Trajectory Planning for Quadrotor Swarms

Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian

***Abstract*—We describe a method for multi-robot trajectory planning in known, obstacle-rich environments. We demonstrate our approach on a quadrotor swarm navigating in a warehouse setting. Our method consists of three stages: 1) roadmap generation which generates sparse roadmaps annotated with possible inter-robot collisions; 2) discrete planning which finds valid execution schedules in discrete time and space; and 3) continuous refinement that creates smooth trajectories. We account for the downwash effect of quadrotors, allowing safe flight in dense formations. We demonstrate computational efficiency in simulation with up to 200 robots and physical plausibility with an experiment on 32 nano-quadrotors. Our approach can compute safe and smooth trajectories for hundreds of quadrotors in dense environments with obstacles in a few minutes.**

***Index Terms*—UAV, quadrotor, swarm, multi-robot systems**

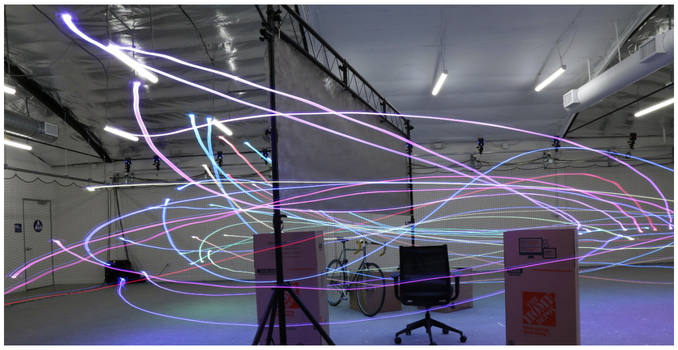


Fig. 1. Long exposure of 32 Crazyflie nano-quadrotors flying through an obstacle-rich environment.

I. Introduction

TRAJECTORY planning is a fundamental problem in multi-robot systems. Given a set of robots with known initial locations and a set of goal locations, the task is to find a set of continuous functions that move each robot from its start position to its goal, while avoiding collisions and respecting dynamic limits. The *unlabeled* case allows robots to swap goal locations with each other; in the *labeled* case the goal assignment is given. Trajectory planning is a core subproblem of various applications including search-and-rescue, inspection, and delivery.

A large body of work has addressed this problem with varied discrete and continuous formulations. However, no existing solution simultaneously satisfies the goals of completeness, physical plausibility, optimality in time or energy usage, and good computational performance. In this work, we present a method that attempts to balance these goals.

Our method uses a graph-based planner to compute a solution for a version of the problem that is discretized in space and time, and then refines this solution into smooth trajectories in a separate, decoupled optimization stage. The roadmap for the discrete planner can be automatically generated from a given environment and robot description. We take the downwash effect of quadrotors into account, preserving safety during dense formation flights. Our method is complete with respect to the resolution of the discretization, and locally optimal with respect to an energy-minimizing integral-squared-derivative objective function. We also present an anytime iterative refinement scheme that improves the trajectories within a given computational budget. We support user-specified smoothness constraints and dynamic limits and provide simulations with up to 200 robots and a physical experiment with 32 quadrotors, see Fig. 1.

We extend our previous work [1] in the following ways:

1) Support of arbitrary continuous environments, i.e., we remove the limitations that start/goal locations and obstacles must be aligned with a user-defined grid.
2) Introduction of a generic multi-robot path planning framework which can be used for non-quadrotor robots as well. Specifically, we introduce Multi-Agent Path Finding with Generalized Conflicts (*MAPF/C*) and an efficient bounded suboptimal solver for MAPF/C problem instances. This also enables us to solve the labeled trajectory planning problem.
3) New experimental results analyzing the computational efficiency of and validating our approach in simulation and on a physical quadrotor swarm.

II. Related Work

A simple approach to multi-robot motion planning is to repurpose a single-robot planner and represent the Cartesian product of the robots' configuration spaces as a single large joint configuration space [2]. Robot-robot collisions are represented as configuration-space obstacles. However, the high-dimensional search space is computationally infeasible for large teams.

Many works have approached the problem from a graph search perspective [3], [4]. These methods are adept at dealing with maze-like environments and scenarios with high congestion. Some represent the search graph implicitly [5], so they are not always restricted to a predefined set of points in configuration space. However, directly interpreting a graph plan as a trajectory results in a piecewise linear path, requiring the robot to fully stop at each graph vertex to maintain dynamic feasibility. It is possible, however, to use these planners to resolve ordering conflicts and refine the output for execution on robots [6].

All authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA, USA.

Email: {whoenig, japreiss}@usc.edu, tkskwork@gmail.com, {gaurav, ayanian}@usc.edu

This work was partially supported by the ONR grants N00014-16-1-2907 and N00014-14-1-0734, and the ARL grant W911NF-14-D-0005.

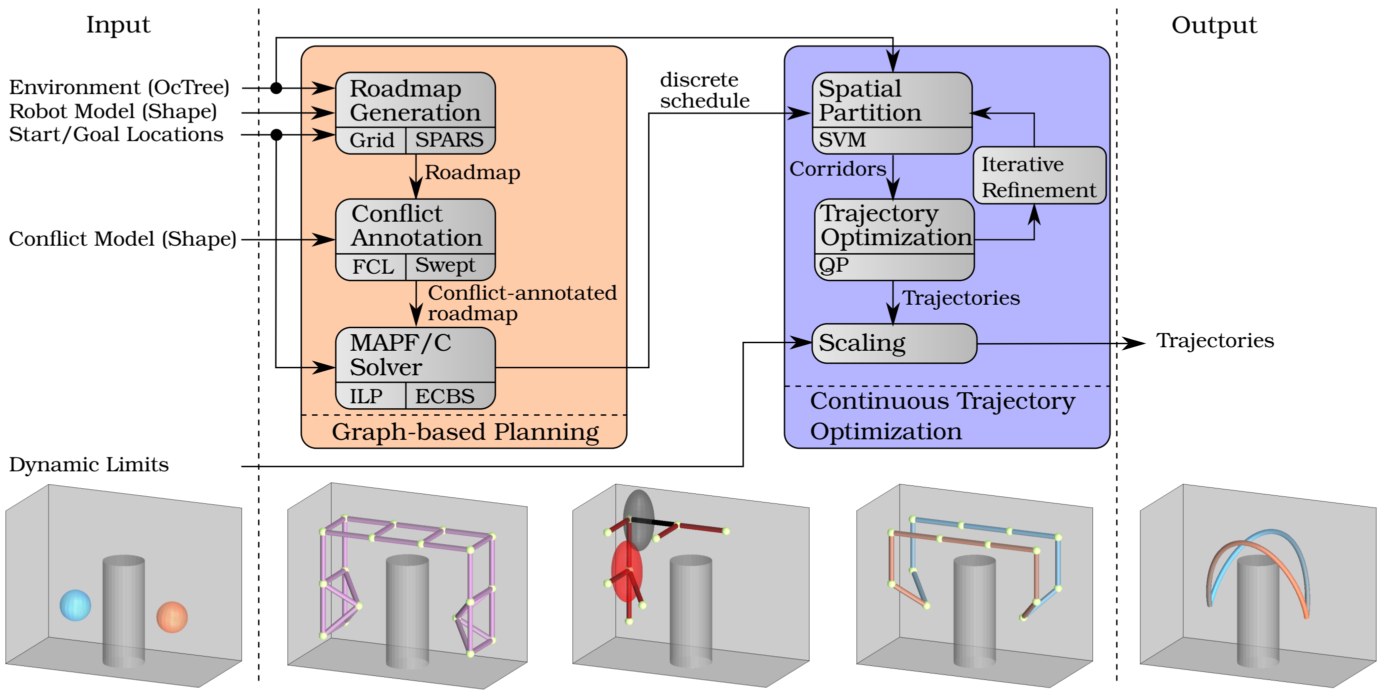


Fig. 2. Components of our approach. The user specifies a model of the environment (e.g. using an octree), a model of the robot (e.g. sphere), start/goal locations, a robot-robot collision model (e.g. ellipsoid), and dynamic limits (e.g. maximum acceleration). We generate a sparse roadmap from the environment, including the start and goal locations as vertices. The collision model is used to annotate the roadmap with additional inter-robot conflicts. This annotated roadmap can be used to find a discrete schedule for each robot. This schedule defines a corridor for each robot in which trajectory optimization is used to generate smooth trajectories. Finally, the trajectories are scaled in order to fulfill the dynamic limits. The pictures on the bottom show one example of two quadrotors swapping their positions.

Some authors have solved the formation change problem in a continuous setting [7], [8], but such methods are often tightly coupled, solving one large optimization problem in which the decision variables define all robots' trajectories. These approaches are typically demonstrated on smaller teams and do not easily scale to the size of a team in which we are interested. The authors of [9] propose a method that, similar to ours, uses conservative separating hyperplanes to decouple the optimization problem. However, the approach requires assigning a priority order to the robots. Others decouple the problem but do not support the level of smoothness in our solution, and do not show results on large teams [10]. The method of Turpin *et al.* [11] is computationally fast, but offsets the different trajectories in time, resulting in much longer time durations. Velocity profile methods [12] handle kinodynamic constraints well but are not able to fully exploit free space in the environment. Collision-avoidance approaches [13], [14] let each robot plan its trajectory independently and resolve conflicts in real time when impending collisions are detected. These methods are distributed and reactive, however, they do not provide any means to optimize the trajectories for objectives such as time or energy use, and they are poorly suited to problems in maze-like environments.

Our method builds upon Spline-based refinement of waypoint plans [15], [16] by adding support for three-dimensional ellipsoidal robots, environmental obstacles, and an anytime refinement stage to further improve the plan after generating an initial set of smooth trajectories. We demonstrate that our iterative refinement produces trajectories with significantly smoother dynamics than the initial trajectories.

III. Approach

We now formalize the trajectory planning problem for any homogeneous robot team and outline our method to solve this class of problems. Furthermore, we introduce the robot model for quadrotors. Later sections discuss the steps of our approach in more detail, using a quadrotor swarm as example.

A. Problem Statement

Consider a team of N robots in a bounded environment containing convex obstacles $\mathcal{O}_1 \ldots \mathcal{O}_{N_{obs}}$. Boundaries of the environment are defined by a convex polytope $\mathcal{W}$. The convex set of points representing a robot at position $q \in \mathbb{R}^3$ is $\mathcal{R}_{\mathcal{E}}(q)$. The free configuration space for a single robot is thus given by

$$\mathcal{F} = (\mathcal{W} \setminus (\bigcup_h \mathcal{O}_h)) \ominus \mathcal{R}_{\mathcal{E}}(\mathbf{0}) \tag{1}$$

where $\ominus$ denotes the Minkowski difference. We allow a separate inter-robot collision model and define $\mathcal{R}_{\mathcal{R}}(q)$ to be the convex set of points a robot at position q requires to operate safely when close to another robot. For example, in the case of quadrotors $\mathcal{R}_{\mathcal{R}}(q)$ can model the downwash effect.

Let $f^i : [0, T] \to \mathbb{R}^3$ be a trajectory for each robot r^i, where $T \in \mathbb{R}_{>0}$ is the total time duration until the last robot reaches its goal. All trajectories are considered *collision-free* if there are no robot-environment collisions, i.e. $f^i(t) \in \mathcal{F}$ and no robot-robot collisions, i.e.

$$\mathcal{R}_{\mathcal{R}}(f^i(t)) \cap \mathcal{R}_{\mathcal{R}}(f^j(t)) = \emptyset \quad \forall\, i \neq j,\ 0 \leq t \leq T. \tag{2}$$

In the *labeled* trajectory planning problem we are given a start and goal position for each robot $s^i, g^i \in \mathcal{F}$, where start and goal inputs must satisfy the robot-robot collision model, i.e. $\mathcal{R}_\mathcal{R}(s^i) \cap \mathcal{R}_\mathcal{R}(s^j) = \emptyset$ and $\mathcal{R}_\mathcal{R}(g^i) \cap \mathcal{R}_\mathcal{R}(g^j) = \emptyset$ for all $i \neq j$.

We seek the total time duration T and collision-free trajectories f^i such that:

- $f^i(0) = s^i$
- $f^i(T) = g^i$
- f^i is continuous up to user-specified derivative order C
- f^i is kinodynamically feasible for robot i.

In the *unlabeled* case we allow the robots to exchange goal locations, i.e. we additionally seek an assignment of each robot to a goal position such that $f^i(T) = g^{\phi(i)}$, where ϕ is a permutation of $1 \dots N$.

B. Components

Our method combines the strengths of two conceptually different approaches to multi-robot trajectory planning problems. The first approach is commonly used in the AI community and uses a graph to represent a roadmap and graph-search based algorithms to find collision-free schedules for all robots. This can be done efficiently for hundreds of robots even in maze-like environments. However, dynamic constraints of robots are ignored. Furthermore, creating an appropriate roadmap for an environment that fulfills all requirements of the search algorithm is challenging. The second approach, commonly used in robotics, is based on trajectory optimization. This approach can deal with kinodynamic constraints, but does not scale well to hundreds of robots.

Our approach is outlined in Fig 2. We use models of the environment and the robots to generate a roadmap suitable for planning for a single robot. We annotate the roadmap with additional edge and vertex conflicts to model constraints caused by inter-robot dependencies. We can then use an extended multi-robot planner to find discrete schedules for each robot. These schedules can be executed on real quadrotors without collisions, but they require the quadrotors to stop at each waypoint. Finally, a trajectory optimization stage generates smooth and conflict-free trajectories based on the discrete schedule. This approach can be used for any multi-robot trajectory planning problem, however the exact details for each step vary. In this paper we present the components needed to plan trajectories for a swarm of quadrotors, directly taking downwash into account.

C. Robot Model for Quadrotor Trajectory Planning

As aerial vehicles, quadrotors have a six-dimensional configuration space. However, as shown in [17], quadrotors are *differentially flat* in the *flat outputs* (x, y, z, ψ), where x, y, z is the robot's position in space and ψ its yaw angle (heading). Differential flatness implies that the control inputs needed to move the robot along a trajectory in the flat outputs are algebraic functions of the flat outputs and a finite number of their derivatives. Furthermore, in many applications, a quadrotor's yaw angle is unimportant and can be fixed at $\psi = 0$. We therefore focus our efforts on planning trajectories in three-dimensional Euclidean space.

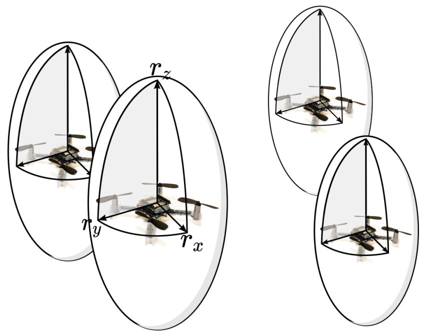


Fig. 3. Axis-aligned ellipsoid model of robot volume. Tall height prevents downwash interference between quadrotors.

While some multi-robot planning work has considered simplified dynamics models such as kinematic agents [6] or double-integrators [7], our method produces trajectories with arbitrary smoothness up to a user-defined derivative. This goal is motivated by [17], where it was shown that a continuous fourth derivative of position is necessary for physically plausible quadrotor trajectories, because it ensures that the quadrotor will not be asked to change its motor speeds instantaneously.

Rotorcraft generate a large, fast-moving volume of air underneath their rotors called *downwash*. The downwash force is large enough to cause a catastrophic loss of stability when one rotorcraft flies underneath another. We model downwash constraints as inter-robot collision constraints by treating each robot as an axis-aligned ellipsoid of radii $0 < r_x = r_y \ll r_z$, illustrated in Fig. 3. Empirical data collected in [18], [19] support this model. The set of points representing a robot at position $q \in \mathbb{R}^3$ is given by

$$\mathcal{R}_\mathcal{R}(q) = \{Ex + q : \|x\|_2 \leq 1\} \tag{3}$$

where $E = \mathbf{diag}(r_x, r_y, r_z)$.

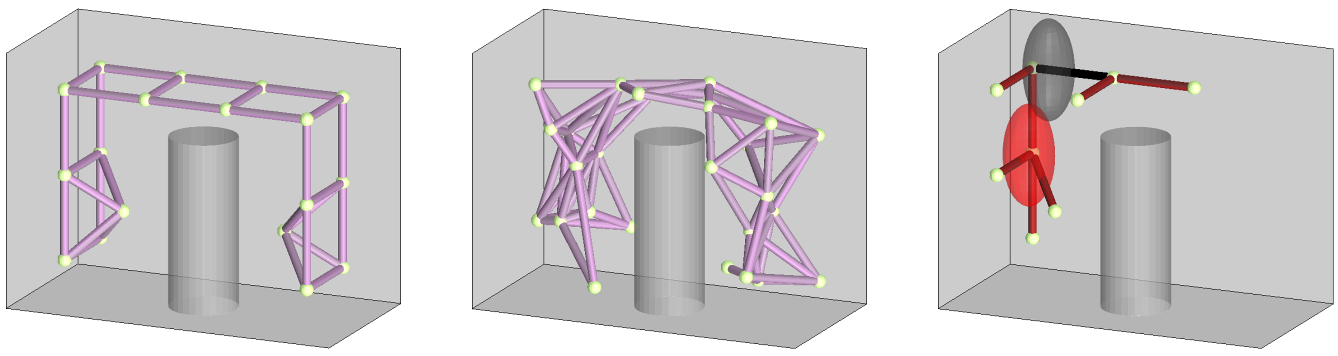
IV. Roadmap Generation

A roadmap is an undirected connected graph of the environment $\mathcal{G}_E = (\mathcal{V}_E, \mathcal{E}_E)$, where each vertex $v \in \mathcal{V}_E$ corresponds to a location in $\mathcal{F}$ and each edge $(u, v) \in \mathcal{E}_E$ denotes that there is a linear path in $\mathcal{F}$ connecting u and v. We also require that there exists a vertex $v_s^i \in \mathcal{V}_E$ corresponding to each start location s^i and that there exists a vertex $v_g^i \in \mathcal{V}_E$ for each goal location g^i. Let $loc : \mathcal{V}_E \to \mathbb{R}^3$ be a function that returns the location for each vertex.

The ideal roadmap should have the following properties:

1) be connected, i.e., if a path between two points in $\mathcal{F}$ exists, there should be a path in the roadmap as well;
2) lead to optimal results, i.e., the shortest path between two points in $\mathcal{F}$ can be approximated well by a path in the roadmap; and
3) be sparse, i.e., have a small number of vertices and edges.

The last property is desired because dense roadmaps result in a higher number of inter-robot conflicts, which can create a significant computational burden for the discrete planning



(a) Grid-based roadmap. (b) SPARS-based roadmap. (c) Conflicting edges.

Fig. 4. Example of generated roadmaps using the grid approach (a) and SPARS (b) with the same desired dispersion. The roadmaps can be annotated with edge and vertex conflicts based on the inter-robot conflict model $\mathcal{R}_\mathcal{R}$. Conflicting edges for black edge are marked in red in (c). The edges are in conflict because it is possible to place the robots on the edge such that (2) is violated.

stage. The first two properties are in conflict with the last property. Thus, roadmap generation should balance those goals. A useful property of a roadmap is its *dispersion*, which is the radius of the largest sphere centered in $\mathcal{F}$ that does not contain any vertex location. Dispersion is a measure of the uniformity of the roadmap. In this work we tested two different approaches, namely 6-connected grid structures, and sparse roadmap spanners. Here, *spanner* refers to a concept from graph theory where a spanner of a graph is a subgraph with fewer edges such that the distance between any two vertices is within a user-provided bound compared to the original graph.

The grid roadmap generator places potential vertices equidistant from each other. A vertex is only added if it is in $\mathcal{F}$. Edges are added to the six closest neighbors if there would be no collision between the robot and the environment when traversing that edge. Roadmaps generated this way have a regular structure, reducing inter-robot conflicts and providing a low dispersion for a fixed number of vertices. However, completeness and optimality are only achieved as the grid-size approaches zero, which is not desirable due to our sparseness requirement. Thus, the generated roadmaps are often missing important edges and vertices and fail to achieve the first two desired properties.

The family of SParse Roadmap Spanner algorithms (*SPARS*) [20] attempts to generate roadmaps that are bounded sub-optimally with respect to distance (up to a user-provided stretch factor). Unlike graph spanners, the algorithm attempts to reduce both vertices and edges, rather than edges only. The SPARS algorithm uses a dense roadmap similar to those generated by PRM* [21] and only keeps a subset of vertices and edges such that the distance sub-optimality is guaranteed. The SPARS2 algorithm removes the requirement of explicitly constructing a dense roadmap, reducing the memory overhead but at the cost of a larger roadmap. We do not have a requirement for a small memory usage and used the SPARS algorithm.

In both grid and SPARS cases we add vertices to the roadmap corresponding to the start and goal locations if such vertices are not already part of the roadmap. We connect those additional vertices to up to six neighbors within a search radius if the resulting edge could be traversed without any collision with the environment. Example roadmaps are shown in Fig. 4.

V. Conflict Annotation

Roadmaps are typically generated to plan the motion for a single robot. If the same roadmap is used by multiple robots, there are additional constraints:

1) Vertex-Vertex Constraints: two robots may not occupy two vertices which are in close proximity to each other at the same time.
2) Edge-Edge Constraints: two robots may not traverse two edges if a collision could occur during the traversal, see Fig. 4(c).
3) Edge-Vertex Constraints: one robot may not traverse an edge if a collision could occur with another stationary robot.

The goal of the conflict annotation is to identify the set of conflicts for each edge and vertex. The output is a graph where each vertex and each edge is annotated with a conflict set.

We define the following functions:

$$\begin{aligned} conVV(v) &= \{u \in \mathcal{V}_E \mid \\ &\qquad \mathcal{R}_\mathcal{R}(loc(u)) \cap \mathcal{R}_\mathcal{R}(loc(v)) \neq \emptyset\} \\ conEE(e) &= \{d \in \mathcal{E}_E \mid \mathcal{R}^*_\mathcal{R}(d) \cap \mathcal{R}^*_\mathcal{R}(e) \neq \emptyset\} \\ conEV(e) &= \{u \in \mathcal{V}_E \mid \mathcal{R}_\mathcal{R}(loc(u)) \cap \mathcal{R}^*_\mathcal{R}(e) \neq \emptyset\} \end{aligned} \tag{4}$$

where $\mathcal{R}^*_\mathcal{R}(e)$ is the set of points swept when traversing edge e. We refer to this method as *swept* collision model.

The swept collision model is conservative, allowing edge traversals with an arbitrary velocity profile. Alternatively, we can assume that the motion on the edge uses a known velocity profile (such as constant velocity), where $mot(e,t)$ refers to the position of the robot traversing edge e for $t \in [0,1]$. We can then use a standard continuous collision definition as used in the Flexible Collision Library (FCL) [22]:

$$\begin{aligned} conEE'(e) &= \{d \in \mathcal{E}_E \mid \exists t \in [0,1] \\ &\qquad \mathcal{R}_\mathcal{R}(mot(d,t)) \cap \mathcal{R}_\mathcal{R}(mot(e,t)) \neq \emptyset\} \\ conEV'(e) &= \{u \in \mathcal{V}_E \mid \exists t \in [0,1] \\ &\qquad \mathcal{R}_\mathcal{R}(loc(u)) \cap \mathcal{R}_\mathcal{R}(mot(e,t)) \neq \emptyset\} \end{aligned} \tag{5}$$

Collision checking is required for all vertex and edge pairs, leading to quadratic time complexity.

VI. Discrete Planning

The annotated roadmap with specified start/goal vertices is the input to the graph planning stage of our method, which we describe in this section. We first state the MAPF/C problem, a generalized version of the Multi-Agent Path Finding problem (MAPF) that explicitly avoids conflicts in a conflict-annotated roadmap. We then introduce two different planners, an optimal planner that solves the unlabeled problem and a bounded-suboptimal planner for the labeled case. Finally, we discuss methods for goal assignment to allow using the bounded-suboptimal planner for the unlabeled case as well.

A. Multi-Agent Path Finding with Generalized Conflicts (MAPF/C)

We are given a roadmap of the environment ($\mathcal{G}_E$) with additional conflict sets for generalized vertex-vertex ($conVV$), edge-edge ($conEE$), and edge-vertex ($conEV$) conflicts as generated in the previous section. Additionally, we are given a unique start vertex for each robot $v_s^i \in \mathcal{V}_E$. In the labeled case we are given a unique goal for each robot $v_g^i \in \mathcal{V}_E$; in the unlabeled case these goals are interchangeable.

At each timestep, a robot can either wait at its current vertex or traverse an edge. A discrete schedule p^i for each robot is composed of a sequence of $K+1$ locations:

$$\begin{aligned} p^i &= loc(u_0^i), loc(u_1^i), \ldots, loc(u_K^i) \\ &\triangleq x_0^i, x_1^i, \ldots, x_K^i \end{aligned} \tag{6}$$

Robots are expected to be synchronized in time, such that robot i is at waypoint x_k^i at timestep k. In between waypoints x_k^i and x_{k+1}^i, we assume that robot i travels on the line segment between x_k^i and x_{k+1}^i. We denote this line segment by ℓ_k^i.

Our goal is to find discrete schedules p^i, such that the following properties hold:

P1: Each robot starts at its start vertex: $\forall i : u_0^i = v_s^i$.
P2: Each robot ends at a unique goal location: $\forall i : x_K^i = v_g^{\phi(i)}$, where ϕ is a permutation of $1 \ldots N$.
P3: At each timestep, each robot either stays at its current position or moves along an edge: $\forall k, \forall i$: $u_k^i = u_{k+1}^i$ or $(u_k^i, u_{k+1}^i) \in \mathcal{E}_E$.
P4: There are no robots occupying the same location at the same time (*vertex collision*): $\forall k, \forall i \neq j$: $u_k^i \neq u_k^j$.
P5: There are no robots traversing the same edge in opposite directions at the same time (*edge collision*): $\forall k, \forall i \neq j$: $u_k^i \neq u_{k+1}^j$ or $u_k^j \neq u_{k+1}^i$.
P6: Robots obey inter-robot constraints when stationary (*generalized vertex-vertex collision*): $\forall k, \forall i \neq j$: $u_k^i \notin conVV(u_k^j)$.
P7: Robots obey inter-robot constraints while traversing an edge (*generalized edge-edge collision*): $\forall k, \forall i \neq j$: $(u_k^i, u_{k+1}^i) \notin conEE((u_k^j, u_{k+1}^j))$.
P8: Robots obey inter-robot constraints between stationary and traversing robots (*generalized edge-vertex collision*): $\forall k, \forall i \neq j, u_k^i = u_{k+1}^i$: $u_k^i \notin conEV((u_k^j, u_{k+1}^j))$.

In the labeled case the permutation $\phi(i) = i$ is given; in the unlabeled case we are seeking $\phi(i)$ and the discrete schedules p^i simultaneously. A solution is optimal with respect to *makespan* if K is minimal. A solution is optimal with respect to cost if $\sum_i pathcost(p^i)$ is minimal. The $pathcost(\cdot)$ is a modular user-provided function, e.g. the total distance traveled or the number of actions taken until the goal is reached.

The first five properties are identical to typical MAPF formulations. Properties 6–8 are newly considered constraints in MAPF/C. In the following two theorems, we prove some hardness results for MAPF/C. Theorem VI.2 shows that makespan minimization for MAPF/C is hard even for the unlabeled case. This is in contrast to the tractability result, established using a reduction to the maximum-flow problem, for makespan minimization for unlabeled MAPF [23].

Theorem VI.1. *Solving labeled MAPF/C optimally with respect to cost or makespan is NP-hard.*

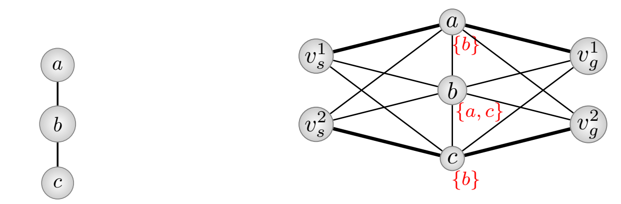
Proof. This follows directly from the NP-hardness proofs of the same problems for labeled MAPF [24]. □

Theorem VI.2. *Solving unlabeled MAPF/C optimally with respect to makespan is NP-hard.*

Proof. Given an undirected graph, an independent set in it is defined as a collection of vertices such that no two of them are connected by an edge. A maximum independent set (MIS) is an independent set of maximum cardinality. Finding an MIS in undirected graphs is a well-known NP-hard problem. It is also NP-hard to decide whether the size of an MIS in graph $\mathcal{G}_{MIS}$ is greater than or equal to k_{MIS}. We reduce this NP-hard decision problem to the makespan minimization problem for unlabeled MAPF/C.

For a given $(\mathcal{G}_{MIS}, k_{MIS})$, we create a new graph $\mathcal{G}_{MAPF/C}$ that includes $\mathcal{G}_{MIS}$ straddled between $2k_{MIS}$ additional vertices $v_s^1, \ldots, v_s^{k_{MIS}}$ and $v_g^1, \ldots, v_g^{k_{MIS}}$. Each additional vertex is connected to all vertices from $\mathcal{G}_{MIS}$. In the corresponding MAPF/C instance, v_s^i is interpreted as the starting vertex of robot r^i; and v_g^i is interpreted as the goal vertex of robot r^i. The edges from $\mathcal{G}_{MIS}$ are interpreted as generalized vertex-vertex conflicts. The edges that connect the additional vertices to vertices from $\mathcal{G}_{MIS}$ are interpreted as edges that robots can traverse in unit time. It is now easy to see that all k_{MIS} robots can go from their start vertices to their goal vertices within the minimum makespan of 2 if and only if the size of an MIS in $\mathcal{G}_{MIS}$ is greater than or equal to k_{MIS}. An example is shown in Fig. 5. Therefore, if solving MAPF/C optimally with respect to makespan can be done in polynomial time, the decision version of the MIS problem can also be solved in polynomial time. Conversely, since the decision version of the MIS problem is NP-hard, makespan minimization for unlabeled MAPF/C is also NP-hard. □

Although this negative result is established for MAPF/C on general graphs (and not on grids), it is relevant for us here since the SPARS-based roadmaps on which MAPF/C instances must be solved are more general graphs that do not resemble grids.



(a) $\mathcal{G}_{MIS}$ (b) $\mathcal{G}_{MAPF/C}$

Fig. 5. Example for MIS to unlabeled MAPF/C reduction with $k_{MIS} = 2$. The red sets define $conVV(\cdot)$ of nodes a, b, c. A solution for the MAPF/C instance is shown with the bold edges, indicating that $\{a, c\}$ is a maximum independent set.

B. Unlabeled Planner

Given the value of K, the decision problem of finding an unlabeled MAPF schedule with makespan K can be solved in polynomial time. This can be achieved by reduction to a maximum-flow problem in a larger graph, derived from $\mathcal{G}_E$, known as a *time-expanded flow-graph* [23]. This graph, denoted by $\mathcal{G}_F$, contains $O(K \cdot |\mathcal{V}_E|)$ vertices and is constructed such that a flow in $\mathcal{G}_F$ represents a solution to the MAPF instance. This maximum-flow problem can also be expressed as an Integer Linear Program (ILP) where each edge is modeled as a binary variable indicating its flow and the objective is to maximize the flow subject to conservation constraints [25]. An ILP formulation allows us to add additional constraints for P6 – P8.

We build the time-expanded flow-graph $\mathcal{G}_F = (\mathcal{V}_F, \mathcal{E}_F)$ as an intermediate step to formulate the ILP. We describe the approach briefly in this paragraph; more detailed discussions are available in related work [23], [25], [26]. For each timestep k and vertex $v \in \mathcal{V}_E$ we add two vertices u_k^v and w_k^v to $\mathcal{V}_F$ and create an edge connecting them (red edges in Fig. 6(c)). For each timestep k and edge $(v_1, v_2) \in \mathcal{E}_E$ we create a "gadget" connecting $u_k^{v_1}, u_k^{v_2}, w_k^{v_1}$, and $w_k^{v_2}$. As shown in Fig 6(b), the "gadget" disallows agents to swap their positions in one timestep, thus enforcing P5. Furthermore, we connect consecutive timesteps with additional edges (w_k^v, u_{k+1}^v) (green edges in Fig. 6(c)) to enforce P4. Additionally we add vertices $source$ and $sink$, which are connected to vertices $\{u_0^{v_s^i} : \forall i\}$ and $\{w_K^{v_g^i} : \forall i\}$ respectively. If a maximum flow is computed on this graph, the flow describes a discrete schedule for each robot, fulfilling P1–P5.

We now extend previous work by introducing annotations $con : \mathcal{E}_F \mapsto 2^{\mathcal{E}_F}$ to some of the edges such that $con(e)$ is the set of edges with which edge e has a generalized conflict with. Consider vertices $v, v' \in \mathcal{V}_E$ that, if simultaneously occupied, would violate P6. Those vertices map to helper edges $e_k = (w_k^v, u_{k+1}^v)$ and $e'_k = (w_k^{v'}, u_{k+1}^{v'})$ in $\mathcal{G}_F$ for all k (green edges in Fig. 6(c)). In that case we insert e'_k into $con(e_k)$ and e_k into $con(e'_k)$ for all k. Similarly, consider $(v_1, v_2) \in \mathcal{E}_E$ and $(v'_1, v'_2) \in \mathcal{E}_E$ that violate P7. These edges map to helper edges e_k and e'_k in $\mathcal{G}_F$ as part of the gadget for all k (blue edges in Fig. 6(c)). As before we insert e'_k into $con(e_k)$ and vice versa for all k. For edge-vertex conflicts, consider $(v_1, v_2) \in \mathcal{E}_E$ and $v' \in \mathcal{V}_E$ that violate P8. The first edge (v_1, v_2) maps to helper edges e_k, while the vertex v' corresponds to an agent waiting at vertex v', which requires the traversal of helper edge e'_k in $\mathcal{G}_F$ (red edges in Fig. 6(c)). As before we insert e'_k into $con(e_k)$ and vice versa for all k. Finally, we disallow that the blue edges of the "gadget" are used for wait actions by adding constraints $con((u_k^{v_1}, u_k^{(v_1,v_2)})) = \{(w_k^{(v_1,v_2)}, w_k^{v_1})\}$ and $con((u_k^{v_2}, u_k^{(v_1,v_2)})) = \{(w_k^{(v_1,v_2)}, w_k^{v_2})\}$.

For each edge $(u, v) \in \mathcal{E}_F$, we introduce a binary variable $z_{(u,v)}$. The ILP can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{(source,v)\in\mathcal{E}_F} z_{(source,v)} \\ \text{subject to} \quad & \sum_{(u,v)\in\mathcal{E}_F} z_{(u,v)} = \sum_{(v,w)\in\mathcal{E}_F} z_{(v,w)} \;\; \forall v \in \mathcal{V}'_F \\ & z_e + \sum_{e' \in con(e)} z_{e'} \leq 1 \quad \forall e \in \mathcal{E}_F \end{aligned} \tag{7}$$

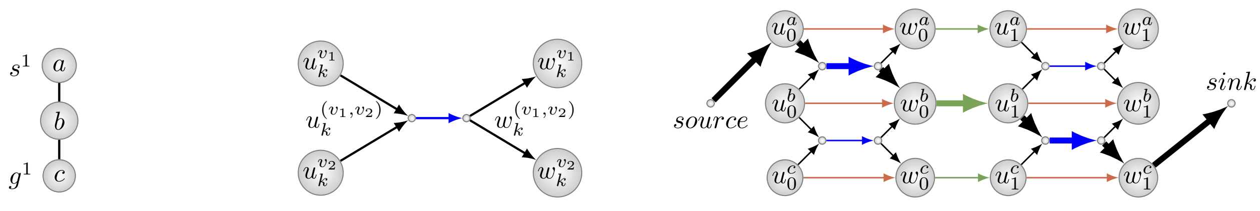
where $\mathcal{V}'_F = \mathcal{V}_F \backslash \{source, sink\}$. The first constraint enforces flow conservation, and thus P3–P5. The second constraint enforces P6–P8. P1 and P2 are implicitly enforced by construction of the flow graph. A solution to the ILP assigns a flow to each edge. We can then easily create the schedule p^i for each robot by setting x_k^i based on the flow in $\mathcal{G}_F$. The permutation $\phi(i)$ is implicitly given by x_K^i.

In order to find an optimal solution for an unknown K, we use a two-step approach. First, we find a lower bound for K by ignoring the generalized constraints. We search the sequence $K = 1, 2, 4, 8, \ldots$ for a minimum feasible value that is a power of 2, and then perform a binary search to find the minimum feasible value $LB(K)$. Because we ignore the generalized constraints, we can check the feasibility in polynomial time using the Edmonds-Karp algorithm on the time-expanded flow-graph. Second, we execute a linear search starting from $LB(K)$, solving the fully constrained ILP. In practice, we found that the lower bound $LB(K)$ is sufficiently close to the final K such that a linear search is faster compared to another modified binary search using the ILP.

C. Labeled Planner

Two common approaches for solving labeled MAPF instances are variants of M* [3] and Conflict-Based Search (CBS) [27]. The core idea of M* is to plan for the robots individually as much as possible. Whenever a conflict occurs, the conflicting robots are joined together as a meta-agent. In contrast, Conflict-Based Search methods try to resolve conflicts one-by-one, starting with a conflict occurring at the earliest timestep. In both methods, the search might grow exponentially in the worst case. Our labeled planner is based on a bounded suboptimal variant of CBS, called Enhanced CBS (ECBS), which has been shown to solve practical instances with hundreds of robots in maze-like environments [28].

A detailed description, pseudocode, and an example of (E)CBS are available in related work [28]. We give a brief overview of the approach in the following. ECBS uses two different search levels: high- and low-level. The high-level search creates a binary constraint tree. Each node in that tree has a set of constraints. A constraint either disallows a robot to occupy a specific vertex or to traverse a specific edge at a



(a) $\mathcal{G}_E$ (b) "Gadget" for flow-graph construction. (c) $\mathcal{G}_F$ with $K = 2$.

Fig. 6. Example flow-graph (c) for environment shown in (a) with a single robot. The construction uses a graph "gadget" (b) for each edge in $\mathcal{G}_E$. The blue edges are annotated with edge-edge and edge-vertex conflicts and the green edges are annotated with vertex-vertex conflicts. The bold arrows in (c) show the maximum flow through the network, which can be used to compute the robots' discrete schedules.

fixed timestep. A node also contains a discrete schedule p^i for each robot that is consistent with the set of constraints, and an associated cost for the discrete schedules. The root node does not have any constraints and thus the discrete schedules p^i can be created by finding the path with the lowest cost in the given roadmap for each robot individually. If a node is valid (i.e., its discrete schedules p^i are consistent with P1–P8), a solution is found. Otherwise, the first violation of P1–P8 between two robots r^i and r^j is found, and two new nodes are created. Both children inherit the constraints from the parent node. The first child imposes an additional constraint on r^i and the second node adds one additional constraint for r^j. In CBS the nodes are traversed using the best-first search strategy with respect to the cost. The low-level search is used to compute a discrete schedule for a single agent that satisfies the constraints for that agent. Each time a new high-level node is created, the low-level search needs to be executed just once for the agent with the newly added constraint. The bounded suboptimality in ECBS is achieved by using focal search[1] with heuristics in both levels.

ECBS can solve MAPF/C instances by using the generalized conflict definition for both high-level and low-level focal searches. The algorithm is identical to the one outlined in the original paper [28], with the following adjustments. For the high-level search, we now need to consider conflicts caused by P6 – P8. In case the first conflict is a violation of P6 between agents r^i and r^j at timestep k, we create two child nodes. The first child adds a constraint for r^i to avoid visiting x_k^i at timestep k. The second child adds a constraint for r^j to avoid visiting x_k^j at timestep k. Similarly, if the first conflict is a violation of P7 between r^i and r^j, we add two child nodes with the additional conflicts of r^i not traversing (u_k^i, u_{k+1}^i) and r^j not traversing (u_k^j, u_{k+1}^j), respectively. In case the first conflict is a violation of P8 where, without loss of generality, r^i is waiting at a vertex and r^j is traversing an edge, we add two child nodes with the additional conflicts of r^i not waiting at u_k^i at timestep k and r^j not traversing (u_k^j, u_{k+1}^j), respectively. Furthermore, we need to adjust the focal search heuristics to count violations caused by P6 – P8 as well. For the high-level focal search we use the number of pairs of robots that have at least one conflict. For the low-level focal search we use the total number of conflicts in the high-level search node.

[1]Focal search maintains two lists in an A* search framework: an open list and a focal list. The focal list contains elements that are in the open list and within a suboptimality factor of the minimum f-value. A node from the focal list is chosen for expansion based on a secondary heuristic.

ECBS is bounded suboptimal with respect to the cost, i.e., for a user-provided suboptimality bound w, the cost of the returned solution will satisfy

$$\sum_i pathcost(p^i) \leq w \sum_i pathcost(p_*^i),$$

where p_*^i is a optimal solution as computed by CBS. The proof in [28] holds for the ECBS/C as well.

D. Goal Assignment

ECBS can be used to solve an unlabeled problem approximately, by finding a goal assignment first. In robotics, the polynomial-time Hungarian method, which finds the optimal assignment with respect to the sum of the costs, is frequently used. This might, for example, correspond to minimizing the total energy usage of the robots. However, in the unlabeled case it is typical to minimize the makespan, i.e., the time until the last robot reaches its goal. This goal assignment problem is also known as the linear bottleneck assignment problem and can also be solved efficiently using, for example, the Threshold algorithm [29].

VII. Trajectory Optimization

In the continuous refinement stage, we convert the waypoint sequences p^i generated by the discrete planner into smooth trajectories f^i. We use the discrete plan to partition the free space $\mathcal{F}$ such that each robot solves an independent smooth trajectory optimization problem in a region that is guaranteed to be collision-free. We assign a time $t_k = k\Delta t$ to each discrete timestep, where Δt is a user-specified parameter. This is an initial guess for $T = K\Delta t$. The exact total time T is computed in a post-processing stage such that all trajectories meet given physical limits such as a maximum thrust constant.

A. Spatial Partition

The continuous refinement method begins by finding *safe corridors* within the free space $\mathcal{F}$ for each robot. An example from a real problem instance is shown in Fig. 7. The safe corridor for robot r^i is a sequence of convex polyhedra $\mathcal{P}_k^i, k \in \{1 \dots K\}$, such that, if each r^i travels within $\mathcal{P}_k^i$ during time interval $[t_{k-1}, t_k]$, both robot-robot and robot-obstacle collision avoidance are guaranteed. For robot r^i in timestep k, the safe polyhedron $\mathcal{P}_k^i$ is the intersection of:

- $N - 1$ half-spaces separating r^i from r^j for $j \neq i$;

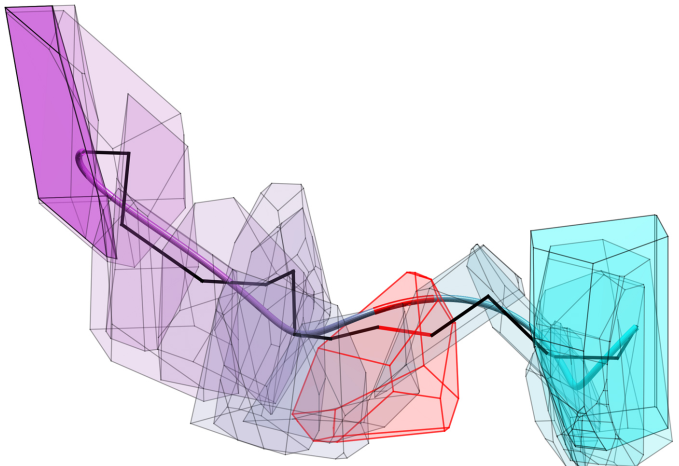


Fig. 7. Safe corridor for one robot over entire flight. Corridor polytopes are colored by timestep. *Black line*: underlying discrete plan. *Shaded tube*: smooth trajectory after first iteration of refinement. *Highlighted in red*: polytope, discrete plan segment, and trajectory polynomial piece for a single timestep.

- N_{obs} half-spaces separating r^i from $\mathcal{O}_1 \dots \mathcal{O}_{N_{obs}}$.

We separate r^i from r^j by finding a separating hyperplane $(\alpha_k^{(i,j)} \in \mathbb{R}^3, \beta_k^{(i,j)} \in \mathbb{R})$ such that:

$$\begin{aligned} \ell_k^i &\subset \{x : {\alpha_k^{(i,j)}}^T x < \beta_k^{(i,j)}\} \\ \ell_k^j &\subset \{x : {\alpha_k^{(i,j)}}^T x > \beta_k^{(i,j)}\}. \end{aligned} \tag{8}$$

While this hyperplane separates the line segments ℓ_k^i and ℓ_k^j, it does not account for the robot ellipsoids. Without loss of generality, suppose the hyperplanes are given in the normalized form where $\|\alpha_k^{(i,j)}\|_2 = 1$. We accommodate the ellipsoids by shifting each hyperplane according to its normal vector, resulting in the final trajectory constraint halfspaces:

$$\begin{aligned} {\alpha_k^{(i,j)}}^T f^i(t) &< \beta_k^{(i,j)} - \|E\alpha_k^{(i,j)}\|_2 \quad \forall t \in [t_{k-1}, t_k] \\ {\alpha_k^{(i,j)}}^T f^j(t) &> \beta_k^{(i,j)} + \|E\alpha_k^{(i,j)}\|_2 \quad \forall t \in [t_{k-1}, t_k] \end{aligned} \tag{9}$$

where $E = \mathbf{diag}(r_x, r_y, r_z)$ is the ellipsoid matrix. Robot-obstacle separating hyperplanes are computed similarly, except using a different ellipsoid E_{obs} for obstacles to model the fact that downwash is only important for robot-robot interactions.

In our implementation, we require that obstacles $\mathcal{O}_i$ are bounded convex polytopes described by vertex lists. Line segments are also convex polytopes described by vertex lists. Computing a separating hyperplane between two disjoint convex polytopes $\Psi = \mathbf{conv}(\psi_1 \dots \psi_{m_\Psi})$ and $\Omega = \mathbf{conv}(\omega_1 \dots \omega_{m_\Omega})$, where **conv** denotes the convex hull, can be posed as an instance of the hard-margin support vector machine (SVM) problem [30]. However, the ellipsoid robot shape alters the problem: for a separating hyperplane with unit normal vector α, the minimal safe margin is $2\|E\alpha\|_2$. Incorporating this constraint in the standard hard-margin SVM formulation yields a slightly modified version of the typical SVM quadratic program:

$$\begin{aligned} \text{minimize} \quad & \alpha^T E^2 \alpha \\ \text{subject to} \quad & \alpha^T \psi_i - \beta \leq 1 \quad \text{for } i \in 1 \dots m_\Psi \\ & \alpha^T \omega_i - \beta \geq 1 \quad \text{for } i \in 1 \dots m_\Omega \end{aligned} \tag{10}$$

We solve a problem of this form for each robot-robot pair and each robot-obstacle pair to yield the safe polyhedron $\mathcal{P}_k^i$ in the form of a set of linear inequalities. Note that the safe polyhedra need not be bounded and that $\mathcal{P}_k^i \cap \mathcal{P}_{k+1}^i \neq \emptyset$ in general. In fact, the overlap between consecutive $\mathcal{P}_k^i$ allows the smooth trajectories to deviate significantly from the discrete plans, which is an advantage when the discrete plan is far from optimal.

B. Bézier Trajectory Basis

After computing safe corridors, we plan a smooth trajectory $f^i(t)$ for each robot, contained within the robot's safe corridor. We represent these trajectories as piecewise polynomials with one piece per time interval $[t_k, t_{k+1}]$. Piecewise polynomials are widely used for trajectory planning: with an appropriate choice of degree and number of pieces, they can represent arbitrarily complex trajectories with an arbitrary number of continuous derivatives.

We denote the k^{th} piece of robot i's piecewise polynomial trajectory as f_k^i. We wish to constrain f_k^i to lie within the safe polyhedron $\mathcal{P}_k^i$. However, when working in the standard monomial basis, i.e., when the decision variables are the a_i in the polynomial expression

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_D t^D,$$

bounding the polynomial inside a convex polyhedron is not a convex constraint. Instead, we formulate trajectories as Bézier curves. A degree-D Bézier curve is defined by a sequence of $D+1$ *control points* $y_i \in \mathbb{R}^3$ and a fixed set of Bernstein polynomials, such that

$$f(t) = b_{0,D}(t) y_0 + b_{1,D}(t) y_1 + \cdots + b_{D,D}(t) y_D \tag{11}$$

where each $b_{i,D}$ is a degree-D Bernstein polynomial with coefficients[2] given in [31]. The curve begins at y_0 and ends at y_D. In between, it does not pass *through* the intervening control points, but rather is guaranteed to lie in the convex hull of all control points. Thus, when using Bézier control points as decision variables instead of monomial coefficients, constraining the control points to lie inside a safe polyhedron guarantees that the resulting polynomial will also lie inside the polyhedron. We define f^i as a K-piece, degree-D Bézier curve and denote the d^{th} control point of f_k^i as $y_{k,d}^i$. The degree parameter D must be sufficiently high to ensure continuity at the user-defined continuity level C.

C. Distributed Optimization Problem

The set of Bézier curves that lie within a given safe corridor describes a family of feasible solutions to a single robot's planning problem. We select an optimal trajectory by minimizing a weighted combination of the integrated squared derivatives:

$$\text{cost}(f^i) = \sum_{c=1}^{C} \gamma_c \int_0^T \left\| \frac{d^c}{dt^c} f^i(t) \right\|_2^2 dt \tag{12}$$

[2]The canonical Bernstein polynomials are defined over the time interval $[0, 1]$, but they are easily modified to span our desired time interval.

where the $\gamma_c \geq 0$ are user-chosen weights on the derivatives.

Our decision vector $\mathbf{y}$ consists of all control points for f^i concatenated together:

$$\mathbf{y} = \left[{y^i_{1,0}}^T \cdots {y^i_{1,D}}^T, \quad \ldots, \quad {y^i_{K,0}}^T \cdots {y^i_{K,D}}^T \right]^T \tag{13}$$

The objective function (12) is a quadratic function of $\mathbf{y}$, which can be expressed in the form:

$$\text{cost}(f^i) = \mathbf{y}^T (B^T Q B) \mathbf{y} \tag{14}$$

where B is a block-diagonal matrix transforming control points into polynomial coefficients, and the formula for Q is given in [32]. The start and goal position constraints, as well as the continuity constraints between successive polynomial pieces, can be expressed as linear equalities. Thus, we solve the quadratic program:

$$\begin{aligned} \text{minimize} \quad & \mathbf{y}^T (B^T Q B) \mathbf{y} \\ \text{subject to} \quad & y^i_{k,d} \in \mathcal{P}^i_k \quad \forall\, i, k, d \\ & f^i(0) = s^i, \ f^i(T) = g^{\phi(i)} \\ & f^i \text{ continuous up to derivative } C \\ & \frac{d^c}{dt^c} f^i(t) = 0 \quad \forall\, c > 0, \ t \in \{0, T\} \end{aligned} \tag{15}$$

It is important to note that there are N quadratic programs which can be solved in parallel, allowing a distributed implementation where each robot receives the halfspace coefficients of its safe corridor (9) and solves the quadratic program (15) onboard. Computation of the spatial partition may also be distributed with a $2\times$ constant factor of redundant work.

The quadratic program (15) may not always have a solution due to our conservative assumptions regarding velocity profiles and the decoupling of robots using hyperplanes. In these cases, we fall back on a solution that follows the discrete plan exactly, coming to a complete stop at corners. Details of this solution are given in [15].

The corridor-constrained Bézier formulation presents one notable shortcoming: for a given safe polyhedron $\mathcal{P}^i_k$, there exist degree-D polynomials that lie inside the polyhedron but cannot be expressed as a Bézier curve with control points that are contained within $\mathcal{P}^i_k$. Empirical exploration of Bézier curves suggests that this problem is most significant when the desired trajectory is near the faces of the polyhedron rather than the center. Further research is needed to characterize this issue more precisely.

D. Iterative Refinement

Solving (15) for each robot converts the discrete plan into a set of smooth trajectories that are locally optimal given the spatial decomposition. However, these trajectories are not globally optimal. In our experiments, we found that the smooth trajectories sometimes lie quite far away from the original discrete plan. Motivated by this observation, we implement an iterative refinement stage where we use the smooth trajectories to define a new spatial decomposition, and use the same optimization method to solve for a new set of smooth trajectories.

For time interval k, we sample f^i_k at S evenly-spaced points in time to generate a set of points $\mathcal{S}^i_k$. The number of sample points S is a user-specified parameter, set to $S = 32$ in our experiments. We then compute the separating hyperplanes as before, except we separate $\mathcal{S}^i_k$ from $\mathcal{S}^j_k$ instead of ℓ^i_k from ℓ^j_k. This problem is also a (slightly larger) ellipsoid-weighted SVM instance. While the sample points $\mathcal{S}^i_k$ are not a complete description of f^i_k, $\mathcal{S}^i_k$ is guaranteed to be linearly separable from $\mathcal{S}^j_k$ for $i \neq j$, because the polynomial pieces f^i_k, f^j_k lie inside their respective disjoint polyhedra $\mathcal{P}^i_k, \mathcal{P}^j_k$.

These new safe corridors are roughly "centered" on the smooth trajectories, rather than on the discrete plan. Intuitively, iterative refinement provides a chance for the smooth trajectories to further minimize the cost of the quadratic objective (15) beyond that allowed by the constraints of the previous spatial partition. Iterative refinement is similar in spirit to sequential quadratic programming (SQP) [33] and sequential convex programming (SCP) [34]. These methods find a local minimum for a nonlinear, nonconvex optimization problem by solving an iterative sequence of convex approximations. Our method is not derived directly from such an underlying problem but follows the same pattern of alternating between updates on the approximated constraints and on the solution. The max-margin spatial partition objective (10) is not directly related to the primary energy minimization objective (14), but updating the separating hyperplanes in a separate step, allows us to decompose each iteration into N independent subproblems for efficient and decentralized computation.

Iterative refinement can be classified as an anytime algorithm. If a solution is needed quickly, the original set of f^i can be obtained in a few seconds. If the budget of computational time is larger, iterative refinement can be repeated until the quadratic program cost (14) converges.

E. Dynamic Limits

Unlike related work (e.g. [35]), we do not take dynamic limits such as acceleration constraints into account during optimization to avoid tight coupling of the robots in the optimization. Instead, we leverage the fact that the discrete planning stage already finds a synchronized solution for the quadrotors and we scale all trajectories uniformly in a post-processing step.

The user-supplied timestep duration Δt directly affects the magnitudes of dynamic quantities such as acceleration and

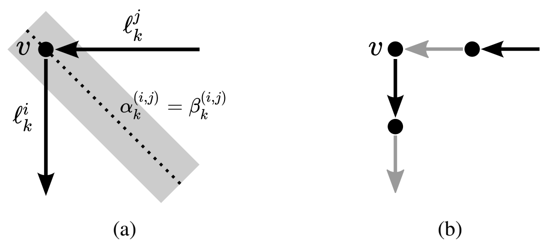


Fig. 8. Illustration of discrete plan postprocessing. (a) In timestep k, robot r^j arrives at a graph vertex v and robot r^i leaves v. The separating hyperplane between ℓ^i_k and ℓ^j_k (with ellipsoid offset shaded in grey) prevents both robots from planning a trajectory that passes through v. (b) Subdivision of discrete plan ensures that this situation cannot occur.

snap that are constrained by the robot's actuation limits. We use a binary search to find a uniform temporal scaling factor such that no trajectory f^i violates a dynamic constraint. For quadrotors, as the temporal scaling goes to infinity, the actuator commands are guaranteed to approach a hover state [17], so kinodynamically feasible trajectories can always be found.

F. Discrete Postprocessing

If the FCL collision model is used, the discrete planner might produce waypoints p^i that require some postprocessing to ensure that they satisfy the collision constraints (2) under arbitrary velocity profiles. In particular, we must deal with the case when one robot r^j arrives at a vertex $v \in \mathcal{V}_E$ in the same timestep k that another robot r^i leaves v. This situation creates a conflict where neither robot's smooth trajectory can pass through v, as illustrated in Fig. 8. We ensure that this situation cannot happen by dividing each discrete line segment in half. In the subdivided discrete plan, at odd timesteps robots exit a graph-vertex waypoint and arrive at a segment-midpoint waypoint, while at even timesteps robots exit a segment-midpoint waypoint and arrive at a graph-vertex waypoint. Under this subdivision, the conflict cannot occur. However, the increased number of timesteps requires more time to solve the quadratic program (15).

In our experiments, we noticed that the continuous trajectories typically experience peak acceleration in the vicinity of $t = 0$ and $t = T$ due to the requirement of accelerating from and to a complete stop. We add an additional wait state at the beginning and end of the discrete plans to reduce the acceleration peak.

VIII. Experiments

We implement the roadmap generation, conflict annotation, and discrete planning in C++. We use FCL [22] for collision checking, OMPL [36] for the SPARS roadmap generator, OctoMap [37] for the environment data structure, Boost Graph for maximum flow computation, and Gurobi 7.0 as ILP solver. For the swept collision model we use a quadratic program for collision checking. The continuous refinement stage is implemented in Matlab.

We use the total number of actions as a cost metric in ECBS, where move actions along an edge and wait actions have uniform cost. Whenever we use ECBS for an unlabeled planning problem, we use the Threshold algorithm to compute the goal assignment first.

When computing the robot-obstacle separating hyperplanes for each robot during each timestep, we limit the search space to a box containing the path segment to reduce computation. We expand the bounding box of the segment by $1\,\mathrm{m}$ to allow the continuous plan to deviate from the discrete plan. We consider only the nodes of the octree that intersect this box.

To compute separating hyperplanes for the safe corridors, our method requires solving $O(KN^2 + KN_{obs}N)$ small ellipsoid-weighted SVM problems. For these problems, we use the CVXGEN package [38] to generate C code optimized for the exact quadratic program specification (10). The per-robot trajectory optimization quadratic programs (15) are solved using Matlab's `quadprog` solver. Since these problems are independent, this stage can take advantage of up to N additional processor cores. In our experiments, we enforce continuity up to the fourth derivative ($C = 4$) by using a polynomial degree of $D = 7$. We evaluate our method in simulation and on the Crazyswarm — a swarm of nano-quadrotors [39].

A. Downwash Characterization

In order to determine the ellipsoid radii E, we executed several flight experiments. For r_z, we fly two Crazyflie quadrotors directly on top of each other and record the average position error of both quadrotors at $100\,\mathrm{Hz}$ for varying distances between the quadrotors. We noticed that high controller gains lead to very low position errors even in this case, but can cause fatal crashes when the quadrotors are close. We determined $r_z = 0.3\,\mathrm{m}$ to be a safe vertical distance. For the horizontal direction, we use $r_x = r_y = 0.12\,\mathrm{m}$. We set E_{obs} to a sphere of radius $0.15\,\mathrm{m}$ based on the size of the Crazyflie quadrotor.

B. Runtime Evaluation

We execute our implementation on a PC running Ubuntu 16.04, with a Xeon E5-2630 $2.2\,\mathrm{GHz}$ CPU and $32\,\mathrm{GB}$ RAM. This CPU has 10 physical cores, which improves the execution runtime for the continuous portion significantly.

Each stage of our framework can be configured and the choices influence the runtime and quality of results in the later stages. We will first use one specific example and discuss the variations at each stage. In later experiments we report the results for one specific choice on different examples.

Our example "Wall32" uses 32 quadrotors, which begin in a grid in the $x - y$ plane, fly through a wall with three windows, and form the letters "USC" in the air. For the roadmap, we report the number of vertices and edges created, the desired dispersion (d_{road}) and the runtime to create the roadmap (t_{rm}). For the conflict annotation we report the average number of conflicting vertices per vertex ($\overline{C_{\mathcal{V}\mathcal{V}}}$), average number of conflicting edges per edge ($\overline{C_{\mathcal{E}\mathcal{E}}}$), average number of conflicting vertices per edge ($\overline{C_{\mathcal{E}\mathcal{V}}}$), and runtime to annotate the roadmap (t_{conf}). For the MAPF/C solver we report the makespan of the solution (K) and the runtime to find a solution (t_{dis}), including solving the assignment problem if ECBS is used for an unlabeled instance. For the continuous trajectory planning we report the runtime of six iterations (t_{con}) and the duration of the trajectories (T).

Each stage might influence the results as follows, see Table I for example results:

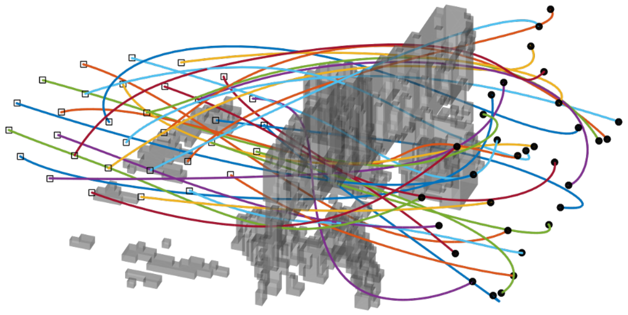
Mapping We use an occupancy grid representation of the environment, stored in an octree. Larger leaf-nodes result in a more compact data structure, but might be overly pessimistic. On the other hand, smaller leaf-nodes require more computation during the roadmap generation and trajectory optimization stages. Row 2 shows an example where a significantly higher number of leaf nodes has no effect on the discrete side, but impacts computation time on the continuous side due to the higher number of hyperplanes that must be considered.

TABLE I
INFLUENCE OF DIFFERENT PARAMETERS FOR "WALL32" PROBLEM INSTANCE, SEE SECTION VIII-B. BOLD ENTRIES INDICATE PARAMETERS CHANGED COMPARED TO ROW 1. GRAY ENTRIES INDICATE RESULTS THAT ARE NOT EXPECTED TO CHANGE COMPARED TO ROW 1.

Row	Mapping		Roadmap					Conflicts					Discrete			Continuous		
	d_{octo}	nodes	Method	d_{road}	$\lvert\mathcal{V}_E\rvert$	$\lvert\mathcal{E}_E\rvert$	t_{rm}	Method	$\overline{C_{\mathcal{VV}}}$	$\overline{C_{\mathcal{EE}}}$	$\overline{C_{\mathcal{EV}}}$	t_{conf}	Method	K	t_{dis}	iter	t_{con}	T
1	0.1	17k	Grid	0.5	978	3331	0.2	FCL	1.5	3.2	2.6	9.0	ECBS(1.5)	24	0.4	6	47	6.5
2	**0.04**	677k	Grid	0.5	978	3331	0.2	FCL	1.5	3.2	2.6	9.0	ECBS(1.5)	24	0.4	6	380	6.5
3	0.1	17k	**SPARS**	0.5	888	3495	36	FCL	0.8	7.5	1.7	9.6	ECBS(2.0)	30	1.5	6	56	11.5
4	0.1	17k	Grid	**0.2**	13k	40k	1.5	FCL	15.9	11.9	24.0	1043	ECBS(1.5)	54	10	6	103	7.7
5	0.1	17k	Grid	0.5	978	3331	0.2	**Swept**	1.5	26	2.6	0.6	ECBS(2.5)	36	1.4	6	29	8.5
6	0.1	17k	Grid	0.5	978	3331	0.2	FCL	1.5	3.2	2.6	9.0	**ILP**	20	222	6	32	5.4
7	0.1	17k	Grid	0.5	978	3331	0.2	FCL	1.5	3.2	2.6	9.0	ECBS(1.5)	24	0.4	**2**	17	9.5

TABLE II
RESULTS FOR DIFFERENT PROBLEM INSTANCES USING SPARS, ECBS AND THE SWEPT COLLISION MODEL, SEE SECTION VIII-B.

Example	labeled	N	Env. Size	occupied	Roadmap			Conflicts				Discrete		Continuous			
					$\lvert\mathcal{V}_E\rvert$	$\lvert\mathcal{E}_E\rvert$	t_{rm}	$\overline{C_{\mathcal{VV}}}$	$\overline{C_{\mathcal{EE}}}$	$\overline{C_{\mathcal{EV}}}$	t_{conf}	K	t_{dis}	$t_{1(hp)}$	$t_{1(qp)}$	t_{con}	T
Flight Test	No	32	$9 \times 5.5 \times 2.2$	4 %	873	3430	50	0.8	28	1.8	0.8	28	0.5	2.0	3.9	28	6.5
Wall32	No	32	$7.5 \times 6.5 \times 2.5$	6 %	921	3536	36	0.8	27.7	1.7	0.8	41	4.5	1.6	3.5	35	11.6
Maze50	No	50	$10 \times 6.5 \times 2.5$	30 %	1045	3221	82	1.1	24.2	2.2	0.7	48	4.4	7.3	12.3	133	16.3
Sort200	No	200	$14.5 \times 14.5 \times 2.5$	31 %	3047	7804	85	1.1	18.8	2.0	3.5	39	25	21	34	411	11.7
Swap50	Yes	50	$7.5 \times 6.5 \times 2.5$	6 %	869	3371	34	0.8	27	1.6	0.7	48	15	3.4	9.4	78	14.4


(a) Full 32-robot trajectory plan after six iterations of refinement. The start and end positions are marked by squares and filled circles, respectively.

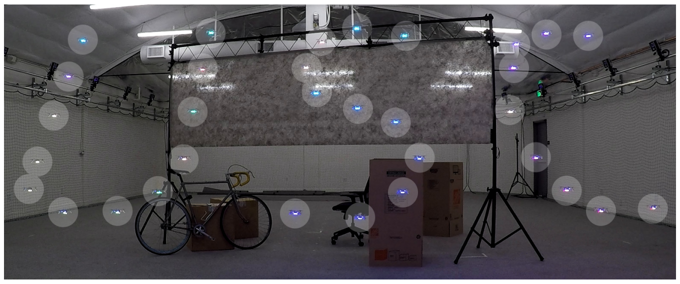

(b) Picture of the final configuration after the test flight. A video is available as supplemental material.

Fig. 9. Formation change example where quadrotors fly from a circle formation to a goal configuration spelling "USC" while avoiding obstacles.

Roadmap Generation A 6-neighbor grid is fast to compute and results in fewer conflicts during the conflict annotation stage, which in turn improves the performance of solving the MAPF/C instance. However, such a grid approach might miss narrow corridors entirely. A SPARS generated roadmap takes longer to compute and results in more conflicts due to its irregular shape, as shown in row 3. Furthermore, the non-uniform edge length might cause the discrete solver to make worse scheduling decisions because the distance traveled by each robot during a single timestep varies widely. In row 3, this caused the continuous solver to find trajectories which had to be executed much more slowly to stay within the physical limits of the quadrotor.

Lower dispersion creates denser roadmaps, which might produce better results. However, the time for conflict annotation and number of identified conflicts increase significantly. Moreover, this results in MAPF/C instances that are harder to solve in practice, see row 4.

Conflict Annotation Using the FCL collision checking model (4) results in fewer edge conflicts on average compared to the swept model, decreasing the MAPF/C solving time, as shown in row 5. However, it requires the continuous side to postprocess the discrete plan as visualized in Fig. 8, resulting in larger solving times. Moreover, we have seen various instances where the FCL model results in infeasible quadratic programs for the continuous stage, thus necessitating reversion to the piecewise linear plan [15]. The FCL collision checking model is slower compared to the swept model because it is generically implemented, supporting a wide range of possible collision models.

Discrete Solver The ILP-based solver can be used to solve the unlabeled problem optimally with respect to makespan. However, ILP takes significantly longer (see row 6) and in cases of large makespans or high number of conflicts might not find a solution in reasonable time at all. ECBS is more versatile and computes bounded suboptimal solutions very quickly. We noted that the suboptimality bound has a big impact on solution time with a smaller impact on the resulting makespan. Therefore, we used several different bounds, as noted in Table I.

Trajectory Optimization The number of refinement stages affects the quality of the trajectories. In our experiments six refinement iterations were sufficient for convergence in all cases. If trajectories are used after fewer iterations,

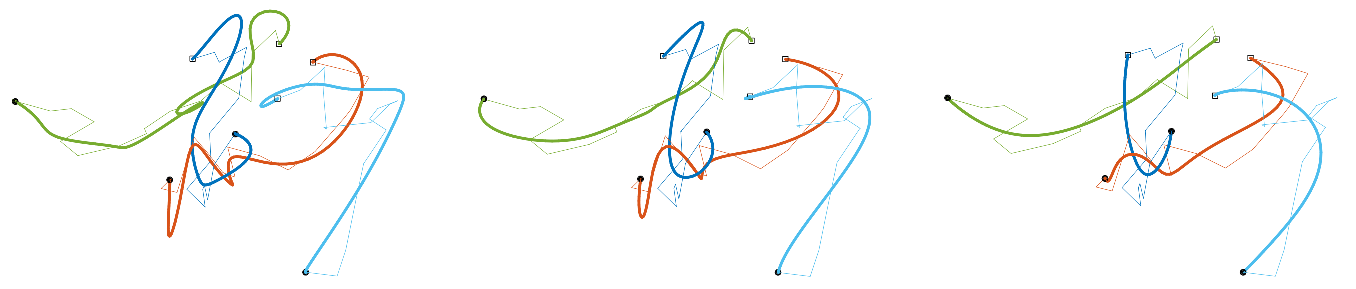


Fig. 10. Subset of results from Fig. 9 after different numbers of refinement iterations. Fine lines represent the discrete plans p^i; heavy curves represent the continuous trajectories f^i. The remaining 28 robots are hidden for clarity. Increased smoothness and directness can be observed with more iterations.

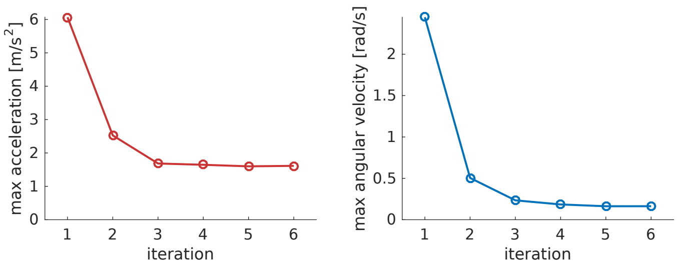


Fig. 11. Illustration of worst-case acceleration and angular velocity over all robots during six iterative refinement cycles. *Left*: peak acceleration was reduced from 6.1 to $1.6\,\mathrm{m/s^2}$. *Right*: peak angular velocity was reduced from 2.4 to $0.2\,\mathrm{rad/s}$.

the resulting time T tends to be larger because the trajectories exhibit larger acceleration terms, see row 7.

In another set of experiments, we fix the parameters and apply our method to different problem instances, see Table II. We use a set of parameters such that we are able to find a good solution quickly. However, tuning the parameters for a particular problem might result in shorter times T at a higher computational cost, as discussed before. We use a leaf-node size of $0.1\,\mathrm{m}$, the SPARS roadmap planner with an average dispersion of $0.5\,\mathrm{m}$, the swept collision model, ECBS as discrete solver and six iterations of continuous refinement. For ECBS we mostly used suboptimality bound 3.0 except for the labeled example (“Swap50”) where we used a higher suboptimality bound of 4.0. We compute plans for five different examples for 32 to 200 robots navigating in obstacle-rich environments. The most significant time portion is the roadmap generation, taking up to $85\,\mathrm{s}$. Conflict annotation and discrete solving can be done within $30\,\mathrm{s}$. The continuous refinement finds the first smooth plan in less than a minute.

C. Flight Test

We discuss the different steps of our approach on a concrete task with 32 quadrotors. In this task, the quadrotors begin on a disc in the $x-y$ plane, fly through randomly placed obstacles (e.g. boxes, bicycle, chair), and form the letters “USC” in the air. We execute the experiment in a space which is $10\,\mathrm{m} \times 16\,\mathrm{m} \times 2.5\,\mathrm{m}$ in size and equipped with a VICON motion capture system with 24 cameras.

In an initial step we use a structured light depth camera tracked by our motion capture system and the `octomap_mapping` ROS stack to map the environment using $0.1\,\mathrm{m}$ as octree resolution. We generate a roadmap within a bounding box of $9\,\mathrm{m} \times 5.5\,\mathrm{m} \times 2.2\,\mathrm{m}$ using SPARS in $50\,\mathrm{s}$, generating 873 vertices and 3430 edges. The conflict annotation and discrete planning take less than one second combined, finding discrete schedules p^i with $K = 28$. The continuous planner needs six seconds to find the first set of smooth trajectories and finishes six iterations of refinement after 28 seconds.

Fig. 11 demonstrates the effect of iterative refinement on the quadrotor dynamics required by the trajectories f^i. For each iteration, we take the maximum acceleration and angular velocity over all robots for the duration of the trajectories. Iterative refinement results in trajectories with significantly smoother dynamics. This effect is also qualitatively visible when plotting a subset of the trajectories, as shown in Fig. 10. The final set of 32 trajectories is shown in Fig. 9(a).

We upload the planned trajectories to a swarm of Crazyflie 2.0 nano-quadrotors before takeoff, and use the Crazyswarm infrastructure [39] to execute the trajectories. State estimation and control run onboard the quadrotor, and the motion capture system information is broadcasted to the UAVs for localization. Figure 9(b) shows a snapshot of the execution when the quadrotors reached their final state. The executed trajectories can be visualized with long-exposure photography, as shown in Fig. 1. The supplemental video shows the full trajectory execution.

IX. Conclusion

We present a trajectory planning method for large robot teams, combining the advantages of graph-based AI planners and trajectory optimization. We validate our approach on a real-world example of downwash-aware planning for quadrotor swarms.

Our approach creates plans where robots can safely fly in close proximity to each other. We create the trajectories using three stages: roadmap generation with inter-robot conflict annotation, discrete planning, and continuous optimization. The roadmap generation stage creates a sparse roadmap for a single robot and can use any existing algorithm for that purpose. We annotate the roadmap with generalized conflicts, describing possible inter-robot dependencies which might be violated if two robots are in close proximity to each other. We can then formulate a MAPF/C instance and either solve it optimally with respect to makespan using an ILP-based formulation, or solve it with bounded suboptimality using a search-based method. The output is a discrete schedule that, if executed, would require quadrotors to stop frequently. The continuous optimization finds smooth trajectories for each robot and is decoupled, allowing easy parallelization and improving performance for large teams.

Our approach can compute safe and arbitrarily smooth trajectories for hundreds of quadrotors in dense continuous environments with obstacles in a few minutes. The trajectory plan outputs have been tested and executed safely in numerous trials on a team of 32 quadrotors, as well as in simulations of up to 200 quadrotors.

In future work, we plan to apply our method to different kinds of robots and heterogeneous robot teams. Moreover, we plan to investigate methods to improve the performance further, for example by generating roadmaps which are better suited for multi-robot path planning. Finally, we are interested in investigating how our method can be adapted for operating in an online setting.

REFERENCES

[1] J. A. Preiss, W. Hönig, N. Ayanian, and G. S. Sukhatme, "Downwash-aware trajectory planning for large quadrotor teams," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 250–257.

[2] S. M. LaValle, *Planning algorithms*. Cambridge Univ. Press, 2006.

[3] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 3260–3267.

[4] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[5] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 591–607.

[6] W. Hönig, T. K. S. Kumar, H. Ma, S. Koenig, and N. Ayanian, "Formation change for robot groups in occluded environments," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 4836–4842.

[7] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 1917–1922.

[8] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 477–483.

[9] D. Morgan, S. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 6, pp. 1725–1740, 2014.

[10] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015, pp. 5954–5961.

[11] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of aerial robots," in *Robotics: Science and Systems (RSS)*, 2013.

[12] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *Int. Journal of Robotics Research (IJRR)*, vol. 24, no. 4, pp. 295–310, 2005.

[13] D. Bareiss and J. van den Berg, "Reciprocal collision avoidance for robots with linear dynamics using LQR-obstacles," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 3847–3853.

[14] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems (DARS)*, 2013, pp. 203–216.

[15] S. Tang and V. Kumar, "Safe and complete trajectory generation for robot teams with higher-order dynamics," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 1894–1901.

[16] M. E. Flores, "Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational b-spline basis functions," Ph.D. dissertation, California Institute of Technology, 2007.

[17] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[18] D. Yeo, E. Shrestha, D. A. Paley, and E. Atkins, "An empirical model of rotorcraft UAV downwash model for disturbance localization and avoidance," in *AIAA Atmospheric Flight Mechanics Conference*, 2015.

[19] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[20] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *Int. Journal of Robotics Research (IJRR)*, vol. 33, no. 1, pp. 18–47, 2014.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. Journal of Robotics Research (IJRR)*, vol. 30, no. 7, pp. 846–894, 2011.

[22] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 3859–3866.

[23] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012, pp. 157–173.

[24] ——, "Structure and intractability of optimal multi-robot path planning on graphs," in *AAAI Conference on Artificial Intelligence*, 2013.

[25] ——, "Planning optimal paths for multiple robots on graphs," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 3612–3617.

[26] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*, 2016, pp. 1144–1152.

[27] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *AAAI Conference on Artificial Intelligence*, 2012.

[28] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Symposium on Combinatorial Search (SOCS)*, 2014.

[29] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[30] S. R. Gunn, "Support vector machines for classification and regression," University of Southhampton, Tech. Rep., 1998.

[31] K. I. Joy, "Bernstein polynomials," *On-Line Geometric Modeling Notes*, vol. 13, 2000.

[32] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 649–666.

[33] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[34] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, vol. 89, no. 1, pp. 149–185, 2000.

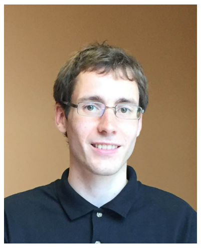
[35] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Int. Symposium of Robotic Research (ISRR)*, 2013, pp. 649–666.

[36] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, software available at http://ompl.kavrakilab.org.

[37] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com.

[38] J. Mattingley and S. Boyd, "CVXGEN: a code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[39] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017, pp. 3299–3304, software available at https://github.com/USC-ACTLab/crazyswarm.



Wolfgang Hönig is a Ph.D. student in the ACT Lab at the University of Southern California. He holds a Diploma in Computer Science from the Technical University Dresden, Germany and an M.S. in Computer Science (Intelligent Robotics) from USC. His research focuses on enabling large teams of physical robots to collaboratively solve real-world tasks by combining methods from artificial intelligence and robotics.

James A. Preiss received the B.S. in Mathematics and B.A. in Photography from The Evergreen State College (2010). He is currently pursuing a Ph.D. in Computer Science at the University of Southern California in Los Angeles. His research interests include trajectory optimization, machine learning for robot control, and multi-robot systems with an emphasis on aerial robots.

T. K. Satish Kumar leads the Collaboratory for Algorithmic Techniques and Artificial Intelligence at USC's Information Sciences Institute. He received his PhD in Computer Science from Stanford University in March 2005. His research interests include constraint reasoning, probabilistic reasoning, planning and scheduling, robotics, knowledge representation, model-based reasoning, heuristic search, algorithms and complexity, and approximation and randomization techniques.

Gaurav S. Sukhatme is Professor of Computer Science and Electrical Engineering and Gordon S. Marshall Chair in Engineering at the Viterbi School of Engineering at the University of Southern California (USC). He received his undergraduate education at IIT Bombay in Computer Science and Engineering, and M.S. and Ph.D. degrees in Computer Science from USC. He is the co-director of the USC Robotics Research Laboratory and the director of the USC Robotic Embedded Systems Laborat ory which he founded in 2000. His research interests are in robot networks with applications to environmental monitoring. He has published extensively in these and related areas. Sukhatme has served as PI on numerous NSF, DARPA and NASA grants. He was a Co-PI on the Center for Embedded Networked Sensing (CENS), an NSF Science and Technology Center. He is a fellow of the IEEE and a recipient of the NSF CAREER award and the Okawa foundation research award. He is one of the founders of the Robotics: Science and Systems conference. He was program chair of the 2008 IEEE International Conference on Robotics and Automation and the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. He is the Editor-in-Chief of Autonomous Robots and has served as Associate Editor of the IEEE Transactions on Robotics and Automation, the IEEE Transactions on Mobile Computing, and on the editorial board of IEEE Pervasive Computing.

Nora Ayanian received the M.S. and Ph.D. degrees in Mechanical Engineering and Applied Mechanics from the University of Pennsylvania, in Philadelphia, PA, USA, in 2008 and 2011, respectively.

Since 2013, she has been with the University of Southern California, Los Angeles, CA, USA as an Assistant Professor of Computer Science and Andrew and Erna Viterbi Early Career Chair (2017-present) and as a WiSE Gabilan Chair (2013-2017). From 2011-2013, she was a Postdoctoral Associate in the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Her research interests include coordination and control for multi-robot systems that can be specified with high-level inputs, with broad applications.

Prof. Ayanian is also a member of the American Society of Mechanical Engineers and the Association for Computing Machinery. She is a co-chair and co-founder of the IEEE Robotics and Automation Society Technical Committee on Multi-Robot Systems, and is Junior Chair of the 2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). She serves on the editorial board of the Swarm Intelligence Journal. She is the recipient of a 2016 National Science Foundation CAREER Award and 2016 Okawa Foundation Research Award, and was named to the 2016 MIT Technology Review 35 Innovators Under 35 (TR-35) and 2013 IEEE Intelligent Systems' "AI's 10 to Watch". She received the 2016 Outstanding Paper in Robotics at the International Conference on Automated Planning and Scheduling (ICAPS), and 2008 Best Student Paper at the IEEE International Conference on Robotics and Automation (ICRA).