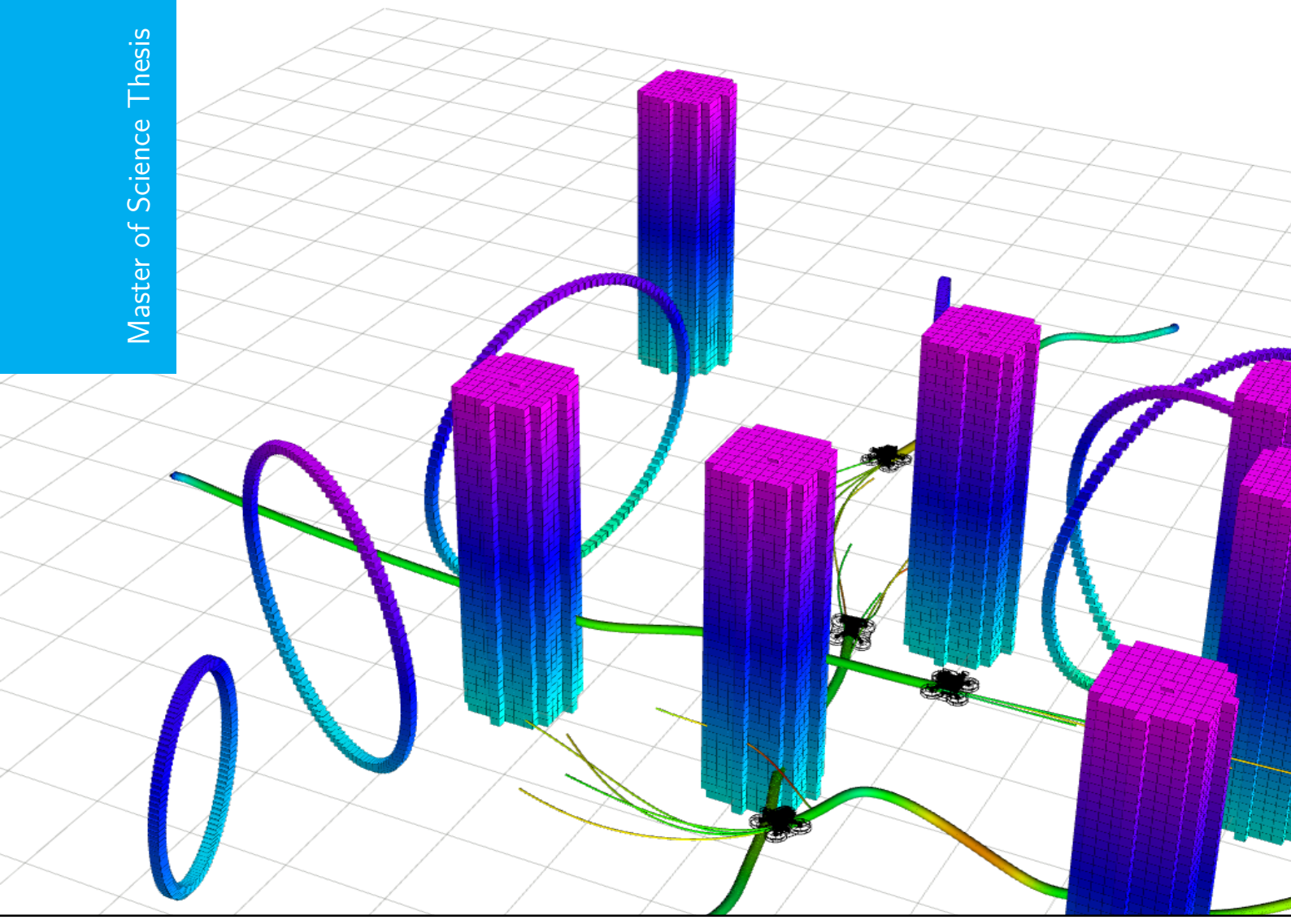
Risk-Aware Decentralized Multi-MAV Planning in Unknown and Dynamic Environments

Siyuan Wu

Master of Science Thesis



TU Delft Delft University of Technology

Cognitive Robotics

Risk-Aware Decentralized Multi-MAV Planning in Unknown and Dynamic Environments

Master of Science Thesis

For the degree of Master of Science in Robotics at Delft University of Technology

Siyuan Wu

August 23, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Cognitive
Robotics

Delft University of Technology
Department of
Cognitive Robotics (CoR)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Risk-Aware Decentralized Multi-MAV Planning in Unknown and Dynamic Environments

by

Siyuan Wu

in partial fulfillment of the requirements for the degree of

Master of Science.

Dated: August 23, 2023

Supervisor:

Prof. Dr. Javier Alonso-Mora

Readers:

Dr. Clarence Chen

Dr. Chris Pek

Dr. Matin Jafarian

Abstract

Recent progress in multiple micro aerial vehicle (MAV) systems has demonstrated autonomous navigation in static environments. Yet, there are limited works regarding the autonomous navigation of multiple MAVs in dynamic and unknown environments. The challenge arises from the complexity of the motion planning problem, which requires the MAVs to coordinate with each other while avoiding dynamic obstacles. This thesis presents a novel risk-aware decentralized multi-agent motion planning framework to address this issue. For perception, we rely on a particle-based dynamic map, which utilizes particles to represent dynamic obstacles and predict future states of dynamic obstacles. Leveraging this map representation and the shared trajectory from other agents, we evaluate the future collision risk with dynamic obstacles and other agents in a coupled manner. During the planning phase, a risk-aware kino-dynamic A* algorithm tailored to the particle-based map representation is developed, ensuring dynamically feasible paths with risks under a given safe level. Subsequently, spatio-temporal safety corridors with maximum volume are optimized by inflating from path segments, taking map particles as constraints. These corridors act as constraints for the trajectory optimization problem, which is simplified to a convex optimization problem by using Bézier splines. The proposed method is thoroughly evaluated in simulation environments featuring various quantities and shapes of dynamic obstacles. Comparative results with state-of-the-art multi-agent planners that rely on precise obstacle observations, demonstrate the efficiency and safety of the proposed method. Furthermore, the effectiveness of the proposed method is validated in a more realistic simulation environment with pedestrians, using depth cameras onboard for perception.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Javier Alonso-Mora for his continuous support, patience and immense knowledge of my master's study and research. I could not have imagined having a better advisor and mentor for my master's study. I would like to thank my daily supervisor Dr. Gang Chen for his guidance and support during my master's thesis. His guidance helped me in all the time of research and writing of this thesis. I would also like to thank my colleagues at the Autonomous Multi-Robots Lab for the stimulating discussions, and for all the work we have done in the last two years. Finally, I would like to thank my parents and all my friends for their support and encouragement throughout my life.

Siyuan Wu

Delft, University of Technology,
August 23, 2023

Chapter 1

Introduction

1-1 Background

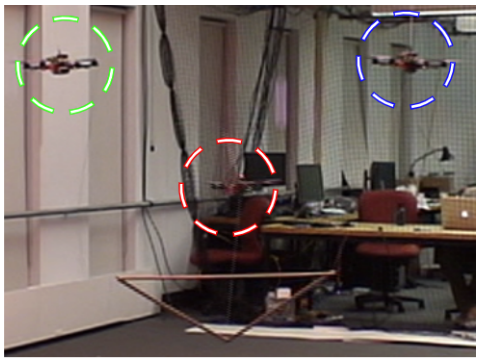
In recent years, the robotics community has increasingly focused on multi-MAV systems, which are groups of micro aerial vehicles (MAV) that can perform complex tasks in a coordinated manner. Compared to a single MAV, a multi-MAV system offers enhanced scalability and usability while maintaining the same mobility and agility. Consequently, the multi-MAV system opens up various challenging applications that a single MAV would find hard to accomplish. These include tasks like conducting search and rescue operations over expansive areas, transporting bulky or sizable loads, capturing aerial shots from multiple viewpoints, and creating three-dimensional reconstructions. For instance, in scenarios like search and exploration, employing multiple MAVs permits the coverage of larger areas in comparison to a lone MAV, resulting in a significant reduction in search time. Additionally, individual MAVs face restrictions in payload capacity, hindering their effectiveness in transportation roles. Yet, by distributing the load across multiple MAVs, the overall transportation payload can be substantially augmented. The above applications are illustrated in Figure 1-1.

To realize these applications in real-world scenarios, a longstanding challenge has been ensuring the safe navigation of multi-MAV systems safely in unknown environments. These environments usually encompass static obstacles, such as trees and buildings, and also dynamic obstacles, such as pedestrians, other robots or moving objects. No prior information regarding the obstacles is available, including factors such as the quantity, shape, location, and velocity of these obstacles.

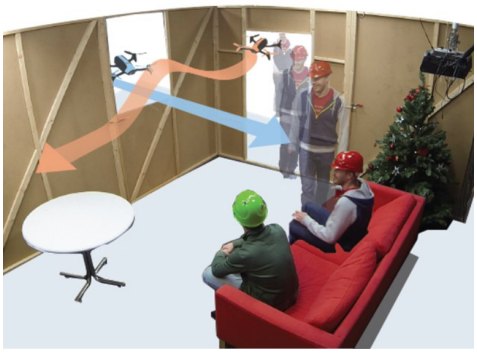
1-2 Motivation

Autonomous navigation has drawn significant attention for decades. Robots utilize onboard sensors autonomously to perceive the environment in real time, and plan tra-

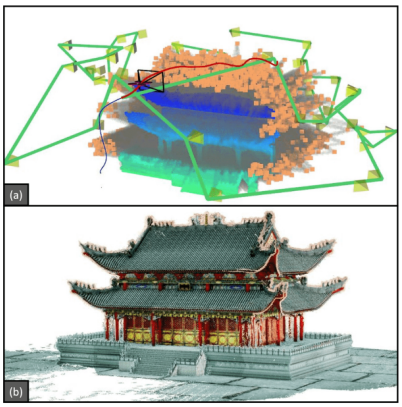
(a) Search and exploration [1]



(b) Transportation [2]



(c) Aerial cinematography [3]



(d) 3D reconstruction [4]

Figure 1-1: Applications of multi-MAV system.

jectories to reach thier goals while avoiding collisions with obstacles and other robots. Recently, with progress in perception and planning technologies, substantial strides have been made concerning autonomous navigation in static environments. For example, a study by [6] demonstrated the capability of small autonomous aerial swarms to operate agilely within a dense forest cluttered with bamboo and branches, as illustrated in Figure 1-3a. Additionally, in another study by [7], a swarm of drones successfully explored a static environment, cooperatively building a 3D map of the unknown terrain, as shown in Figure 1-3b. In these studies, robots coordinate in a distributed and asynchronous manner by sharing their trajectories. Safe trajectories are initially generated on local maps, which are then refined by incorporating the trajectories received from other robots. Although these approaches are demonstrated to be effective in static environments, they are not sufficient in unknown and dynamic environments.

In unknown and dynamic environments, there are dynamic obstacles with unpredictable quantities, shapes and motions, which challenges the robustness and efficiency of the planning algorithms. The planning algorithm should guarantee the safety of the robots in the presence of unknown and dynamic obstacles. Since the robot does not have prior knowledge of the environment, the future motions of the obstacles should be predicted and potential collisions should be handled. However, predicting the motions of dynamic obstacles requires solving a data association problem, which matches the raw sensor

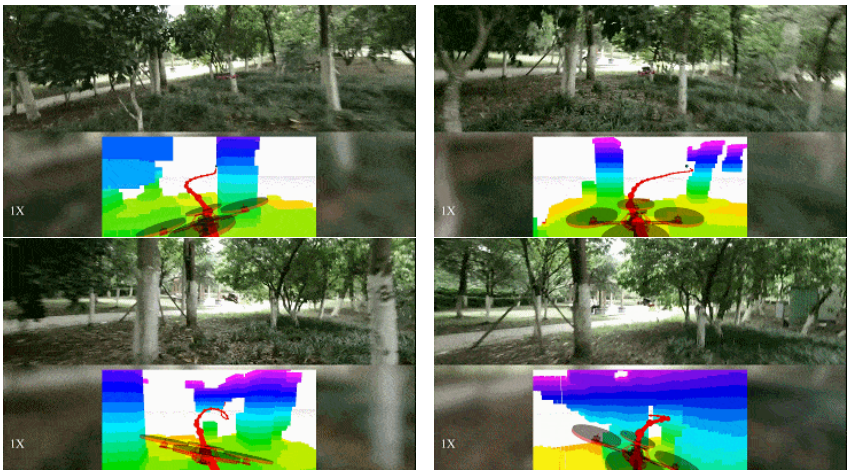


Figure 1-2: Autonomous navigation in a dense forest [5]. The time interval is 0.5 seconds.

(a) Multiple MAVs navigating in a dense forest. **(b)** Multiple MAVs exploring a room.

Figure 1-3: Recent progress in aerial swarm navigation.

measurements with previous observations to update the predicted future trajectories. Due to inevitable noise in unknown environments, the predicted future trajectories are often inaccurate. This issue introduces uncertainties in the perception process, which challenges the safety of the planning algorithm.

Existing planning frameworks in dynamic environments fail to address the uncertainties, primarily due to their map representation. Most of them [6, 8, 9] are designed on top of a static occupancy map which cannot model the dynamic obstacles and their future motions. Particle-based dynamic map representation in [10] is a promising perception approach to address this issue. It represents obstacles as a set of particles distributed in the workspace. Each particle carries a weight indicating the probability of the existence of an obstacle at the corresponding location. The distribution of particles is updated by observations from the onboard sensors. By applying a motion model to the particles, using observed position and velocity, the future states of dynamic obstacles can be predicted. This map representation has many advantages: First, it can model arbitrary shapes of static and dynamic obstacles without making any assumptions a priori. Second, the future collision risk in the environment can be predicted directly. Furthermore, observation and localization noises can be handled naturally in this representation. Based on these properties, the particle-based dynamic map representation addresses the aforementioned challenges in multi-robot planning

for unknown and dynamic environments. This research gap motivates us to develop a motion planner based on the particle-based dynamic map representation, specifically for multi-MAV systems.

1-3 Contribution

Based on the motivation above, this thesis aims to develop a risk-aware decentralized multi-MAV system that can safely navigate in unknown dynamic environments using only onboard sensors and wireless communications.

To achieve this goal, the particle-based dynamic map representation is leveraged to model the environment. By utilizing this map representation, multi-MAV planning can be simplified into a unified framework. Collision avoidance among collaborative robots can be achieved by mapping robots to the particle-based dynamic map with trajectory sharing. Then, a risk-aware kino-dynamic path planning algorithm is designed to search for safe initial trajectories for each robot. The resulting path contains time allocation, which can be used to generate safety corridors in the particle-based dynamic map. The generation of safety corridors is formulated as an optimization problem to maximize the free space where the robot can safely navigate. These safety corridors can be used as constraints to ensure the safety of refined trajectories. Finally, the flight trajectory is optimized by refining the initial trajectories, which is composed of Bézier splines. The deconfliction between robots can be achieved by checking the linear separability of the Bézier control points. Our approach can accommodate multiple robots and dynamic obstacles in the environment and is robust against perception uncertainties.

The contribution of this thesis can be summarized as follows:

- A novel risk-aware decentralized motion planning framework in uncertain and dynamic environments, which leverages a particle-based map to model the environment.
- A risk-aware kinodynamic path search method designed for the particle-base map that enables safe planning in dynamic environments.
- An efficient corridor-constrained trajectory optimization algorithm that can efficiently generate safe trajectories for multiple robots.

1-4 Structure of the Report

The report will be organized as follows: Chapter 2 will review the related works in decentralized motion planning and planning in dynamic environments. Chapter 3 will introduce the preliminaries that are necessary to understand the proposed method. Chapter 4 will present the proposed method in detail. Chapter 5 will present the experimental results of the proposed method in simulation. Finally, Chapter 7 will conclude the thesis and discuss future work.

Chapter 2

Related Works

2-1 Decentralized Multi-Robot Motion Planning

Decentralized multi-robot motion planning has been an active research topic for years. Major challenges, as outlined in [10], include: 1) the difficulty of searching for optimal or suboptimal paths in a high-dimensional space, and 2) the need for multi-robot planning to take other robots into account, necessitating the inclusion of nonconvex constraints. Current multi-robot motion planning algorithms can be classified into two main categories: centralized and decentralized methods. Centralized methods, such as coupled methods [11, 12], prioritized planning [13, 14, 15], plan trajectories for all MAVs on a centralized server. In contrast, for decentralized methods, robots compute their ego trajectories locally and communicate with other robots. Generally, decentralized methods are more scalable and robust than centralized methods.

Decentralized methods can be divided into two categories[16, 6]: reactive approaches and optimization-based approaches. Reactive approaches reflect the reactive autonomy of drones, whereas optimization-based approaches reflect the cognitive autonomy of autonomous drones. Reactive approaches utilize the instantaneous state of the system, specifically the positions and velocities of the agents, to generate safe control inputs. These approaches plan at a high frequency, typically between 50-100 Hz. Typical reactive approaches include Artificial Potential Field [17], Flocking Method [18], Velocity Obstacles [19, 20, 21, 22], Control Barrier Functions [23], and Buffered Voronoi Cells [24, 25]. Optimization-based approaches [26, 27, 28, 29] utilize the future trajectories of agents to plan safely. They plan at a lower frequency, usually between 1-20 Hz. These approaches can apply more complex constraints and objective functions, as well as deal with delays and asynchronous communication. However, both approaches failed to tackle the problem of dynamic obstacles, neglecting the noise in perception and localization modules. Therefore, novel methods are required to address these problems.

2-2 Autonomous Navigation in Dynamic Environments

In the context of navigation in complex environments, there exist unknown dynamic obstacles. Safe navigation in such environments requires predictions of the environment states in the future. The predictions tell the future states of the dynamic obstacles, which are essential for the robot to avoid collisions.

In the past decades, there have been a variety of methods proposed for prediction problems in such environments, which can be categorized into two groups. One category distinguishes dynamic obstacles from the static background and predicts the future states of these detections There are several methods for detecting moving obstacles from depth images. [30] detects obstacles from U-depth maps, which are computed with the column depth value histograms of the original depth image. The detections are then represented by three-dimensional boxes with estimated dimensions. The positions of the detections are predicted by a Kalman filter. [31] clusters the point clouds by DBSCAN and then proposes an occlusion-aware Kalman filter to track the detections. [9] proposes a vision-aided 3D mapping method in dynamic environments. It improves [30] by applying a continuity filter to remove objects that exhibit jerky motions, using the history of object velocities. [32] and [33] adopt the YOLO detector to detect dynamic obstacles. [34] combines the U-depth detector and the DBSCAN detector to obtain more accurate results with high efficiency. The primary drawback of these methods is their applicability to well-defined obstacles of simple shapes. It may lead to potential failure when faced with unknown obstacles with complex shapes.

Numerous studies have investigated motion planning in dynamic environments. These methods can be divided into two categories, depending on whether they consider static and dynamic obstacles separately or uniformly. The first category detects dynamic obstacles from static environments and then plans a collision-free trajectory that takes into account both static and dynamic obstacles. Recent work [6, 35, 8, 9, 36] employs a static occupancy map [37] to model static obstacles, taking advantage of its ability to represent arbitrary 3D shapes [38]. However, dynamic obstacles are represented separately in a fixed shape [9]; each obstacle is approximated by a single ellipsoid [30, 39, 8], sphere [28], or bounding box [40]. To estimate the trajectory of moving obstacles, polynomial fitting methods [41, 42] or the constant velocity model [43] are commonly used. Since dynamic obstacles are represented separately with fixed shapes, these methods narrow the free space and require two collision avoidance pipelines, making the planner conservative and complex. Furthermore, data association and tracking noise [10, 44] in this representation will lead to unsafe trajectories.

The other category considers the dynamic obstacles and possible static obstacles uniformly, and predicts the future states of all the obstacles. For example, [45] clusters the point clouds and then estimates the velocities of dynamic obstacles. Then a dynamic Hilbert map is created using nonstationary kernels in the Hilbert space based on these estimations. [10] proposes a particle-based method to represent the environments. Recently, learning-based methods such as spatio-temporal network [46] and generative models [47] are proposed to forecast the position of dynamic obstacles. The perception parts of our work will be based on the method in [10].

Chapter 3

Preliminaries

This chapter provides the essential theoretical background required to understand the proposed motion planning algorithm. First, I will introduce the map representation upon which our planner is built. Next, I will provide an overview of the existing motion planning pipeline for autonomous navigation in cluttered environments in Section 3-2.

3-1 Particle-based Dynamic Environment Representation

Our system is built upon a particle-based map that provides a risk-aware dynamic environment representation. The particle-based map utilizes particles to represent states of the dynamic obstacles. Future maps are predicted by applying the constant velocity model to the particles, and the collision risk can be estimated by calculating the expectation of the particles within the robot's safety distance. The dual-structured particle-based (DSP) map [10] is an example of such a representation. It is built upon the random finite set (RFS) theory, which can be used to represent a random number of elements whose states are random variables. Obstacles are represented by a RFS of point objects, which are finite but randomly distributed in the environment. The DSP map estimates the probability hypothesis density (PHD) of the RFS from the point cloud input, rather than individually estimating the state of each point object. The estimation is performed by a sequential Monte Carlo PHD (SMC-PHD) filter, which iteratively predicts and updates a multitude of particles. Then, a risk map in dynamic environments can be accurately constructed by querying the number of point objects in each voxel, even when there exist non-negligible noises in both sensing and localization modules of the robot [48]. The velocities of these particles are modeled by a constant velocity model. The future risk map of the dynamic environments is predicted by propagating the particles with the constant velocity model. We use this property in our system to predict the future risk value of the environment when checking collisions, and design a corresponding risk-aware planning algorithm to avoid collisions with dynamic obstacles. We will discuss the details later in Chapter 4-2-2.

3-2 Motion Planning for Autonomous MAVs

To find a safe trajectory in cluttered environments, a coarse-to-fine pipeline is commonly used [49, 50, 51]. The pipeline consists of global path planning and local trajectory optimization. The global path plans a spatial discrete path from the start to the goal, which provides a rough initial guess of the trajectory optimization problem. The trajectory then optimizes the path by incorporating the robot dynamics while satisfying the environmental constraints. The path searching is necessary because the trajectory optimization can be trapped in local minima easily therefore the robot will never reach the goal. Trajectory optimization aims to refine the path to make it dynamically feasible and energy efficient. However, a global path is hard to obtain in our scenarios because the robot faces an unknown dynamic environment, where only a local region is observed and long-term changes are not predictable. This problem can be solved by applying local replanning [50].

3-2-1 Path Planning

Sampling-based Methods Sampling-based methods are widely used in MAV motion planning. These methods are based on rapidly exploring random tree (RRT) [38] and RRT* [52] considering the dynamic constraints of the MAVs. A common approach has two steps, first generating a low-dimensional path in position space and then optimizing the trajectory to generate a smooth and dynamically feasible trajectory. [53] demonstrated that this approach is more efficient than kinodynamic RRT* [54] which applies a polynomial steering function to the quadrotor system to address linear differential constraints. However, recent research [55] improved the efficiency of kinodynamic RRT * by applying topological guided search to speed up the sampling. [56] further improved the efficiency by applying a bidirectional search strategy to reduce the complexity and reduced the search time to 16ms.

Search-based Methods Search-based methods are based on the A * algorithm, which is a heuristic search algorithm to find the shortest path between two nodes in a graph. Nodes are the voxels and edges are connections between the voxels in the occupancy map, therefore, paths generated by A* are discrete and not smooth. To make the searched path smooth, a state lattice with motion primitives can be applied to form the edges in the graph [57, 58], and A* can be applied to search the path in the state lattice [50]. [59] discussed state-based primitives and control-based primitives to demonstrate the advantages of state-based primitives since FOV constraints can be simply applied. Kinodynamic A* [51] further improved this method by deriving the heuristic cost function based on Pontryagins minimum principle that can be used to find the trajectory with minimum time duration.

Gradient-based Methods Gradient-based methods utilize obstacle information to generate a gradient that guides trajectory optimization to avoid obstacles. CHOMP [60]

is a discrete gradient-based method that applies both smoothness and collision cost which performs gradient descent with positions of discrete waypoints as parameters. Traditionally, gradient-based methods rely on a pre-build Euclidean signed distance field (ESDF)[61] to evaluate the direction and magnitude of the gradient [62, 63, 64]. However, computing the ESDF is time-consuming and requires a lot of memory. EGO-Planner [5] presented an ESDF-free local planning framework that projects forces onto the colliding trajectory and generates an estimated gradient to wrap the trajectory free from obstacles. It successfully reduces computation time by over an order of magnitude and achieves state-of-the-art performance in terms of computation time and robustness.

3-2-2 Safety Corridors

A safety corridor is defined as an obstacle-free region in the workspace. In the literature, safety corridors are usually formed as ellipsoids [65, 66] or polyhedra [67, 68, 49, 69, 70], since these shapes are convex and easy to confine in most optimization problems. Safety corridors provide geometric constraints for the trajectory optimization problem to ensure the safety of the robot. Since the structured information of the environments is not available in most planning tasks, generating safety corridors directly from the point cloud is widely applied in many literature. However, this is a time-consuming task due to the large number of points in the point cloud. There exist two types of methods, geometric methods and inflation methods.

Inflation Methods Inflation methods normally start with a small safety region around the robot and iteratively expand the safety region until it reaches obstacles. A common approach is IRIS [67], which represents the safety region as a polyhedron and iteratively expands the polyhedron and its inscribed ellipsoid. However, this method is computationally expensive because it requires solving a quadratic programming (QP) problem to find the polyhedron that is closest to the inscribed ellipsoid, as well as solve a semidefinite programming (SDP) problem to find the inscribed ellipsoid of the polyhedron. In [49], the authors simplified the problem by shirking and dilating the initial ellipsoid to find a feasible ellipsoid. Then the polyhedron is generated by finding the intersection of the ellipsoid and the hyperplanes that exclude the obstacles. It avoids solving an expensive SDP problem but loses the optimality of the ellipsoid. FIRI [71] improves the efficiency by providing a second-order conic programming (SOCP) formulation of the problem, which can be solved by an L-BFGS solver efficiently. Experimental results show that this approach reduces the computation time to 0.1ms, which is two orders of magnitude faster than IRIS.

Geometric Methods Geometric methods generate safety corridors by excavating the geometric properties of the environments. Geometric methods are usually faster than inflation methods, taking the consequence of losing the optimality of the safety corridor. For example, Galaxy [72] generates a safety corridor utilizing the sphere flipping method to compute an obstacle-free star convex polytope inside the point cloud. Since the star convex polytope is not convex, it needs to be modified to a convex polytope by shrinking

the concave edges of the convex hull of the star convex polytope. Due to the property of star convex, each extreme edge can form a simplex with the vertex of the star convex polytope. Therefore, by pushing the extreme edges inward, the concave edges of the convex hull can be shrunk.

3-2-3 Trajectory Optimization

Given the safety corridors $\mathcal{P}$ of the environment, we can formulate the trajectory optimization problem as a constrained optimization problem as follows:

$$\min_{\boldsymbol{c}} \sum_{j=1}^{T} \int_{t_{j-1}}^{t_j} \left\| \frac{\mathrm{d}^3 \boldsymbol{p}_j(t)}{\mathrm{d}t^4} \right\|^2 dt \tag{3-1a}$$

$$\text{s.t.} \quad \boldsymbol{p}(t_0) = \boldsymbol{p}_0, \quad \boldsymbol{p}(t_T) = \boldsymbol{p}_T \tag{3-1b}$$

$$\boldsymbol{p}_j(t) \in \mathcal{P}_j, \quad \forall t \in [t_0, t_T] \tag{3-1c}$$

$$\boldsymbol{p}_j(t_j) = \boldsymbol{p}_{j+1}(t_j) \tag{3-1d}$$

$$\boldsymbol{p}_j^{(1)}(t) \leq \boldsymbol{p}_{max}^{(1)}, \ \boldsymbol{p}_j^{(2)}(t) \leq \boldsymbol{p}_{max}^{(2)}, \ \forall t \in [t_0, t_T], \tag{3-1e}$$

The cost function (Eq. 3-1a) is the integral of the third derivative of the trajectory, which minimizes the jerk of the trajectory [73]. where $\boldsymbol{c}$ is the parameter of trajectory $\boldsymbol{p}(t)$, equation (3-1b) provides the initial value, (3-1d) is continuity constraints of two adjoint pieces of trajectory, (3-1e) limits the maximum derivatives of the trajectory. The corridor constraints (Eq. 3-1c) bounds the trajectory for all valid time t continuously. Eq. 3-1d provides the continuity constraints of two pieces of trajectory. Eq. 3-1e bounds the maximum velocity and acceleration of the trajectory.

Since corridor constraints (Eq. 3-1c) enforce the entire trajectory of all time steps to be inside the safety corridor, this optimization problem cannot be converted to a standard QP form which can be accepted by most solvers. Therefore, various methods are proposed to simplify the problem. The idea is to find a proper trajectory representation that can reformulate the corridor constraints to a standard affine form. Different polynomial curves have different properties and can be used to simplify the optimization problem. As for trajectory parameterization, most literature uses polynomial splines, Bézier splines, and B-splines [51, 74, 75]. MINVO [40, 76, 77] and MINCO [71, 6, 78] are two novel methods, one provides minimum guarantees on the volume of the convex hull of the trajectory, and the other minimizes the control efforts of the trajectory. In the following section, typical trajectory representations are introduced and their properties are discussed.

Polynomial Trajectory

A polynomial trajectory is the most common trajectory representation in the literature:

$$p(t) = p_i(t - t_{i-1}) = \mathbf{c}_i^{\mathsf{T}} \beta(t - t_{i-1}), , \quad t \in [t_{i-1}, t_i) \tag{3-2}$$

in which $p_i(t)$ is an N degree polynomial trajectory with coefficient $\mathbf{c}_i \in \mathbb{R}^{(N+1)\times n}$ and natural basis $\beta(t) = [1, t, \cdots, t^N]^\intercal$. The parameter needed in this polynomial curve is a coefficient matrix $\mathbf{c} \in \mathbb{R}^{(N+1)M\times n}$of the whole trajectory is defined by

$$\mathbf{c} = [\mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \cdots, \mathbf{c}_M^\intercal]^\intercal \tag{3-3}$$

and a time allocation represented by time vector $\mathbf{T} \in \mathbb{R}^M_{\geq 0} = [T_1, T_2, \cdots, T_M]$. Given the time allocation $\mathbf{T}$ and boundary states of all the trajectory pieces, the coefficient matrix $\mathbf{c}$ can be uniquely determined. Enforcing the corridor constraints (Eq. 3-1c) explicitly is difficult, but there are various techniques to address them. [73] samples N time steps and confine positions at this sampled timestamp in the corridor. [68] proposed an iterative approach that first solves a QP problem without constraints and then adds time stamps not satisfying constraints to solve the optimization. [79] splits the safety corridor into two intersecting polyhedra if the constraints are violated on the given piece.

Bézier Trajectory

A trajectory can be formulated as a piece-wise Bézier spline which is a linear combination of Bernstein basis. A N-degree Bernstein basis polynomial is defined as:

$$\mathcal{B}_N^k(t) = \begin{pmatrix} N \\ k \end{pmatrix} t^k (1-t)^{N-k}, \quad t \in [0,1],\ k = 0, 1, \cdots, N \tag{3-4}$$

where $\begin{pmatrix} N \\ k \end{pmatrix}$ is the binomial coefficient. The i-th piece of trajectory $p_i(t)$ can be represented as

$$p_i(t) = \sum_{k=0}^{N} c_{i,k} \mathcal{B}_N^k(\tau_i) \quad t \in [t_{i-1}, t_i) \tag{3-5}$$

where $\tau_i = \frac{t - t_{i-1}}{t_i - t_{i-1}} \in [0,1]$ is the normalized time and $c_{i,k}$ is the k-th control point of the i-th piece trajectory. The Bézier spline has several properties [62]:

1. **Convex hull property**: The Bézier spline is entirely confined within the convex hull defined by control points.
2. **Fixed interval property**: The parameter t of Bézier spline is defined with $t \in [0,1]$.
3. **Hodograph property**: The hodograph of the Bézier spline, which is its derivative spline, is still a Bézier spline.
4. **Start and end point property**: The Bézier spline starts from the first control point $c_{i,1}$ and ends at the final control point $c_{i,N}$.

These properties are significantly helpful for trajectory optimization. Due to the convex hull property, the safety constraints can be simplified by confining the control points in the corridors. Recent methods [62, 80] are developed based on this property.

Discussion

Bézier splines, B-splines, and MINVO trajectories are fully equivalent and linearly convertible [81]. The difference lies in their parameterization form and functions. Here, we summarize these trajectory parameterization methods in Table 3-1. As indicated in the table, polynomial splines do not have control points, MINCO trajectory is based on control points on the trajectory. Bézier spline passes through the initial and final control point, B-spline can guarantee this by setting the first three control points as the same, however, MINVO cannot guarantee this property. For the volume of the convex hull, MINVO can achieve the smallest simplex convex hull enclosing the trajectory, where the control points are close to the trajectory. These properties are vital in trajectory optimization because we can only check the control points to ensure the safety of the planned trajectory. Therefore, the trajectory optimization problem can be simplified to a control point optimization problem, which can be formulated as quadratic programming (QP) problem and solved efficiently.

Method/Feature	Continuity	Safety Constraints	Dim.	Temporal Opt.	Ctrl. Points Passed
Polynomial Splines	Equality Constraints	Sample	High	Coupled	None
Bézier Splines	Equality Constraints	Convex Hull	High	Coupled	Start & End
B-Splines	Satisfied by default	Convex Hull	Low	Highly Coupled	Start & End
MINVO	Not Satisfied	Convex Hull	Low	Highly Coupled	No
MINCO	Satisfied by default	Sample	Low	Decoupled	All

Table 3-1: Comparison of different trajectory parameterization methods regarding how to achieve continuity, how to satisfy safety constraints, dimensionality, how to optimize the trajectory in the time domain, and whether the control points are passed by the trajectory.

Chapter 4

Methods

In this chapter, the proposed method for risk-aware multi-agent planning in dynamic environments is presented. This method is built on top of the particle-based map [10] which is a novel map representation that can efficiently take into account the sensing and localization uncertainties in the dynamic environment. Thanks to the particle-based map representation, the proposed method can tackle planning problems in unknown dynamic environments with arbitrary-shaped obstacles. In this map representation, the collision risk can be quantified by the number of particles in the region, and the future state can be predicted by applying the motion model to the particles. Additionally, this map representation is also able to incorporate the risk information of other agents by modeling them as particles in the map. Prediction of other agents' particles can be achieved by using shared trajectories. Thus, both obstacles and other agents are represented in a unified manner, enhancing the method's capability to address the multi-robot planning problem. The proposed method can be divided into three stages: particle-based trajectory mapping, risk-aware kinodynamic path planning, and corridor-constrained trajectory optimization.

In this chapter, first, the general pipeline of the proposed framework is introduced. Then, the particle-based map is presented and the collision risk definition is discussed. Finally, the details of the three stages of the proposed method are presented.

4-1 Pipeline Overview

As indicated in Figure 4-1, the classical pipeline of multi-robot collision avoidance in dynamic environments consists of building a map of the static environment, tracking and predicting dynamic obstacles, and coordinating multiple robots to avoid collisions with each other and with dynamic obstacles [40, 82, 30, 83]. Path planning, usually performed on a static map, plans a global path from the start point to the goal point while avoiding static obstacles. Collision avoidance and trajectory coordination are

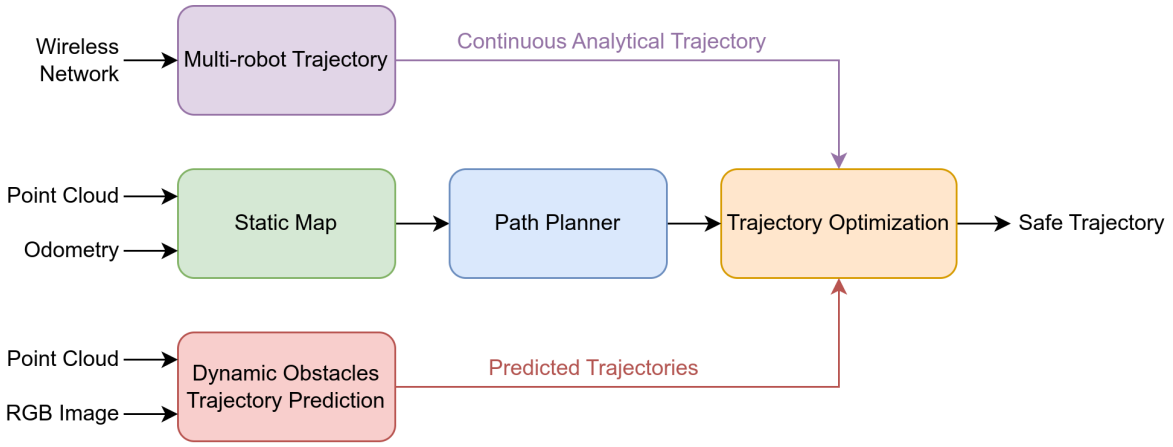


Figure 4-1: Classical pipeline of multi-robot collision avoidance in dynamic environments.

achieved locally at the trajectory optimization stage, which is performed in a receding horizon manner, optimizing the trajectory over a short time horizon. However, this pipeline has several drawbacks:

1. Due to the uncertainty in tracking and predicting dynamic obstacles, the static map may be disrupted by these obstacles, limiting the planner's feasibility.

2. The uncertainty arising from dynamic obstacle detection and trajectory prediction compromises the system's robustness, and incorporating this noise into the pipeline is challenging.

3. Given the non-convexity from the mutual collision avoidance constraints, the trajectory optimization problem turns out to be a non-convex optimization problem. It is not only computationally expensive, but also depends sensitively on the initial guess of the trajectory.

To address the above issues in the classical pipeline, the particle-based map representation can be utilized to unify the static map, dynamic obstacles, and cooperative agents into a single map. The proposed pipeline with the particle-based map representation is shown in Figure 4-2. To represent static and dynamic obstacles, we use the map representation introduced in Section 4-2. It models obstacles as particles with velocity and risk information. Since cooperative robots communicate their future trajectories, they can be mapped as particles with known future trajectories on the map. Finally, after representing all the obstacles and cooperative agents on a single map, motion planning and collision avoidance can be performed directly. Since the trajectory of other agents is considered in the map, it's not necessary to consider the trajectory in the optimization stage. In this manner, the path planner first plans a kinodynamic path in the particle map. The kinodynamic path embeds the kinematic limits of the robot as well as the time allocation of the trajectory, which can be regarded as a sparse initial guess for trajectory optimization. Then spatial-temporal safety corridors are expanded around the path to ensure the safety of the trajectory during the optimization.

Finally, trajectory optimization is performed in the particle map to generate a smooth trajectory that satisfies the kinodynamic constraints.

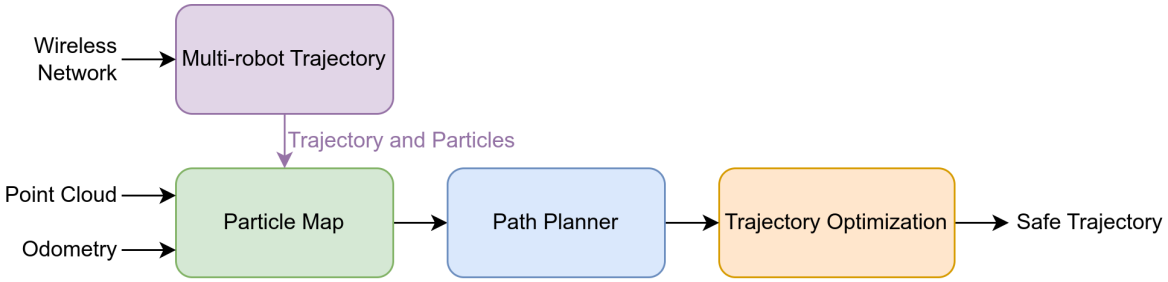


Figure 4-2: Proposed pipeline of multi-robot collision avoidance in dynamic environments.

The proposed pipeline offers several advantages: 1) Thanks to continuous risk-aware map representation, the measurement uncertainty and the localization uncertainty by the sensors are naturally incorporated into the map. 2) The kinodynamic path searching provides a sparse initial guess for the trajectory optimization, which overcomes the non-convexity of the trajectory optimization problem. 3) The expansion and shrinking of the spatial-temporal corridor ensure the safety of the trajectory without imposing more conservative constraints. 4) The trajectory optimization is formulated as a quadratic programming problem, which is computationally efficient even on a low-cost embedded system with limited computational resources.

4-2 Particle-based Map and Risk Evaluation

In this section, we will demonstrate how to evaluate the risk within dynamic environments. To do so, we will introduce the DSP map, which is a particle-map representation upon which our planner is built.

4-2-1 Particle-based Map

The DSP map [10] is an egocentric local map that uses particles to represent the obstacles in the environment. In this map, the obstacles are first modeled by many independent point objects, as shown by green dots in Figure 4-3. The state of any point object k at time t can be represented by

$$\mathbf{x}_k(t) = [\mathbf{p}_k(t), \mathbf{v}_k(t)]^\top, \quad \mathbf{p}_k(t) \in \mathbb{R}^3, \mathbf{v}_k(t) \in \mathbb{R}^3, \tag{4-1}$$

by doing this, the map is able to represent arbitrary-shaped obstacles and the future environment is predictable by applying the motion model to the particles. The point objects in the map space at time t can be formulated as a random finite set (RFS) as $\mathbf{X}(t) = \{\mathbf{x}_1(t), \mathbf{x}_2(t), \mathbf{x}_3(t), \cdots, \mathbf{x}_K(t)\}$. K denotes the number of point objects in the set $X(t)$. Since estimating the states of all point objects is computationally expensive, the DSP map only computes the probability hypothesis density (PHD) $\mathbf{D}_\mathbf{X}(t)$ of the

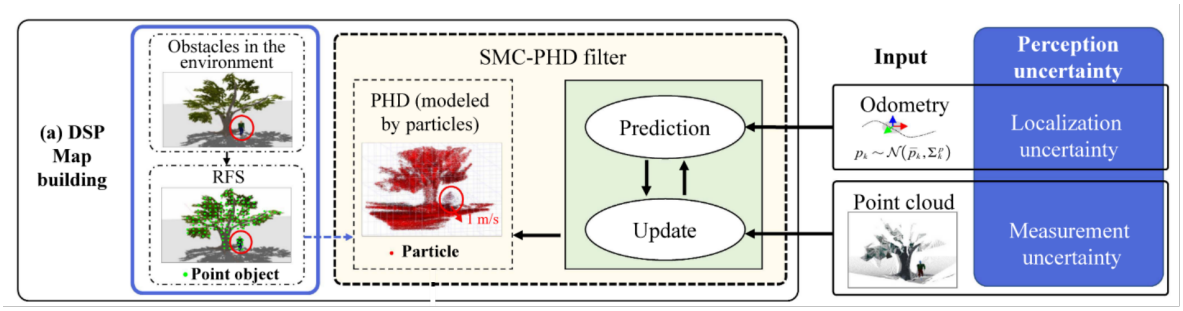


Figure 4-3: The pipeline of the DSP map

random finite set $\mathbf{X}(t)$. The PHD $\mathbf{D}_{\mathbf{X}}(t)$ is widely used in multi-object tracking, which is estimated by a sequential Monte Carlo (SMC-PHD) filter. In our case, the SMC-PHD filter requires the input of the point cloud measurement and 6DoF odometry, and outputs $n(t)$ particles at time t, where $n(t)$ is a large number normally greater than 10^6. Each particle contains a scalar weight $w_i(t)$ state $\tilde{\mathbf{x}}_i(t) = [\tilde{\mathbf{p}}_i(t), \tilde{\mathbf{v}}_i(t)]$. We can derive the estimation of the PHD $\mathbf{D}_{\mathbf{X}}(t)$ according to [10] using Dirac delta function $\delta(\cdot)$ to represent the particles,

$$\mathbf{D}_{\mathbf{X}}(t) = \sum_{i=1}^{n(t)} w_i(t)\delta(\tilde{\mathbf{x}}_i(t) - \mathbf{x}(t)). \tag{4-2}$$

4-2-2 Map Prediction

Thanks to the particle representation, the future environment is predictable. For simplicity, we assume particles are following a constant velocity model (CVM) with Gaussian noise. The state of particle i at time t can be predicted by

$$\tilde{\mathbf{x}}_i(t) = \begin{bmatrix} \mathbf{I}_{3\times3} & (t-t_0)\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tilde{\mathbf{x}}_i(t_0) + \eta_i(t), \tag{4-3}$$

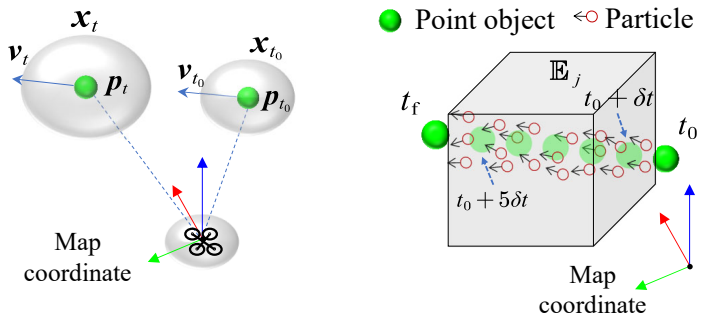
where $\eta_i(t) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ is the Gaussian noise with covariance matrix $\mathbf{\Sigma}$. We assume the robot only suffers from the localization error from noisy odometry, therefore we have

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_p & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}, \tag{4-4}$$

where $\mathbf{\Sigma}_p$ denotes the covariance matrix of the odometry. Since the number of particles is large, we can neglect the velocity noise of each single particle and focus on the behavior of numerous particles.

4-2-3 Risk Evaluation

Next, we will discuss the risk definition in this particle-based environmental representation. Risk is defined as the probability of collision between the robot and the obstacles.



(a) State transition (b) Risk calculation

Figure 4-4: State transition and risk evaluation in the DSP map

Since we model the obstacles as point objects, the collision probability depends on the number of point objects near the robot. In the DSP map, the number of point objects within a given region $\mathbb{E}_j$ can be calculated by the expectation of the cardinality of the random finite set $\mathbf{X}(t)$, which is

$$E\left[|\mathrm{X}_t^{\mathbb{E}_j}|\right] = \int D_{\mathrm{X}_t^{\mathbb{E}_j}(t)}(\mathbf{x}_t)\mathrm{d}\mathbf{x}_t = \sum_{i=1}^{n_t^{\mathbb{E}_j}} w_t^{(i)}. \tag{4-5}$$

where $w_t^{(i)}$ is the weight of the particle $\mathbf{x}_i$ at time t and $n_t^{\mathbb{E}_j}$ is the number of particles within the region $\mathbb{E}_j$ at time t. Therefore, the continuous collision risk of the region $\mathbb{E}_j$ from t_0 to t_f can be represented as

$$\mathrm{Risk}(\mathbb{E}_j, t_0, t_f) = \int_{t_0}^{t_f} E\left[|\mathbf{X}_t^{\mathbb{E}_j}|\right] \mathrm{d}t. \tag{4-6}$$

In practice, a discrete version of the collision risk is used for computation efficiency, which is

$$\mathrm{Risk}(\mathbb{E}_j, t_0, t_f) = \int_{t_0}^{t_f} E\left[|\mathbf{X}_t^{\mathbb{E}_j}|\right] \mathrm{d}t \approx \sum_{t=\{t_0, t_0+\delta t, \ldots, t_f\}} \sum_{i=1}^{n_t^{\mathbb{E}_j}} w_t^{(i)} \delta t. \tag{4-7}$$

Hence, the collision risk of a region $\mathbb{E}_j$ is calculated by the summation of the weights of particles within the region.

To achieve multi-robot collision avoidance in this particle-based map, shared trajectories of other robots are considered during the risk evaluation. These trajectories are parameterized by their start time and Bezier control points and shared via a broadcast network. Robots are represented as particles with predetermined risk levels. These risks are communicated to other robots through wireless transmission. Using the shared trajectories denoted as $\boldsymbol{p}^\star(t)$, future positions and velocities of these particles at time t_f are inferred, projecting the particles to the position $\boldsymbol{p}^\star(t_f)$ and adopting the velocity $\dot{\boldsymbol{p}}^\star(t_f)$. Then the risk of collision with all other robots in the environment from t_0 to t_f can be calculated by equation 4-7 similar to that for obstacles. Figure 4-5 shows the risk evaluation in the dynamic environment with other robots. The trajectories

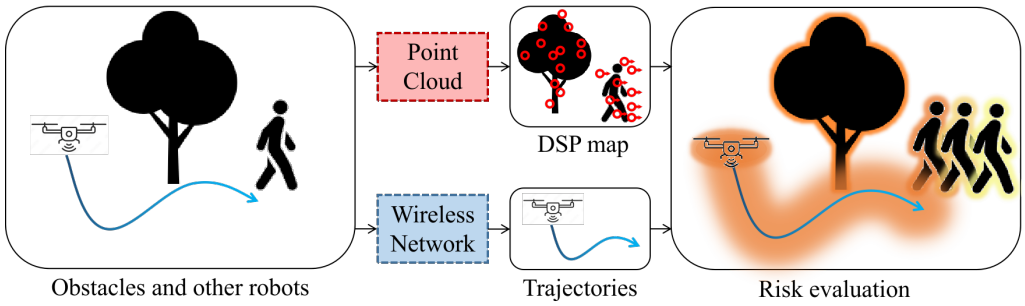


Figure 4-5: Risk evaluation in the dynamic environment with other robots.

shared by these robots are depicted in blue. The figure uses a color gradient to indicate collision risks at various future time steps: orange for imminent risks and yellow for those further in the future.

4-3 Risk-Aware Spatial-Temporal Kinodynamic A*

In order to search for a kinodynamically-feasible path in the particle-based map, we develop a risk-aware spatial-temporal kinodynamic A* algorithm based on the kinodynamic A* algorithm in [51]. This process offers a rough estimate of the trajectory prior to optimization, reducing its complexity. It not only searches for a safe and kinodynamic feasible path but also finds a time allocation for the trajectory. To provide a clear understanding of the algorithm, Algorithm 1 [48] is summarized with an overview visualized in Figure 4-6.

4-3-1 Primitive Expansion

Due to the differential flatness property of the robot, the robot can be described by a double integrator model in 3D space with constant yaw, we can then generate a set of primitives by discretizing the control inputs of the robot [51, 48]. We use $\boldsymbol{p}(t)$, $\dot{\boldsymbol{p}}(t)$, and $\ddot{\boldsymbol{p}}(t)$ to denote the position, velocity, and acceleration of the robot at time t, respectively. The robot state at time t is indicated by $\boldsymbol{x}(t) = [\boldsymbol{p}_x(t), \boldsymbol{p}_y(t), \boldsymbol{p}_z(t), \dot{\boldsymbol{p}}_x(t), \dot{\boldsymbol{p}}_y(t), \dot{\boldsymbol{p}}_z(t)]^\top$, and the control input is indicated by $\mathbf{u} = [\ddot{\boldsymbol{p}}_x, \ddot{\boldsymbol{p}}_y, \ddot{\boldsymbol{p}}_z]^\top$. The robot dynamics is given by the following linear time-invariant system

$$\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\mathbf{u}, \tag{4-8}$$

where $\mathbf{A} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} \end{bmatrix}$. Given the initial state $\boldsymbol{x}(0)$, input $\mathbf{u}(\tau)$ and time duration τ, the robot state at time τ is expressed as

$$\boldsymbol{x}(\tau) = e^{\mathbf{A}\tau}\boldsymbol{x}(0) + \int_0^\tau e^{\mathbf{A}(\tau-t)}\mathbf{B}\mathbf{u}dt. \tag{4-9}$$

$$= \begin{bmatrix} \mathbf{I}_{3\times3} & \tau\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \boldsymbol{x}(0) + \begin{bmatrix} \frac{\tau^2}{2}\mathbf{I}_{3\times3} \\ \tau\mathbf{I}_{3\times3} \end{bmatrix} \mathbf{u}. \tag{4-10}$$

Algorithm 1: Risk-Aware Spatial-Temporal Kinodynamic A* Algorithm

Data: p_{start}, v_{start}, τ, T, p_{goal}
Result: A path from p_{start} to p_{goal} or an indication of failure

$openList \leftarrow$ initialize with $(p_{\text{start}}, v_{\text{start}}, t_{\text{start}})$;
$closedList \leftarrow empty$;
while $openList \neq empty$ **do**
  $current \leftarrow$ node in $openList$ with lowest f;
  if $p_{current}$ *is close enough to* p_{goal} **then**
    return *reconstructed path*; `// Reach the goal`
  if $t_{current} > T$ *and* $p_{current}$ *is out of the map* **then**
    return *reconstructed path*; `// Reach the boundary of the map`
  $openList \leftarrow openList \setminus current$;
  $closedList \leftarrow closedList \cup current$;
  $primitiveList \leftarrow$ `Expansion`$(current, \tau)$;
  foreach $successor \in primitiveList$ **do**
    if `RiskCheck`$(successor) \wedge successor \notin closedList$ **then**
      $succesor.g \leftarrow current.g +$ `Cost`$(successor)$;
      $succesor.h \leftarrow$ `Heuristic`$(successor, p_{\text{goal}})$;
      $succesor.f \leftarrow successor.g + successor.h$;
      add $successor$ to $openList$;
return *failure (no path)*;

By selecting a set of control inputs $\mathbf{u}$ given a fixed time duration τ, we can generate a set of primitives $\boldsymbol{P} = \{\boldsymbol{x}(u, \tau) | u \in \boldsymbol{U}\}$, where $\boldsymbol{U}$ is the set of control inputs discretized uniformly from $[u_{\min}, u_{\max}]$ in each dimension.

4-3-2 Risk Check

In order to check the feasibility of the sampled primitive, we need to check whether primitives collide with the obstacle. Taking into account the shape of the robot and the uncertainty of the obstacle, primitives are inflated with the shape of the robot and checked if the collision risk of the inflated subspace from t_0 to $t_0 + \tau$ is below a threshold ϵ, given by

$$\text{Risk}(\mathbb{E}_j, t_0, t_f) = \sum_{t=\left\{t_0, t_0+\delta t, \ldots, t_f\right\}} \sum_{i=1}^{n_t^{\mathbb{E}_j}} w_t^{(i)} \delta t. \leq \epsilon \tag{4-11}$$

t_0 is the starting time of the primitive and τ is the duration of the primitive. The collision risk of the subspace is defined in Section 4-2-3. If the collision risk is below the threshold, the primitive is considered feasible. Nonfeasible primitives are discarded and rejected to be added to the open list.

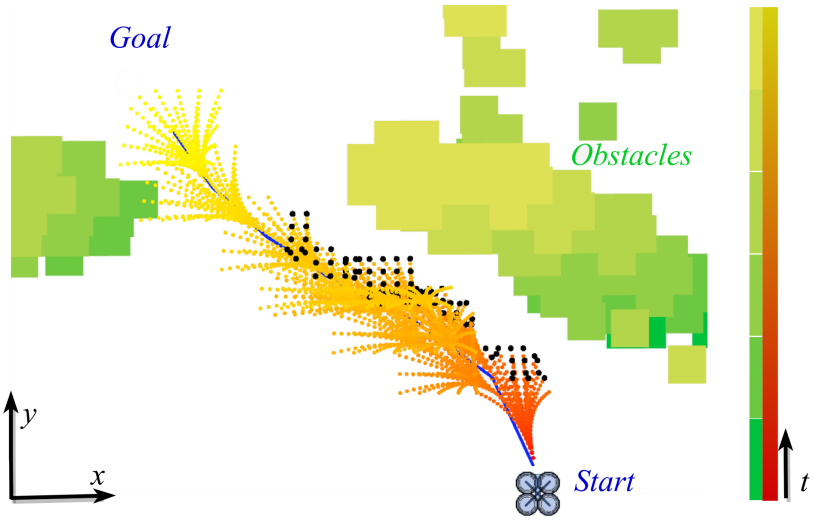


Figure 4-6: Our algorithm finds a collision-free path that avoids the obstacles by sampling a set of primitives (represented by circles in red) and checking the collision risk of each primitive. Obstacles are moving from the right edge toward the center. At the current timestamp, obstacles are shown in green, transitioning to future positions in yellow through a color gradient. Sampled primitives are shown from red to orange through a color gradient, where red signifies the most recent primitives and orange indicates those in the future The black dots represent collision primitives which failed the risk check. The blue curve represents the final collision-free trajectory.

4-3-3 Cost and Heuristic

To provide a smooth path and a reasonable speed profile, we penalize the total acceleration of the trajectory. The cost function is defined as the cumulative square of control inputs over the entire trajectory as

$$\text{PathCost} = \int_0^T \mathbf{u}(t)^\top \mathbf{u}(t) dt. \tag{4-12}$$

Correspondingly, the cost of a primitive can be defined as

$$\text{PrimitiveCost} = \|\mathbf{u}\|^2 \tau. \tag{4-13}$$

The heuristic function is computed by applying Pontryagins minimum principle [84] which yields the optimal control input to reach the goal state $\boldsymbol{x}_{\text{goal}}$. The optimal trajectory from the current state $\boldsymbol{x}_c = \begin{bmatrix} \mathbf{p}_c \\ \mathbf{v}_c \end{bmatrix}$ to the goal state $\boldsymbol{x}_{\text{goal}} = \begin{bmatrix} \mathbf{p}_{\text{goal}} \\ \mathbf{v}_{\text{goal}} \end{bmatrix}$ is given by

$$\mathbf{p}^*(t) = \mathbf{p}_c + \mathbf{v}_c t + \frac{1}{2}\beta t^2 + \frac{1}{6}\alpha t^3, \tag{4-14}$$

where α and β is given by boundary values

$$\alpha = \frac{12}{T^3}\left(\mathbf{p}_c - \mathbf{p}_{\text{goal}}\right) + \frac{6}{T^2}\left(\mathbf{v}_{\text{goal}} + \mathbf{v}_c\right), \tag{4-15}$$

$$\beta = \frac{6}{T^2}\left(\mathbf{p}_{\text{goal}} - \mathbf{p}_c\right) - \frac{2}{T}\left(\mathbf{v}_{\text{goal}} + 2\mathbf{v}_c\right). \tag{4-16}$$

By taking α and β into the cost function, we can get the minimum cost to reach the goal state given the current state and time duration T as

$$J^*(\boldsymbol{x}_c, \boldsymbol{x}_g, T) = \frac{1}{3}\alpha^\top \alpha T^3 + \alpha^\top \beta T^2 + \beta^\top \beta T. \tag{4-17}$$

The heuristic function is defined as the minimum cost to reach the goal state given the current state. We can derive the heuristic cost by finding the minimum cost over all possible time duration T as

$$\text{Heuristic} = \min_T J^*(\boldsymbol{x}_c, \boldsymbol{x}_g, T). \tag{4-18}$$

Since this function is the minimum cost to reach the goal state, it is admissible and consistent. Note this is a nonlinear optimization problem thus the optimal time duration T is computed numerically. Therefore, the A* algorithm is guaranteed to find the optimal path.

4-4 Corridor Constrained Trajectory Optimization

To generate collision-free trajectories, a trajectory optimization method is developed on kinodynamic paths. Initially, spatio-temporal corridors are generated which enclose the obstacle-free region along the path. Subsequently, the trajectory generation problem is formulated as a quadratic programming (QP) problem with linear constraints by applying the convex hull property of Bézier spline trajectory parameterization to simplify the problem.

4-4-1 Optimization-Based Spatio-Temporal Corridor Generation

Since we have found a dynamically-feasible and collision-free path in the particle-based map, we would like to get the obstacle-free region along the path before trajectory generation. Due to the dynamic obstacles in the environment, the safety region of the path may change with time, so the safe flight corridor introduced in [49] is no longer applicable. Therefore, we generate spatio-temporal corridors that enclose the obstacle-free region along each path segment. The spatio-temporal corridor is defined as a convex polyhedron $\mathcal{P} = \{x \in \mathbb{R}^n | \boldsymbol{A}_\mathcal{P} x \leq \boldsymbol{b}_\mathcal{P}\}$ that encloses the safety region within the time window $[t_{i-1}, t_i]$. The corridor is derived from the path segment from t_{i-1} to t_i, factoring in the predicted particle states for the same time interval. This is illustrated in Figure 4-7, where blue curves are kinodynamic paths from the previous step.

To maximize the obstacle-free region and avoid conservative approximation, we form the spatio-temporal corridor generation problem as an optimization problem that maximizes the volume of the corridor. The problem of finding the obstacle-free convex polyhedron in the given time window $[t_{i-1}, t_i]$. can be formulated as the following

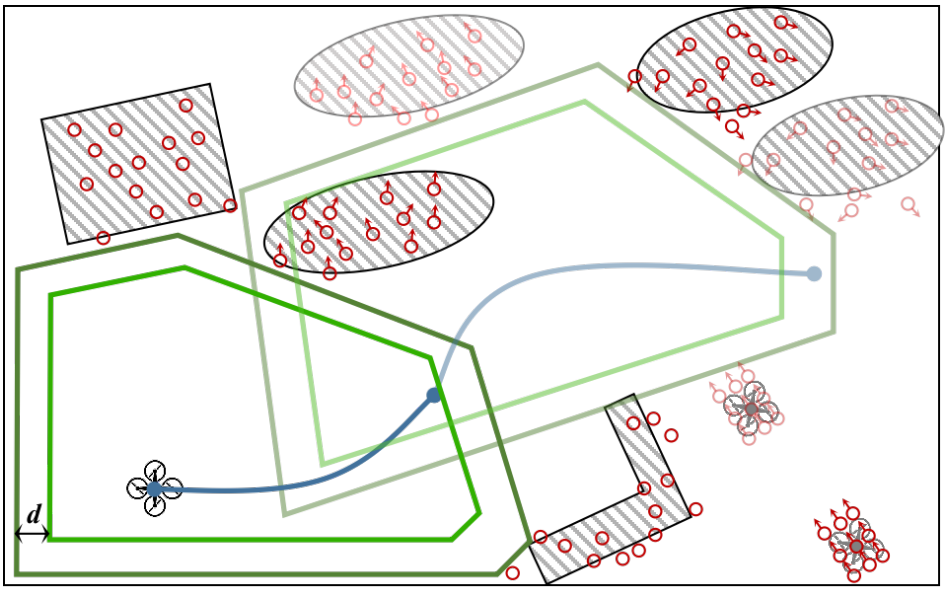


Figure 4-7: Corridor generation and shrinking on the particle-based map, upcoming obstacles and corridors are depicted with transparency.

optimization problem:

$$\max_{\boldsymbol{A}_{\mathcal{P}},\boldsymbol{b}_{\mathcal{P}}} \text{vol}(\mathcal{P}), \tag{4-19}$$

$$\text{s.t. } R_k \subset \mathcal{P}, \tag{4-20}$$

$$\mathcal{O}_{k,t_{i-1},t_i} \subset \mathbb{M} \setminus \mathcal{P}, \tag{4-21}$$

$$\mathcal{O}_{k,t_{i-1},t_i} = \{\mathbf{x}(t) | \mathbf{x}(t) \in \mathbf{X}(t), w_t^{(\mathbf{x}(t))} > w_{\max},\ t \in [t_{i-1}, t_i]\} \tag{4-22}$$

where $\mathbf{X}(t)$ represents the particle sets, $\mathbb{M}$ is the particle-based map, $w_t^{(\mathbf{x}(t))}$ is the weight of the particle $\mathbf{x}(t)$ at time t, $\mathcal{O}_{k,t_{i-1},t_i}$ represents particles whose weight is larger than $w_{\max}$ between time t_{i-1} and t_i, and R_k is the size of robot k.

We apply Fast Iterative Region Inflation (FIRI) algorithm [71] to efficiently solve this optimization problem. Similar to the IRIS method [67], FIRI finds the maximum volume polyhedron $\mathcal{P}$ by iteratively and monotonically increasing the volume of its inscribed ellipsoid using low-dimensional geometry. FIRI reformulates the problem as an equivalent second-order conic programming (SOCP) problem to decrease the dimension of the problem and reduce the computational cost. Thanks to the efficiency of FIRI, we can generate a feasible spatio-temporal corridor from sampled particles in the map within a few milliseconds.

Since the corridor generated by FIRI only encloses the obstacle-free region without considering the volume of the robot, we need to shrink the corridor to ensure that the robot can pass through it. This can be done by pushing the edges of the corridor inwards by the radius of the robot d. Considering the polyhedron $\mathcal{P}$ represented by H-representation, the shrinking operation can be formulated as:

$$\boldsymbol{A}_{\mathcal{P}} x \leq \boldsymbol{b}_{\mathcal{P}} - d\boldsymbol{a}'. \tag{4-23}$$

where each row of vector $\boldsymbol{a}'$ satisfies $\boldsymbol{a}'_j = \left(\boldsymbol{A}_{\mathcal{P}_j}\boldsymbol{A}_{\mathcal{P}_j}^\top\right)^{-\frac{1}{2}}$. Figure 4-7 demonstrates the concept of corridor shrinking. The original corridors, optimized by FIRI, are depicted in dark green, while the modified corridors appear in light green. Corridors for future trajectories are depicted in a translucent manner.

To ensure that corridors are feasible after shrinking, we need to check if inequalities $\boldsymbol{A}_{\mathcal{P}}x \leq \boldsymbol{b}_{\mathcal{P}} - d\boldsymbol{a}'$ are still satisfied. Specifically, we can form a linear programming (LP) problem to efficiently check the feasibility of the resulting corridors:

$$\min_{x} \quad \mathbf{0}^T x \tag{4-24}$$

$$\text{s.t.} \quad \boldsymbol{A}_{\mathcal{P}}x \leq \boldsymbol{b}_{\mathcal{P}} - d\boldsymbol{a}'. \tag{4-25}$$

If the LP problem is non-feasible, which means the corridor is too narrow for the robot to pass through, we discard it and any incoming corridors then proceed directly to the trajectory generation step. This avoids generating narrow corridors that the robots may not be able to pass through and helps ensure the safety and feasibility of the generated trajectories.

4-4-2 Minimum Jerk Trajectory Optimization

The trajectory optimization problem can be formulated as a constrained optimization problem with the cost function defined as the cumulative square of the third derivative of the position over the time horizon. The trajectory is discretized into T segments with the same duration τ.

$$\min_{\mathbf{c}} \sum_{j=1}^{T} \int_{t_{j-1}}^{t_j} \left\| \frac{\mathrm{d}^3 \boldsymbol{p}_j(t)}{\mathrm{d}t^3} \right\|^2 dt \tag{4-26a}$$

$$\text{s.t.} \quad \boldsymbol{p}(t_0) = \boldsymbol{p}_0, \quad \boldsymbol{p}(t_T) = \boldsymbol{p}_T \tag{4-26b}$$

$$\boldsymbol{p}_j(t) \in \mathcal{P}_j, \forall t \in [t_{j-1}, t_j] \tag{4-26c}$$

$$\boldsymbol{p}_j(t_j) = \boldsymbol{p}_{j+1}(t_j) \tag{4-26d}$$

$$\boldsymbol{p}_j^{(1)}(t) \leq \boldsymbol{p}_{max}^{(1)}, \ \boldsymbol{p}_j^{(2)}(t) \leq \boldsymbol{p}_{max}^{(2)}, \ \forall t \in [t_0, t_T], \tag{4-26e}$$

To simplify the corridor constraint (4-26c), we employ the Bézier curve to parametrize the trajectory. Bézier curve is a n-th order piecewise polynomial defined as:

$$\boldsymbol{p}_j(t) = \sum_{i=0}^{n} \mathcal{B}_n^i\left(\frac{t - t_{j-1}}{t_j - t_{j-1}}\right)\boldsymbol{c}_j^i, \quad t \in [t_{j-1}, t_j), \tag{4-27}$$

where $\boldsymbol{c}_j^i$ denotes the i-th control point at the j-th piece of the Bézier curve. $\mathcal{B}_n^i(t) = \binom{n}{i} t^i (1-t)^{n-i}$ is the Bernstein basis. As mentioned in Section 3-2-3, the Bézier curve carries two important properties [62]: hodograph property and convex hull property. The hodograph property shows that the derivative of the Bézier curve is also a Bézier

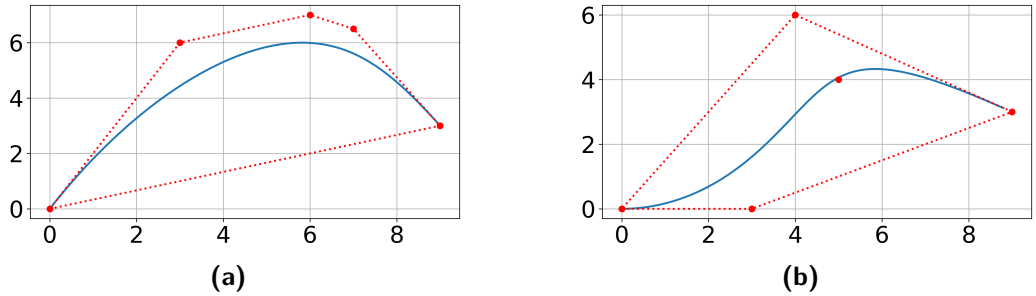


Figure 4-8: Two 4th-order Bézier splines (blue) and their control points. Both Bézier splines are confined within the convex hulls in red dashed lines.

curve, which can be used to limit velocity and acceleration. By applying this property, the first and second derivatives of the given Bézier trajectory can be written as:

$$\frac{d\boldsymbol{p}_j}{dt} = \frac{n}{t_j - t_{j-1}} \sum_{i=0}^{n-1} \mathcal{B}_{n-1}^i \left(\frac{t - t_{j-1}}{t_j - t_{j-1}} \right) (\boldsymbol{c}_j^{i+1} - \boldsymbol{c}_j^i), \quad t \in [t_{j-1}, t_j) \tag{4-28}$$

$$\frac{d^2\boldsymbol{p}_j}{dt^2} = \frac{n(n-1)}{(t_j - t_{j-1})^2} \sum_{i=0}^{n-2} \mathcal{B}_{n-2}^i \left(\frac{t - t_{j-1}}{t_j - t_{j-1}} \right) (\boldsymbol{c}_j^{i+2} - 2\boldsymbol{c}_j^{i+1} + \boldsymbol{c}_j^i), \quad t \in [t_{j-1}, t_j) \tag{4-29}$$

The convex hull property guarantees that the trajectory is entirely inside the convex hull made by the control points. Therefore, we can simply confine the j-th piece of trajectory $\boldsymbol{p}_j(t)$ within the spatio-temporal corridors $\mathcal{P}_j$ by constraining the corresponding control points $\{\boldsymbol{c}_j^0, \boldsymbol{c}_j^1, \cdots, \boldsymbol{c}_j^n\}$ as follows:

$$\boldsymbol{A}_{\mathcal{P}_j} \boldsymbol{c}_j^i \leq \boldsymbol{b}_{\mathcal{P}_j}, \quad i \in \{0, 1, \cdots, N\}, \tag{4-30}$$

As shown in Figure 4-8, both quartic Bézier curve (visualized in blue) is confined within the convex hull (depicted in red dashed lines) constructed by the control points (red dots).

In the following section, we will show how Bézier spline can help to simplify the optimization problem.

Cost function Given the quartic Bernstein basis with $n = 4$ as $\mathcal{B}_4^i(t) = \binom{4}{i} t^i (1-t)^{4-i}$, the trajectory $\boldsymbol{p}_j(t)$ can be written in the following matrix form:

$$\boldsymbol{p}_j(t) = \begin{bmatrix} \boldsymbol{c}_j^0 & \boldsymbol{c}_j^1 & \boldsymbol{c}_j^2 & \boldsymbol{c}_j^3 & \boldsymbol{c}_j^4 \end{bmatrix} \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ 0 & 4 & -12 & 12 & -4 \\ 0 & 0 & 6 & -12 & 6 \\ 0 & 0 & 0 & 4 & -4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \\ t^4 \end{bmatrix} \quad t \in [0, 1) \tag{4-31}$$

where $\boldsymbol{c}_j^i$ is a vector, denotes the i-th control point of the j-th piece.

To derive the cost function, we first derive the first, second and third-order derivative of the trajectory $\boldsymbol{p}_j(t)$ as

$$\frac{d^1\boldsymbol{p}_j(t)}{dt^1} = \begin{bmatrix}\boldsymbol{c}_j^0 & \boldsymbol{c}_j^1 & \boldsymbol{c}_j^2 & \boldsymbol{c}_j^3 & \boldsymbol{c}_j^4\end{bmatrix} \begin{bmatrix}-4 & 12 & -12 & 4 & 0\\ 4 & -24 & 36 & -16 & 0\\ 0 & 12 & -36 & 24 & 0\\ 0 & 0 & 12 & -16 & 0\\ 0 & 0 & 0 & 4 & 0\end{bmatrix} \begin{bmatrix}1\\ t\\ t^2\\ t^3\\ t^4\end{bmatrix} \tag{4-32}$$

$$\frac{d^2\boldsymbol{p}_j(t)}{dt^2} = \begin{bmatrix}\boldsymbol{c}_j^0 & \boldsymbol{c}_j^1 & \boldsymbol{c}_j^2 & \boldsymbol{c}_j^3 & \boldsymbol{c}_j^4\end{bmatrix} \begin{bmatrix}12 & -24 & 12 & 0 & 0\\ -24 & 72 & -48 & 0 & 0\\ 12 & -72 & 72 & 0 & 0\\ 0 & 24 & -48 & 0 & 0\\ 0 & 0 & 12 & 0 & 0\end{bmatrix} \begin{bmatrix}1\\ t\\ t^2\\ t^3\\ t^4\end{bmatrix} \tag{4-33}$$

$$\frac{d^3\boldsymbol{p}_j(t)}{dt^3} = \begin{bmatrix}\boldsymbol{c}_j^0 & \boldsymbol{c}_j^1 & \boldsymbol{c}_j^2 & \boldsymbol{c}_j^3 & \boldsymbol{c}_j^4\end{bmatrix} \begin{bmatrix}-24 & 24 & 0 & 0 & 0\\ 72 & -96 & 0 & 0 & 0\\ -72 & 144 & 0 & 0 & 0\\ 24 & -96 & 0 & 0 & 0\\ 0 & 24 & 0 & 0 & 0\end{bmatrix} \begin{bmatrix}1\\ t\\ t^2\\ t^3\\ t^4\end{bmatrix} = \boldsymbol{C}\boldsymbol{E}_3\boldsymbol{T} \tag{4-34}$$

where $\boldsymbol{T} = \begin{bmatrix}1\\ t\\ t^2\\ t^3\\ t^4\end{bmatrix}$ is a column vector, $\boldsymbol{E}_3 = \begin{bmatrix}-24 & 24 & 0 & 0 & 0\\ 72 & -96 & 0 & 0 & 0\\ -72 & 144 & 0 & 0 & 0\\ 24 & -96 & 0 & 0 & 0\\ 0 & 24 & 0 & 0 & 0\end{bmatrix}$ is a 5-by-5 matrix, and $\boldsymbol{C} = \begin{bmatrix}\boldsymbol{c}_j^0 & \boldsymbol{c}_j^1 & \boldsymbol{c}_j^2 & \boldsymbol{c}_j^3 & \boldsymbol{c}_j^4\end{bmatrix}$ is a 5-by-3 matrix with each column being a control point vector.

Before taking the integral, we need to do some modifications to the equation 4-34. Note matrix $\boldsymbol{C}$ can be represented as three sub-matrics, each representing a row vector of control points in x, y, and z axis respectively, i.e.

$$\boldsymbol{C} = \begin{bmatrix}\boldsymbol{C}_x & \boldsymbol{C}_y & \boldsymbol{C}_z\end{bmatrix}^\top \tag{4-35}$$

Then we can rewrite the equation 4-34 as

$$\frac{d^3\boldsymbol{p}_j(t)}{dt^3} = \boldsymbol{C}\boldsymbol{E}_3\boldsymbol{T} = \begin{bmatrix}\boldsymbol{C}_x\\ \boldsymbol{C}_y\\ \boldsymbol{C}_z\end{bmatrix} \boldsymbol{E}_3\boldsymbol{T} = \begin{bmatrix}\boldsymbol{C}_x\boldsymbol{E}_3\boldsymbol{T}\\ \boldsymbol{C}_y\boldsymbol{E}_3\boldsymbol{T}\\ \boldsymbol{C}_z\boldsymbol{E}_3\boldsymbol{T}\end{bmatrix} = \begin{bmatrix}\boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_x^\top\\ \boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_y^\top\\ \boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_z^\top\end{bmatrix} \tag{4-36}$$

Taking above equations into the cost function, we have

$$\sum_{j=1}^{T}\int_{t_{j-1}}^{t_j}\left\|\frac{\mathrm{d}^3\boldsymbol{p}_j(t)}{\mathrm{d}t^3}\right\|^2 dt \tag{4-37}$$

$$=\sum_{j=1}^{T}\int_0^1\left(\boldsymbol{C}_x\boldsymbol{E}_3\boldsymbol{T}\boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_x^\top+\boldsymbol{C}_y\boldsymbol{E}_3\boldsymbol{T}\boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_y^\top+\boldsymbol{C}_z\boldsymbol{E}_3\boldsymbol{T}\boldsymbol{T}^\top\boldsymbol{E}_3^\top\boldsymbol{C}_z^\top\right)dt \tag{4-38}$$

$$=\sum_{j=1}^{T}\boldsymbol{C}_x\boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top\boldsymbol{C}_x^\top+\boldsymbol{C}_y\boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top\boldsymbol{C}_y^\top+\boldsymbol{C}_z\boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top\boldsymbol{C}_z^\top \tag{4-39}$$

$$=\sum_{j=1}^{T}\begin{bmatrix}\boldsymbol{C}_x,\boldsymbol{C}_y,\boldsymbol{C}_z\end{bmatrix}\begin{bmatrix}\boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top & & \\ & \boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top & \\ & & \boldsymbol{E}_3\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt\boldsymbol{E}_3^\top\end{bmatrix}\begin{bmatrix}\boldsymbol{C}_x^\top\\ \boldsymbol{C}_y^\top\\ \boldsymbol{C}_z^\top\end{bmatrix} \tag{4-40}$$

$$=\sum_{j=1}^{T}\begin{bmatrix}\boldsymbol{C}_x,\boldsymbol{C}_y,\boldsymbol{C}_z\end{bmatrix}\begin{bmatrix}\boldsymbol{E}_3\boldsymbol{E}_{\boldsymbol{T}}\boldsymbol{E}_3^\top & & \\ & \boldsymbol{E}_3\boldsymbol{E}_{\boldsymbol{T}}\boldsymbol{E}_3^\top & \\ & & \boldsymbol{E}_3\boldsymbol{E}_{\boldsymbol{T}}\boldsymbol{E}_3^\top\end{bmatrix}\begin{bmatrix}\boldsymbol{C}_x^\top\\ \boldsymbol{C}_y^\top\\ \boldsymbol{C}_z^\top\end{bmatrix} \tag{4-41}$$

$$\boldsymbol{E}_{\boldsymbol{T}}=\int_0^1\left(\boldsymbol{T}\boldsymbol{T}^\top\right)dt=\int_0^1\begin{bmatrix}1\\ t\\ t^2\\ t^3\\ t^4\end{bmatrix}\begin{bmatrix}1 & t & t^2 & t^3 & t^4\end{bmatrix}dt=\begin{bmatrix}1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5}\\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6}\\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7}\\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8}\\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9}\end{bmatrix} \tag{4-42}$$

Here we can get the cost matrix $\boldsymbol{Q}$ as

$$\boldsymbol{Q}=\boldsymbol{E}_3\boldsymbol{E}_{\boldsymbol{T}}\boldsymbol{E}_3^\top=\begin{bmatrix}192 & -480 & 288 & 96 & -96\\ -480 & 1343 & -1152 & 192 & 96\\ 288 & -1152 & 1728 & -1152 & 288\\ 96 & 192 & -1152 & 1344 & -480\\ -96 & 96 & 288 & -480 & 192\end{bmatrix} \tag{4-43}$$

Then we can derive the cost function as follows:

$$\sum_{j=1}^{T}\int_{t_{j-1}}^{t_j}\left\|\frac{\mathrm{d}^3\boldsymbol{p}_j(t)}{\mathrm{d}t^3}\right\|^2 dt=\sum_{j=1}^{T}\begin{bmatrix}\boldsymbol{C}_x,\boldsymbol{C}_y,\boldsymbol{C}_z\end{bmatrix}\begin{bmatrix}Q & & \\ & Q & \\ & & Q\end{bmatrix}\begin{bmatrix}\boldsymbol{C}_x^\top\\ \boldsymbol{C}_y^\top\\ \boldsymbol{C}_z^\top\end{bmatrix} \tag{4-44}$$

This is a quadratic form cost taking control points $\boldsymbol{C}_x$, $\boldsymbol{C}_y$, $\boldsymbol{C}_z$ as decision variables.

Corridor Constraints Thanks to the convex hull property of Bézier curve, we can easily derive the corridor constraints by constraining the control points to be inside the convex

hull of the corridor. As discussed in Section 4-4-1, given the spatio-temporal corridor $\mathcal{P}_j$ at time interval j represented by linear inequality constraints

$$\boldsymbol{A}_{\mathcal{P}_j}\boldsymbol{p} \leq \boldsymbol{b}_j, \tag{4-45}$$

where $\boldsymbol{p}$ denotes a trajectory point in the 3D space. In our case, by applying the convex hull property of Bézier curve, we can derive the corridor constraints which guarantee the safety of optimized trajectory by enforcing all the control points to be inside the polyhedron $\mathcal{P}_j$:

$$\boldsymbol{A}_{\mathcal{P}_j}\boldsymbol{c}_j^i \leq \boldsymbol{b}_j \tag{4-46}$$

Continuous Constraints As introduced in Section 3-2-3, the Bézier spline satisfies the continuity property. The Bézier spline starts from the first control point $c_{i,1}$ and ends at the final control point $c_{i,N}$. By enforcing the continuity property, we can derive the following constraints to ensure the continuity of the optimized trajectory at the boundary of each time interval:

$$\boldsymbol{c}_j^N = \boldsymbol{c}_{j+1}^0 \quad \forall j \in \{1, \ldots, T-1\} \tag{4-47}$$

where $\boldsymbol{c}_j^N$ denotes the final control point of the j-th time interval, and $\boldsymbol{c}_{j+1}^0$ denotes the initial control point of the $(j+1)$-th time interval.

Boundary Constraints We also need to enforce the boundary constraints to ensure the optimized trajectory starts from the initial position $\boldsymbol{p}_0$ with initial velocity $\dot{\boldsymbol{p}}_0$ and initial acceleration $\ddot{\boldsymbol{p}}_0$, and ends at the final position $\boldsymbol{p}_T$ with velocity $\dot{\boldsymbol{p}}_T$ and acceleration $\ddot{\boldsymbol{p}}_T$. Positional boundary constraints can be easily achieved by constraining the first and last control points to be the initial and final positions.

$$\boldsymbol{c}_1^1 = \boldsymbol{p}_0, \quad \boldsymbol{c}_T^N = \boldsymbol{p}_T \tag{4-48}$$

For velocity and acceleration, we can constrain the control points in the velocity and acceleration trajectory. As shown in Section 3-2-3, the derivative of the Bézier spline is also a Bézier spline with one degree lower. Therefore, we can derive the following boundary constraints:

$$4\boldsymbol{c}_1^2 - 4\boldsymbol{c}_1^1 = \dot{\boldsymbol{p}}_0, \tag{4-49}$$

$$4\boldsymbol{c}_T^N - 4\boldsymbol{c}_T^{N-1} = \dot{\boldsymbol{p}}_T \tag{4-50}$$

$$12\boldsymbol{c}_1^3 - 24\boldsymbol{c}_1^2 + 12\boldsymbol{c}_1^1 = \ddot{\boldsymbol{p}}_0, \tag{4-51}$$

$$12\boldsymbol{c}_T^N - 24\boldsymbol{c}_T^{N-1} + 12\boldsymbol{c}_T^{N-2} = \ddot{\boldsymbol{p}}_T \tag{4-52}$$

Velocity and Acceleration Constraints In order to generate a dynamically feasible trajectory that can be tracked by the drone, we should also constrain the maximum velocity and acceleration of the trajectory. To keep the linearity of the constraints, we limit the velocity and acceleration of the control points instead of the trajectory itself. For simplicity, an axis-wise decouple of the velocity and acceleration constraints is applied by constraining the velocity and acceleration of each axis separately.

Numerical Solution Since the cost function is a quadratic form, and all the constraints are linear, we can formulate the trajectory optimization problem as a quadratic programming (QP) problem. This problem can be solved efficiently by modern QP solvers, e.g. OSQP [85]. In our experiment, we use 4th-order Bézier Splines ($N = 4$) with 5 control points for each time interval ($\tau = 0.2$). The number of time intervals T is determined by the risk-aware kino-dynamic planning algorithm, which is normally less than 9.

Chapter 5

Results and Discussion

In this chapter, I will present the results of the simulation experiments conducted to evaluate the performance of the proposed method. First, I will compare the performance of the proposed method with two state-of-the-art methods in a simulation environment with ground truth information of obstacles. Then, I will discuss the results in a simulation where no prior knowledge of the environment is given. Robots are equipped with onboard sensors, allowing them to perceive the environment as point clouds.

5-1 Evaluation without Perception Module

In this section, the performance of the proposed method is evaluated in a simulation environment where the ground truth information of obstacles is given. This evaluation serves to validate the planning capability of the proposed method in dynamic environments when the perception module is not integrated.

5-1-1 Simulation Environment

First, a 3D simulation environment is built based on a quadrotor simulator in [51]. This simulator is designed for motion planning in a cluttered static environment, which is implemented in C++ and ROS. The simulator is capable of simulating multiple drones with different motion planning algorithms. In this simulator, the drone is modeled as a point mass with 6 degrees of freedom, and the measurement noise and localization noise are neglected. Since this thesis focuses on the motion planning part, I assume the perception, localization, and tracking control are perfect.

The simulator is built to support the simulation of dynamic obstacles. Dynamic obstacles are modeled as cylindrical and circular obstacles with random sizes, initial positions,

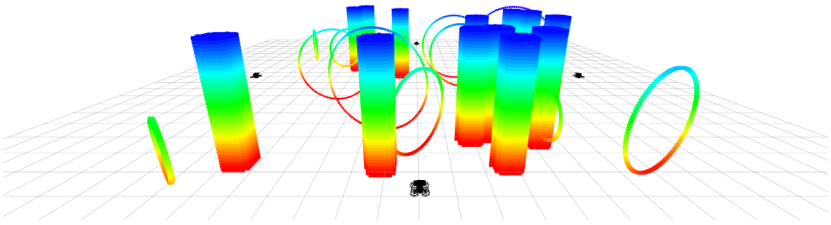


Figure 5-1: A front view of the simulation environment, where the obstacles are represented as point clouds colored by the z-axis. Four drones are initialized at the four edges of the map.

and velocities. The widths of the cylinders are uniformly sampled from 0.5 to 1.0 m. All the cylinders are initialized with a height of 4.0 m. The radius of the circles is uniformly sampled from 0.7 to 2.5 m, with a fixed width of 0.1 m. The initial positions of obstacles are uniformly sampled in a 16m×16m×4m 3D map space. The velocities of the obstacles are randomly sampled from 0 to 1.0 m/s, and the directions are uniformly sampled from 0 to 2π. Obstacles move with constant velocities and directions, and they rebound when they reach the boundary of the map space.

In our simulation, I use point clouds to represent obstacles. This approach mirrors the real-world perception of drones, where onboard depth cameras or LiDAR sensors perceive the environment as point clouds. The point cloud is generated by sampling points on the obstacle surfaces at an update rate of 10 Hz. Additionally, the distance between any two points in the point cloud is set at 0.1 m.

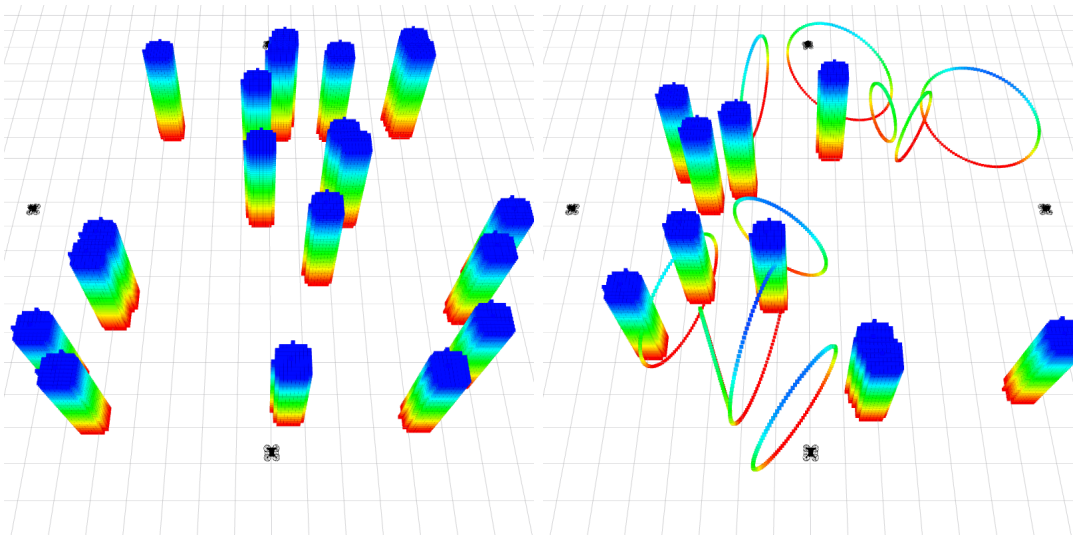
The simulator is lightweight and can be run in real-time. I use this simulation environment to evaluate the performance of the selected baselines and our proposed method. All simulation experiments were performed on a laptop equipped with a 16-core AMD R7-5800H CPU.

5-1-2 Experimental Setups

To evaluate the performance of our proposed method in unknown and dynamic environments, extensive simulation experiments are conducted in different scenarios. A 3D environment is built based on the simulator mentioned in Section 5-1-1, where 4 drones are simulated to perform collision avoidance in a dynamic environment with moving obstacles. Drones are initialized in the hovering state at a height of 1.0 m, outside the obstacle space. The goal positions are carefully selected to ensure that the quadrotors will go through the obstacle space and their trajectories will intersect with each other.

The experiments are divided into two parts across environments of varying difficulty levels. In the first part, extensive simulations are conducted in a cluttered environment with cylindrical obstacles. (Figure 5-2a) By varying the number of obstacles from 10 to 50, the performance is evaluated to challenge the scalability of the proposed method. The second part of the experiments is performed in a mixed environment with an equal

mix of cylindrical and circular obstacles, leading to a more challenging collision avoidance problem. (Figure 5-2b) This setup aims to create a more cluttered environment by breaking the convexity and symmetry of the obstacles. Robots can choose either side to avoid cylindrical obstacles; however, for circular obstacles, they must plan a 3D trajectory to avoid the edges. Note that robots are allowed to pass through circular obstacles from center to center, but they must avoid the edges, otherwise, a collision will occur. To quantify the difficulty, I calculate the average obstacle density, defined as the ratio of total obstacle volume to the entire volume of the obstacle space. Table 5-1 shows the average obstacle density for each environment setting.



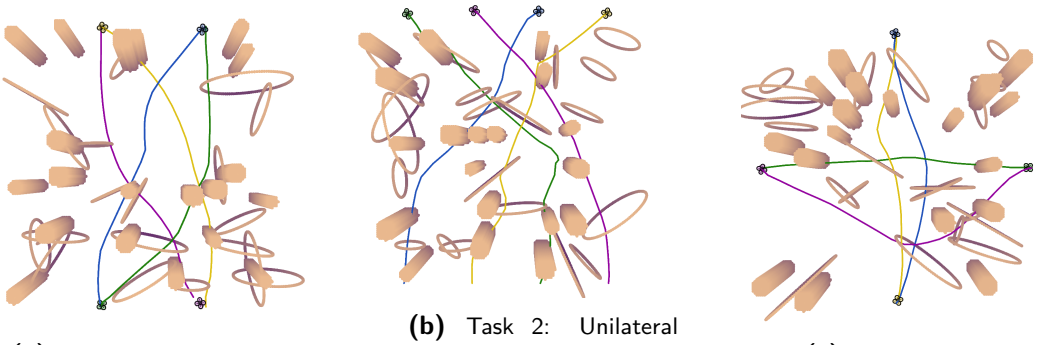
(a) Cylindrical obstacle environment **(b)** Complex environment

Figure 5-2: A visualization of two different simulation environments

Num. of Obstacles	10	20	30	40	50
Cylindrical obstacle environment (%)	1.73	3.45	5.18	7.00	8.62
Complex environment (%)	0.86	1.73	3.71	4.53	5.76

Table 5-1: Average obstacle density in different environments

Three different position swap tasks are designed in the test, as depicted in Fig. 5-3a–5-3c. The first task is a bilateral position swap, where two drones are initialized on one side of the obstacle space and the other two drones are initialized on the opposite side. The goal positions of the drones are swapped, so they need to pass through the obstacle space to reach their goals. Fig. 5-3a depicts the final state in which drones have successfully swapped positions, with intersecting trajectories. Obstacles are colored by the z-axis with a color map ranging from purple to orange. The second task is a unilateral position swap, where drones are initialized on the same side of the obstacle space and the goal positions are on the other side with different orders to ensure trajectory intersection. To validate the capability of the proposed method to handle collision avoidance with cooperative robots, I design the third task as a cross-quadrant position swap, where drones are initialized at four different edges of the obstacle space



(a) Task 1: Bilateral Swap

(b) Task 2: Unilateral Swap

(c) Task 3: Cross Swap

Figure 5-3: Illustration of three different task settings (Fig. 5-3a–5-3c)

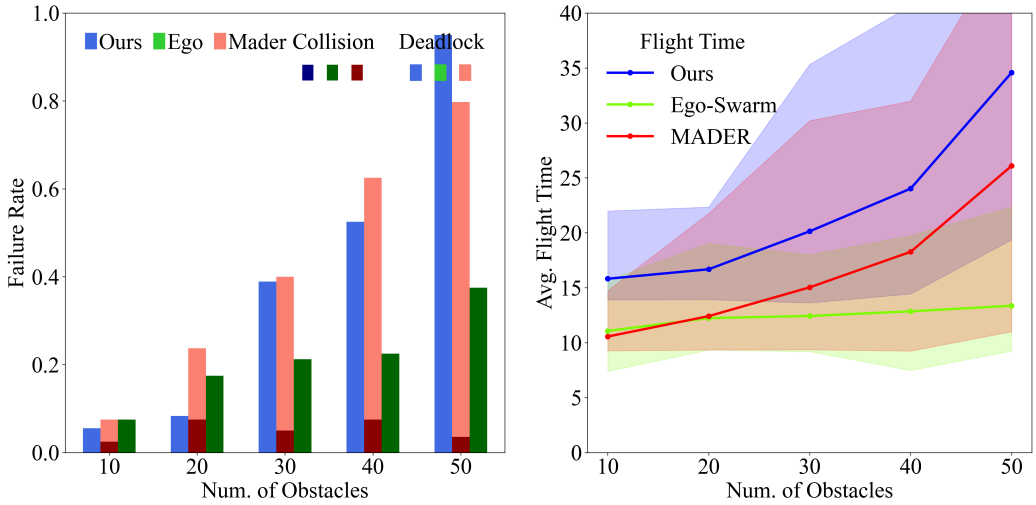
Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.93	**0.00**	0.08	7.40	15.69	**11.07±1.55**	0.46	0.47
	MADER	0.93	0.05	0.03	9.26	**14.74**	10.56±1.30	**0.60**	0.82
	Proposed	**0.98**	0.03	**0.00**	**2.31**	30.50	19.27±3.20	0.46	**1.15**
Unilateral Swap	EGO-Swarm	**0.96**	**0.00**	0.04	9.40	27.16	11.23±2.10	0.52	0.87
	MADER	0.93	0.04	0.04	9.26	**16.70**	**10.35±1.58**	**0.63**	0.89
	Proposed	0.95	0.01	0.04	**3.76**	24.46	18.98±2.56	0.47	**1.05**
Cross Swap	EGO-Swarm	**0.99**	0.00	0.01	**8.42**	24.05	11.10±2.12	0.50	0.46
	MADER	0.93	**0.06**	0.01	9.46	**17.76**	**11.03±1.61**	**0.69**	0.87
	Proposed	0.90	0.05	0.05	9.15	26.24	19.06±2.14	0.41	**0.92**

Table 5-2: Comparison of the performance of different planners across different tasks in the pure cylinder environment with 10 obstacles.

and the goal positions are at the opposite edges. The robots are expected to encounter each other in the center of the obstacle space. Fig. 5-3a–5-3c presents a bird's eye view of these settings in a mixed environment with 20 cylinders and 20 circles.

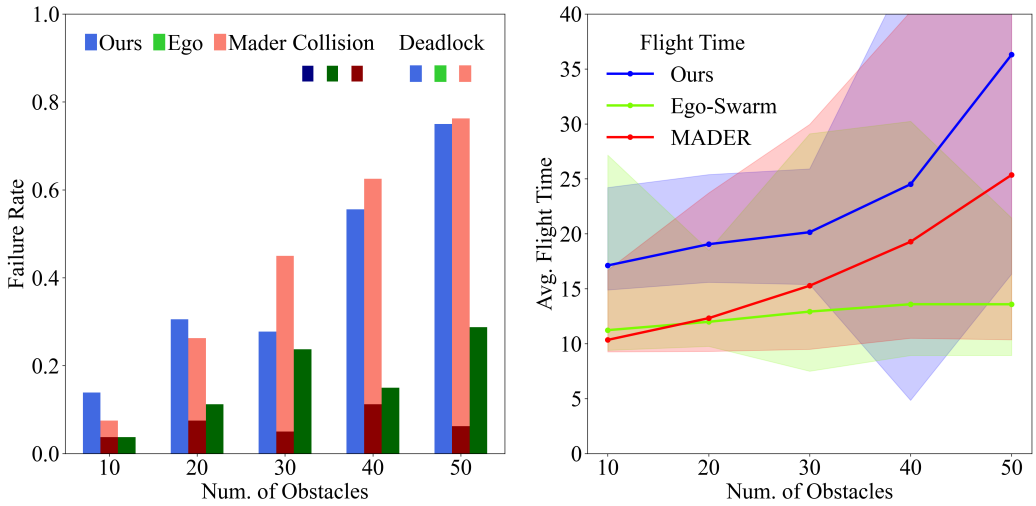
The proposed method is compared with two baselines: MADER [40] and EGO-Swarm [6]. MADER is a hard-constrained method that uses axis-aligned bounding boxes (AABBs) to represent obstacles, and EGO-Swarm is a soft-constrained method formulating collision cost to push the trajectory away from the obstacles by minimizing the total cost. Since [40] and [6] rely on obstacle sizes and trajectories, the ground truth trajectories of the obstacles are used for these two baselines. To make a fair comparison, I use the ground truth DSP map, which is a local map with the ground truth positions and velocities of the particles. The local map covers a 4m×4m area with a resolution of 0.1m. The update frequency of the DSP map is 10Hz. Similarly, the ground truth trajectories of the obstacles that are used in the baselines are also updated at 10Hz. All methods are constrained to use local obstacle information. Specifically, robots can only perceive obstacles within a radius of 5.0 m and have a time horizon of 2.0 s. The maximum velocity of the drones is set to 2.0 m/s, and the maximum acceleration is set to 6.0 m/s^2

5-1-3 Results in Cylindrical Obstacle Environment



(a) Failure rate **(b)** Trajectory time

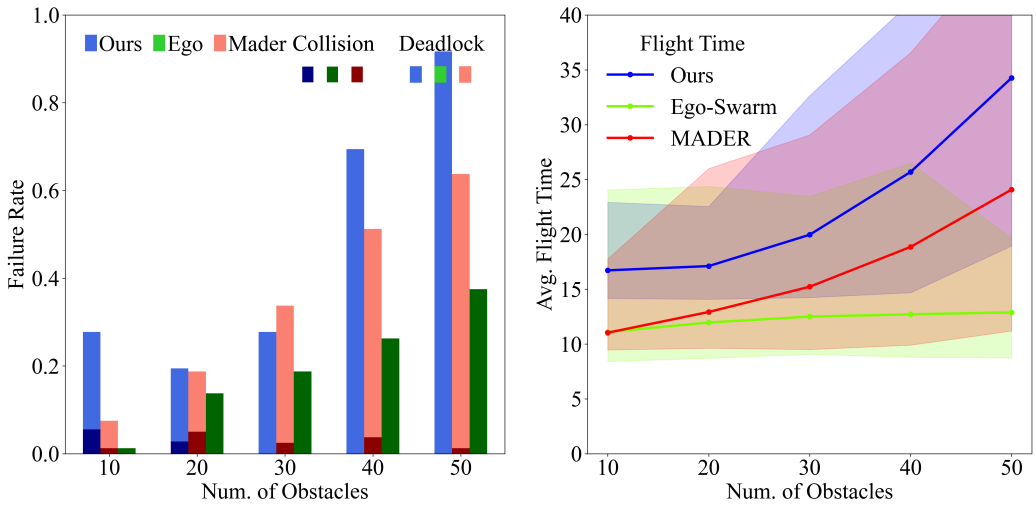
Figure 5-4: Comparison of planners' performance in the bilateral swap task with varying numbers of cylindrical obstacles.



(a) Failure rate **(b)** Trajectory time

Figure 5-5: Comparative analysis of various planners' performance in the unilateral task with varying numbers of cylindrical obstacles.

The performance is evaluated based on failure rate, trajectory time, and average minimum distance to obstacles. Table 5-2 shows the results of all tasks in the environment with 10 cylindrical obstacles. More results are shown in the appendix (Table A-1 to A-4). In general, our method closely matches the performance of the state-of-the-art methods in terms of success rate. It significantly outperforms the baselines in the bilateral swap task; however, its performance is slightly inferior to the baselines in the



(a) Failure rate **(b)** Trajectory time

Figure 5-6: Comparative analysis of various planners' performance in the cross-swap task with varying numbers of cylindrical obstacles.

cross-quadrant swap task. From the results shown in Table 5-2, we can see our planner achieves the largest averaged minimum distance to other robots, which means our planner can guarantee a safe distance between different robots, However, the average flight time of our method is worse than other methods, but but our method has a greater variance in flight time compared to other methods, as depicted in Table 5-2. This is because our method is more conservative than others, opting to wait for obstacles until a safe path is found. This waiting time contributes to the variance in flight time.

Figure 5-4 to Figure 5-6 compare the performance of different planners in the three tasks with different numbers of obstacles. The failure rate and trajectory time increase with the number of obstacles in the environment. As the number of obstacles increases, the failure rate of our method increases faster than other methods, indicating that our method is more conservative than other methods therefore the safe trajectories are harder to find in obstacle-dense environments. Comparing the results in Figure 5-5 and Figure 5-6 can also tell that our method is more sensitive to the density of obstacles than other methods.

To further understand the failure reason, we distinguish failures into two different types: collision and deadlock. Collision only happens when the agent is following a planned trajectory. Deadlock is defined as a situation when a dynamic obstacle hits an agent after the agent has stopped due to the inability to generate a feasible trajectory. In many real-world scenarios, e.g., robots are passing through crowded areas full of pedestrians, deadlock is not a problem since the pedestrians will not directly hit the robot. As shown in Figure 5-4 to Figure 5-6, deadlock accounts for almost all failures in our method. In contrast, EGO-Swarm hardly encounters deadlock, but it has a higher collision rate than other methods. Safety is assured because our method confines the planned trajectory within the corridors.

5-1-4 Results in the Complex Environment

From the results shown in Figures 5-7 to 5-9, we can see that the proposed planner outperforms the other baseline planners in terms of success rate. The proposed method achieves the highest success rate in most of the tasks, and most of the failures are caused by deadlock. Furthermore, in obstacle-dense environments, the trajectory time of the proposed method is shorter than that of MADER but remains comparable to EGO-Swarm across all tasks.

Similar to the results in previous environments, the failure rate, trajectory time, and averaged minimum distance to obstacles are shown in Table A-6. Results can be found in the appendix (Table A-5–A-9). Table A-6 indicates that our method surpasses other baseline methods in success rate for bilateral swap and unilateral tasks. Regarding trajectory time, our method is close to EGO-Swarm in all tasks. Results show that our method plans trajectories that are closer to the obstacles, indicating a more efficient utilization of space.

Furthermore, our method performs better in the complex environment than in the cylindrical obstacles environment. As depicted in Figure 5-4 and Figure 5-7, a significant decrease in failure rate can be seen in obstacle-rich environments. These results reveal that the ability of the particle-based map to represent obstacles with arbitrary shapes allows for greater flexibility in finding feasible paths in such environments. Using the particle-based map to represent the environment is less conservative than axis-aligned bounding boxes in MADER and ellipsoidal costs in EGO-Swarm.

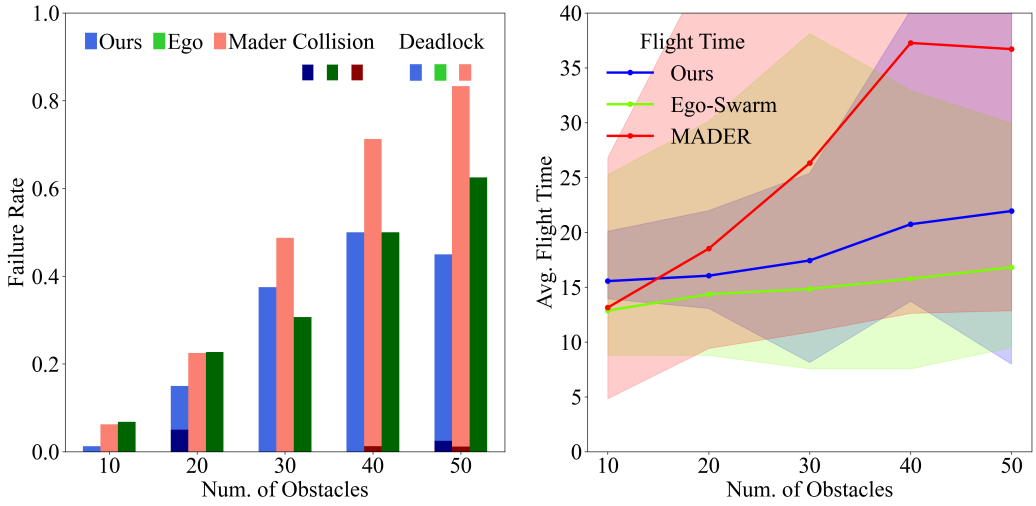
Table 5-4 compares the success rate and trajectory time across different obstacle levels in the complex environment.

Task	**Planner**	**Succ. rate**	**Fail. rate**		**Trajectory time (s)**			**Avg. min. dist (m)**	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.77	**0.00**	0.23	**8.79**	30.15	**14.36±3.364**	0.19	0.48
	MADER	0.78	0.23	**0.00**	9.42	44.68	18.51±6.985	**0.21**	**1.02**
	Proposed	**0.85**	0.10	0.05	13.06	**21.98**	16.05±1.888	0.17	0.90
Unilateral Swap	EGO-Swarm	0.78	**0.02**	0.19	**4.32**	34.99	**14.7±4.889**	0.17	1.13
	MADER	0.75	0.20	0.05	9.44	42.54	19.06±6.929	0.18	**1.53**
	Proposed	**0.90**	0.10	**0.00**	12.26	**23.30**	17.02±2.077	**0.19**	1.06
Cross Swap	EGO-Swarm	**0.87**	**0.00**	0.13	**8.83**	33.82	**13.67±4.323**	0.22	0.53
	MADER	0.84	0.16	0.00	9.58	39.36	18.51±7.147	0.22	0.96
	Proposed	0.79	0.21	0.00	13.76	**23.26**	16.93±2.368	0.13	**1.00**

Table 5-3: Comparison of the performance of different planners across different tasks in the complex environment with 20 obstacles.

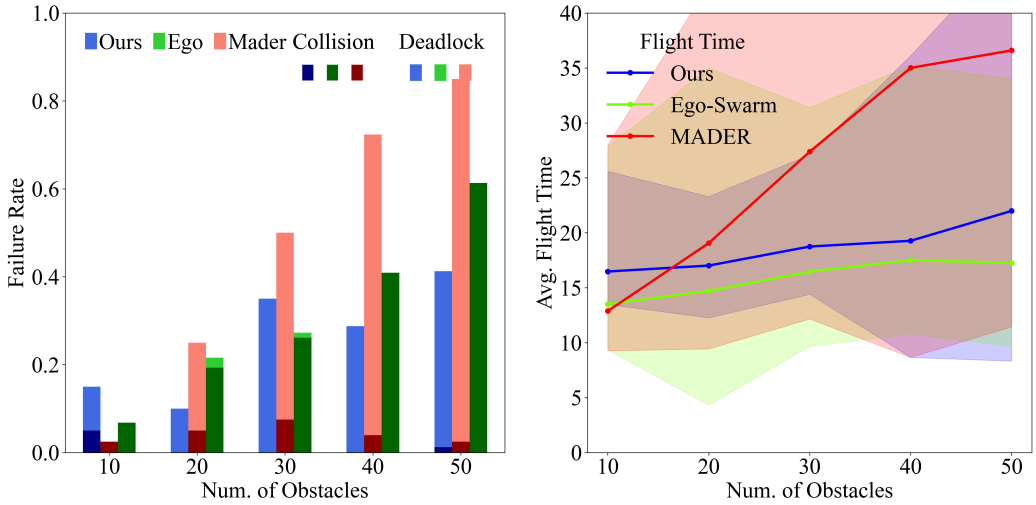
5-1-5 Discussion

In general, our method outperforms MADER with a 98.5% higher success rate and a 21.9% reduction in flight time. When compared to EGO-Swarm, we see an 88.2% decrease in the collision rate and a 13.6% increase in the success rate.



(a) Failure rate **(b)** Trajectory time

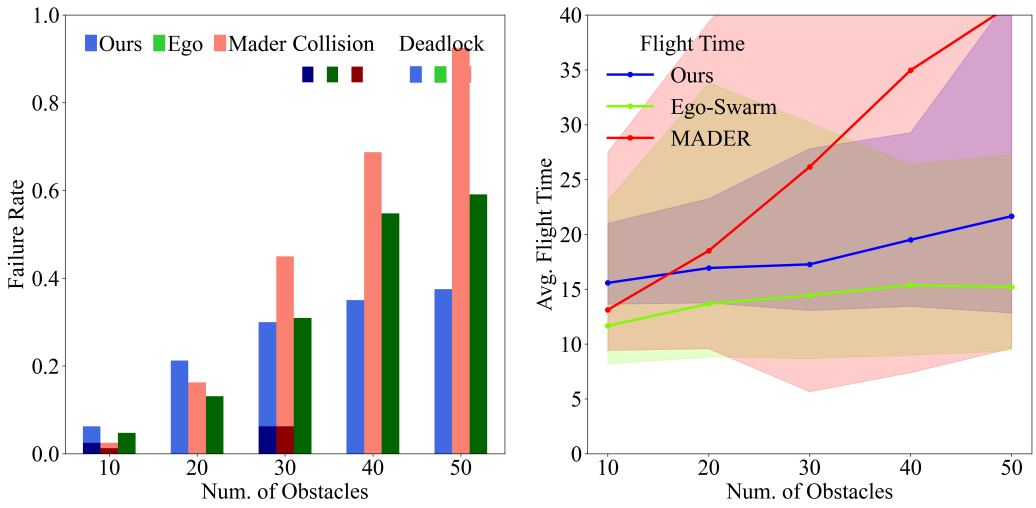
Figure 5-7: Comparative analysis of planners' performance in the bilateral swap task with varying numbers of obstacles in the complex environment.



(a) Failure rate **(b)** Trajectory time

Figure 5-8: Comparative analysis of various planners' performance in the unilateral task with varying numbers of obstacles in the complex environment.

Based on the results in both environments, we can draw the following conclusions. In both environments, MADER is more prone to local deadlocks because it demands future trajectories of obstacles at every time step. However, this is not applicable in many real-world scenarios since robots usually have local perceptions and cannot predict the future trajectory of unknown obstacles. EGO-Swarm adjusts its time allocation to prevent sudden stops, which is why it rarely encounters deadlocks. This adjustment also accounts for EGO-Swarm having the shortest trajectory time. However, being a gradient-based method to generate trajectories by solving a soft-constrained opti-



(a) Failure rate **(b)** Trajectory time

Figure 5-9: Comparative analysis of various planners' performance in a cross-swap task with varying numbers of obstacles in the complex environment.

	Success Rate					Trajectory Time (s)				
Num. Obs.	10	20	30	40	50	10	20	30	40	50
Proposed	0.93	0.85	0.66	0.62	0.59	15.87	16.67	17.82	19.84	21.87
EGO-Swarm	0.94	0.81	0.70	0.51	0.39	12.68	14.24	15.25	16.22	16.42
MADER	0.96	0.79	0.52	0.29	0.13	13.05	18.69	26.62	35.75	38.09

Table 5-4: Comparison of the performance of different planners across different obstacle levels in the complex environment.

mization problem, EGO-Swarm is at a greater risk of colliding with obstacles. The results validate this conclusion where the collision rate is highest among the three. Our method stands out by achieving the lowest collision rate in both environments, although with more conservative trajectories. The failure rate of our method is in the same range as EGO-Swarm in pure cylinder environments, but it is better in mixed environments. Despite utilizing ground truth information in experiments, collision rates are not eliminated. This limitation arises from our constant velocity assumption for trajectory prediction, which doesn't accurately capture the sudden velocity changes occurring when objects rebound at space boundaries.

5-2 Evaluation with the Perception Module

To validate the effectiveness of our planner as a complete system, we integrate our planner with the perception module from [10] and conduct experiments in a simulated environment with moving pedestrians.

5-2-1 Experimental Setup

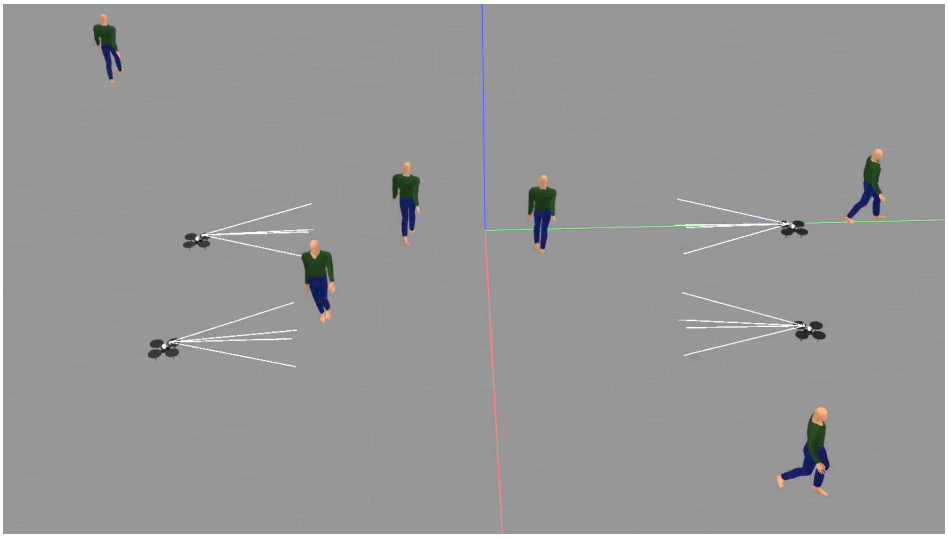


Figure 5-10: Screenshot of the simulated environment, where four drones are flying in a dynamic environment with 6 pedestrians.

The experiment is conducted in a simulated environment with moving pedestrians, as shown in Figure 5-10. In this environment, four drones are initialized at four corners of the space and are required to fly to the opposite corner similar to the bilateral swap task in Section 6-3-2. The size of each drone is 0.5 m $\times$ 0.5 m $\times$ 0.3 m. Each drone is equipped with a depth camera with an 85.2-degree horizontal field of view (FOV) to sense the environment. The orientation of the depth camera is marked as a white cone in Figure 5-10. We use DSP map [10] to generate the particle-based map for the planner. In the planning stage, we modify the risk threshold to 0.20 to allow the planner to generate more conservative trajectories with uncertain perceptions. The maximum velocity and acceleration of the drone are limited to 1.5 m/s and 4.0 m/s^2 respectively.

5-2-2 Results

In Figure 5-11, we show the planning results at a certain time step when 4 drones are coordinating to avoid collision with each other. The figure on the left is a bird-eye view of the environment in Gazebo, and the figure on the right is the corresponding visualization of the bottom-right drone. Within the visualization, the discretized particle-based map is shown, colored by the z-axis value of the particles. The particles of pedestrians and other drones are shown on the map. The red path represents the planned path searched by the risk-aware kino-dynamic A* algorithm. Additionally, safety corridors generated by the searched path are depicted in transparent blue. As shown in the figure, the particles and upcoming trajectories of other drones are represented as particles in the map, where a collision-free path is searched from. Safety corridors are generated to avoid the people on the upper right, as well as other drones on the left. The corridor encompasses the area currently occupied by the drone on the left because the shared

trajectory shows that it will move downward. The searched A* path moves upward to avoid collision with the drone on the left as shown in Figure 5-11.

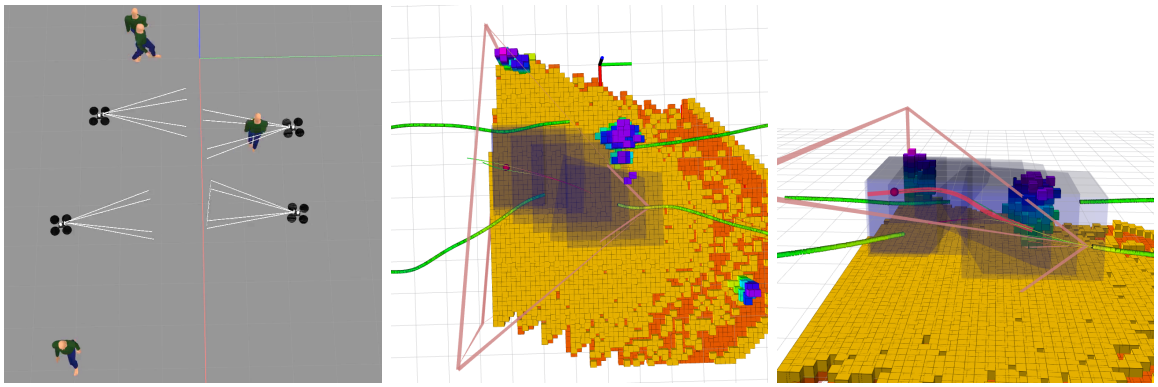


Figure 5-11: Visualization of the planning results of the bottom-right drone in Gazebo and RViz when 4 drones are encountering each other in a dynamic environment with 6 pedestrians.

Based on the data presented in Table 5-5, it's evident that the success rate diminishes as the number of pedestrians in the environment increases. There are two main reasons for these collisions. First, the camera's restricted field of view (FOV) can lead to undetected pedestrians outside this view. As shown in Figure 5-11, there are several pedestrians whose walking directions are perpendicular to the drone camera direction. The drone is not able to detect these pedestrians if they are walking from the side or rear. Second, the time delay in the perception module also contributes to the collision. In our case, the computation time of the DSP map fluctuates between 20 ms and 100 ms, depending on the size of the point cloud input. The planned trajectories will be affected by these perception delays, compromising the evasive maneuvers and leading to potential collisions.

Num. Ped.	Success rate	Failure rate		Trajectory time (s)			Avg. min. dist (m)	
		Dead. rate	Coll. rate	Avg.	Min.	Max.	Obs	MAV
4	0.833	0.000	0.167	19.021 ± 1.688	16.500	23.176	0.990	1.173
6	0.611	0.037	0.352	19.016 ± 1.765	13.908	24.100	0.270	1.167
8	0.444	0.009	0.546	19.024 ± 2.053	15.460	25.160	0.170	1.153

Table 5-5: Results of the proposed planner in the simulation with perception module.

5-3 Computational Efficiency Analysis

The run time of each step is evaluated on a laptop with an AMD R7-5800H CPU. As shown in Figure 5-12, the search step takes less than 2 ms in most cases. The corridor generation step takes 2.34 ms on average. The trajectory optimization step takes 6.93 ms on average, which is time-consuming due to the large number of polygon planes as corridor constraints. In general, the execution time of our planner averages 10.04 ms, ranging from a minimum of 0.53 ms to a maximum of 114 ms. The results

demonstrate that our method can be deployed on robots with limited computational resources. Additionally, it is significantly faster than 31.04 ms reported by MADER [40].

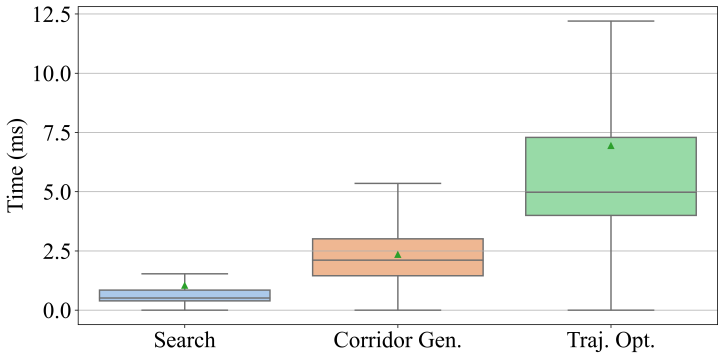


Figure 5-12: The execution time of each step in our method.

Chapter 6

Prototype Implementation

In this chapter, the engineering implementation of the experimental multi-MAV platform for future real-world experiments is presented. This platform is used to validate the utility and feasibility of the proposed multi-robot planning algorithms. First, the hardware design will be introduced, including the selection and discussion of sensors, motors, and computational boards mounted. Next, we will present the software architecture used in this thesis work. Finally, the network communication setup will be discussed.

6-1 Hardware Platform

The method proposed in Chapter 4 requires an autonomous MAV platform that has onboard sensing, computing, and communication capabilities to achieve real-time environment sensing, obstacle avoidance, and safe navigation. Therefore, we need to design a fully-equipped and versatile MAV platform to validate the proposed algorithm in real-world experiments. To reach this goal, the designed MAV is required to carry a depth sensor to provide environmental perception based on depth images and is equipped with powerful computing modules. Safety is an important aspect of robotics research, therefore, the MAV design should consider sufficient strength to withstand crushes and the incorporation of propeller guards to prevent potential harm to humans. Furthermore, because of the scale limitations of the experimental environments, the size and weight of the designed MAV are limited. To balance total weight and computation powers, we will discuss the hardware selection in the following sections.

6-1-1 Overview

To meet the above requirements, there exists a variety of open-sourced research platforms. A widely-used platform in many MAV works is Crazyflie 2.0 [86], which has a

Framework	Onboard Computer	Low-Level Controller	CPU Mark	GPU Mark (TFLOPS)	Frame Size (mm)	Total Weight (g)	Thrust/Weight
Crazyflie [86]	✗	custom	N/A	✗	92	27	≈2.26
Parrot Bebop2 [30]	✓	custom	1343	1.33	-	≈750	≈2
ASL-Flight [87]	✗	DJI	3383	✗	887	3620	≈2.32
RPG-Quad [88]	✓	Betaflight	633	✗	-	-	≈4.00
MRS UAV [89]	✓	PX4	8846	✗	450	-	≈2.50
FAST-Drone-250 [5]	✓	PX4	9954	✗	250	≈1500	≈4.00
RPG-Agilicious[90]	✓	custom	1343	1.33	264	750	≈5.00
Ours	✓	PX4	2000	2.12	180	502	≈3.50

Table 6-1: A comparison of different MAV platforms

92 mm wheelbase and only weighs 27g. However, it is too small to carry additional resources for onboard perception and computation. Although commercial companies, such as Parrot, Skydio, and DJI, have provided various drone platforms, however, due to their limited software development kits (SDK), researchers have limited access to their onboard sensors and computers. Thus, robotics research based on these types of drones is inconvenient. Some researchers managed to design their platform based on these drones, such as [91], who designed a mount for the extra sensor and the onboard computer. Control commands are sent to onboard drivers via a ROS wrapper, which translates input commands into internal flight commands. However, due to the limited functionalities of the ROS wrapper, some special requirements, such as yaw control, high-speed flying, and agile maneuverability, may not be met. More and more laboratories around the world have developed their own research platform, such as ASL-Flight [87], CUVT-MRS-UAV [89], ZJU-FAST-Drone-250 [5, 6] and RPG-Agilicious [90]. However, these platforms are designed for specific research purposes, such as drone racing [88, 90] and outdoor exploration and inspection [87, 89]. These platforms are compared on the basis of their onboard capability, size, weight, and agility in Table 6-1.

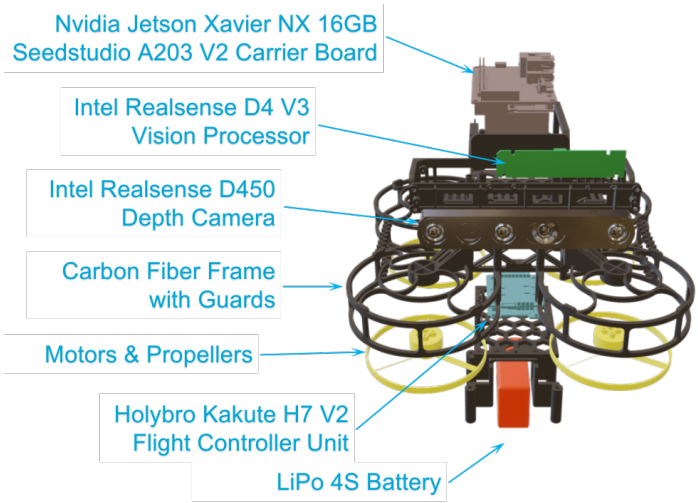


Figure 6-1: 3D model of the designed drone platform

Here, we introduce our human-friendly MAV system designed for indoor autonomous

navigation. The MAV platform is designed to be versatile and can be used for various research, such as cooperative exploration, cooperative localization and mapping, and autonomous navigation in unknown dynamic environments, which is what we discussed in this thesis. Our drone platform has a wheelbase size of 180 mm and a total weight of 500g, which is small enough for indoor experiments and large enough to carry essential sensors and computing modules for our tasks. It is equipped with a RealSense D455 depth camera for onboard perception and an NVIDIA Jetson Xavier NX board for onboard computation. We use the Holybro Kakute H7 V2 flight controller, which integrates a 6-axis inertial measurement unit (IMU) and a barometer, to provide onboard attitude estimation and altitude control. The drone is powered by a 4S 850mAh LiPo battery, which only has 120g weight and can provide enough power for 5 minutes of flight time. The system architecture is shown in Figure 6-1 as a 3D model.

6-1-2 System Architecture

An autonomous drone consists of multiple modules, including the frame, motors, propellers, flight controller, sensors, and payload. This equipment can be categorized into three parts: propulsion system, flight control system, and computation system. Figure 6-2 provides a comprehensive overview of our MAV system.

The propulsion system includes an electronic speed controller (ESC), motors, propellers, and batteries. It thrust the drone and provides the drone with the ability to fly in the air. In our design, we use four BetaFPV 1508 3600KV brushless motors with 3-inch propellers. According to the manufacturer, each motor can provide 150g of thrust at 50% throttle. The motors are driven by four Holybro Tekko32 F3 35A ESCs. These modules are powered by a 4S 2000mAh LiPo battery, which can allow the drone to fly aggressively for 5 minutes.

The flight control system includes the flight controller, accelerometer, gyroscope, magnetometer, barometer, and GPS. It provides the drone with the ability to estimate its state and control its attitude and altitude, as well as communication with the joystick and the ground station. Our drone uses a Holybro Kakute H7 V2 flight controller unit which integrates IMU and gyroscope with a total weight of 8g. The Wi-Fi telemetry modules are connected to the flight controller via a serial port, which builds a UDP connection with the ground station. It can also communicate with onboard computers via a TTL to USB converter, which receives upper-level control commands from the onboard computer and sends back the state estimation of the drone.

The computation system includes an onboard computer and a depth camera. It provides the drone with the ability to perceive the environment and plan a trajectory. The onboard computer is an NVIDIA Jetson Xavier NX board, which has a 6-core ARM CPU and a 384-core NVIDIA Volta GPU. The depth camera is a RealSense D455 camera, which is able to provide a depth range of 0.2 to 6 m with a resolution of 640x480 at 30Hz. The total weight of the computation system is 260g, which is more than half of the total weight of the drone.

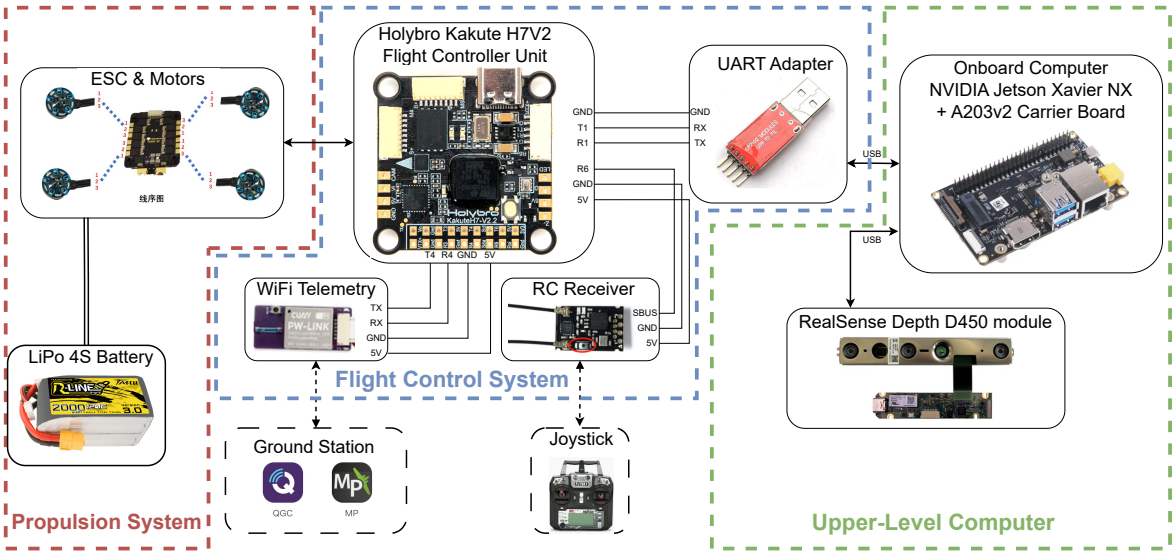


Figure 6-2: The electrical settings of our MAV system.

6-1-3 Perception

The perception system provides the drone with the ability to perceive the environment and estimate its state, which can be summarized as two research tasks: localization and mapping. The localization algorithms fuse data from different sensors to provide the real-time state estimation of the drone. The mapping algorithms build a map of the environment from the raw point cloud data, which is usually represented as a 3D occupancy map in practice [37, 10].

There are various perception approaches in the community, two popular approaches are vision-based and lidar-based approaches based on the type of sensors used. The vision-based approach is the most common one, which employs RGB-D cameras to perceive the environment. The RGB image provides the drone with the ability to detect and track objects, while the depth image gives the distance estimation. The localization task can be accomplished by employing various visual-inertial odometry (VIO) algorithms that fuse IMU data with images, using monocular [92, 93] or stereo [94, 95] cameras. Intel RealSense cameras are commonly used for MAVs as they are small, lightweight, and cost-effective. It should be noted that with RealSense depth cameras, the protective casing can be detached, enabling us to utilize the camera modules and vision processing units individually. This modular approach significantly minimizes the comprehensive mass of the camera, reducing it to a mere 30g. Meanwhile, lidar-based approaches utilize LiDARs for perception, and various corresponding localization algorithms have been proposed recently [96, 97]. Since LiDARs can provide a much larger perception range both in field-of-view and detection range. This increased perception scope facilitates the detection of a larger number of features, enhancing the accuracy and robustness of the odometry estimates produced by Lidar-Inertial Odometry (LIO) systems. Consequently, these LIOs often outperform Visual-Inertial Odometry (VIO) systems in this aspect. Traditionally, LiDARs were heavy and expensive, making them

Approach	Sensor	Weight	Perception Range	Frequency	Horizontal FOV	Price
Vision	Intel RealSense D435i	72g	0.2-10m	60Hz	87ř	365$
	Intel RealSense D455	103g	0.5-10m	60Hz	87ř	419$
LiDAR	Livox Mid-360	265g	0.1-40m		360ř	749$
	Velodyne Puck LITE	590g	0.5~100m	20 Hz	360ř	7999$
	Ouster OSDome	200g	0.5-20m	100Hz	360ř	16000$

Table 6-2: A comparison between vision-based and lidar-based approaches

unsuitable for miniaturized MAVs. However, recently introduced commercial LiDARs, like LIVOX Mid-360 [1], weigh less than 300g, which is comparable to most RGB-D cameras. Hence, we compare the two types of sensors in Table 6-2.

As introduced in the previous section, we use RealSense D455 to provide onboard perception for our drone. The consideration of choosing this sensor includes the following aspects:

- **Weight**: Considering the flight time of our drone platform, the total weight of the system should be as light as possible. The weight of Lidar is significantly heavier than the depth camera, which may influence the overall performance of the drone.
- **Price**: Another reason for opting against LiDAR is its prohibitive cost.
- **Usability**: The choice of perception module should be aligned with the requirements of the implemented perception algorithm. Given that our DSP-map [10] demonstrates improved compatibility with depth cameras in dynamic environments due to its pyramid structure for particle estimation, we have chosen to use depth cameras as our main onboard perception sensors.
- **Perception Accuracy**: Taking into account two different types of RealSense cameras, the depth estimation of D455 is twice as accurate as D435i in the same perception range, although it is slightly longer and heavier.

Based on the aforementioned considerations, the Intel RealSense D455 depth camera is the best fit for our drone platform.

6-1-4 Assembly Process

The process of assembling the drone hardware is visually depicted in Figures 6-3 and 6-4. Firstly, the power supply is soldered to both the flight controller and the Electronic Speed Controller (ESC) to provide the necessary power to the drone. Following this, the motor is securely attached to the drone's base and integrated with the ESC. The flight controller and ESC are then mounted onto the drone base. Next, you should connect the propellers to the motors carefully by considering the mounting direction to ensure the propulsion system can function correctly. Subsequently, the supporting

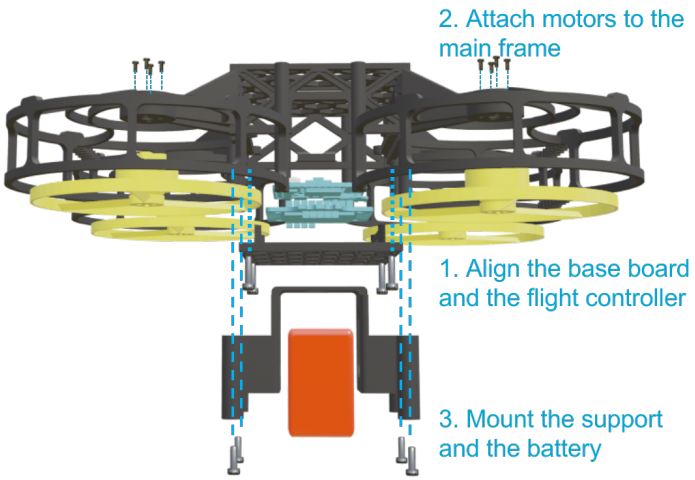


Figure 6-3: The mounting of the drone base

components and the battery are positioned on the drone base, ensuring that the drone's center of gravity aligns with the central point of the base.

To verify the correct rotational order of the propeller, a manual flight test can be conducted. A misaligned propeller will have unbalanced torque, which will dangerously cause the drone to spin uncontrollably. In the final step of assembly, the onboard computer and the depth camera are installed on the drone base, as depicted in Figure 6-4.

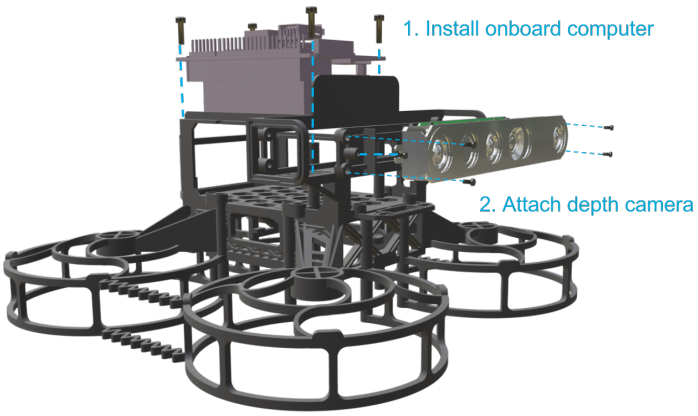


Figure 6-4: The mounting of the onboard computer and the depth camera

6-2 Software Architecture

To address the complexity of the drone platform, it is essential to design a software architecture that is modular and extensible, enabling the reusability of software com-

[1] https://www.livoxtech.com/mid-360

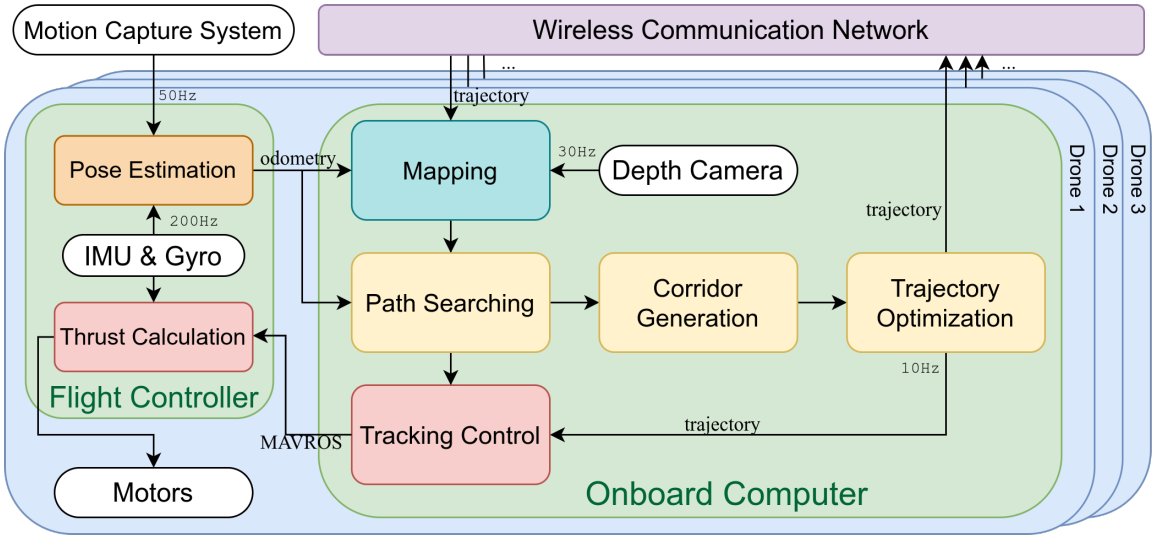


Figure 6-5: The software architecture of our drone platform

ponents. The architecture should support the integration of various functionalities and promote flexibility. As depicted in Figure 6-5, the software architecture of our drone platform comprises five primary components: perception, localization, planning, control, and communication. Each component serves a specific function and is color-coded to distinguish its role within the system visually. These components are implemented on the onboard computer or embedded within the flight controller. The flight controller itself is programmed using the PX4 firmware, which is a widely used open-source autopilot software suite specifically designed for drones. On the onboard computer, the remaining components are implemented using the Robot Operating System (ROS). Sensors and actuators are depicted in white in Figure 6-5.

The localization module (marked in orange in Figure 6-5) is responsible for estimating the pose of the drone in the global coordinate system. In our system, the localization module utilizes a motion capture system to obtain precise global position and altitude information for the drone. By detecting the reflective markers placed on the drone, it is able to accurately track and capture the drone's pose, which is then published at a rate of 50 Hz. To further enhance the accuracy of the pose estimation and reduce latency, the module fuses this pose information with high-frequency data from the onboard Inertial Measurement Unit (IMU). This fusion process performed within the flight controller results in a more refined and precise estimation of the drone's pose.

The perception module (marked in blue) is responsible for processing the point cloud data from the depth camera and generating the environment map. It is implemented based on the DSP-map [10] algorithm, which is a novel probabilistic mapping algorithm for dynamic environments. It is able to build a dynamic occupancy map with predictions of the future states of the environment.

The planning module (marked in yellow) generates a collision-free trajectory based on the DSP map and the current pose of the drone. A detailed discussion can be found in Chapter 4.

Then, the trajectory is sent to the control module (marked in cyan), which is responsible for controlling the drone to follow the trajectory. The optimized trajectory is first discretized into a series of target state points, which are then sent to the tracking controller. The tracking controller is a model-based cascaded PID controller implemented in ROS that is responsible for calculating the desired state input for the drone to track the target state points. The target state points contain the desired position, altitude, velocity, and acceleration of the drone at each time step. These inputs are then sent to the low-level controller implemented in the flight controller to calculate the desired thrust of each motor. We use MAVROS to communicate with the ROS nodes running on the onboard computer and PX4 autopilot running on the flight controller.

The trajectory is also sent to the communication module (marked in purple) to be broadcasted to other drones in the network. The details of the communication modules will be discussed in Section 6-3. In the past decades, there have been a variety of methods proposed for prediction problems in such environments, which can be categorized into two groups. One category of methods considers the dynamic obstacles and possible static obstacles uniformly and predicts the future states of all the obstacles. The other category detects the dynamic obstacles from the static background and only predicts the future state of the detections.

6-3 Network Communication

Multi-robot navigation requires the coordination of multiple robots, in which the robots need to communicate to exchange information. The robustness of the communication is significant since communication failure may lead to the collision of the robots. In our system, robots communicate with each other through the 5GHz wireless network with ROS. The network is able to achieve a communication bandwidth from 10MB/s to 300MB/s, which is sufficient for the exchange of trajectory information. The exchanged information includes 10 Hz planned trajectories sent from the planning node, as well as 100 Hz pose estimation from the OptiTrack system. The architecture of the multi-MAV communication network is illustrated in Figure 6-6.

6-3-1 Existing Solutions

There are several methods to establish a communication network for multi-robot systems via ROS, including

1. ROS 1 XML-RPC middleware [2]
2. ROS 2 DDS middleware [3]
3. Custom middleware

[2] http://wiki.ros.org/ROS/Technical%20Overview#Topic_Transports
[3] https://design.ros2.org/articles/ros_on_dds.html

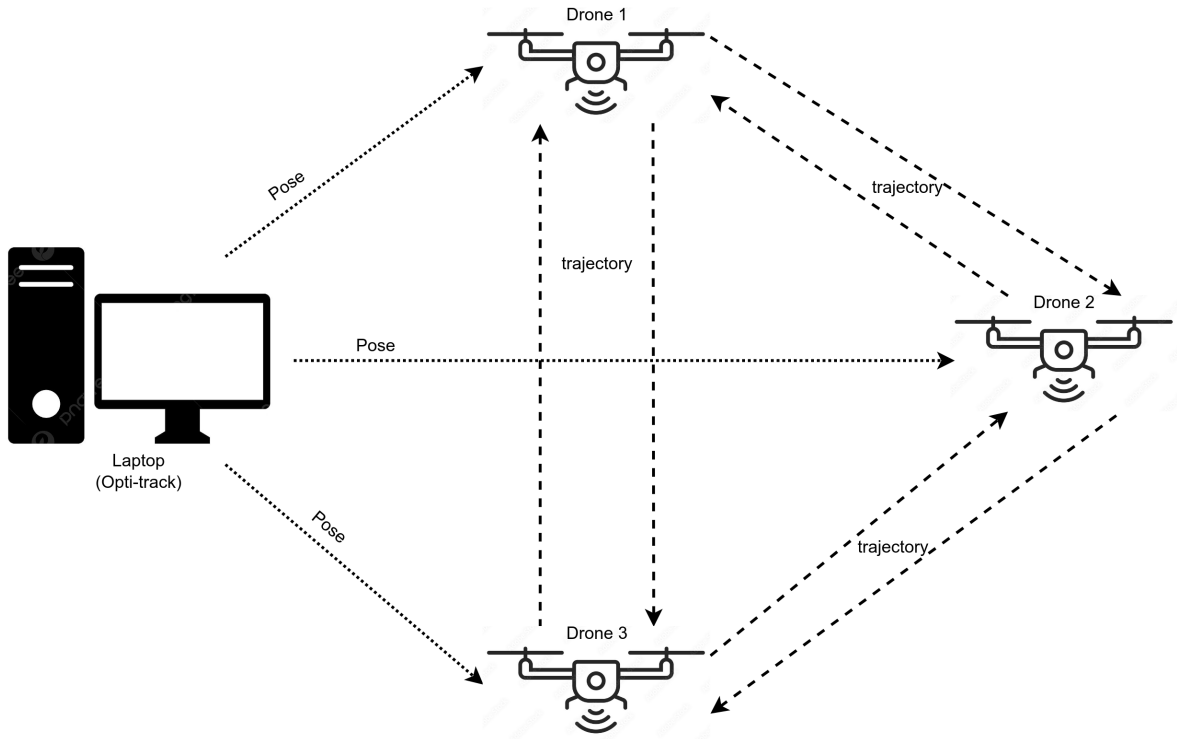


Figure 6-6: Communication network of the designed multi-robot system.

The ROS 1 XML-RPC middleware is the default middleware for ROS 1. It uses a master-slave-based architecture to establish the communication network between multiple robots. It is mandatory to have a central master machine to manage the communication between nodes. The master node is responsible for registering the topics and the nodes within and between robots, and it also provides the service to resolve the IP address of each robot. The major drawback is the existence of the master node, which limits its robustness and scalability, making it challenging to deploy on multi-robot systems when the status of the communication network is not ideal [98].

The communication configurations are complex: each robot has to set up the correct IP address under the network, as well as the correct ROS master URI and hostname of each robot. When launching the ROS node, it is required to find the ROS core on the master machine first to register the node and resolve the topic subscription relationship. In addition, since all the ROS node relies on communication with the master machine, the failure of communication between the robot and the master machine will lead to the failure of the whole system. The communication architecture of ROS 1 XML-RPC middleware is illustrated in Figure 6-7. A centralized master node is responsible for managing topic subscriptions between nodes. In this example, two drones rely on the communication network to exchange trajectory information. Each drone consists of a mapping node, which constructs a local map using perception data, and a planning node, responsible for generating safe trajectories.

The ROS 2 DDS middleware is a distributed middleware that relies on the fast Data

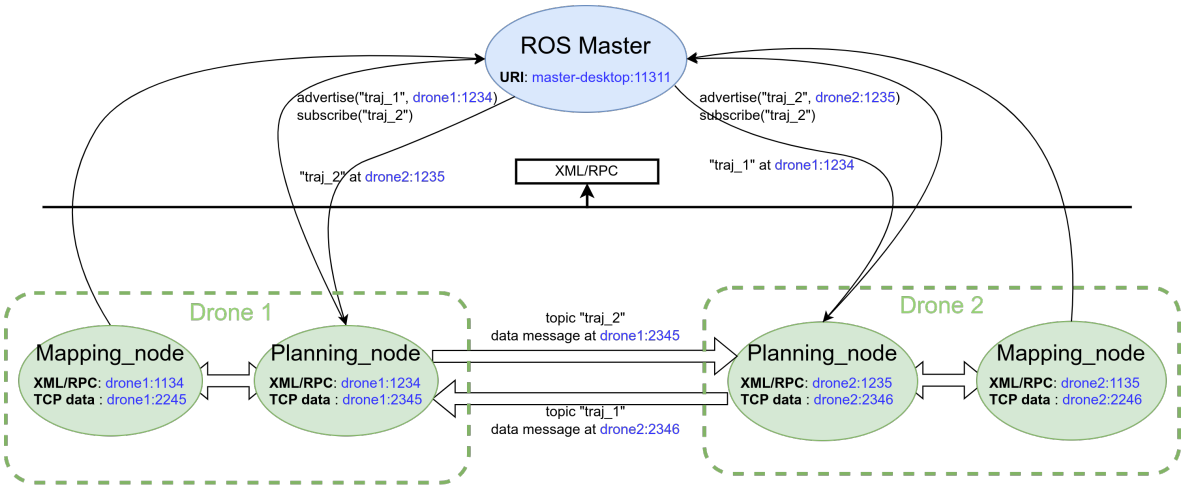


Figure 6-7: Communication Architecture of ROS1 XML-RPC Middleware.

Distribution Services (DDS) protocol to establish the communication network. Due to the distributed architecture, it is not necessary to have a master machine for topic registration and node resolution. Furthermore, users can easily configure the communication network by launching arbitrary nodes on any machine. However, the major problem of ROS 2 DDS middleware is its reliability. [4] A primary problem is the intermittent disruption of the communication network. The DDS middleware in ROS 2 is typically developed for data exchange among multiple processes on a local machine, rather than for wireless communication across multiple machines. By default, fast DDS utilizes the UDP communication protocol, which is less reliable than TCP when it comes to wireless communication since UDP does not guarantee the delivery of data. The unstable connection of the communication network will lead to the loss of data, which is unacceptable for our multi-robot system which relies on trajectory sharing to avoid mutual collision. Due to this reason, the ROS 2 DDS middleware is not suitable for multi-robot communication in our system.

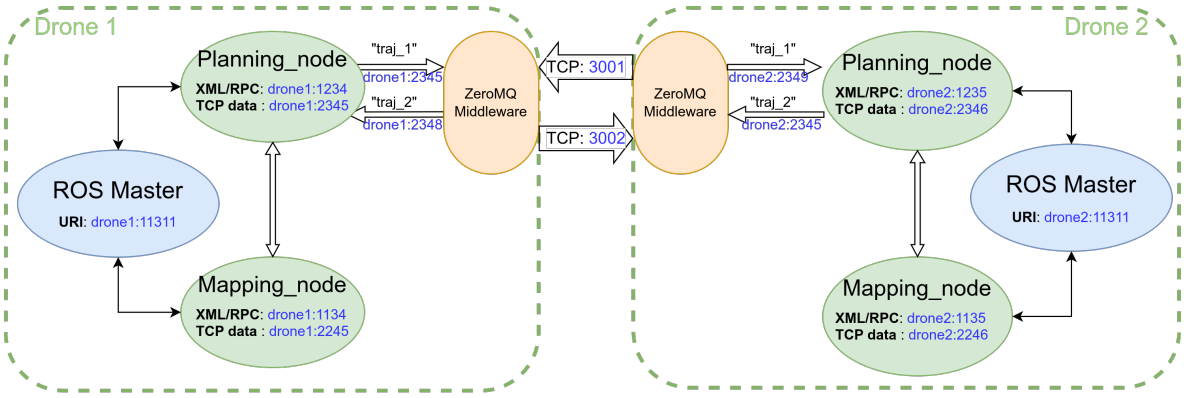


Figure 6-8: Communication Architecture of ZeroMQ-based Custom Middleware.

A more reliable and practical solution for multi-robot communication is to use a custom

[4] https://answers.ros.org/question/403517/ros2-network-communication-does-it-even-work-reliably/

socket-based middleware. Using libraries such as `sys/socket.h`, one can write custom socket communication code for TCP or UDP to convert ROS messages into TCP data streams and send them to a specific robot. As mentioned before, TCP is more reliable than UDP because UDP is only responsible for sending and receiving data without considering whether the recipient is online or has received the data. To avoid writing the socket communication code from scratch, we use ZeroMQ [5] as our middleware.

ZeroMQ provides a lightweight, high-performance messaging library for TCP communication that simplifies the implementation of communication patterns between distributed components. It encapsulates all the complexities of socket communication and provides a set of "data-oriented" API interfaces for us to utilize. The ZeroMQ-based communication network is illustrated in Figure 6-8. Each robot can start autonomously and connect in random order. The communication network is established by the ZeroMQ-based custom middleware. Figure 6-9 shows the communication architecture of the ZeroMQ-based custom middleware in our system. Similar to the system in Figure 6-7, two drones rely on the communication network to exchange trajectory information. Each drone consists of a mapping node, which constructs a local map using perception data, and a planning node, responsible for generating safe trajectories

In the following sections, we will compare the communication performance of the ZeroMQ-based custom middleware and the ROS 1 XML-RPC middleware in a multi-robot system with 3 drones.

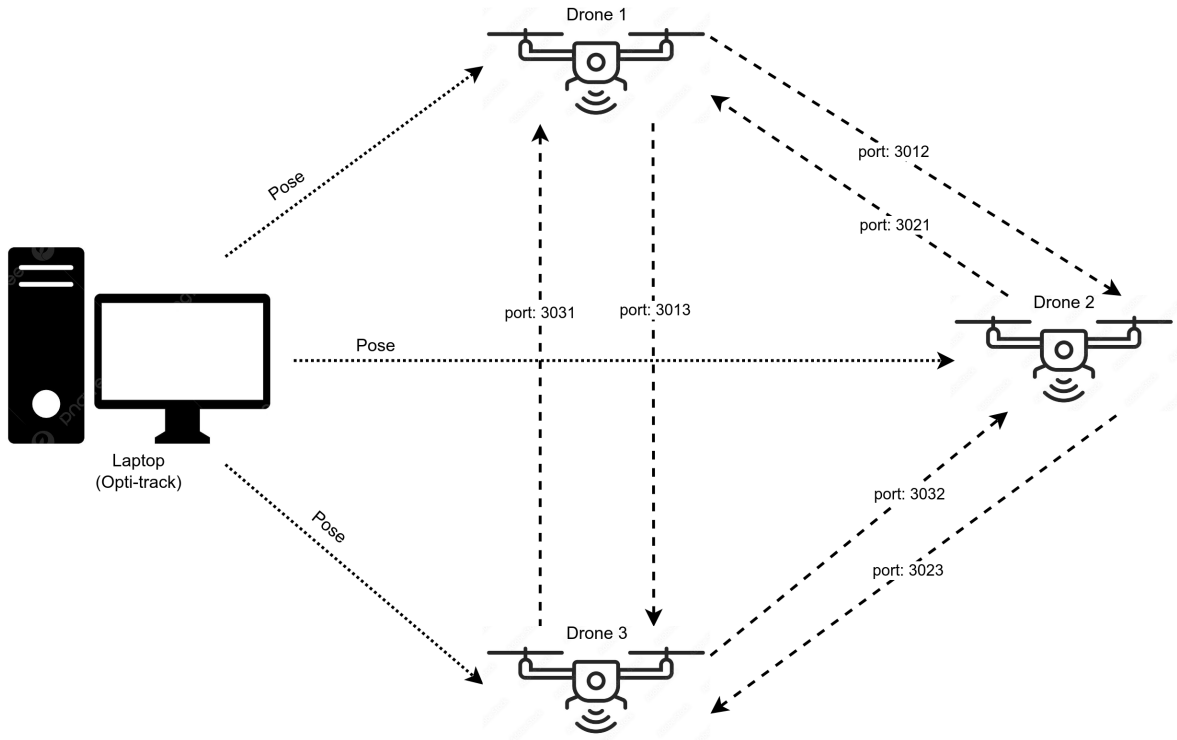


Figure 6-9: Communication experiment setting under ZeroMQ-based network

[5] https://zeromq.org/

6-3-2 Verification

Experimental Setup

The compare the performance of the ROS 1 XML-RPC middleware and the ZeroMQ-based custom middleware in real-world scenarios, we conduct a simple experiment to show the latency of both methods. We use three drones with the same hardware configuration mentioned in Section 6-1, and build a simple communication network under a 5GHz wireless network. The architecture of this network can be seen in Figure 6-10. Each drone is equipped with a ROS talker node and a ROS listener node. The talker node publishes a text message with a local timestamp t_0 to the listener node, and the listener node sends back the message immediately after receiving it. Figure 6-10 illustrates the experiment setup. The message is sent at 100 Hz with a bandwidth of 2 KB/s which is similar to the bandwidth of the trajectory message in our system. When the talker node receives the message, it will record the timestamp it receives the returned message as t_2. The local communication latency is calculated by half of the time difference between the received timestamps and the initial timestamp recorded by the talker node, i.e.,

$$\Delta t = \frac{t_2 - t_0}{2} \tag{6-1}$$

The reason is due to the time synchronization problem between the two machines. Accurate time synchronization between both machines is hard to achieve due to the communication delay and the clock drift. Therefore, the time delay was recorded on the talker machine but only one-way communication latency is calculated. This experiment is performed in 30 seconds and the average, maximal, and minimal latency are recorded.

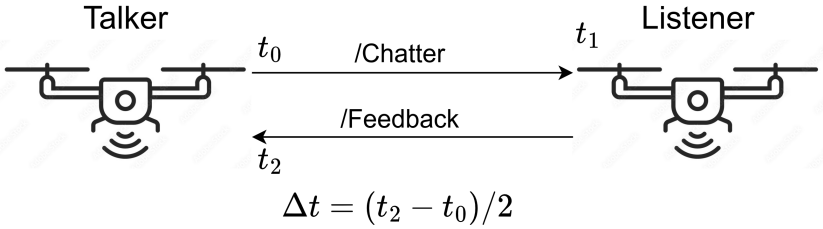


Figure 6-10: Illustration of communication experiment

Experiment Results

Figures 6-11 and 6-12 show the comparison of communication latency between ROS 1 XML-RPC middleware and the custom middleware based on ZeroMQ. The label *communication13* in both figures indicates the communication between drone 1 and drone 3 where drone 1 is the talker and drone 3 is the listener. The changes in communication latency over time for each connection are depicted in the figure, represented by accompanying error bars. According to the results, the ZeroMQ-based middleware achieves a latency of up to 200 ms. This latency is acceptable, as the trajectory plans

every 0.1 seconds with a planning horizon of 2 seconds. However, the vanilla ROS 1 XML-RPC communication setup produces increased latency. It starts with a latency of 200ms but increases to 10s in the end. The reason is obvious: ROS 1 XML-RPC middleware cannot handle such a large amount of messages in a slow network with limited bandwidth. Messages cannot be sent and received in time, so they were queued in a buffer. Latency increases as the buffer size increases.

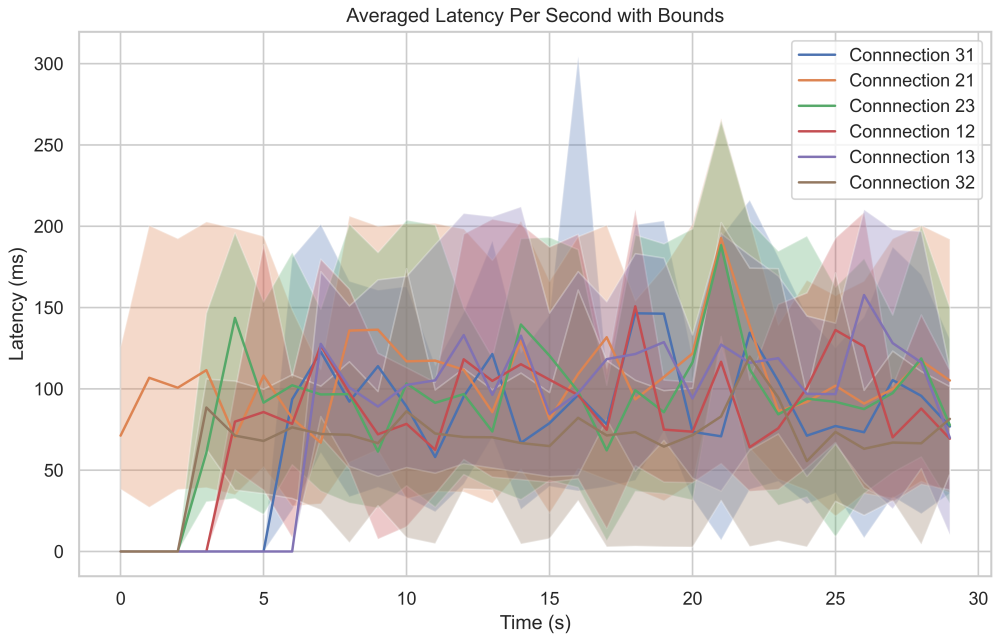


Figure 6-11: Average communication latency under ZeroMQ-based custom middleware network

To further test the robustness of the ZeroMQ-based middleware, we perform a stress test which is illustrated in Figure 6-13 and 6-14. The test aims to discover the influence of other running ROS nodes on the communication latency and to verify the On each drone, we started ROS nodes of mapping and planning modules, as well as the talker and listener nodes mentioned above. The mapping and planning modules will occupy CPU resources and onboard communication bandwidth, which may affect the multi-MAV communication latency.

According to the results, the communication latency of the ZeroMQ-based middleware increases but remains stable. The average communication latency between drone 1 and drone 2 increases to 150 ms. However, the ROS 1 XML-RPC middleware failed to handle the test. Similarly to previous results, it cannot deal with communication and the buffer size of waiting messages increases over time.

6-3-3 Discussion

Compared to the ROS 1 XML-RPC middleware, ZeroMQ-based custom middleware offers the following advantages:

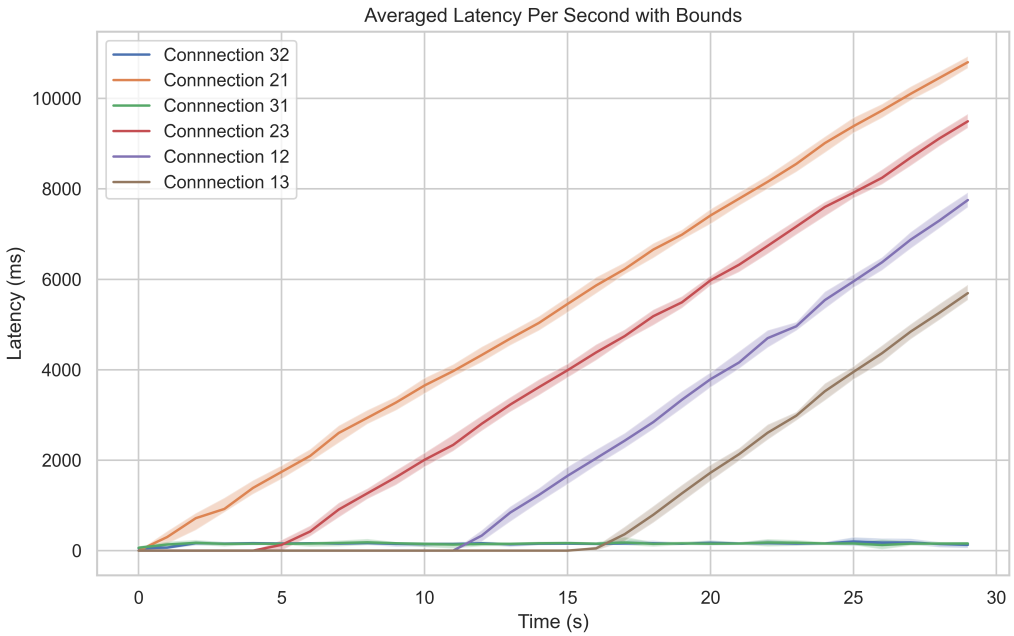


Figure 6-12: Average communication latency under ROS 1 XML-RPC middleware network

1. **Robustness**: It eliminates the need to start a master ROS node as a base station. Each robot can start autonomously and connect in random order.
2. **Flexibility**: Users can send and receive specific ROS topics instead of transmitting all topics to the master machine, as required in ROS 1.
3. **Usability**: All IP configurations and ROS topics can be specified together, which avoids setting up the hostname and ROS master URI of each robot thus simplifying the setup process.

Additionally, this custom middleware is more reliable than ROS 2 DDS communication, since it utilizes socket communication based on the TCP protocol, which has a better performance in wireless communication. Based on the above experimental results, our system will use the ZeroMQ-based custom middleware for multi-robot communication.

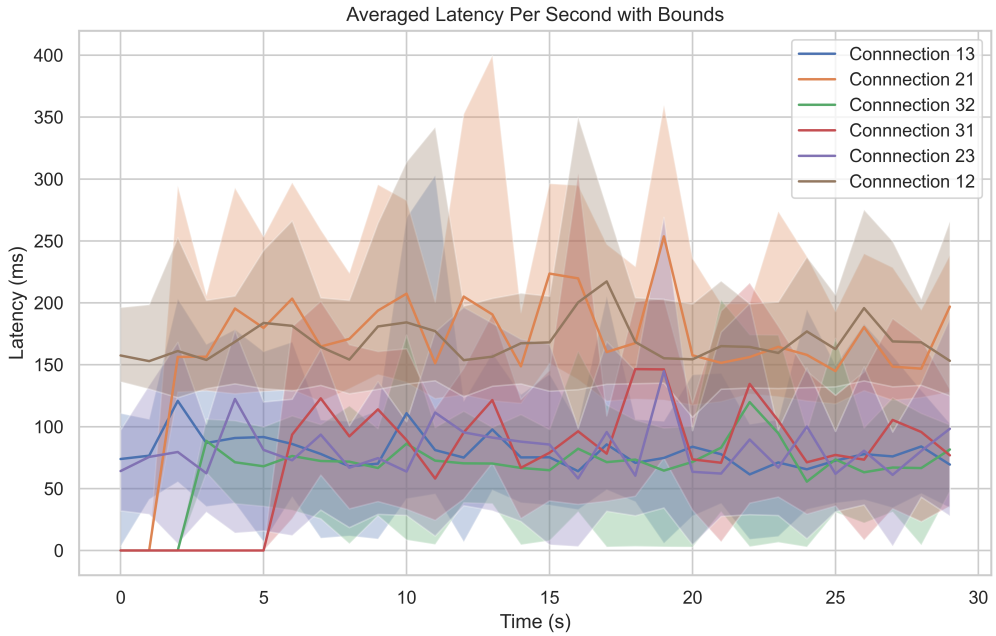


Figure 6-13: Average communication latency under ZeroMQ-based custom middleware network in the stress test

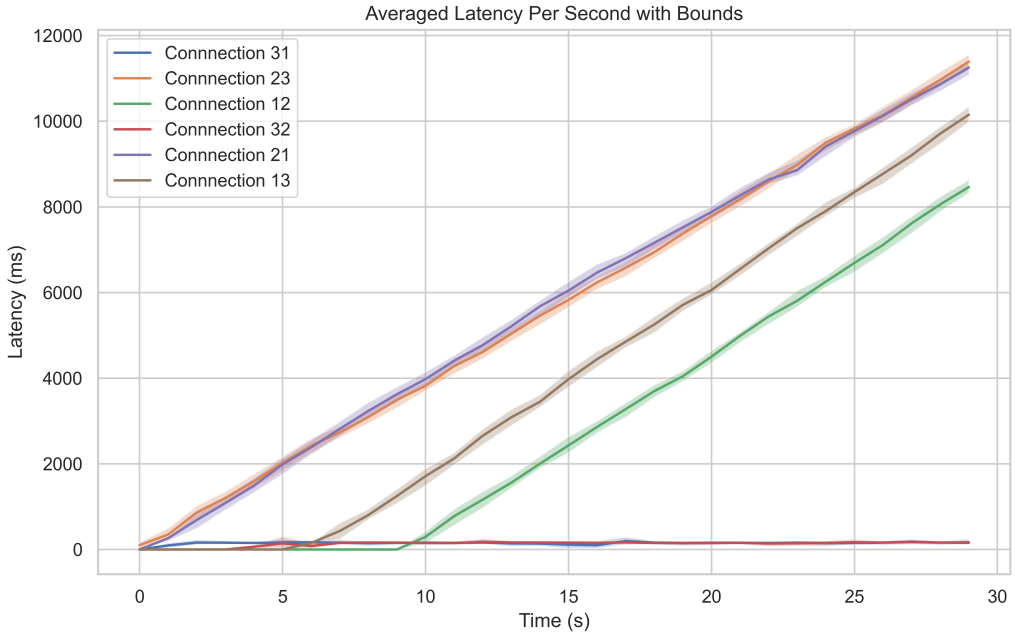


Figure 6-14: Average communication latency under ROS 1 XML-RPC middleware network in the stress test

Chapter 7

Conclusion

7-1 Summary

In this thesis, a risk-aware decentralized planning algorithm is proposed for multiple micro-aerial vehicles in unknown and dynamic environments, based on the particle-based map representation which predicts and evaluates future collision risks. Using this map representation, the proposed planning method seamlessly integrates both robot coordination and dynamic obstacle avoidance. Dynamically feasible trajectories are planned in real-time without any prior information about the dynamic environment.

In the experiment section, we first compare the proposed method with two state-of-the-art works. To this end, simulation environments with randomly generated dynamic obstacles are built. Given that the methods being compared concentrate solely on planning without a perception front-end, the experiments are carried out with ground truth local trajectories of dynamic obstacles known to every planner. The result shows that our method achieves competing results with the baselines in dense environments with cylindrical obstacles, and outperforms the baselines in a more complex environment with both cylindrical and circular obstacles, in terms of the success rate and trajectory time. Moreover, by classifying the failure cases into collision and deadlock, we find that our method has a lower collision rate compared to the baselines, and the majority of failures are due to deadlock cases. Our method outperforms MADER with a 98.5% higher success rate and a 21.9% reduction in flight time in the complex environment. Compared to EGO-Swarm, we see an 88.2% decrease in the collision rate and a 13.6% increase in the success rate. This result indicates that the proposed method while being conservative, ensures safety against potential collisions, making it well-suited for real-world applications with safety-critical requirements.

To understand how the proposed method performs integrating with perception modules, simulation experiments in a more realistic environment with pedestrians are conducted with the onboard perception. The results show that the proposed method is able

to handle collision avoidance with pedestrians while coordinating with other robots. However, the performance of the proposed method is constrained by the FOV of the onboard depth camera, which affects successful avoidance in environments with high pedestrian density.

7-2 Future Works

The proposed method can be improved in the future in several aspects.

During the experiment, a major problem of the proposed method is the occurrence of deadlocks. This can be attributed to the limitations of the risk-aware spatial-temporal kinodynamic path-searching algorithm, which searches for motion primitives that can minimize the control effort and total time. The heuristic function, based on Pontryagin's minimum principle, typically identifies the shortest path to the goal. The algorithm has a tendency to choose primitive paths that are straight to the goal, which is proven to be effective in static environments. However, in dynamic environments where the obstacles are moving, the shortest path normally is not the optimal path. This greedy strategy makes the robot prone to deadlock situations. To solve this problem, the cost and heuristic function of the path-searching algorithm can be modified to consider the blocking risk of the primitive paths. By leveraging this information, I believe the deadlock cases can be reduced.

The second observation from the experiments is that the planner does not effectively utilize available temporary free spaces for faster flight. One reason for this is that the feasible region for trajectory optimization is constrained by the corridor's size, which in turn is determined by the path found using kinodynamic A*. The searched path is not time optimal, and the planner design cannot effectively optimize the trajectory in terms of time. To solve this problem, the corridor generation process can be enhanced by factoring in the timing of the particles on the map. However, introducing such improvements would necessitate the addition of spatial-temporal constraints to trajectory optimization, which is still an open question in this area.

Another potential improvement involves accounting for communication delays and failures when mapping multi-robot trajectories. Currently, the proposed method assumes that communication is perfect without delay or failure, which limits the application in real-world scenarios. The quality of trajectory mapping could be enhanced by incorporating models of potential communication delays and necessary replanning into the risk function. By considering this risk in the map, robots can plan safer trajectories in advance to avoid potential collisions with other robots caused by communication failures. Future work will emphasize improving communication robustness, paving the way for practical demonstrations of the proposed method in real-world settings.

Appendix A

Additional Results

A-1 Performance of Different Planners in the Cylindrical Obstacle Environment

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.83	0.00	0.18	9.33	19.03	12.24±2.01	0.27	0.49
	MADER	0.76	0.16	0.08	9.34	21.76	12.42±2.76	0.38	0.90
	Proposed	0.83	0.05	0.13	16.08	26.86	20.79±2.13	0.23	1.07
Unilateral Swap	EGO-Swarm	0.89	0.00	0.11	9.76	18.50	11.99±1.66	0.28	0.96
	MADER	0.74	0.19	0.08	9.30	23.72	12.33±2.89	0.39	1.10
	Proposed	0.78	0.08	0.15	3.76	28.76	20.07±3.93	0.25	1.36
Cross Swap	EGO-Swarm	0.86	0.00	0.14	8.70	24.37	11.96±2.38	0.28	0.47
	MADER	0.81	0.14	0.05	9.62	26.00	12.92±3.04	0.41	0.88
	Proposed	0.83	0.06	0.11	5.65	30.06	19.98±3.37	0.23	1.02

Table A-1: Comparison of the performance of different planners across different tasks in the cylindrical obstacle environment with 20 obstacles.

A-2 Performance of Different Planners in the Complex Environment

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.79	0.00	0.21	9.19	18.00	12.43±1.96	0.24	0.48
	MADER	0.60	0.35	0.05	9.36	30.20	15.04±4.61	0.29	0.93
	Proposed	0.71	0.06	0.23	4.56	30.52	22.23±3.81	0.19	1.11
Unilateral Swap	EGO-Swarm	0.76	0.00	0.24	7.50	29.10	12.92±3.27	0.23	0.99
	MADER	0.55	0.40	0.05	9.49	29.96	15.28±4.16	0.22	1.21
	Proposed	0.71	0.10	0.19	5.85	43.42	23.19±4.95	0.19	1.37
Cross Swap	EGO-Swarm	0.81	0.00	0.19	9.07	23.46	12.50±2.23	0.23	0.53
	MADER	0.66	0.31	0.03	9.50	29.06	15.22±4.35	0.27	0.94
	Proposed	0.74	0.09	0.17	13.13	36.04	21.70±3.72	0.17	1.23

Table A-2: Comparison of the performance of different planners across different tasks in the cylindrical obstacle environment with 30 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.78	0.00	0.23	7.48	19.74	12.86±2.59	0.18	0.50
	MADER	0.38	0.55	0.08	9.24	31.98	18.27±5.72	0.16	1.00
	Proposed	0.45	0.24	0.31	17.74	37.58	24.89±5.05	0.10	1.16
Unilateral Swap	EGO-Swarm	0.85	0.00	0.15	8.93	30.23	13.59±2.94	0.20	1.16
	MADER	0.38	0.51	0.11	10.50	40.24	19.28±6.17	0.12	1.25
	Proposed	0.54	0.15	0.31	5.99	40.36	24.00±5.05	0.12	1.23
Cross Swap	EGO-Swarm	0.74	0.00	0.26	8.80	26.48	12.71±2.79	0.19	0.50
	MADER	0.49	0.48	0.04	9.90	36.56	18.86±5.79	0.18	0.99
	Proposed	0.48	0.24	0.29	3.46	40.18	24.09±5.37	0.10	1.18

Table A-3: Comparison of the performance of different planners across different tasks in the cylindrical obstacle environment with 40 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.63	0.00	0.38	9.24	22.33	13.36±2.36	0.14	0.48
	MADER	0.20	0.76	0.04	11.00	47.34	26.09±8.89	0.06	0.96
	Proposed	0.36	0.29	0.35	3.36	47.88	26.72±6.47	0.06	1.19
Unilateral Swap	EGO-Swarm	0.71	0.00	0.29	8.93	21.44	13.59±2.71	0.16	1.15
	MADER	0.24	0.70	0.06	10.36	46.64	25.36±9.39	0.08	1.18
	Proposed	0.40	0.30	0.30	4.84	48.16	27.42±7.08	0.08	1.40
Cross Swap	EGO-Swarm	0.63	0.00	0.38	8.74	19.65	12.89±2.18	0.15	0.52
	MADER	0.36	0.63	0.01	11.19	47.40	24.08±8.56	0.10	1.10
	Proposed	0.44	0.24	0.33	16.18	41.38	25.24±5.13	0.06	1.24

Table A-4: Comparison of the performance of different planners across different tasks in the cylindrical obstacle environment with 50 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.93	0.00	0.07	8.80	25.26	12.87±2.621	0.29	0.47
	MADER	0.94	0.06	0.00	4.84	26.82	13.14±4.256	0.46	0.86
	Proposed	0.99	0.01	0.00	13.94	20.10	15.55±1.218	0.30	0.88
Unilateral Swap	EGO-Swarm	0.93	0.00	0.07	9.36	27.88	13.52±3.471	0.37	1.14
	MADER	0.98	0.00	0.03	9.26	27.98	12.88±4.146	0.47	1.26
	Proposed	0.85	0.10	0.05	13.48	25.60	16.48±2.1	0.27	0.99
Cross Swap	EGO-Swarm	0.95	0.00	0.05	8.20	23.10	11.66±2.358	0.47	0.51
	MADER	0.98	0.01	0.01	9.42	27.48	13.12±4.303	0.52	0.93
	Proposed	0.94	0.04	0.03	13.64	21.02	15.58±1.231	0.49	0.87

Table A-5: Comparison of the performance of different planners across different tasks in the complex environment with 10 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.77	0.00	0.23	8.79	30.15	14.36±3.364	0.19	0.48
	MADER	0.78	0.23	0.00	9.42	44.68	18.51±6.985	0.21	1.02
	Proposed	0.85	0.10	0.05	13.06	21.98	16.05±1.888	0.17	0.90
Unilateral Swap	EGO-Swarm	0.78	0.02	0.19	4.32	34.99	14.7±4.889	0.17	1.13
	MADER	0.75	0.20	0.05	9.44	42.54	19.06±6.929	0.18	1.53
	Proposed	0.90	0.10	0.00	12.26	23.30	17.02±2.077	0.19	1.06
Cross Swap	EGO-Swarm	0.87	0.00	0.13	8.83	33.82	13.67±4.323	0.22	0.53
	MADER	0.84	0.16	0.00	9.58	39.36	18.51±7.147	0.22	0.96
	Proposed	0.79	0.21	0.00	13.76	23.26	16.93±2.368	0.13	1.00

Table A-6: Comparison of the performance of different planners across different tasks in the complex environment with 20 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.69	0.00	0.31	7.56	38.11	14.84±4.42	0.12	0.51
	MADER	0.51	0.49	0.00	10.90	48.46	26.32±9.571	0.11	1.14
	Proposed	0.63	0.38	0.00	8.14	25.38	17.44±3.004	0.08	1.07
Unilateral Swap	EGO-Swarm	0.73	0.01	0.26	9.67	31.41	16.47±4.378	0.14	1.09
	MADER	0.50	0.43	0.08	12.16	49.96	27.38±9.724	0.11	1.18
	Proposed	0.65	0.35	0.00	14.40	27.10	18.75±2.604	0.11	1.27
Cross Swap	EGO-Swarm	0.69	0.00	0.31	8.69	30.18	14.45±3.739	0.14	0.55
	MADER	0.55	0.39	0.06	5.66	49.18	26.15±10.086	0.10	1.15
	Proposed	0.70	0.24	0.06	13.06	27.82	17.26±2.92	0.13	0.97

Table A-7: Comparison of the performance of different planners across different tasks in the complex environment with 30 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.50	0.00	0.50	7.56	32.94	15.77±4.572	0.06	0.58
	MADER	0.29	0.70	0.01	12.62	50.63	37.27±10.095	0.04	1.05
	Proposed	0.50	0.50	0.00	13.72	40.24	20.74±5.348	0.06	1.71
Unilateral Swap	EGO-Swarm	0.59	0.00	0.41	10.79	35.23	17.51±4.619	0.09	0.86
	MADER	0.28	0.68	0.04	8.66	50.20	35.01±11.873	0.03	1.03
	Proposed	0.71	0.29	0.00	8.66	36.11	19.27±3.884	0.10	1.24
Cross Swap	EGO-Swarm	0.45	0.00	0.55	8.98	26.38	15.37±3.958	0.06	0.59
	MADER	0.31	0.69	0.00	7.40	49.66	34.96±11.514	0.04	0.91
	Proposed	0.65	0.35	0.00	13.44	29.28	19.5±4.255	0.07	1.23

Table A-8: Comparison of the performance of different planners across different tasks in the complex environment with 40 obstacles.

Task	Planner	Succ. rate	Fail. rate		Trajectory time (s)			Avg. min. dist (m)	
			Deadlock rate	Coll. rate	Min.	Max.	Avg.	obs.	MAV
Bilateral Swap	EGO-Swarm	0.38	0.00	0.63	9.51	29.94	16.81±4.662	0.04	0.55
	MADER	0.17	0.82	0.01	12.86	50.90	36.7±11.52	0.02	1.12
	Proposed	0.55	0.43	0.03	7.98	133.04	21.95±13.532	0.07	1.33
Unilateral Swap	EGO-Swarm	0.39	0.00	0.61	9.68	34.00	17.26±4.993	0.04	1.09
	MADER	0.15	0.83	0.03	11.44	50.50	36.6±10.724	0.02	1.27
	Proposed	0.59	0.40	0.01	8.33	45.98	21.99±6.312	0.07	1.54
Cross Swap	EGO-Swarm	0.41	0.00	0.59	9.40	27.28	15.19±3.753	0.05	0.53
	MADER	0.08	0.93	0.00	9.62	50.94	40.96±10.345	0.01	1.29
	Proposed	0.63	0.38	0.00	12.84	41.86	21.66±5.28	0.07	1.23

Table A-9: Comparison of the performance of different planners across different tasks in the complex environment with 50 obstacles.

Bibliography

[1] K. N. McGuire et al. "Minimal Navigation Solution for a Swarm of Tiny Flying Robots to Explore an Unknown Environment". en. In: *Sci. Robot.* 4.35 (Oct. 30, 2019), eaaw9710. ISSN: 2470-9476. DOI: 10.1126/scirobotics.aaw9710. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aaw9710. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aaw9710 (visited on 08/13/2022).

[2] Nathan Michael, Jonathan Fink, and Vijay Kumar. "Cooperative manipulation and transportation with aerial robots". In: *Autonomous Robots* 30 (2011), pp. 73–86.

[3] T. Nägeli et al. "Real-Time Motion Planning for Aerial Videography With Dynamic Obstacle Avoidance and Viewpoint Optimization". en. In: *IEEE Robot. Autom. Lett.* 2.3 (July 2017), pp. 1696–1703. ISSN: 2377-3766. DOI: 10.1109/lra.2017.2665693. (Visited on 05/06/2021).

[4] Feng Chen et al. "PredRecon: A Prediction-boosted Planning Framework for Fast and High-quality Autonomous Aerial Reconstruction". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023. DOI: 10.48550/arXiv.2302.04488.

[5] Xin Zhou et al. "EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors". In: *IEEE Robot. Autom. Lett.* 6.2 (Apr. 2021), pp. 478–485. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/lra.2020.3047728. URL: https://doi.org/10.1109/LRA.2020.3047728 (visited on 08/20/2022).

[6] Xin Zhou et al. "Swarm of Micro Flying Robots in the Wild". en. In: *Sci. Robot.* 7.66 (May 11, 2022), eabm5954. ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm5954. (Visited on 05/19/2022).

[7] Boyu Zhou, Hao Xu, and Shaojie Shen. *RACER: Rapid Collaborative Exploration with a Decentralized Multi-UAV System*. Sept. 18, 2022. arXiv: 2209.08533 [cs]. URL: http://arxiv.org/abs/2209.08533 (visited on 01/11/2023). preprint.

[8] Zhefan Xu et al. "DPMPC-Planner: A Real-Time UAV Trajectory Planning Framework for Complex Static Environments with Dynamic Obstacles". In: *2022 Int. Conf. Robot. Autom. ICRA*. 2022 International Conference on Robotics and Automation (ICRA). May 2022, pp. 250–256. DOI: 10.1109/icra46639.2022.9811886.

[9] Zhefan Xu et al. *Vision-Aided UAV Navigation and Dynamic Obstacle Avoidance Using Gradient-based B-spline Trajectory Optimization*. Sept. 14, 2022. arXiv: 2209.07003 [cs]. URL: http://arxiv.org/abs/2209.07003 (visited on 09/17/2022). preprint.

[10] Gang Chen et al. *Continuous Occupancy Mapping in Dynamic Environments Using Particles*. Feb. 13, 2022. DOI: 10.48550/arXiv.2202.06273. arXiv: 2202.06273 [cs]. (Visited on 08/17/2022). preprint.

[11] Glenn Wagner and Howie Choset. "M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds". In: *2011 IEEERSJ Int. Conf. Intell. Robots Syst.* 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. Sept. 2011, pp. 3260–3267. DOI: 10.1109/iros.2011.6095022.

[12] Guni Sharon et al. "Conflict-Based Search for Optimal Multi-Agent Pathfinding". In: *Artif. Intell.* 219 (2015), pp. 40–66. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.11.006.

[13] Anna Mannucci, Lucia Pallottino, and Federico Pecora. "On Provably Safe and Live Multirobot Coordination With Online Goal Posting". In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1973–1991. DOI: 10.1109/tro.2021.3075371.

[14] Michal áp et al. "Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots". In: *IEEE Transactions on Automation Science and Engineering* 12.3 (2015), pp. 835–849. DOI: 10.1109/tase.2015.2445780.

[15] Anna Mannucci, Lucia Pallottino, and Federico Pecora. "Provably Safe Multi-Robot Coordination With Unreliable Communication". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3232–3239. DOI: 10.1109/lra.2019.2924849.

[16] Soon-Jo Chung et al. "A Survey on Aerial Swarm Robotics". en. In: *IEEE Trans. Robot.* 34.4 (Aug. 2018), pp. 837–855. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/tro.2018.2857475. (Visited on 06/15/2021).

[17] Herbert G Tanner and Amit Kumar. "Formation stabilization of multiple agents using decentralized navigation functions." In: *Robotics: Science and systems*. Vol. 1. Boston. 2005, pp. 49–56.

[18] Tamás Vicsek et al. "Novel type of phase transition in a system of self-driven particles". In: *Physical review letters* 75.6 (1995), p. 1226.

[19] Paolo Fiorini and Zvi Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". en. In: *The International Journal of Robotics Research* 17.7 (July 1998), pp. 760–772. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836499801700706. (Visited on 11/21/2022).

[20] Daniel Claes et al. "Collision Avoidance under Bounded Localization Uncertainty". In: *2012 IEEERSJ Int. Conf. Intell. Robots Syst.* 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. Oct. 2012, pp. 1192–1198. DOI: 10.1109/iros.2012.6386125.

[21] J. Alonso-Mora, P. Beardsley, and R. Siegwart. "Cooperative Collision Avoidance for Nonholonomic Robots". In: *IEEE Trans. Robot.* 34.2 (Apr. 2018), pp. 404–420. ISSN: 1941-0468. DOI: 10.1109/tro.2018.2793890.

[22] Javier Alonso-Mora et al. "Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots". en. In: *Distributed Autonomous Robotic Systems*. Ed. by Alcherio Martinoli et al. Vol. 83. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 203–216. ISBN: 978-3-642-32722-3 978-3-642-32723-0. DOI: 10.1007/978-3-642-32723-0_15. (Visited on 11/21/2022).

[23] Wenhao Luo, Wen Sun, and Ashish Kapoor. "Multi-robot collision avoidance under uncertainty with probabilistic safety barrier certificates". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 372–383.

[24] Dingjiang Zhou et al. "Fast, On-line Collision Avoidance for Dynamic Vehicles Using Buffered Voronoi Cells". en. In: *IEEE Robot. Autom. Lett.* 2.2 (Apr. 2017), pp. 1047–1054. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/lra.2017.2656241. (Visited on 04/16/2021).

[25] Hai Zhu and Javier Alonso-Mora. "B-UAVC: Buffered Uncertainty-Aware Voronoi Cells for Probabilistic Multi-Robot Collision Avoidance". In: *2019 Int. Symp. Multi-Robot Multi-Agent Syst. MRS*. 2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). Aug. 2019, pp. 162–168. DOI: 10.1109/mrs.2019.8901092.

[26] Yufan Chen, Mark Cutler, and Jonathan P. How. "Decoupled Multiagent Path Planning via Incremental Sequential Convex Programming". In: *2015 IEEE Int. Conf. Robot. Autom. ICRA*. 2015 IEEE International Conference on Robotics and Automation (ICRA). May 2015, pp. 5954–5961. DOI: 10.1109/icra.2015.7140034.

[27] Daniel Morgan et al. "Swarm Assignment and Trajectory Optimization Using Variable-Swarm, Distributed Auction Assignment and Sequential Convex Programming". en. In: *The International Journal of Robotics Research* 35.10 (Sept. 1, 2016), pp. 1261–1285. ISSN: 0278-3649. DOI: 10.1177/0278364916632065. (Visited on 08/20/2022).

[28] Mina Kamel et al. "Robust Collision Avoidance for Multiple Micro Aerial Vehicles Using Nonlinear Model Predictive Control". In: *2017 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sept. 2017, pp. 236–243. DOI: 10.1109/iros.2017.8202163.

[29] Carlos E. Luis, Marijan Vukosavljev, and Angela P. Schoellig. "Online Trajectory Generation With Distributed Model Predictive Control for Multi-Robot Motion Planning". In: *IEEE Robot. Autom. Lett.* 5.2 (Apr. 2020), pp. 604–611. ISSN: 2377-3766. DOI: 10.1109/lra.2020.2964159.

[30] Jiahao Lin, Hai Zhu, and Javier Alonso-Mora. "Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments". In: *2020 IEEE Int. Conf. Robot. Autom. ICRA*. 2020 IEEE International Conference on Robotics and Automation (ICRA). 2020, pp. 2682–2688. DOI: 10.1109/icra40945.2020.9197481.

[31] Yingjian Wang et al. "Autonomous Flights in Dynamic Environments with Onboard Vision". In: *2021 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sept. 2021, pp. 1966–1973. DOI: 10.1109/iros51168.2021.9636117.

[32] Minghao Lu, Han Chen, and Peng Lu. "Perception and Avoidance of Multiple Small Fast Moving Objects for Quadrotors With Only Low-Cost RGBD Camera". In: *IEEE Robotics and Automation Letters* 7.4 (Oct. 2022), pp. 11657–11664. ISSN: 2377-3766. DOI: 10.1109/lra.2022.3205114.

[33] Thomas Eppenberger et al. "Leveraging Stereo-Camera Data for Real-Time Dynamic Obstacle Detection and Tracking". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2020, pp. 10528–10535. DOI: 10.1109/iros45743.2020.9340699.

[34] Zhefan Xu et al. *Onboard Dynamic-Object Detection and Tracking for Autonomous Robot Navigation with RGB-D Camera*. en. Feb. 28, 2023. arXiv: 2303.00132 [cs]. URL: http://arxiv.org/abs/2303.00132 (visited on 06/18/2023). preprint.

[35] Xin Zhou et al. "EGO-Swarm: A Fully Autonomous and Decentralized Quadrotor Swarm System in Cluttered Environments". In: *2021 IEEE Int. Conf. Robot. Autom. ICRA*. 2021 IEEE International Conference on Robotics and Automation (ICRA). May 2021, pp. 4101–4107. DOI: 10.1109/icra48506.2021.9561902.

[36] Jialiang Hou et al. *Enhanced Decentralized Autonomous Aerial Swarm with Group Planning*. Mar. 2, 2022. arXiv: 2203.01069 [cs]. URL: http://arxiv.org/abs/2203.01069 (visited on 06/29/2022). preprint.

[37] Armin Hornung et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous robots* 34.3 (2013), pp. 189–206.

[38] Steven M. LaValle. *Planning Algorithms*. en. Cambridge: Cambridge University Press, 2006. ISBN: 978-0-511-54687-7 978-0-521-86205-9. DOI: 10.1017/cbo9780511546877. (Visited on 07/12/2021).

[39] Hai Zhu and Javier Alonso-Mora. "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments". In: *IEEE Robot. Autom. Lett.* 4.2 (Apr. 2019), pp. 776–783. ISSN: 2377-3766. DOI: 10.1109/lra.2019.2893494.

[40] Jesus Tordesillas and Jonathan P. How. "MADER: Trajectory Planner in Multiagent and Dynamic Environments". In: *Tro* 38.1 (Feb. 2022), pp. 463–476. ISSN: 1941-0468. DOI: 10.1109/tro.2021.3080235.

[41] Fei Gao and Shaojie Shen. "Quadrotor Trajectory Generation in Dynamic Environments Using Semi-Definite Relaxation on Nonconvex QCQP". In: *2017 IEEE Int. Conf. Robot. Autom. ICRA*. 2017 IEEE International Conference on Robotics and Automation (ICRA). May 2017, pp. 6354–6361. DOI: 10.1109/icra.2017.7989750.

[42] Botao He et al. "FAST-Dynamic-Vision: Detection and Tracking Dynamic Objects with Event and Depth Sensing". In: *2021 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sept. 2021, pp. 3071–3078. DOI: 10.1109/iros51168.2021.9636448.

[43] Han Chen and Peng Lu. "Real-Time Identification and Avoidance of Simultaneous Static and Dynamic Obstacles on Point Cloud for UAVs Navigation". en. In: *Robotics and Autonomous Systems* 154 (Aug. 1, 2022), p. 104124. ISSN: 0921-8890. DOI: 10.1016/j.robot.2022.104124. (Visited on 02/23/2023).

[44] Georg Tanzmeister and Dirk Wollherr. "Evidential Grid-Based Tracking and Mapping". In: *IEEE Transactions on Intelligent Transportation Systems* 18.6 (2017), pp. 1454–1467. DOI: 10.1109/tits.2016.2608919.

[45] Vitor Guizilini, Ransalu Senanayake, and Fabio Ramos. "Dynamic Hilbert roMaps: Real-Time Occupancy Predictions in Changing Environments". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019 International Conference on Robotics and Automation (ICRA). May 2019, pp. 4091–4097. DOI: 10.1109/icra.2019.8793914.

[46] Khushdeep Singh Mann et al. *Predicting Future Occupancy Grids in Dynamic Environment with Spatio-Temporal Learning*. May 6, 2022. DOI: 10.48550/arXiv.2205.03212. arXiv: 2205.03212 [cs]. (Visited on 02/20/2023). preprint.

[47] Bernard Lange, Masha Itkina, and Mykel J. Kochenderfer. *LOPR: Latent Occupancy PRediction Using Generative Models*. Oct. 3, 2022. DOI: 10.48550/arXiv.2210.01249. arXiv: 2210.01249 [cs]. (Visited on 02/20/2023). preprint.

[48] Gang Chen et al. "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.

[49] Sikang Liu et al. "Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments". In: *Ral* 2.3 (July 2017), pp. 1688–1695. ISSN: 2377-3766. DOI: 10.1109/lra.2017.2663526.

[50] Helen Oleynikova et al. "Continuous-Time Trajectory Optimization for Online UAV Replanning". en. In: *2016 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Daejeon, South Korea: Ieee, Oct. 2016, pp. 5332–5339. ISBN: 978-1-5090-3762-9. DOI: 10.1109/iros.2016.7759784. (Visited on 11/30/2021).

[51] Boyu Zhou et al. "Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight". In: *Ral* 4.4 (Oct. 2019), pp. 3529–3536. ISSN: 2377-3766. DOI: 10.1109/lra.2019.2927938.

[52] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[53] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments". en. In: *Robotics Research*. Ed. by Masayuki Inaba and Peter Corke. Vol. 114. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2016, pp. 649–666. ISBN: 978-3-319-28870-3 978-3-319-28872-7. DOI: 10.1007/978-3-319-28872-7_37. (Visited on 11/30/2021).

[54] Dustin J Webb and Jur van den Berg. "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints". In: *arXiv preprint arXiv:1205.5088* (2012).

[55] Hongkai Ye et al. "TGK-Planner: An Efficient Topology Guided Kinodynamic Planner for Autonomous Quadrotors". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 494–501. DOI: 10.1109/lra.2020.3047798.

[56] Hongkai Ye et al. "Efficient Sampling-based Multirotors Kinodynamic Planning with Fast Regional Optimization and Post Refining". In: *2022 IEEE RSJ Int. Conf. Intell. Robots Syst. IROS*. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Oct. 2022, pp. 3356–3363. DOI: 10.1109/iros47612.2022.9981707.

[57] Brian MacAllister et al. “Path planning for non-circular micro aerial vehicles in constrained environments”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 3933–3940. DOI: 10.1109/icra.2013.6631131.

[58] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar. “Incremental micro-UAV motion replanning for exploring unknown environments”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 2452–2458. DOI: 10.1109/icra.2013.6630910.

[59] Brett T. Lopez and Jonathan P. How. “Aggressive 3-D Collision Avoidance for High-Speed Navigation”. In: *2017 IEEE Int. Conf. Robot. Autom. ICRA*. 2017 IEEE International Conference on Robotics and Automation (ICRA). Ieee. May 2017, pp. 5759–5765. DOI: 10.1109/icra.2017.7989677.

[60] Nathan Ratliff et al. “CHOMP: Gradient Optimization Techniques for Efficient Motion Planning”. In: *2009 IEEE Int. Conf. Robot. Autom.* 2009 IEEE International Conference on Robotics and Automation. May 2009, pp. 489–494. DOI: 10.1109/robot.2009.5152817.

[61] Helen Oleynikova et al. “Signed Distance Fields: A Natural Representation for Both Mapping and Planning”. en. In: *RSS 2016 Workshop: Geometry and Beyond - Representations, Physics, and Scene Understanding for Robotics*. RSS 2016 Workshop: Geometry and Beyond - Representations, Physics, and Scene Understanding for Robotics; Conference Location: Ann Arbor, MI, USA; Conference Date: June 19, 2016. Ann Arbor, MI: University of Michigan, 2016. DOI: 10.3929/ethz-a-010820134.

[62] Fei Gao et al. “Online Safe Trajectory Generation for Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial”. In: *2018 IEEE Int. Conf. Robot. Autom. ICRA*. 2018 IEEE International Conference on Robotics and Automation (ICRA). May 2018, pp. 344–351. DOI: 10.1109/icra.2018.8462878.

[63] Fei Gao, Yi Lin, and Shaojie Shen. “Gradient-based online safe trajectory generation for quadrotor flight in complex environments”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 3681–3688. DOI: 10.1109/iros.2017.8206214.

[64] Wenchao Ding et al. “An efficient b-spline-based kinodynamic replanning framework for quadrotors”. In: *IEEE Transactions on Robotics* 35.6 (2019), pp. 1287–1306.

[65] Fei Gao and Shaojie Shen. “Online Quadrotor Trajectory Generation and Autonomous Navigation on Point Clouds”. In: *2016 IEEE Int. Symp. Saf. Secur. Rescue Robot. SSRR*. 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). Oct. 2016, pp. 139–146. DOI: 10.1109/ssrr.2016.7784290.

[66] Yunfan Ren et al. “Bubble Planner: Planning High-speed Smooth Quadrotor Trajectories Using Receding Corridors”. In: *ArXiv220212177 Cs* (Feb. 24, 2022). arXiv: 2202.12177 [cs]. URL: http://arxiv.org/abs/2202.12177 (visited on 04/11/2022).

[67] Robin Deits and Russ Tedrake. “Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming”. In: *Algorithmic Foundations of Robotics XI*. Ed. by H. Levent Akin et al. Vol. 107. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2015, pp. 109–124. ISBN: 978-3-319-16594-3 978-3-319-16595-0. DOI: 10.1007/978-3-319-16595-0_7. (Visited on 11/17/2022).

[68] Jing Chen, Tianbo Liu, and Shaojie Shen. "Online Generation of Collision-Free Trajectories for Quadrotor Flight in Unknown Cluttered Environments". In: *2016 IEEE Int. Conf. Robot. Autom. ICRA*. 2016 IEEE International Conference on Robotics and Automation (ICRA). Ieee. Ieee, May 2016, pp. 1476–1483. DOI: 10.1109/icra.2016.7487283.

[69] Javier Alonso-Mora et al. "Distributed Multi-Robot Formation Control in Dynamic Environments". en. In: *Auton Robot* 43.5 (June 1, 2019), pp. 1079–1100. ISSN: 1573-7527. DOI: 10.1007/s10514-018-9783-9. (Visited on 04/21/2022).

[70] Javier Alonso-Mora et al. "Collision Avoidance for Aerial Vehicles in Multi-Agent Scenarios". en. In: *Auton Robot* 39.1 (June 2015), pp. 101–121. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-015-9429-0. (Visited on 08/20/2022).

[71] Zhepei Wang et al. "Geometrically Constrained Trajectory Optimization for Multicopters". In: *IEEE Trans. Robot.* (2022), pp. 1–10. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/tro.2022.3160022. (Visited on 07/07/2022).

[72] Xingguang Zhong et al. "Generating Large Convex Polytopes Directly on Point Clouds". In: *ArXiv201008744 Cs* (Nov. 16, 2020). arXiv: 2010.08744 [cs]. URL: http://arxiv.org/abs/2010.08744 (visited on 02/25/2021).

[73] Daniel Mellinger and Vijay Kumar. "Minimum Snap Trajectory Generation and Control for Quadrotors". In: *2011 IEEE Int. Conf. Robot. Autom.* 2011 IEEE International Conference on Robotics and Automation. Ieee, May 2011, pp. 2520–2525. DOI: 10.1109/icra.2011.5980409.

[74] Wenchao Ding et al. "An Efficient B-Spline-Based Kinodynamic Replanning Framework for Quadrotors". In: *IEEE Trans. Robot.* 35.6 (Dec. 2019), pp. 1287–1306. ISSN: 1941-0468. DOI: 10.1109/tro.2019.2926390.

[75] Kaihuai Qin. "General Matrix Representations for B-splines". In: *Proc. Pac. Graph. 98 Sixth Pac. Conf. Comput. Graph. Appl. Cat No98EX208*. Proceedings Pacific Graphics '98. Sixth Pacific Conference on Computer Graphics and Applications (Cat. No.98EX208). Oct. 1998, pp. 37–43. DOI: 10.1109/pccga.1998.731996.

[76] Jesus Tordesillas and Jonathan P. How. "PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments". en. In: *ArXiv210306372 Cs* (Mar. 10, 2021). arXiv: 2103.06372 [cs]. URL: http://arxiv.org/abs/2103.06372 (visited on 05/06/2021).

[77] Kota Kondo et al. *Robust MADER: Decentralized and Asynchronous Multiagent Trajectory Planner Robust to Communication Delay*. Oct. 26, 2022. arXiv: 2209.13667 [cs, eess]. URL: http://arxiv.org/abs/2209.13667 (visited on 12/17/2022). preprint.

[78] "Fast-Racing: An Open-Source Strong Baseline for $\mathrm{SE}(3)$ Planning in Autonomous Drone Racing". In: ().

[79] Zhepei Wang et al. "Generating Large-Scale Trajectories Efficiently Using Double Descriptions of Polynomials". In: *2021 IEEE Int. Conf. Robot. Autom. ICRA*. 2021 IEEE International Conference on Robotics and Automation (ICRA). Xi'an, China: Ieee, May 30, 2021, pp. 7436–7442. ISBN: 978-1-72819-077-8. DOI: 10.1109/icra48506.2021.9561585. (Visited on 01/27/2022).

[80] Jungwon Park et al. "Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee Using Relative Bernstein Polynomial". In: *2020 IEEE Int. Conf. Robot. Autom. ICRA*. 2020 IEEE International Conference on Robotics and Automation (ICRA). May 2020, pp. 434–440. DOI: 10.1109/icra40945.2020.9197162.

[81] Jesus Tordesillas and Jonathan P. How. "MINVO Basis: Finding Simplexes with Minimum Volume Enclosing Polynomial Curves". In: *ArXiv201010726 Cs* (Sept. 15, 2021). arXiv: 2010.10726 [cs]. URL: http://arxiv.org/abs/2010.10726 (visited on 11/11/2021).

[82] Hao Xu et al. "Decentralized Visual-Inertial-UWB Fusion for Relative State Estimation of Aerial Swarm". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020 IEEE International Conference on Robotics and Automation (ICRA). Paris, France: Ieee, May 2020, pp. 8776–8782. ISBN: 978-1-72817-395-5. DOI: 10.1109/icra40945.2020.9196944.

[83] Vivek K. Adajania et al. "AMSwarm: An Alternating Minimization Approach for Safe Motion Planning of Quadrotor Swarms in Cluttered Environments". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1421–1427. DOI: 10.1109/ICRA48891.2023.10161063.

[84] Mark W. Mueller, Markus Hehn, and Raffaello D'Andrea. "A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation". In: *IEEE Transactions on Robotics* 31.6 (2015), pp. 1294–1310. DOI: 10.1109/TRO.2015.2479878.

[85] Bartolomeo Stellato et al. "OSQP: An Operator Splitting Solver for Quadratic Programs". en. In: *Math. Prog. Comp.* 12.4 (Dec. 2020), pp. 637–672. ISSN: 1867-2949, 1867-2957. DOI: 10.1007/s12532-020-00179-2. (Visited on 08/21/2022).

[86] Wojciech Giernacki et al. "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering". In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. Ieee. 2017, pp. 37–42.

[87] Inkyu Sa et al. "Build your own visual-inertial drone: A cost-effective and open-source autonomous drone". In: *IEEE Robotics & Automation Magazine* 25.1 (2017), pp. 89–103.

[88] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories". In: *IEEE Robot. Autom. Lett.* 3.2 (Apr. 2018), pp. 620–626. ISSN: 2377-3766. DOI: 10.1109/lra.2017.2776353.

[89] Tomas Baca et al. "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles". In: *Journal of Intelligent & Robotic Systems* 102.1 (2021), p. 26.

[90] Philipp Foehn et al. "Agilicious: Open-source and Open-Hardware Agile Quadrotor for Vision-Based Flight". en. In: *Sci. Robot.* 7.67 (June 22, 2022), eabl6259. ISSN: 2470-9476. DOI: 10.1126/scirobotics.abl6259. (Visited on 04/30/2023).

[91] Jiahao Lin. "Real-Time Vision-based Autonomous Navigation of MAV in Dynamic Environments". en. In: (2019). URL: https://repository.tudelft.nl/islandora/object/uuid%5C%3A91f200f7-4966-4504-bc83-5a87e5550a91 (visited on 08/17/2022).

[92] Tong Qin, Peiliang Li, and Shaojie Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Trans. Robot.* 34.4 (Aug. 2018), pp. 1004–1020. ISSN: 1941-0468. DOI: 10.1109/tro.2018.2853729.

[93] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *2014 IEEE international conference on robotics and automation (ICRA)*. Ieee. 2014, pp. 15–22.

[94] Tong Qin et al. *A General Optimization-based Framework for Global Pose Estimation with Multiple Sensors*. 2019. eprint: arXiv:1901.03642.

[95] Carlos Campos et al. "ORB-SLAM3: An Accurate Open-Source Library for Visual, VisualInertial, and Multimap SLAM". In: *IEEE Trans. Robot.* 37.6 (Dec. 2021), pp. 1874–1890. ISSN: 1941-0468. DOI: 10.1109/tro.2021.3075644.

[96] Ji Zhang and Sanjiv Singh. "LOAM: Lidar odometry and mapping in real-time." In: *Robotics: Science and Systems*. Vol. 2. 9. Berkeley, CA. 2014, pp. 1–9.

[97] Wei Xu et al. "Fast-lio2: Fast direct lidar-inertial odometry". In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073.

[98] Muhammad Fadhil Ginting et al. "CHORD: Distributed Data-Sharing via Hybrid ROS 1 and 2 for Multi-Robot Exploration of Large-Scale Complex Environments". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5064–5071. DOI: 10.1109/LRA.2021.3061393.

Glossary

List of Acronyms

MAV	micro aerial vehicles
MPC	model predictive control
QP	quadratic programming
LP	linear programming
SDP	Semidefinite programming
SOCP	second-order conic programming
VO	velocity obstacles
BVC	buffered voronoi cell
RFS	random finite set
PHD	probability hypothesis density
SMC	sequential Monte Carlo
DSP	dual-structured particle-based
RRT	rapidly exploring random tree
FOV	field of view
ESDF	Euclidean signed distance field
IRIS	Iterative Region Inflation by Semidefinite Programming
FIRI	Fast iterative region inflation
DOF	degree of freedom
ROS	Robot Operating System
IMU	Inertial Measurement Unit

List of Symbols

$\mathcal{P}$	safety corridor
$\mathbf{M}$	map
$\mathcal{B}_N^k(t)$	Bernstein basis
$\mathbf{x}_k(t)$	point object state used in the DSP map
$\mathbf{X}(t)$	random finite set of point objects
$\mathbf{D}_\mathbf{X}(t)$	probability hypothesis density of the random finite set
$w_t^{(\mathbf{x}(t))}$	weight of the particle $\mathbf{x}(t)$ at time t
$\mathbf{\Sigma}_p$	covariance matrix of the odometry
$\mathbb{E}_j$	region to be evaluated for collision risk
$\boldsymbol{x}(t)$	state of the MAV when searching for a kinodynamic path
$\boldsymbol{p}(t)$	trajectory
τ	time duration of the trajectory piece
ϵ	threshold for risk check
d	radius of the MAV when shrinking the corridor
$\mathcal{O}_{k,t_{i-1},t_i}$	obstacle set from time t_{i-1} to t_i.
$\boldsymbol{c}_j^i$	the i-th control point of the j-th trajectory segment