Efficient Kinodynamic Multi-Robot Replanning in Known Workspaces

Arjav Desai, Matthew Collins, and Nathan Michael

***Abstract*— In this work, we consider the problem of online centralized kinodynamic multi-robot replanning (from potentially non-stationary initial states) and coordination in known and cluttered workspaces. Offline state lattice reachability analysis is leveraged to decouple the planning problem into two sequential graph searches—one in the explicit geometric graph of the environment and the other in the graph of the higher-order derivatives of the robot's state—in a manner such that the intermediate vertices of a safe set of geometric paths are guaranteed to have a feasible assignment of higher-order derivatives. Without additional iterative refinement procedures, the resulting time parameterized polynomial trajectories are dynamically feasible and collision-free. Planning results with up to 20 robots in two and three dimensional workspaces suggest the suitability of the proposed approach for multi-robot replanning in known environments.**

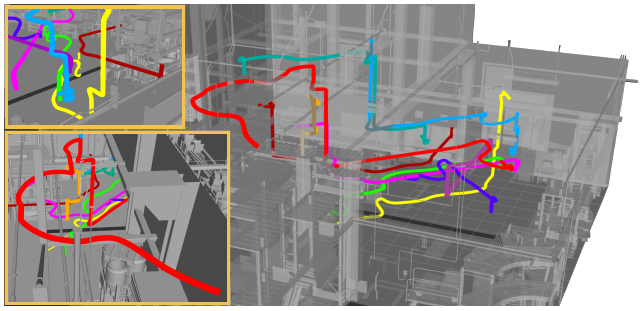


Fig. 1: Trajectories for 10 robots in a three dimensional warehouse environment from non-stationary initial states. (Top) Avoiding inter-robot collisions. (Bottom) Avoiding obstacles in warehouse. Representative video: `https://bit.ly/2IzeDde`

I. INTRODUCTION

An efficient online planning or replanning methodology is a critical requirement for safe, responsive, and robust real world multi-robot deployments. The need to replan typically stems from invalidation of existing plans due to partial knowledge of the environment (e.g., unmapped regions or dynamic obstacles) or in scenarios that necessitate online changes in the goal locations (e.g., monitoring [17] or theatrical [4] applications). This work considers the problem of multi-robot replanning from potentially non-stationary initial states, specifically, in response to online changes in goal locations in a known and static workspace.

In contrast to planning from stationary states, non-stationary initial states increase the complexity of both the motion-planning and the coordination problem. Planning dynamically feasible and collision-free trajectories from non-stationary states necessitates reasoning about the higher-order derivatives of position, increasing the dimensionality of the search space. In this case, geometric planning followed by iterative refinement as in [21] may restrict trajectories to infeasible homotopies. The coordination problem is made additionally challenging as robots cannot be assigned time offsets as in [27] in order to avoid path conflicts.

The state-of-the-art approaches in kinodynamic motion-planning can be classified into search-based [16], sampling-based [13], [14], or optimization-based methods [18], [22]. For multi-robot motion-planning, search and sampling-based methods [2], [5], [9], [15], [27] typically use coupled [24], decoupled [8], [25], or hybrid [26], [28] graph-theoretic multi-agent path-finding algorithms or variants of the RRT algorithm [12] to find an initial set of collision-free geometric paths. These geometric paths are then iteratively refined to generate feasible and safe trajectories for the multi-robot team. While the complexity of decoupled discrete methods [25], [27] scales well with the number of robots, state-of-the-art graph-theoretic multi-robot planning approaches such as [27] typically assume stationary initial states. This limits their applicability in scenarios that necessitate replanning from non-stationary states due to a loss of guarantees on feasibility and safety. The authors in [15] tackle this issue by exploring alternate geometric routes in case the original routes are deemed infeasible or unsafe; however, the evaluation of each candidate control input for dynamic feasibility and collisions with other robots restricts the applicability of this approach in online planning scenarios. Optimization-based methods [18], [20], [22] do not impose any restrictions on the initial state and find the optimal solution for the planning problem in a continuous representation of the composite state space. However, high computational complexity renders these approaches impractical for online use.

This paper presents a graph-theoretic kinodynamic planning and coordination approach for multi-robot teams in known and cluttered workspaces without assuming stationary initial states. We leverage offline state lattice reachability analysis to find geometric paths in the explicit graph of the environment in a manner that ensures a feasible assignment of higher-order derivatives at intermediate vertices of the geometric paths. The resulting polynomial trajectories are guaranteed to be dynamically feasible and collision-free without additional refinement. Within the context of the state-of-the-art in kinodynamic multi-robot motion-planning, the proposed methodology can be used as (**A**) a standalone planning or replanning strategy in sparsely cluttered envi-

The authors are affiliated with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. {arjavdesai, mcollin1, nmichael} @cmu.edu

We gratefully acknowledge support from DOE (DE-EM0004067) and industry.

ronments (Fig. 1) for certain classes of dynamical systems, (**B**) a computationally efficient method for generating informative guides in obstacle-rich environments for sampling-based methods [15], and (**C**) for seeding optimization-based methods [18], [22].

The paper is organized as follows. Section II describes the problem statement. The proposed approach is detailed in Section III and evaluated in Section IV. The conclusion and future work is discussed in Section V.

II. Problem Statement

$\mathbb{R}$ and $\mathbb{N}$ denote the set of real and natural numbers respectively. $\mathcal{S}$ represents a set and $\mathbf{M}$, $\mathbf{v}$, and s denote matrices, vectors, and scalars respectively. $|\mathcal{S}|$ refers to the cardinality of the set $\mathcal{S}$.

We consider the *homogeneous*, *labelled* [30] multi-robot kinodynamic motion-planning problem in a known workspace defined by the tuple $(\mathcal{I}, \mathcal{F}, \mathcal{W})$, where $\mathcal{I} \in \mathbb{R}^p$ and $\mathcal{F} \in \mathbb{R}^p$ denote the set of initial and final states respectively and $p \in \mathbb{N}$ is the dimensionality of the robot's state space. $\mathcal{W}$ is the known workspace. $\mathcal{W}_{\text{free}}$ and $\mathcal{W}_{\text{occ}}$ denote the set of points in the free and occupied spaces respectively, i.e., $\mathcal{W} = \mathcal{W}_{\text{free}} \cup \mathcal{W}_{\text{occ}}$.

We seek to generate a time parameterized polynomial trajectory set, Ξ, for the multi-robot team such that all trajectories in Ξ are dynamically feasible (requirement **R1**), collision-free (requirement **R2**), and terminate at the goal state (requirement **R3**).

III. Approach

This section is organized as follows:

1) Sect. III-A describes the environment representation.
2) In Sect. III-B, the one step reachability for a discrete set of higher-order derivatives, $\mathcal{S}$, is evaluated within a geometric lattice structure [19].
3) In Sect. III-C, the one step reachability information is used to construct k-step *position invariant* reachability trees; one for each state $\mathbf{s}_i \in \mathcal{S}$. At a high level, reachability trees encode the local connectivity between states in $\mathcal{S}$ induced from an initial state in an obstacle-free workspace over k time steps.
4) Sect. III-D presents the two-stage single robot planning algorithm. The first stage leverages the offline reachability information to concurrently find a geometric path in the explicit graph of the environment and construct a graph in the space of the higher-order derivatives of position (we refer to this graph as the *derivative graph*). In the second stage, higher-order derivatives are assigned to the vertices of the geometric path by searching for a path in the derivative graph that minimizes a predefined cost metric.
5) In Sect. III-E, a prioritized, cooperative planning strategy (similar to [25]) is used to extend the single robot planning algorithm for the multi-robot case.

A. Environment Representation

A k_L connected state lattice structure (e.g., an 8-connected lattice as shown in Fig. 2) is translated across the workspace $\mathcal{W}$ to create an explicit geometric workspace graph $G_{\mathcal{W}} = (V_{\mathcal{W}}, E_{\mathcal{W}})$. $V_{\mathcal{W}}$ represents the set of vertices within free space and $E_{\mathcal{W}}$ denotes the set of edges. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V_{\mathcal{W}}| \times |V_{\mathcal{W}}|}$ encodes the weighted connectivity between the vertices in $V_{\mathcal{W}}$ along the edges in $E_{\mathcal{W}}$. The start ($\mathcal{I}^i_{\text{pos}}$) and goal ($\mathcal{F}^i_{\text{pos}}$) positions of the ith robot are assumed to lie within the same connected component [29] of $G_{\mathcal{W}}$.

B. Offline Reachability Analysis

Let $\mathcal{S}$ denote the discretized feasible set of higher-order derivatives in the state-space of the robot. For an acceleration controlled system, $\mathcal{S}$ consists of feasible velocities; for a jerk controlled system, $\mathcal{S}$ consists of feasible velocities and accelerations.

For each initial state $\mathbf{s}_i \in \mathcal{S}$ and for each edge in the lattice, $|\mathcal{S}|$ time parameterized polynomial trajectories of a fixed time duration, t_{fixed}, are generated from $\mathbf{s}_i$ to the final state $\mathbf{s}_j$, $\forall j \in \{1, \ldots, |\mathcal{S}|\}$.

Each polynomial trajectory is evaluated for violation of differential (constraint $\mathcal{C}_1$) and curvature constraints (constraint $\mathcal{C}_2$). The curvature constraints are imposed by a rectangular convex region of fixed width around each edge in the lattice (Fig. 2). While constraint $\mathcal{C}_1$ ensures dynamic feasibility, constraint $\mathcal{C}_2$ is leveraged in the multi-robot planner (Sect. III-E) for efficient collision checking. $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2\}$ denotes the cumulative constraint set.

For each $\mathbf{s}_i \in \mathcal{S}$, a cost matrix $\mathbf{C} \in \mathbb{R}^{|k_L+1| \times |\mathcal{S}|}$ and an adjacency vector $\mathbf{a} \in \{0,1\}^{k_L+1}$ is maintained. The cost matrix captures the cost of transitioning from state $\mathbf{s}_i$ to $\mathbf{s}_j$ ($\forall j \in \{1, \ldots, |\mathcal{S}|$) along an edge in the lattice structure. Let ξ^k_{ij} be the polynomial coefficients of the trajectory connecting $\mathbf{s}_i$ to $\mathbf{s}_j$ along the kth edge. The cost matrix is updated as follows:

$$\mathbf{C}(k,j) = \begin{cases} c(\mathbf{s}_i, \mathbf{s}_j), & \text{if } f_{\text{eval}}(\xi^k_{ij}, \mathcal{C}) = 1 \\ \infty, & \text{if } f_{\text{eval}}(\xi^k_{ij}, \mathcal{C}) = 0 \end{cases} \tag{1}$$

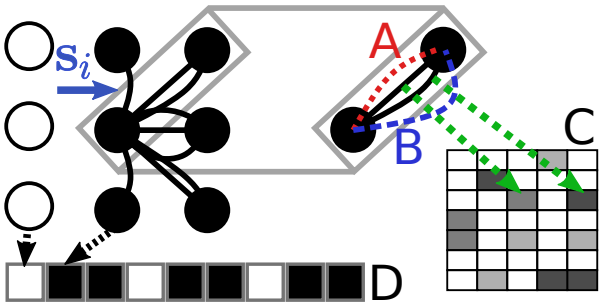


Fig. 2: Polynomial trajectories are generated from a fixed state $\mathbf{s}_i \in \mathcal{S}$ to $\mathbf{s}_j \in \mathcal{S}$ $\forall j \in \{1, \ldots |\mathcal{S}|\}$ within the geometric lattice structure. Trajectories that violate differential constraint $\mathcal{C}_1$ (A) or curvature constraint $\mathcal{C}_2$ (B) are deemed infeasible. The trajectory costs are stored in the $\mathbf{C}$ matrix and the one step adjacency is stored in the boolean valued vector $\mathbf{a}$ (D), where reachable vertices are denoted in black.

where $c(\mathbf{s}_i, \mathbf{s}_j)$ is the trajectory cost function and $f_{\text{eval}} \rightarrow \{0, 1\}$ checks the polynomial trajectory ξ_{ij}^k for violation of the constraint set $\mathcal{C}$. The total control effort along the polynomial trajectory (e.g., acceleration, jerk, etc. depending on the control input for the system) is used as a measure of cost. The one step adjacency that an initial state $\mathbf{s}_i$ induces within the lattice structure is stored in the boolean valued adjacency vector $\mathbf{a}$.

A library L_i stores the cost matrix $\mathbf{C}$ and the adjacency vector $\mathbf{a}$ for each $\mathbf{s}_i \in S$. We denote the set of all libraries by $\mathcal{L} = \{L_1, \ldots, L_{|\mathcal{S}|}\}$.

C. Offline k-Step Forward Reachability Tree Construction

A *position invariant* k-step reachability tree is constructed for each library in the library set by evaluating its forward reachability over a k-step horizon in an obstacle-free unbounded workspace.

Each reachability tree T_i is defined by the tuple $(\mathbf{A}_{T_i}, \mathbf{R}_{T_i})$. The matrix $\mathbf{A}_{T_i} \in \mathbb{R}^{|V_T| \times |V_T|}$ is the geometric adjacency matrix that state $\mathbf{s}_i$ induces in an obstacle-free environment over k time steps. Here, V_T represents the origin-centered time-indexed reachable position vertices in an obstacle-free workspace over k time steps assuming a stationary initial state. $\mathbf{R}_T \in \{0, 1\}^{|V_T| \times |\mathcal{S}|}$ denotes the boolean valued *reachability* matrix that maps the vertices in V_T to reachable states in $\mathcal{S}$ given the initial (root) state $\mathbf{s}_{\text{init}}$. The set of all the k-step reachability trees is denoted by $\mathcal{T} = \{T_1, \ldots, T_{|\mathcal{S}|}\}$.

Fig. 3 illustrates the rationale behind the k-step reachability evaluation. Let $\mathbf{s}_i \in \mathcal{S}$ be the higher-order derivatives associated with the current position $\mathbf{x}$ of the robot and $\mathbf{A}_T^{\mathbf{s}_i}$ be the corresponding k-step adjacency. Let $\mathbf{A}_{\mathcal{W}}^{\mathbf{x}}$ be the adjacency matrix of the k-step graph of the workspace $\mathcal{W}$ centered around $\mathbf{x}$. While all the edges in $\mathbf{A}_{\mathcal{W}}^{\mathbf{x}}$ are collision-free, some edges might be dynamically infeasible given $\mathbf{s}_i$ and constraint $\mathcal{C}_1$. The adjacency matrix, $\mathbf{A}^{\mathbf{x}}$, corresponding to the k-step graph centered at $\mathbf{x}$ that contains dynamically feasible *and* collision-free edges is obtained by taking the element-wise product of $\mathbf{A}_{\mathcal{W}}^{\mathbf{x}}$ and $\mathbf{A}_T^{\mathbf{s}_i}$:

$$\mathbf{A}^{\mathbf{x}} = (\mathbf{A}_{\mathcal{W}}^{\mathbf{x}} \odot \mathbf{A}_T^{\mathbf{s}_i})_{ij}. \tag{2}$$

Using (2) allows for efficiently obtaining the set of dynamically feasible and safe geometric edges over multiple time steps in the single and multi-robot planners.

D. Single Robot Planner

An iterative, receding horizon planning strategy is used for the single robot planner. Let $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{final}}$ denote the initial and final positions and $\mathbf{s}_{\text{init}}$ and $\mathbf{s}_{\text{final}}$ denote the initial and final higher-order derivatives. Algorithm 1 describes the single robot planning procedure.

In each iteration, a k-step tree $\mathcal{T}_{\text{active}}$ is constructed from the k-step trees corresponding to all the available (active) libraries ($\mathcal{L}_{\text{active}}^{\mathbf{x}_{\text{curr}}}$) at the current position vertex. In the first iteration $\mathcal{T}_{\text{active}}$ corresponds to the k-step tree of $\mathbf{s}_{\text{init}}$ (line 4). In each iteration, $V_{\mathcal{T}_{\text{active}}}$ is translated by the robot's current position into the world frame and the workspace constraints are incorporated into $\mathbf{A}_{\mathcal{T}_{\text{active}}}$ (line 7) using (2).

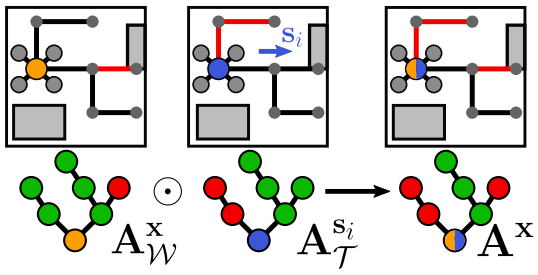


Fig. 3: The k-step graph in $\mathbb{R}^d$ with dynamically feasible and collision-free edges, $\mathbf{A}^x$, can be obtained by taking the element-wise product of $\mathbf{A}_{\mathcal{W}}^x$ (which encodes the collision constraints) and $\mathbf{A}_{\mathcal{T}}^{s_i}$ (which encodes the feasibility constraints).

The adjacency and reachability matrices of $\mathcal{T}_{\text{active}}$ are queried to check if the goal state ($\mathbf{x}_{\text{final}}$, $\mathbf{s}_{\text{final}}$) is reachable within the k-step horizon (line 8). If the goal is reachable, the minimum cost path [6] is found from the root node and the geometric planning terminates (line 10).

If the goal state is unreachable within the k-step horizon, a hierarchical graph $G_H = (V_H, E_H)$ is constructed that considers the dynamic feasibility and the workspace collision constraints for the first k time steps and only the workspace collision constraints after the kth step (line 16). The vertex in V_H after the kth time step that corresponds to $\mathbf{x}_{\text{final}}$ is chosen as the goal vertex and the shortest path is found from the root node. The first k steps of this path are retained, providing a partial geometric path P_{iter}. If no path exists, the algorithm reports failure and terminates.

Once a partial geometric path P_{iter} is found for the planning iteration, the active libraries $\mathcal{L}_{\text{active}}$ at the root vertex are propagated along the edges of the partial path to update a directed *derivative graph* $G_{\mathcal{D}} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ (line 12, 18). $\mathcal{L}_{\text{active}}$ is reinitialized to include the active libraries associated with the last vertex of the partial path (line 20). The vertices of $G_{\mathcal{D}}$ are the time-indexed, higher-order derivatives, and the edges encode the connectivity between these higher-order derivatives over successive time instances. The edge weights between pairs of vertices are queried from the relevant cost matrices computed in section III-B.

Instead of selecting a library from the active library set after each planning iteration, all the k-step trees from the libraries in $\mathcal{L}_{\text{active}}$ are *merged* (line 21). Since the adjacency matrix of the k-step trees encode the dynamics of the underlying state, the resultant tree ($\mathcal{T}_{\text{active}}$) will maintain all the higher-order information from the active libraries used for its construction. This biases the construction of the geometric tree in a direction of dynamic feasibility, ensuring the existence of a continuous kinodynamic path if a geometric solution can be found. The merge operation updates $\mathbf{R}_{\mathcal{T}_{\text{active}}}$ and $\mathbf{A}_{\mathcal{T}_{\text{active}}}$ according to (3).

Algorithm 1 Find a dynamically feasible and safe trajectory ξ, given the initial and final positions, $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{final}}$ and derivatives, $\mathbf{s}_{\text{init}}$ and $\mathbf{s}_{\text{final}}$ respectively.

1: **procedure** SINGLEROBOTPLANNER($\mathbf{x}_{\text{init}}$, $\mathbf{x}_{\text{final}}$, $\mathbf{s}_{\text{init}}$, $\mathbf{s}_{\text{final}}$)
2: initialize empty path P
3: initialize empty derivative graph $G_{\mathcal{S}}$
4: set $\mathcal{L}_{\text{active}}$ to L_{init} and $\mathcal{T}_{\text{active}}$ to T_{init}
5: path_found $\leftarrow$ False
6: **while** iter $<$ max_iter **do**
7: incorporate workspace constraints in $\mathcal{T}_{\text{active}}$ using (2)
8: goal_found $\leftarrow$ is goal state ($\mathbf{x}_{\text{final}}$, $\mathbf{s}_{\text{final}}$) is in $\mathcal{T}_{\text{active}}$
9: **if** goal_found **then**
10: $P_{\text{iter}} \leftarrow$ find shortest path to goal
11: path_found $\leftarrow$ True
12: $G_{\mathcal{S}} \leftarrow$update derivative graph using $P_{\text{iter}}, \mathcal{L}$
13: append P_{iter} to P
14: **break**
15: **else**
16: $G_H \leftarrow$construct hierarchical graph using $\mathcal{T}_{\text{active}}$
17: $P_{\text{iter}} \leftarrow$ find shortest path to goal in G_H
18: $G_{\mathcal{S}} \leftarrow$update derivative graph using $P_{\text{iter}}, \mathcal{L}$
19: append P_{iter} to P
20: reset $\mathcal{L}_{\text{active}}$ by propagating current $\mathcal{L}_{\text{active}}$ along P_{iter}
21: reinitialize new $\mathcal{T}_{\text{active}}$ using (3)
22: update $\mathbf{x}_{\text{init}} \leftarrow P_{\text{iter}}(\text{end})$
23: **if** path_found **then**
24: $D \leftarrow$ find shortest path in derivative graph $G_{\mathcal{S}}$
25: $\xi \leftarrow$ compute polynomial trajectory using P, D, t_{fixed}
26: **return** ξ
27: **else**
28: **return** $\emptyset$

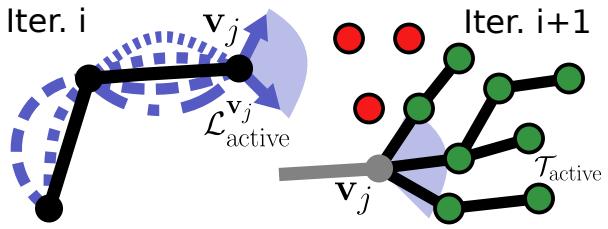


Fig. 4: Once a partial geometric path (shown in black) is found for the ith iteration, the k-step trees corresponding to the available libraries at the last vertex of the partial path (Left) are merged to construct one k-step tree for the subsequent iteration. This biases the geometric tree within a dynamically feasible cone (Right) to ensure the existence of a continuous kinodynamic path.

The merge operation allows for the deferment of the assignment of higher-order derivatives until after a safe geometric path is found. Afterwards, the higher-order derivatives are assigned to each vertex by finding the least cost path in the derivative graph $G_{\mathcal{D}}$.

$$\mathbf{R}_{\mathcal{T}_{\text{active}}}(i,j) = \begin{cases} 1, \text{ if } \exists\ \mathbf{R}_{\mathcal{T}_{\mathcal{L}_k}}(i,j) = 1\ \forall \mathcal{L}_k \in \mathcal{L}_{\text{active}} \\ 0, \text{ otherwise} \end{cases}$$
$$\mathbf{A}_{\mathcal{T}_{\text{active}}}(i,j) = \begin{cases} 1, \text{ if } \exists\ \mathbf{A}_{\mathcal{T}_{\mathcal{L}_k}}(i,j) = 1\ \forall \mathcal{L}_k \in \mathcal{L}_{\text{active}} \\ \infty, \text{ otherwise} \end{cases} \quad (3)$$

Algorithm 2 Find a dynamically feasible and safe polynomial trajectory set Ξ for a robot team R given the initial and final states $\mathcal{I}$ and $\mathcal{F}$ respectively.

1: **procedure** MULTIROBOTPLANNER(I, F)
2: **for** $\forall r_i \in R$ **do**
3: initialize empty path P^i
4: initialize empty derivative graph $G^i_{\mathcal{S}}$
5: set $\mathcal{L}^i_{\text{active}}$ to L^i_{init} and $\mathcal{T}^i_{\text{active}}$ to T^i_{init}
6: path_found_i $\leftarrow$ False
7: **while** iter $<$ max_iter **do**
8: $\mathcal{O} \leftarrow$ assign priority as in III-E.1
9: **for** $\forall i \in \mathcal{O}$ **do**
10: **Algorithm 1**: lines 7-23
11: **if** $\forall$ path_found_i = True **then**
12: **for** $\forall r_i \in R$ **do**
13: $D^i \leftarrow$ find shortest path in derivative graph $G^i_{\mathcal{S}}$
14: $\xi^i \leftarrow$ compute trajectory using $P^i, D^i, t_{\text{fixed}}$
15: $\Xi \leftarrow$ Append ξ^i
16: **return** Ξ
17: **else**
18: **return** $\emptyset$

E. Multi-Robot Planner

A prioritized, cooperative planning strategy (similar to [25]) is used for the multi-robot case. In each iteration, a priority ordering (discussed in III-E.1) $\mathcal{O}$ is computed for the robot team and k-step paths are computed sequentially for each robot in a manner similar to the single robot planner described in Algorithm 1 (lines 6-24). The multi-robot algorithm is summarized in Algorithm 2.

1) Priority Assignment: In order to compute $\mathcal{O}$, the set of robots R is partitioned into two disjoint subsets: R_{s} (stationary) and R_{ns} (non-stationary). If the active library set $\mathcal{L}^{r_i}_{\text{active}}$ of robot r_i contains the rest state—having zero higher-order derivatives—r_i is added to the R_{s} set. Otherwise, the robot r_i is added to the R_{ns} set. All the robots in R_{ns} are assigned a higher priority than the robots in R_{s}. Within R_{ns}, robots are assigned priority based upon their lowest achievable speed, where robots that are bounded to move faster have higher priority due to lower maneuverability from having fewer available edges. The robots in R_{s} are assigned priority according to the length of the shortest geometric path (ignoring robot interactions) from their position at the beginning of an iteration to their respective goal positions, with robots needing to cover longer distances having higher priority. Searching the workspace graph $G_{\mathcal{W}}$ using standard graph search techniques [6] provides the shortest path for each robot.

2) Hierarchical Graph Construction: We note one difference in the hierarchical graph construction step. For a robot r_i, the partial geometric paths of robots with a higher priority are queried, and the edges in the first k time steps from the hierarchical graph of robot r_i that intersect with these partial paths are removed. Since the offline reachability analysis constrains the curvature of the resulting trajectories to a rectangular convex hull of fixed width (constraint $\mathcal{C}_2$), the edges in collision with higher priority paths can be pruned using a line-polytope intersection algorithm (e.g., [7]).

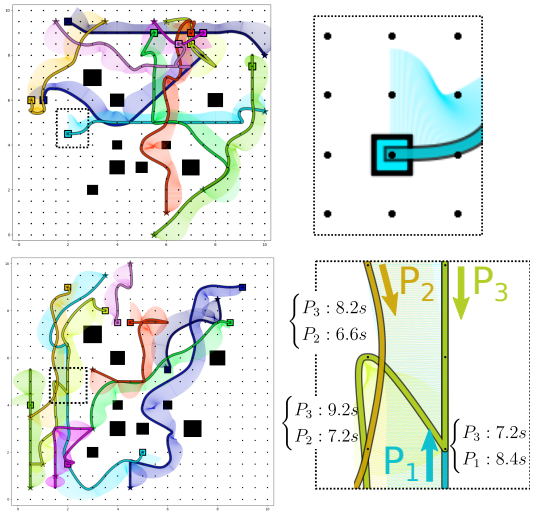


Fig. 5: Representative experiments conducted in **S1**. The two figures on the left show examples of the resulting polynomial trajectories for 10 robots w/ non-stationary starts in a 10×10 m^2 environment. (Top Right) Inset showing the non-zero initial velocity profile. (Bottom Right) Inset showing three robots (priorities P_1 through P_3) navigating the environment and avoiding collision. Times of arrival are shown for each robot at trajectory intersections.

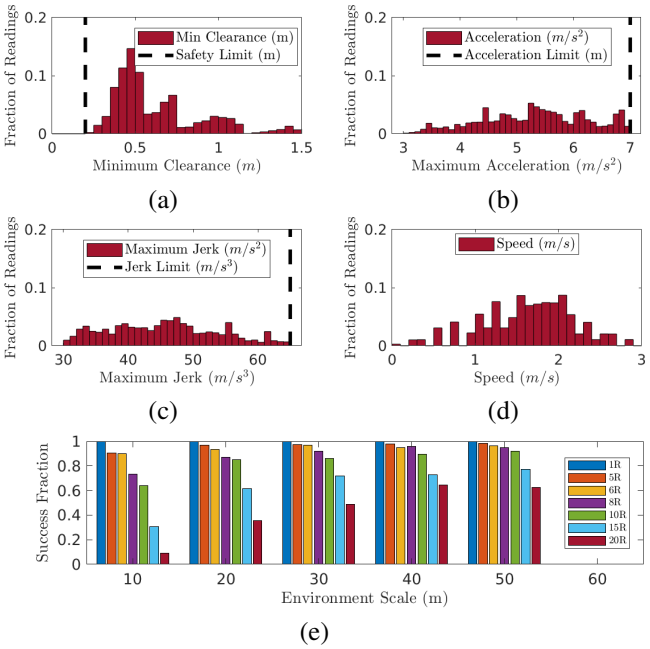


Fig. 6: **S1**: Tests employ robots of radius 0.1 m, an acceleration limit of 7 m/s^2, and jerk limit of 65 m/s^3. Sub-figures (a), (b), and (c) show the distribution of inter-robot clearance distances, maximum accelerations, and jerks, while sub-figure (d) shows the distribution of initial speeds for all experiments in which geometric paths were successfully found. Sub-figure (e) shows the variation in success rates for $\forall n_i \in N$ against the environment scale. Results verify the claim that if a set of geometric paths is found the resulting trajectories are both feasible (**R1**) and safe (**R2**).

IV. Evaluation

A. Implementation Details and Experimental Design

The single and multi-robot planners were implemented in Julia [1] and evaluated on a Lenovo Thinkpad with a Intel 4-Core 2.80GHz i7 CPU and 16 GB RAM.

The proposed method is qualitatively and quantitatively evaluated by conducting two studies. For all evaluations, a tree depth (Sect. III-C) of 3 was used. Study **S1** experimentally shows that if the planner successfully finds a solution (i.e., a set of geometric paths) for a multi-robot planning problem (Sect. II), the resulting trajectories are dynamically feasible and collision-free. Study **S2** characterizes the variation in the memory requirements, computational complexity, and the solution quality of the proposed approach for different cardinalities of the discrete set of higher-order derivatives, $\mathcal{S}$. Sect. IV-D discusses the completeness of the single and multi-robot planning algorithms. In Sect. IV-E, we compare our algorithm to the multi-robot variants of state-of-the-art search [16] and sampling-based [23] algorithms. Finally, in Fig. 8 qualitative results of the proposed approach applied to online multi-robot replanning are shown in a realistic three dimensional environment.

B. Dynamic Feasibility and Safety

For **S1** (Fig. 6), 1000 multi-robot planning experiments are conducted for each $n_i \in N = \{1, 5, 6, 8, 10, 15, 20\}$ robots in two dimensional environments with different obstacle densities and scales (minimum scale 10×10 m^2, maximum scale 50×50 m^2, lattice resolution 0.5 m). The results verify that if the multi-robot planner successfully finds a set of collision-free geometric paths for the multi-robot team, the resulting polynomial trajectories are dynamically feasible and collision-free.

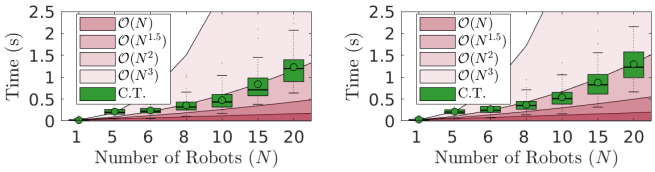
C. Computational Complexity

For **S2** (Table I, Fig. 7), 200 multi-robot planning experiments are conducted for different cardinalities of the discrete set of higher-order derivatives ($|\mathcal{S}| = \{81, 289, 1089, 1681\}$) for each $n_i \in N$. Table I compares the percentage variation in the solution cost (total jerk across all trajectories) and the planning time baselined against $|\mathcal{S}| = 81$ for 5 robots. The baseline implementation of the proposed approach, as described in Sect. III-E, leads to roughly a 200 % increase in the planning time for $|\mathcal{S}| = \{1089, 1681\}$, and is primarily attributed to the large matrix operations in (3).

To reduce the computational cost, only a sub-sample of the active states are chosen at each vertex along the partial path (Algorithm 1, line 18). A kd-tree is constructed from the active state velocities at each vertex and the n closest vertices to the weighted expected velocity for the next two steps are chosen. This sub-sampling leads to a significant reduction in the planning time. For the experiments in **S1** with $|\mathcal{S}| = 289$, a maximum of 10 libraries were selected, leading to a $\sim 90\%$ reduction in the planning time, while only increasing the solution cost by approximately 7 %. There is a planning time reduction between $|\mathcal{S}| = 81$ and $|\mathcal{S}| = 289$, as there are more reachable states allowing for connections to more vertices in the tree. Scaling further, however, increases the

TABLE I: Mean percentage variation (+:increase, -:decrease) in the solution cost (S.C.) and the planning time (P.T) for different cardinalities of $\mathcal{S}$ compared against $|\mathcal{S}| = 81$ for 200 experiments with 5 robots. We assume an acceleration controlled system with $v_{\max} = 2.82$ m/s to construct $\mathcal{S}$.

Card($\mathcal{S}$)	289	1089	1681
Memory requirements for $\mathcal{L}$ and $\mathcal{T}$	5.8MB	109MB	231.4MB
Offline computation time	70 s	1 hour	3 hours
Pct. change in P.T. (**w/o sub-sampling**)	90.53	211.5	214.9
Pct. change in P.T. (**w/ sub-sampling**)	-1.83	14.66	12.88
Pct. change in S.C. (**w/o sub-sampling**)	-23.19	-31.24	-31.39
Pct. change in S.C. (**w/ sub-sampling**)	-20.06	-23.94	-23.69



(a) (b)

Fig. 7: **S2**: Sub-figures show characterization of computational complexity and scaling of computation times (C.T.) for different cardinalities of the discrete set $\mathcal{S}$: (a) 289 and (b) 1681 with 10 sub-sampled states. The complexity of the proposed approach is approximately $\mathcal{O}(N^2)$ for up to 20 robots. The planning times (< 1 s for up to 10 robots) suggest the viability of the proposed approach for online replanning.

time as finer discretization results in similar states that are chosen by the kd-tree. This leads to redundancy rather than new information about available connections in the tree. Fig. 7 shows the absolute computation times for $|S| = \{81, 289, 1089, 1681\}$ (w/ sub-sampled states) as well as the scaling of algorithm complexity (approximately $\mathcal{O}(N^2)$ for $N \leq 20$) for different team sizes. These results suggest the viability of the proposed approach for online multi-robot planning in known, cluttered, and static workspaces.

D. Algorithm Completeness

The single robot planner is resolution complete with respect to the constraint set $\mathcal{C}$ and discretization of the environment and the state space. This guarantee stems from the completeness of Dijkstra's algorithm [6] and the merge operation 3—shown in Algorithm 1, line 21—where each planning iteration captures the geometric k-step adjacencies induced by all possible higher-order derivatives at a geometric vertex given the initial state of the robot. For the case of multi-robot planning, this method is incomplete due to the use of a prioritized strategy [3].

E. Comparison with Search and Sampling-Based Techniques

We compare our approach (approach **A1**) to two decoupled and prioritized multi-robot extensions: (A) a search-based motion-planning approach using motion primitives[1] [16] (approach **A2**) that discretizes the control input space as opposed to the state space as in our approach and (B) a sampling-based approach that uses the differential fast marching tree algorithm [23] (approach **A3**). For this comparison, 18 primitives were used per robot for approach **A2** and 1000 nodes were used per robot for approach **A3**.

[1] https://github.com/sikang/mpl_ros

TABLE II: Comparison results for success rate, computation times, and solution cost. 100 experiments were conducted for team sizes of 1, 5, and 10 in 10 randomly generated 10×10 m^2 workspaces.

N	Success %			Avg. Time (s)			Avg. Cost		
	A1	A2	A3	A1	A2	A3	A1	A2	A3
1	**100**	91	95	**0.006**	0.021	0.046	18.4	**6.27**	11.6
5	**98**	81	85	**0.102**	0.522	0.244	88.9	**30.3**	57.1
10	**82**	77	71	**0.349**	1.10	0.702	189.3	**60.4**	116

As summarized in Table II, our approach achieves lower computation times and higher success rates. However, the resulting paths incur a higher cost (approximately 3 times the cost of [16]). For this comparison, an acceleration controlled system is assumed and the cumulative control effort for the entire robot team is used as a measure of cost. The high cost in the proposed approach is primarily attributed to trajectories being constrained to the spatial lattice.

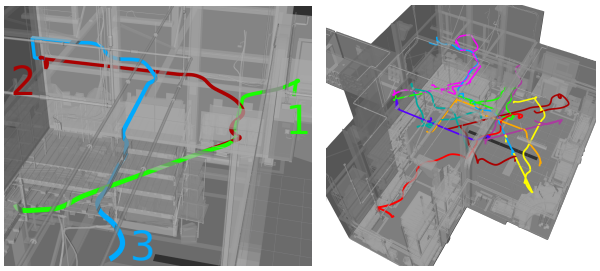


Fig. 8: Replanning with non-stationary initial states for 10 robots in a 3D workspace. (Left) Two replanning instances for one robot starting at the bottom left corner and planning to the goal indicated by the number for the planning period (green to red to blue). (Right) The final trajectories for 10 robots over two replanning instances. Each robot is represented by a distinct color. The average replanning time for 10 robots in the 3300 m^3 warehouse environment with $11,000$ vertices and $|S| = 1089$ was approximately one second.

V. Conclusion and Future Work

This paper presents a computationally efficient method for motion-planning of multi-robot teams from non-stationary initial states in known, static environments. Results show that the proposed approach generates feasible and collision-free trajectories for up to 10 robots from non-stationary states in under one second, suggesting its viability for online planning or replanning.

Since there exists a trade-off between the solution cost and memory usage as the discretization for constructing the set of higher-order derivatives increases, we intend to explore more efficient set construction techniques—identifying representative states that express the reachability of similar states—to better select or generate states. Another avenue that we plan to explore is how the proposed approach can be combined with search or sampling-based methods that discretize the control input space [15], [16] for efficient single robot local planning in partially known [10], [11] and dynamic environments.

REFERENCES

[1] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

[2] James Bruce and Manuela Veloso. Real-time multi-robot motion planning with safe dynamics. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 159–170. Springer, 2005.

[3] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, 12(3):835–849, 2015.

[4] Ellen A Cappo, Arjav Desai, Matthew Collins, and Nathan Michael. Online planning for human–multi-robot interactive theatrical performance. *Autonomous Robots*, pages 1–16, 2018.

[5] Marcello Cirillo, Tansel Uras, and Sven Koenig. A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 232–239. IEEE, 2014.

[6] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[7] David P Dobkin and David G Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[8] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[9] Wolfgang Hönig, TK Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *ICAPS*, pages 477–485, 2016.

[10] Lucas Janson, Tommy Hu, and Marco Pavone. Safe motion planning in unknown environments: Optimality benchmarks and tractable policies. *arXiv preprint arXiv:1804.05804*, 2018.

[11] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[12] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. Institute of Electrical and Electronics Engineers, 2011.

[13] Steven M LaValle and Seth A Hutchinson. Optimal motion planning for multiple robots having independent goals. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2847–2852. IEEE, 1996.

[14] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[15] Duong Le and Erion Plaku. Multi-robot motion planning with dynamics guided by multi-agent search. In *IJCAI*, pages 5314–5318, 2018.

[16] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2872–2879. IEEE, 2017.

[17] Kai-Chieh Ma, Zhibei Ma, Lantao Liu, and Gaurav S Sukhatme. Multi-robot informative and adaptive planning for persistent environmental monitoring. In *Distributed Autonomous Robotic Systems*, pages 285–298. Springer, 2018.

[18] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 477–483. IEEE, 2012.

[19] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[20] James A Preiss, Wolfgang Hönig, Nora Ayanian, and Gaurav S Sukhatme. Downwash-aware trajectory planning for large quadrotor teams. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 250–257. IEEE, 2017.

[21] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[22] D Reed Robinson, Robert T Mar, Katia Estabridis, and Gary Hewer. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. *IEEE Robotics and Automation Letters*, 3(2):1215–1222, 2018.

[23] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 2574–2581. IEEE, 2015.

[24] Micha Sharir and Shmuel Sifrony. Coordinated motion planning for two independent robots. *Annals of Mathematics and Artificial Intelligence*, 3(1):107–130, 1991.

[25] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[26] Kiril Solovey, Oren Salzman, and Dan Halperin. Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning. In *Algorithmic Foundations of Robotics XI*, pages 591–607. Springer, 2015.

[27] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4):401–415, 2014.

[28] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[29] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[30] Jingjin Yu and Steven M LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.