou, Chao Xu, and Fei Gao. "Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments". In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, pp. 4101–4107.

[137] Yichao Zhou and Jianyang Zeng. "Massively parallel A* search on a GPU". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015. DOI: 10.1609/aaai.v29i1.9367.

[138] Zhou Zhou, Gang Huang, Zhaoxin Su, Yongfu Li, and Wei Hua. "Dynamic Attention-Based CVAE-GAN for Pedestrian Trajectory Prediction". In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 704–711. DOI: 10.1109/LRA.2022.3231531.

[139] Hai Zhu and Javier Alonso-Mora. "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 776–783. DOI: 10.1109/LRA.2019.2893494.

[140] Qidan Zhu, Yongjie Yan, and Zhuoyi Xing. "Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing". In: *Sixth International Conference on Intelligent Systems Design and Applications*. Vol. 2. 2006, pp. 622–627. DOI: 10.1109/ISDA.2006.253908.

# Appendices

## A Sweep of a Convex Shape Along a Line Segment is Convex

**Lemma 5.** *Let $\mathcal{R}(\mathbf{x}) = \{\mathbf{x}\} \oplus \mathcal{R}_0$ be the Minkowski sum of $\{\mathbf{x}\} \in \mathbb{R}^d$ and $\mathcal{R}_0 \subseteq \mathbb{R}^d$, and let $\mathcal{R}_0$ be a convex set. Let $\mathbf{p}(t) = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$ where $t \in [0, 1]$ be the line segment from $\mathbf{a} \in \mathbb{R}^d$ to $\mathbf{b} \in \mathbb{R}^d$. Then, the swept volume of $\mathcal{R}$ along $\mathbf{p}(t)$ is convex.*

*Proof.* The swept volume $\zeta$ of $\mathcal{R}$ from $\mathbf{a}$ to $\mathbf{b}$ can be defined as $\zeta = \cup_{t=0}^{1} \mathcal{R}(\mathbf{p}(t)) = \cup_{t=0}^{1} \{\mathbf{p}(t)\} \oplus \mathcal{R}_0$. Choose two points $\mathbf{q}_1, \mathbf{q}_2 \in \zeta$. $\exists t_1 \in [0, 1] \exists \tilde{\mathbf{q}}_1 \in \mathcal{R}_0 \; \mathbf{q}_1 = \mathbf{p}(t_1) + \tilde{\mathbf{q}_1}$ and $\exists t_2 \in [0, 1] \exists \tilde{\mathbf{q}}_2 \in \mathcal{R}_0 \; \mathbf{q}_2 = \mathbf{p}(t_2) + \tilde{\mathbf{q}_2}$. Let $\mathbf{q}' = (1 - t')\mathbf{q}_1 + t'\mathbf{q}_2$ be a point on the line segment connecting $\mathbf{q}_1$ and $\mathbf{q}_2$ where $t' \in [0, 1]$.

$$
\begin{aligned}
\mathbf{q}' &= (1 - t')(\mathbf{p}(t_1) + \tilde{\mathbf{q}}_1) + t'(\mathbf{p}(t_2) + \tilde{\mathbf{q}}_2) \\
&= (1 - t')(\mathbf{a} + t_1(\mathbf{b} - \mathbf{a}) + \tilde{\mathbf{q}}_1) \\
&\quad + t'(\mathbf{a} + t_2(\mathbf{b} - \mathbf{a}) + \tilde{\mathbf{q}}_2) \\
&= (1 - t')\tilde{\mathbf{q}}_1 + t'\tilde{\mathbf{q}}_2 \\
&\quad + \mathbf{a} + (t_1(1 - t') + t_2 t')(\mathbf{b} - \mathbf{a})
\end{aligned}
$$

$(t_1(1 - t') + t_2 t') \in [0, 1]$ because it is a convex combination of $t_1, t_2 \in [0, 1]$. Therefore $\mathbf{a} + (t_1(1 - t') + t_2 t')(\mathbf{b} - \mathbf{a}) \in \mathbf{p}(t)$. Also, $(1 - t')\tilde{\mathbf{q}}_1 + t'\tilde{\mathbf{q}}_2 \in \mathcal{R}_0$, because it is a convex combination of $\tilde{\mathbf{q}}_1, \tilde{\mathbf{q}}_2 \in \mathcal{R}_0$ and $\mathcal{R}_0$ is convex. Therefore, $\mathbf{q}' \in \zeta$ since $\exists t \in [0, 1] \; \mathbf{q}' \in \{\mathbf{p}(t)\} \oplus \mathcal{R}_0$. Hence, $\zeta$ is convex. □

## B Ordered Constraint Generation for Low Number of Constraints in DREAM

We introduce an ordered constraint generation scheme that allows us to decrease the number of collision avoidance constraints against static and dynamic obstacles from thousands to less than ten for each piece during trajectory optimization in obstacle-rich environments in DREAM (Chapter 6), while also still preserving interpolated collision probabilities $P_s$ and $P_d$.

Let $\mathcal{H}_l$ be the set of hyperplanes constraining $l^{th}$ piece $\mathbf{f}_{i,l}$. We initialize it to $\tilde{\mathcal{H}}_i^{active} \setminus x^{l+1}.\mathcal{H}$ if $x^l.t < T_i^{team}$ and $\emptyset$ otherwise.

Let $A_l$ be the set of all regions (composed of regions occupied by static obstacles and regions swept by dynamic obstacles) that $l^{th}$ segment from $x^l.\mathbf{p}$ to $x^{l+1}.\mathbf{p}$ avoids, i.e., $\zeta_{i,l}^{robot} \cap A = \emptyset \; \forall A \in A_l$. We add constraints to avoid elements of $A_l$ during trajectory optimization.

Instead of computing separating hyperplanes between $\zeta_{i,l}^{robot}$ and each element of $A_l$ and using them as constraints, we compute separating hyperplanes in an ordered fashion from closest element to the farthest element in $A_l$ to $\zeta_{i,l}^{robot}$ and adding them to $\mathcal{H}_l$, skipping the element if it is already avoided by the previous hyperplanes $\mathcal{H}_l$. This allows us to avoid the elements of $A_l$ with significantly lesser number of constraints as shown in Figure 8.1.

## C Implementation Details of the Multi-robot Aware Planning and Control Stack (MRNAV)

The ROS nodes, topics, and their connections are shown in Figure 8.2. The nodes and topics in the green boxes are repeated for each robot, and executed within individual robot ROS namespaces.

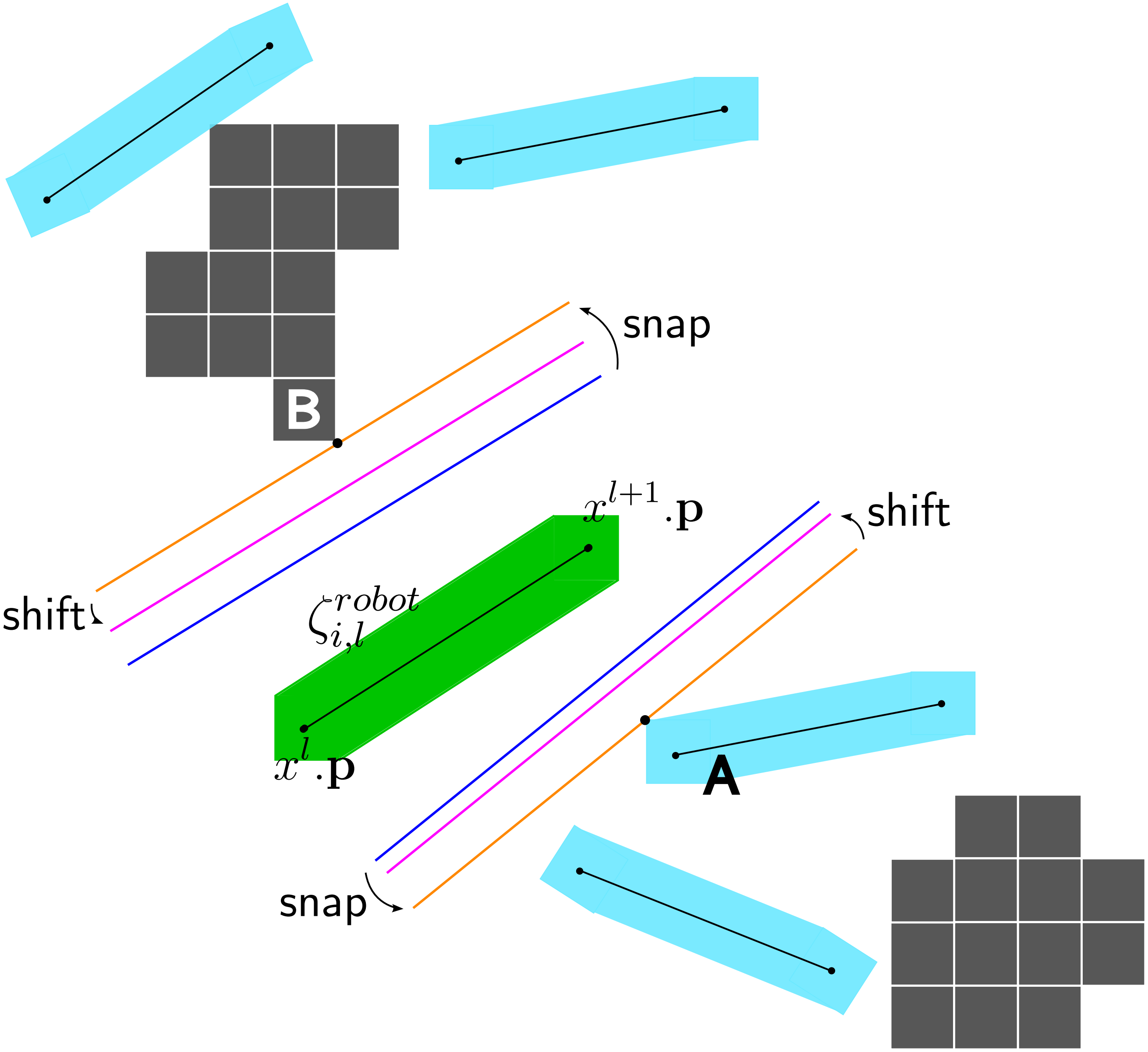


Figure 8.1: In order to decrease the number of constraints for static and dynamic obstacle avoidance, we utilize an ordered constraint generation method. First, we sort regions to avoid according to their distance to $\zeta_{i,l}^{robot}$. Then, for each element in to sorted list, we check whether it is avoided by the constraints computed so far, and compute the hyperplane constraint as described in Section 6.3.4.1 if not. In the figure, we first compute the separating hyperplane against the region A swept by a dynamic obstacle, which already allows avoidance of all regions below A. Then, we compute the separating hyperplane against static obstacle B. The hyperplane avoids all regions above B. Therefore, two computed constraints are enough to avoid all obstacles, compared to 30 that would be required without the ordered constraint generation method.

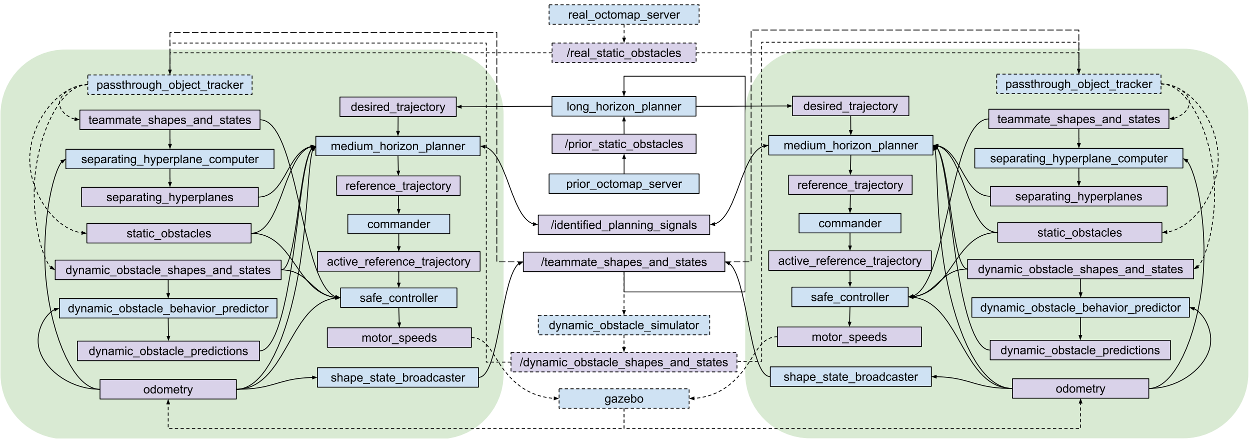

real_octomap_server
/real_static_obstacles
passthrough_object_tracker
teammate_shapes_and_states
separating_hyperplane_computer
separating_hyperplanes
static_obstacles
dynamic_obstacle_shapes_and_states
dynamic_obstacle_behavior_predictor
dynamic_obstacle_predictions
odometry
desired_trajectory
medium_horizon_planner
reference_trajectory
commander
active_reference_trajectory
safe_controller
motor_speeds
shape_state_broadcaster
long_horizon_planner
/prior_static_obstacles
prior_octomap_server
/identified_planning_signals
/teammate_shapes_and_states
dynamic_obstacle_simulator
/dynamic_obstacle_shapes_and_states
gazebo
desired_trajectory
medium_horizon_planner
reference_trajectory
commander
active_reference_trajectory
safe_controller
motor_speeds
shape_state_broadcaster
passthrough_object_tracker
teammate_shapes_and_states
separating_hyperplane_computer
separating_hyperplanes
static_obstacles
dynamic_obstacle_shapes_and_states
dynamic_obstacle_behavior_predictor
dynamic_obstacle_predictions
odometry

Figure 8.2: **Implemented System.** The ROS nodes (blue), topics (purple), and their connections in our implementation of the stack for quadrotor flight are shown. The nodes and topics in the green boxes are repeated for each robot. The connections that would exist in a full system, i.e., the system with perception, mapping, and localization subsystems, are shown with solid lines. The connections that could optionally exist but are not necessary in the full system are shown with dashed lines. The connections, nodes, and topics that would not exist in the full system and are introduced to mock sensing, estimation, or simulation related procedures are shown with dotted lines.

## C.1 Mocking State Estimation and Teammate Sensing

We use position and velocities to describe the states of the robots, which is required by the on-board decision making modules. To mock on-board state estimation, we use the ground truth position and velocity of the robot from Gazebo (gazebo node), and provide it to the on-board modules via the odometry topic. Each quadrotor's shape is modeled as an axis aligned box, which is provided to all on-board nodes as a ROS parameter. Each robot broadcasts its state and shape whenever it receives a new odometry message via shape_state_broadcaster node to the global /teammate_shapes_and_states topic. Each robot has a passthrough_object_tracker node that mocks object perception, classification, and mapping by relaying global information to the local nodes. For teammate sensing, it connects to the global /teammate_shapes_and_states topic, and relays shapes and states of all teammates (except itself) to teammate_shapes_and_states topic.

## C.2 Simulating Dynamic Obstacles, Mocking Dynamic Obstacle Sensing, and Predicting Dynamic Obstacle Behaviors

Each dynamic obstacle's behavior is simulated using a behavior model, and its shape is an axis aligned box. The dynamic_obstacle_simulator node simulates the movement of dynamic obstacles in the environment, obtaining position and velocities of the teammates from the /teammate_shapes_and_states topic, which it uses to execute the interaction models. It outputs the shapes and states of dynamic obstacles, which involves their positions and velocities, at high frequency to the global /dynamic_obstacle_shapes_and_states topic. The passthrough_object_tracker receives messages from /dynamic_obstacle_shapes_and_states, and relays them to dynamic_obstacle_shapes_and_states, simulating perfect dynamic obstacle shape sensing and state estimation.

The on-board systems do not have explicit access to the behavior models of the dynamic obstacles, but run on-board probabilistic behavior model estimation in dynamic_obstacle_behavior_predictor, by running the behavior prediction algorithms described in Section 6.4.1.2. The output of the behavior prediction is a set of behavior models $\mathcal{B}_{i,j}$ for each dynamic obstacle $i$ and probabilities $p_{i,j}$ assigned to them, reflecting that dynamic obstacle $i$ moves according to behavior model $\mathcal{B}_{i,j}$ with probability $p_{i,j}$. Necessarily, $\sum_j p_{i,j} \leq 1$ holds for all $i$. In terms of the interaction model, on-board behavior predictor only predicts dynamic obstacle's interaction with the ego robot, and does not consider multi-robot interactions. Hence, it is necessarily inaccurate when describing dynamic obstacle behaviors stemming from interactions with multiple robots. The inputs of the behavior predictor are states of the ego robot and states of dynamic obstacles, which it receives from odometry and dynamic_obstacle_shapes_and_states topics, respectively. The behavior model predictions are published to dynamic_obstacle_predictions topic.

## C.3 Mocking Static Obstacle Sensing

We use octrees from the octomap [41] library to represent static obstacles, which allow for fast neighborhood occupancy queries. In order to model potential imperfect prior static obstacle information, there are two octomap servers: real_octomap_server and prior_octomap_server. real_octomap_server provides the real static obstacles in the environment to the on-board modules, through the global /real_static_obstacles topic. The on-board passthrough_object_tracker node relays these obstacles to the static_obstacles topic. We do not mock local sensing static obstacles within passthrough_object_tracker, but we mock it by using static obstacles up to certain distance during on-board decision making. prior_octomap_server provides static obstacles known a priori to the navigation, which may be different than the real static obstacles, to /prior_static_obstacles topic, which is used by the long_horizon_planner node only. During experiments, we do not relay local static obstacles obtained from /real_static_obstacles to the long_horizon_planner in order to show the performance of the system when a realiable connection to the centralized system cannot be maintained.

## C.4 Long Horizon Decision Making

The long horizon decision making module, i.e., the long_horizon_planner node, randomly generates goal positions for the robots in the environment, computes shortest paths from robots' current positions to the goal positions avoiding only the static obstacles using A* search and assigns durations to each segment of the computed path. The computed paths are used as desired trajectories and provided to desired_trajectory topics of robots. During decision making, it does not enforce dynamic feasibility, and enforces collision avoidance against only the prior static obstacles, which it obtains from /prior_static_obstacles topic. We choose not to enforce dynamic feasibility during long horizon decision making because i) long horizon decision making module does not enforce collision avoidance against dynamic obstacles or teammates, which requires necessary divergence from desired trajectories during execution, and ii) quadrotors are agile systems, which allows them to navigate back to their desired trajectories easily when possible. When a robot reaches to its goal position, the long horizon decision making module randomly generates a new goal position, and computes a new desired trajectory for the robot. To know the existence of and track the progress of robots, it subscribes to /teammate_shapes_and_states topic.

## C.5 Medium Horizon Decision Making

We use DREAM (Chapter 6) as the medium horizon decision making module, implemented within the medium_horizon_planner node. To recap, DREAM is decentralized real-time trajectory planning algorithm for multi-robot teams, computing reference trajectories that avoids static obstacles, interactive or non-interactive dynamic obstacles, as well as teammates.

For static obstacle avoidance, it requires shapes of the obstacles and their existence probabilities. Since we do not implement probabilistic sensing, we provide static obstacles obtained from static_obstacles with existence probabilities $1.0$ to the medium horizon planner.

For dynamic obstacle avoidance, i) it requires a probability distribution across possible behavior models, which are provided via dynamic_obstacle_predictions topic by dynamic_obstacle_behavior_predictor node, and ii) current shapes and positions of the dynamic obstacles, which is provided via the dynamic_obstacle_shapes_and_states topic by the passthrough_object_tracker node.

For teammate avoidance, it relies on discretized separating hyperplane trajectories (DSHTs) (Chapter 5), which is a constraint generation method that allows the active trajectories of each pair of robots to share a separating hyperplane constraint at all times under asynchronous planning and communication imperfections. DSHTs require computing separating hyperplanes between the ego robot and other robots at a high frequency and pruning separating hyperplane trajectories whenever the ego robot knows another robot has planned successfully. The separating_hyperplane_computer node computes hard-margin support vector machine [19] hyperplanes between the ego robot and each other robot in the environment at a high frequency, and outputs the computed hyperplanes to separating_hyperplanes topic. It obtains ego robot's state from the odometry topic, and the states of teammates from teammate_shapes_and_states. The medium_horizon_planner constructs DSHTs for each other robot in the environment using the computed separating hyperplanes. Whenever medium_horizon_planner node plans successfully, it broadcasts that it planned successfully on a planning iteration that started in a specific time via the global /identified_planning_signals topic. Whenever medium_horizon_planner nodes in other robots receive the message, they prune their DSHTs against the sender robot as explained in Chapters 5 and 6.

DREAM employs widely a used [99, 114, 50, 58, 94, 17, 33] goal selection, discrete search, and trajectory optimization pipeline. The trajectories generated by the long_horizon_planner are fed to the medium_horizon_planner as desired trajectories over the desired_trajectory topic. In each planning iteration, goal selection step chooses a goal position on the desired trajectory to plan to. In the discrete search stage, DREAM computes a spatiotemporal path to the selected goal position by minimizing collision probabilities against static obstacles, dynamic obstacles, as well as DSHT violations. In the trajectory optimization stage, quadratic programming based spline optimization is executing, smoothing the computed discrete spatiotemporal path while preserving collision probabilities computed and DSHT hyperplanes not violated during search. The dynamic feasibility of the trajectories are accounted for by enforcing desired continuity and maximum derivative magnitudes of the trajectories, which are sufficient to ensure dynamic feasibility of the trajectories for differentially flat systems [69], which quadrotors are. The generated trajectories are published to the reference_trajectory topic. If a planning iteration fails, the robot continues using the latest reference trajectory that is successfully computed.

### C.5.1 Environment Projection

One important consideration for medium horizon decision making is its relative low replanning frequency. Since it typically runs in 1 Hz to 10 Hz, it takes 0.1 s to 1 s until the robot produces a plan, during which the robot continues moving. Hence, when planning starts at robot's and environment's current states, by the time planning ends, the ego robot might have moved substantially and the environment might have already changed.

This requires projecting the environment to the future, and starting planning from a future predicted state of the environment. The projection duration should at least be the re-planning period of the medium horizon planner. We project the environment to the future by i) projecting the initial state of the ego robot that is used by the planner, and ii) projecting the current positions of dynamic obstacles for each of their behavior models. We do not project the teammate states to the future, because mutually computable separating hyperplane based constraints enforce teammate safety without requiring projection (Chapters 5 and 6).

To project the state of the ego robot, we employ open-loop and close-loop projection. If the robot's current position is within a predefined distance of the expected position on the previously planned reference trajectory at the current time point, we use open-loop projection, in which the robot's initial state is set to the state on the previously planned reference trajectory that is projection duration away from the current time point. If the robot is not within the predefined distance of the expected position on the previously planned trajectory, i.e., it diverged from its reference trajectory, we project its position to the future by assuming it moves with its current velocity for projection duration, which closes the loop between state estimation and planning. We prefer to have open-loop projection because close-loop projection fails to accurately describe the future state of the robot as the robot does not move with constant velocity in reality, which causes non-smooth movement while switching reference trajectories.

The tuple of behavior model current position pairs $(\mathcal{B}_{i,j}, \mathbf{p}_i^{dyn})$ of dynamic obstacle $i$ with probabilities $p_{i,j}$ describe the behavior modes of the dynamic obstacle. We project each behavior mode by running the movement and the interaction models, and setting the initial position of the mode to $\mathbf{p}_{i,j}^{proj} = \mathbf{p}_i^{dyn} + \mathcal{I}_{i,j}(\mathbf{p}_i^{dyn}, \mathcal{M}(\mathbf{p}_i^{dyn}), \mathbf{p}, \mathbf{v})t_{proj}$ where $\mathbf{p}$ and $\mathbf{v}$ are the current position and velocity of the ego robot respectively, and $t_{proj}$ is the projection duration, and provide $(\mathcal{B}_{i,j}, \mathbf{p}_{i,j}^{proj})$ with probabilities $p_{i,j}$ as behavior modes to the planning algorithm.

### C.5.2 Commanding

While the medium horizon planner is configured to run at a fixed frequency, it may take longer or shorter than the projection duration to complete a planning iteration as there is generally no completion time guarantee for a planning iteration. If a planning iteration takes longer than the projection duration, we discard the generated trajectory as the robot might have moved considerably away from the start state of the newly generated trajectory and the environment may have changed substiantially. In that case, the robot continues following its previous reference trajectory. If a planning iteration takes shorter than the projection duration, the newly generated trajectory should not be activated until the projection duration is elapsed. The commander node manages the activation logic. Each generated trajectory is associated with an activation time point and published to reference_trajectory topic. The commander subscribes to the reference_trajectory, and creates a timer for each of the trajectories, and activates them in order. The activated trajectories are published to active_reference_trajectory topic for safe_controller to track.

## C.6 Controller and Short Horizon Decision Making

RotorS simulator accepts angular motor speed inputs for each of the four propellers of the quadrotor. The forces and moments applied by the propellers are simulated based on the model of a single propeller near hovering [65], which in turn are based on classical blade element theory [45] as mentioned in Martin and Salaün [65]'s work. The forces and moments applied by the propellers are quadratically related to the motor speeds. The simulator also models rotor drag while computing forces. Gravity is applied externally to the quadrotors by Gazebo.

Our design (Figure 7.2) necessitates a short horizon decision making module that accepts desired control inputs and outputs safe control inputs in a minimally invasive manner, considering static and dynamic obstacles as well as teammates. We propose integrating a quadrotor controller [54] that internally computes desired acceleration commands with a short horizon decision making module for double-integrator systems [121] allowing static and dynamic obstacle and teammate avoidance. By doing so, we diverge from the exact design, because we do not make final motor speeds safe, but ensure that the intermediate desired accelerations computed by the controller are safe. The augmented controller is implemented in the safe_controller node. It receives active reference trajectories from active_reference_trajectory, and provides samples of it to the safe control algorithm as desired states.

The controller computes a desired body acceleration using a PD controller to track reference trajectory samples, which involve desired position and velocity. We set the desired yaw of the quadrotors to zero. Then, controller computes the desired angular acceleration of the quadrotor in each of the three frame axes in order to produce the given desired acceleration and yaw. Then, it computes the desired total body thrust and torques to achieve desired acceleration and angular accelerations. Next, it computes the forces that need to be generated by the propellers in order to produce the desired total body thrust and torques. The conversion of the desired forces to motor speed commands are done assuming a quadratic relation, similarly to the simulator.

The short horizon decision making module [121] accepts desired accelerations, and makes them safe in a minimally invasive manner. We plug this decision making module within the controller after desired body acceleration computation. The static obstacles are modeled using their positions and radii, which are obtained and computed from static_obstacles topic. The dynamic obstacles are modeled using their positions, current velocities and radii, which are obtained and computed from dynamic_obstacle_shapes_and_states topic. The decision making module assumes that the dynamic obstacles move with constant velocities. This is not the case in general, but is a good approximation for the immediate future of the environment.

Since the safe controller runs at a high frequency, e.g., $50\,\mathrm{Hz} - 100\,\mathrm{Hz}$, we find that this approximation is effective. The teammates are modeled using their positions, velocities, and radii, which are obtained and computed from teammate_shapes_and_states topic. In addition to these, SBC requires i) maximum dynamically feasible acceleration for the ego robot, and ii) a lower bound on maximum accelerations of the teammates. We use the maximum acceleration for the ego robot as the lower bound of maximum accelerations of the teammates, since the team is homogeneous. For heterogeneous robot teams, acceleration estimation of the teammates can be added, from which the lower bounds on maximum accelerations can be tightened.

The computed motor commands are published to motor_speeds topic, which the gazebo node subscribes to and simulates the quadrotor flight.