Implan: Scalable Incremental Motion Planning for Multi-Robot Systems

Indranil Saha[1], Rattanachai Ramaithitima[2],
Vijay Kumar[2], George J. Pappas[2], Sanjit A. Seshia[3]
[1] Department of Computer Science and Engineering, Indian Institute of Technology Kanpur
[2]GRASP Lab, University of Pennsylvania
[3]Department of Electrical Engineering and Computer Science, University of California Berkeley
Email: isaha@cse.iitk.ac.in, {ramar, kumar, pappasg}@seas.upenn.edu, sseshia@eecs.berkeley.edu

***Abstract*—We consider the collision-free motion planning problem for a group of robots using a library of motion primitives. To cope with the complexity of the problem, we introduce an incremental algorithm based on an SMT solver, where we divide the robots into small groups based on a priority assignment algorithm. The priority assignment algorithm assigns priorities to the robots in such a way that the robots do not block the cost-optimal trajectories of the other robots. While the priority assignment algorithm attempts to assign distinct priorities to the robots, the algorithm ends up with assigning the same priority to some robots due to the dependencies among themselves. The algorithm includes the robots with the same priority in the same group. Our incremental algorithm then considers the robot groups one by one based on their priority and synthesizes the trajectories for the group of robots together. While synthesizing the trajectories for the robots in one group, the algorithm considers the higher priority robots as dynamic obstacles, and introduces a minimal delay in executing the cost-optimal trajectories to avoid collision with the higher priority robots. We apply our method to synthesize trajectories for a group of quadrotors in our lab space. Experimental results show that we can synthesize trajectories for tens of robots with complex dynamics in a reasonable time.**

I. Introduction

A major outstanding problem in cyber-physical system (CPS) design is the effective programming of large teams ("swarms") of autonomous systems that must perform coordinated tasks. In particular, many applications of multi-robot systems, in disaster management, law enforcement, exploration of unknown environments, etc., fall into this category. In all these applications, a group of robots need to move from their initial locations to their final locations while avoiding obstacles and collision with each other. Though the multi-robot motion planning problem has been studied extensively in the past for robots without modeling dynamics or with only simple models of dynamics [9], [3], [15], [23], [22], [6], [16], [26], [4], [5], there is no effective solution that is applicable for a large group of robots with complex dynamics.

In this paper, we present an approach based on satisfiability modulo theories (SMT) [1] to solve the motion planning problem for large- scale multi-robot systems. Our objective is to dispatch all the robots to their designated destinations following their cost-optimal trajectories, where the cost of a path for a robot represents the total energy consumption by the robot to reach its destination. One way of achieving this is to synthesize the cost-optimal trajectory for each robot independently, and then dispatch the robots one-by-one following their cost-optimal trajectories. However, one robot's optimal trajectory may be blocked by the initial or the final location of another robot. For example, Figure 1(a) presents an instance of multi-robot motion planning where robot R_1 has to move from its initial location $I(R_1)$ to its final location $F(R_1)$, and robot R_2 has to move from its initial location $I(R_2)$ to its final location $F(R_2)$. If we dispatch robot R_1 first, it may block the (optimal) trajectory of robot R_2. As the example in Figure 1(a) suggests, the core challenge of incremental multi-robot motion planning is to find an ordering of the robots that will enable us to dispatch all the robots to their destination one-by-one following their optimal trajectory, if feasible.

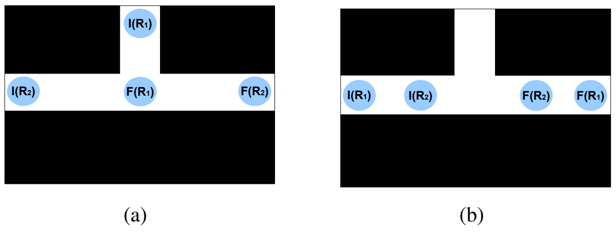


Fig. 1. Instances of multi-robot motion planning problem

A total ordering of the robots may not always exist even if there is a way to move the robots from their initial locations to their final locations. For example, in Figure 1(b) there is no way to move the robots R_1 and R_2 one by one to their destinations. However, trajectories for both robots exist if we synthesize their trajectories together — robot R_2 first moves in the vertical space in the middle, and waits for robot R_1 to move to its destination, and then moves to its final destination. Thus, in finding the ordering of the robots, we might need to group some robots together.

Our incremental algorithm uses an SMT-based priority assignment algorithm at its core to find a feasible ordering for the robots. The algorithm first synthesizes the cost-optimal trajectories for all the robots independently by using the

This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and by the NSF ExCAPE project (grants CCF-1138996 and CCF-1139138).

SMT-based algorithm introduced in [20]. Then the algorithm uses the SMT-based priority assignment algorithm to assign priorities to the robots in such a way that the ordering induced by the priority assignment enables the robots not to block the trajectories of the other robots. While the priority assignment algorithm attempts to assign distinct priorities to the robots, the algorithm ends up with assigning the same priority to some robots due to the dependencies among themselves, an example of which is shown in Figure 1(b). The outcome of the priority assignment algorithm is a set of groups of robots, where the robots with the same priority are included in the same group. If a group has more than one robot, we synthesize the cost-optimal trajectory of the group of robots together using the SMT-based multi-robot motion planning algorithm described in [20]. Now, one can dispatch the robots to their destinations according to their priority (robots with the same priority are dispatched together), and it is guaranteed that all robots will be able to reach their destination successfully.

A secondary objective of our incremental multi-robot motion planning algorithm is to minimize the maximum time to send all the robots to their destinations. Once we have cost-optimal trajectories for all the groups, we want the robots to reach their destinations as early as possible. To achieve this, we adapt the idea proposed in [26] for priority-based motion planning for multi-robot systems. We synthesize the final trajectories of the robot groups one by one based on their priorities. While synthesizing the trajectories for the robots in one group, we consider the higher priority robots as dynamic obstacles, and introduce minimal delay at the beginning of the cost-optimal trajectories to avoid collision with the higher-priority robots. We model this minimum-delay trajectory synthesis problem as an SMT formula in quantifier free integer linear arithmetic.

The major drawback of the above-mentioned algorithm is that the algorithm might entail synthesizing trajectories for a large group of robots together due to their dependencies. This situation arises when we have a large number of robots in a compact workspace. If we need to synthesize the trajectories for a large group of robots together, then the benefit of using the incremental algorithm is not perceived. To alleviate this problem, we present a practical incremental motion planning algorithm where we trade optimality of the trajectories for reducing computational effort. More specifically, we settle for suboptimal trajectories for individual robots to reduce the size of the group of robots for which we need to synthesize trajectories together.

We have implemented both our optimal and practical incremental motion planning algorithms in a tool named Implan. Implan provides an automated mechanism to synthesize motion plans in the form of software code that can be directly used to control the robots using a centralized computer. We compare our incremental trajectory planning algorithm with the SMT-based multi-robot motion planning algorithm in [20]. For a group of 6 robots, the monolithic approach in [20] takes more than 3 hours to generate the (sub-optimal) trajectories, whereas our incremental algorithm can generate the collision free optimal trajectories in less than 40 mins. Our algorithm also scales well with number of robots. We have synthesized collision-free motion trajectories for 25 quadrotors in a compact workspace in around 4 hours and up to 50 quadrotors in obstacle free workspace in around 30 minutes.

II. Problem

A. Preliminaries

1) Workspace: We represent the workspace as a 2D occupancy grid map, where we decompose the workspace into rectangular blocks using a uniform grid. We specify the size of the workspace by the number of blocks in each dimension. Each block is assigned a unique identifier. The identifier of the lower left block is assigned the identifier $(0, 0)$. If the identifier of a block is $ID = (I_x, I_y)$ where I_x and I_y are non-negative integers, then the identifiers of the neighboring blocks are obtained by adding or subtracting 1 to the appropriate component(s) of ID. Each block may either be free, or may be occupied by an obstacle.

Example 1: Figure 2(a) shows a 2D workspace. The black blocks denote the locations of the obstacles.

2) State of a Robot: The state of a robot has two components: its position on the grid and a velocity configuration. A velocity configuration of a robot represents a velocity with a specific magnitude and direction. We denote by $\mathbb{V}$ the finite set of velocity configurations for all robots. We denote by $V_0 \in \mathbb{V}$ the velocity configuration of a robot at the stationary position (zero velocity).

Definition 1 (State of a Robot): The state of a robot is the pair $\langle V, X \rangle$, where $V \in \mathbb{V}$ is the velocity configuration of the robot and $X \in \mathbb{N}_0^2$ denotes its position on the grid representing the workspace.

3) Motion Primitives: Motion primitives are a set of short closed-loop trajectories of a robot under the action of a set of precomputed control laws. The set of motion primitives form the basis of the motion for a robot.

Definition 2 (Motion Primitive): A motion primitive is defined based on the grid structure representing the workspace. A motion primitive for a robot is formally defined as a 6-tuple: $\langle \tau, q_i, q_f, X_{rf}, W, cost \rangle$. The symbol τ denotes the duration of motion of the motion primitive. The symbols $q_i \in \mathbb{V}$ and $q_f \in \mathbb{V}$ are the initial and the final velocity configurations of the robot, respectively. The symbol $X_{rf} \in \mathbb{N}_0^2$ denotes the relative final position of the robot in the workspace grid with respect to the grid block where the motion primitive is applied. The symbol W denotes the set of relative grid blocks through which the robot may pass to move from its initial location to its final location. The execution cost $cost \in \mathbb{R}_+$ is the estimated energy consumption to execute the associated control law in free space.

Example 2: Let us consider a motion primitive for moving a quadrotor in a 2D space. Let us assume that the set of velocity configurations $\mathbb{V} = \{V_0, V_1, \ldots, V_8\}$ contains the zero velocity (V_0) and the velocities with constant magnitude in 8 uniform directions $(V_1, \ldots, V_8)$. Let us assume that for a motion primitive, $\tau = 1\text{s}$, $q_i = V_1$, $q_f = V_3$, $X_{rf} = (2, -1)$,

$W = \{(0,0),(0,-1),(1,0),(1,-1),(2,-1)\}$. The controller associated with the primitive can be applied to the robot at location **I** in the workspace in Figure 2(a), if the velocity configuration of the robot is V_1. After 1s, the quadrotor will move to the block denoted by **F** and it will be in the velocity configuration V_3. While moving, the quadrotor may pass through the blocks indicated by triangles (in Figure 2(a)) decided by the set W.

Definition 3 (Rest Primitive): For each robot, there exists a motion primitive that can be applied when the robot is at the velocity configuration V_0 and it keeps the robot in the same state. This special primitive is called the *rest primitive*.

B. Problem Definition

The *input motion planning problem* is given by a five tuple $\mathcal{P} = \langle R, I, F, PRIM, OBS\rangle$, where $R = \{R_1, \ldots, R_N\}$ is the set of robots, $I : R \to \mathbb{N}_0^2$ maps each robot to an initial location, $F : R \to \mathbb{N}_0^2$ maps each robot to a final location, $PRIM$ is a vector $[PRIM_1, \ldots, PRIM_N]$, where $PRIM_i$ denotes the set of motion primitives available for the i-th robot, and OBS is the set of blocks in the workspace that are occupied by obstacles.

For $R' \subseteq R$, we denote by $I|R'$, $F|R'$ and $PRIM|R'$ the restriction of I, F and $PRIM$ on R' respectively. In this work, we assume that the time durations for all motion primitives are the same. Henceforth, we will denote by τ the common time duration of all the motion primitives.

The runtime behavior of a multi-robot system is given by a discrete transition system $\mathcal{T}^{\mathcal{R}}$. The state of the transition system is denoted by $\Phi = [\phi_1, \ldots, \phi_N]$, where ϕ_i denotes the state of the i-th robot.

Let $\Phi_1 = [\phi_{11}, \ldots, \phi_{1N}]$ and $\Phi_2 = [\phi_{21}, \ldots, \phi_{2N}]$ be two states of the group of robots, and $Prim = [prim_1, \ldots, prim_N]$, where $prim_i \in PRIM_i$, be a vector containing the primitives applied to individual robots in state Φ_1. The transitions of the transition system are given by the following rule:

$$\Phi_1 \xrightarrow{Prim} \Phi_2$$

iff $\forall i \in \{1, \ldots, N\}$: $\phi_{1i}.V = prim_i.q_i$, $\phi_{2i}.V = prim_i.q_f$, $\phi_{2i}.X = \phi_{1i}.X + prim_i.X_{rf}$, $\forall R_i \in R$ *obstacle_avoidance* $(\phi_{1i}, prim_i, OBS)$, and *collision_avoidance*$(\Phi_1, Prim)$. The inputs to the predicate *obstacle_avoidance* are ϕ_{1i}, the state of robot R_i before the transition, $prim_i$, the primitive applied to the robot in state ϕ_{1i}, and OBS, the set of obstacles. This predicate is $true$ if the trajectory of robot R_i between the states ϕ_{i1} and ϕ_{2i} does not overlap with any obstacle. The inputs to the predicate *collision_avoidance* are Φ_1 and $Prim$. The predicate is $true$ if no two robots collide with each other while moving from state Φ_1 to state Φ_2.

Definition 4 (Trajectory of a Multi-Robot System): A *trajectory* of a multi-robot system for an input problem $\mathcal{P} = \langle R, I, F, PRIM, OBS\rangle$ is defined as a sequence of states $\Phi = (\Phi(0), \Phi(1), \ldots, \Phi(L))$, where $\Phi(i) = [\phi_1(i), \ldots, \phi_N(i)]$, such that for all $R_j \in R$, $\phi_j(0) = \langle V_0, I(R_j)\rangle$ and $\phi_j(L) = \langle V_0, F(R_j)\rangle$ and the states are related by the transitions in the following way:

$$\Phi(0) \xrightarrow{Prim_1} \Phi(1) \xrightarrow{Prim_2} \Phi(2) \ldots \Phi(L-1) \xrightarrow{Prim_L} \Phi(L),$$

where $Prim_i = [prim_{1i}, \ldots, prim_{Ni}]$, $prim_{ji} \in PRIM_j$.

Definition 5 (Length of a Trajectory): The length of a trajectory Φ of a multi-robot system is the number of transitions in the trajectory. We denote the length of a multi-robot trajectory Φ as $Length(\Phi)$. If the trajectory is $\Phi = (\Phi(0), \Phi(1), \ldots, \Phi(L))$, $Length(\Phi) = L$.

The total duration to move all robots from their initial state to final state is given by $L \times \tau$. Note that the previous definition does not require the lengths of the trajectories of all the robot to be equal to L when they are synthesized independently. Using the rest primitive, a robot can wait in its initial state or remain in its final state for an arbitrary amount of time to stretch its length to match with that of the multi-robot system.

Definition 6 (Cost of a Trajectory): The cost of a trajectory of a multi-robot system is equal to the cumulative cost of all the primitives used in the trajectory. For a trajectory Φ:

$$\Phi(0) \xrightarrow{Prim_1} \Phi(1) \xrightarrow{Prim_2} \Phi(2) \ldots \Phi(L-1) \xrightarrow{Prim_L} \Phi(L),$$

where $Prim_i = [prim_{1i}, \ldots, prim_{Ni}]$, $prim_{ji} \in PRIM_j$, the cost of the trajectory Φ, denoted by $Cost(\Phi)$, is given by $Cost(\Phi) = \sum_{i=1}^{L} \sum_{j=1}^{N} prim_{ji}.cost.$

Definition 7 (Cost-optimal Trajectory): A multi-robot trajectory Φ is cost optimal if there does not exist another trajectory Φ' that can be synthesized using the motion primitives in $PRIM$ such that $Cost(\Phi') < Cost(\Phi)$.

Now we formally define the motion planning problem we study in this paper.

Problem (*Trajectory Planning for a Multi-Robot System*). Given an input problem $\mathcal{P}$, synthesize a trajectory Φ which is *cost-optimal* and $Length(\Phi)$ is also minimized.

Our primary objective is to ensure that all the robots reach their destination following the cost optimal trajectory. Our secondary objective is to minimize the maximum time required for all the robots to reach their destinations.

Example 3: Two instances of multi-robot motion planning problem have been shown in Figure 2(a) and Figure 3(a). The grid blocks denoted by $i_1, \ldots, i_6$ are the initial locations of robots $R_1, \ldots, R_6$, and the grid blocks denoted by $f_1, \ldots, f_6$ are the final locations of the robots. The black blocks denote the locations covered by the obstacles.

III. Incremental Trajectory Generation

We present an incremental algorithm for solving the multi-robot motion planning problem introduced in Section II. The algorithm is shown in Algorithm 1. The algorithm is described in detail in the following subsections.

A. Finding Optimal Trajectories and Obstacle Robots

In this subsection, we describe the first phase of our incremental motion planning algorithm (Lines 2-6). In this phase, we first synthesize optimal trajectories for

Algorithm 1 Incremental Motion Planning Algorithm

```
1: procedure IncrementalMotionPlanning($\mathcal{P}$)
2:     for $i = 1$ to $|R|$ do
3:         $\mathcal{P}_i \leftarrow \langle\{R_i\}, I \mid \{R_i\}, F \mid \{R_i\}, PRIM_i, OBS\rangle$, optTraj($R_i$) ← findOptimalTrajectory($\mathcal{P}_i$)
4:         InitObs($R_i$) ← findInitialPositionObstacles(optTraj($R_i$), $I$)
5:         FinObs($R_i$) ← findFinalPosisionObstacles(optTraj($R_i$), $F$)
6:     end for
7:     $G$ ← assignPrioritiesToRobots($R$, InitObs, FinObs, optTraj)
8:     for $i = 1$ to $|G|$ do
9:         $G[i] \leftarrow [R_{k_1}, \ldots, R_{k_{|G|}}]$
10:        if $|G[i]| = 1$ then
11:            optGroupTraj[$i$] ← optTraj($R_{k_1}$)
12:        else
13:            $\mathcal{P}_G \leftarrow \langle G[i], I \mid G[i], F \mid G[i], PRIM \mid G[i], OBS\rangle$, optGroupTraj[$i$] ← findOptimalTrajectory($\mathcal{P}_G$)
14:        end if
15:    end for
16:    FinTraj ← synthesizeFinalTrajectories($G$, $PRIM$, optGroupTraj)
17: end procedure
```

all the robots independently. To synthesize optimal trajectory for robot R_i, we invoke the algorithm described in Section III-D in [20] with input motion planning problem $\mathcal{P}_i = \langle\{R_i\}, I \mid \{R_i\}, F \mid \{R_i\}, PRIM_i, OBS\rangle$. Once we synthesize optimal trajectory for robot R_i, we invoke the function `findInitialPositionObstacles` (`findFinalPositionObstacles`) to find the subset of robots whose initial (final) locations block the optimal trajectory of robot R_i. The set of robots whose initial (final) locations block the optimal trajectory of robot R_i is stored in `InitObs`(R_i) (`FinalObs`(R_i)). To ensure safety, we assume that any two robots have to always maintain one block distance from each other, that is, if a robot is on a block in the grid, all its neighboring blocks cannot be occupied by any other robot. For a grid location X, let $\mathcal{N}(X)$ denote the set of all the neighboring grid locations and X itself. Let X_j denote the initial location of robot R_j. If any $Y \in \mathcal{N}(X_j)$ blocks the optimal trajectory of robot R_i, then R_j is included in `InitObs`(R_i). The elements of `FinalObs`(R_i) are selected similarly.

Let the optimal trajectory of robot R_i is given as a sequence of states $\phi = (\phi(0), \phi(1), \ldots, \phi(L_i))$, where the states are related by the transitions in the following way:

$$\phi(0) \xrightarrow{prim_1} \phi(1) \xrightarrow{prim_2} \phi(2) \ldots \phi(L_{i-1}) \xrightarrow{prim_{L_i}} \phi(L_i),$$

where $prim_t \in PRIM_i$ for $t \in \{1, 2, \ldots L_i\}$. Let X_{test} denote the initial or final location of some robot $R_j \in R\backslash\{R_i\}$. To find if X_{test} blocks the optimal trajectory of robot R_i, we check the satisfiability of the following constraint:

$$\bigvee_{t=0}^{L_i-1} \bigvee_{w \in prim_{t+1}.W} \bigvee_{Y \in \mathcal{N}(\phi(t).X+w)} Y = X_{test} \qquad (1)$$

Here, the symbol '$\bigvee$' denotes *logical disjunction*. If the above constraint is satisfiable, then X_{test} blocks the optimal trajectory of robot R_i.

Example 4: For the example in Figure 2(a), for each robot $R_i \in R$, the sets `InitObs`(R_i) and `FinalObs`(R_i) are as follows: `InitObs`(R_1) = $\{R_2, R_3\}$, `FinalObs`(R_1) = { }, `InitObs`(R_2) = $\{R_4\}$, `FinalObs`(R_2) = { }, `InitObs`(R_3) = $\{R_1, R_2\}$, `FinalObs`(R_3) = { }, `InitObs`(R_4) = $\{R_6\}$, `FinalObs`(R_4) = { }, `InitObs`(R_5) = $\{R_4\}$, `FinalObs`(R_5) = $\{R_6\}$, `InitObs`(R_6) = { } and `FinalObs`(R_6) = { }. The computation of these sets requires total 4.724s.

B. Priority Assignment

In this subsection, we describe the function `assignprioritiesToRobots` invoked at Line 7 of Algorithm 1. The inputs to the function are a trajectory for each robot, the initial obstacles list `InitObs` and the final obstacle list `FinalObs`. The function `assignPrioritiesToRobots` is shown in Algorithm 2. The function assigns priorities to the robots in such a way that if a robot's path is blocked by the initial (final) location of another robot, then the first robot has lower (higher) priority than the second robot.

Let $\texttt{prio}_i$ denote the priority of robot $R_i \in R$. If $\texttt{prio}_j < \texttt{prio}_k$, then Robot R_j is dispatched before robot R_k. Let η denote the set of pairs of robots for which the priorities have to be equal. The role of η in our algorithm will be clear later. Initially, η does not contain any element. Now, we generate the following set of constraints:

$$\forall R_i \in R.\ \texttt{prio}_i \geq 1\ \wedge\ \texttt{prio}_i \leq |R| \qquad (2)$$

$$\forall R_i \in R, \forall R_j \in R\backslash\{R_i\}.\ (R_i, R_j) \in \eta \Leftrightarrow \texttt{prio}_i = \texttt{prio}_j \qquad (3)$$

$$\forall R_j \in R, \forall R_k \in R\backslash\{R_j\}.\quad R_k \in \texttt{InitObs}(R_j) \wedge (R_j, R_k) \notin \eta \Rightarrow \texttt{prio}_k < \texttt{prio}_j \qquad (4)$$

$$\forall R_j \in R, \forall R_k \in R\backslash\{R_j\}.\quad R_k \in \texttt{FinalObs}(R_j) \wedge (R_j, R_k) \notin \eta \Rightarrow \texttt{prio}_j < \texttt{prio}_k \qquad (5)$$

We name the constraint for robot R_j and robot R_k in (4) as $\texttt{init\_obs\_constraint}_{j,k}$ and each constraint in (5) as $\texttt{final\_obs\_constraint}_{j,k}$. If the set of constraints has a solution, then we obtain a priority assignment to the robots. However, if the set of constraints does not have a solution, it implies that there exists a subset of the set of constraints that cannot be satisfied together. Such a subset is called an "unsatisfiable core". If a constraint named $\texttt{init\_obs\_constraint}_{j,k}$ or $\texttt{final\_obs\_constraint}_{j,k}$ is in the unsatisfiable core, the priority of robot R_j and robot R_k have to be the same for all constraints being satisfiable. In such a case, we add the pair (R_j, R_k) to η. Now, with the new η, we generate the constraints in (2)–(5) and solve them again. The above process continues until either we find a solution of the constraints, or we are left with only equality constraints in (3).

Example 5: For the example in Figure 2(a), using the sets $\texttt{InitObs}(R_i)$ and $\texttt{FinalObs}(R_i)$ for each robot $R_i \in R$, we generate the following constraints:
$(\forall R_i \in R.\ \texttt{prio}_i \geq 1 \wedge \texttt{prio}_i \leq |R|) \wedge$
$(\forall R_i \in R, \forall R_j \in R \backslash \{R_i\}.\ \texttt{prio}_i \neq \texttt{prio}_j) \wedge$
$\texttt{prio}_2 < \texttt{prio}_1 \wedge \texttt{prio}_3 < \texttt{prio}_1 \wedge \texttt{prio}_4 < \texttt{prio}_2 \wedge \texttt{prio}_1 < \texttt{prio}_3 \wedge \texttt{prio}_2 < \texttt{prio}_3 \wedge \texttt{prio}_6 < \texttt{prio}_4 \wedge \texttt{prio}_4 < \texttt{prio}_5 \wedge \texttt{prio}_5 < \texttt{prio}_6$.

These set of constraints is unsatisfiable and the solver produces the following two constraints to be in the unsatisfiable core: $\texttt{init\_obs\_constraint}_{1,3}$ and $\texttt{init\_obs\_constraint}_{3,1}$. We now add the pair (R_1, R_3) to η. Based on the contents of η, our algorithm now removes the constraints: $\texttt{prio}_1 \neq \texttt{prio}_3$, $\texttt{prio}_3 < \texttt{prio}_1$, $\texttt{prio}_1 < \texttt{prio}_3$, and adds the following constraint: $\texttt{prio}_1 = \texttt{prio}_3$ in the constraints generation phase of the next iteration. When the algorithm attempts to solve the new set of constraints, the solver again generates an unsatisfiable core with the following constraints: $\texttt{prio}_6 < \texttt{prio}_4$, $\texttt{prio}_4 < \texttt{prio}_5$ and $\texttt{prio}_5 < \texttt{prio}_6$. We add (R_4, R_6), (R_4, R_5) and (R_5, R_6) to η. Based on this new η, our algorithm now removes the constraints: $\texttt{prio}_4 \neq \texttt{prio}_5$, $\texttt{prio}_4 \neq \texttt{prio}_6$, $\texttt{prio}_5 \neq \texttt{prio}_6$, $\texttt{prio}_6 < \texttt{prio}_4$, $\texttt{prio}_4 < \texttt{prio}_5$ and $\texttt{prio}_5 < \texttt{prio}_6$, and add the following constraints: $\texttt{prio}_4 = \texttt{prio}_5$, $\texttt{prio}_4 = \texttt{prio}_6$, $\texttt{prio}_5 = \texttt{prio}_6$. The new set of constraints is now satisfiable and we obtain the following priority assignment to the robots: $\texttt{prio}_4 = \texttt{prio}_5 = \texttt{prio}_6 = 1$, $\texttt{prio}_2 = 2$ and $\texttt{prio}_1 = \texttt{prio}_3 = 3$. The output G is given by $[\{4, 5, 6\}, \{2\}, \{1, 3\}]$. The priority assignment step for this example requires $\texttt{0.056}$s.

C. Synthesizing Trajectories for Robot Groups

The priority assignment algorithm in Section III-B outputs an ordered list of disjoint subsets of R. Algorithm 1 synthesizes optimal trajectories for each of such subsets (Line 8-15).

Let the function `assignPrioritiesToRobots` produces as output the ordered list $G = [G_1, G_2, \ldots, G_M]$, where, $M \leq |R|$, $G_i \subseteq R$, $G_i \cap G_j = \emptyset$ for $i, j \in \{1, 2, \ldots M\}$. If $|G_i| = 1$, we already have the optimal trajectory for the only robot in this group from the first phase of Algorithm 1 (Line 3). If $|G_i| > 1$, we invoke the optimal trajectory synthesis algorithm in [20] with the set of robots to be G_i to synthesize optimal trajectory for the robots in G_i.

Example 6: For the example in Figure 2(a), we invoke the optimal trajectory planning algorithm presented in [20] for the robots groups $\{R_4, R_5, R_6\}$ and $\{R_1, R_3\}$. The third phase for this example is the most time consuming and requires $\texttt{35}\text{min}\texttt{34}$s of computation time.

D. Final Trajectory Generation

In the final phase of Algorithm 1, we synthesize the final trajectories for each robot by invoking the function `synthesizeFinalTrajectories` (Line 16).

The final trajectories are synthesized based on the ordering induced by the priority assignment by minimally introducing delays at the beginning of the trajectories. We consider the groups of robots one by one for trajectory synthesis based on the ordering induced by the priority assignment described in Section III-B. While synthesizing a trajectory for the robots in a group, we need to ensure that the synthesized trajectory does not interfere with any of the previously synthesized ones. To ensure this, we maintain a hash table that captures the blocks in the workspace that are occupied by some robot at different time instants. The indices of the hash table corresponds to discrete time instances starting from 0, and the value corresponding to an index represents the set of blocks in the workspace that are occupied by some robot at the corresponding time instant. At any time instant t, we denote by $\rho(t)$ the set of blocks that are occupied by some robot at time instant t. When we synthesize motion plan for a robot, we ensure that the robot does not pass through any block in $\rho(t)$ at time instant t.

Let `max_len` denote the maximum of the lengths of the trajectories synthesized so far, i.e., the size of the hash table is `max_len`. When we synthesize trajectory for a group of robots G_k with length l, the following set of constraints ensures that the synthesized trajectory does not intersect with any of the previously synthesized ones.

$$\begin{aligned} &\forall R_j \in G_k,\ \forall t \in \{0, \ldots, \min(\texttt{max\_len}, l) - 1\}, \\ &\forall w \in prim_{j(t+1)}.W,\ \forall Y \in \mathcal{N}(\phi_j(t).X + w), \\ &\qquad\qquad \forall Z \in \rho(t) : Y \neq Z \end{aligned} \tag{6}$$

Algorithm 4 presents how we synthesize the final trajectories for all the robots in an incremental way by using the collision avoidance hash table ρ. In this algorithm, `max_len` denotes the maximum length of the trajectories synthesized so far. If the length of the trajectory of a group of robots is `len`, then the length of the final trajectory cannot be more than `len` + `max_len`. This is due to the fact that if the robots in the current group wait up to `max_len` time, the workspace will be clear for the current group of robots. For each robot group G_i, we start with its optimal trajectory $\texttt{optGroupTraj}[i]$ and check using `checkTrajectoryFeasibility` function whether the trajectories of the already considered robots allow the current group of robots reach their destination following their optimal trajectories. If the function `checkTrajectoryFeasibility` returns $false$, we insert a delay to $\texttt{optGroupTraj}[i]$ and invoke `checkTrajectoryFeasibility` again. If at any iteration `checkTrajectoryFeasibility` function returns $true$, the current trajectory is the final one. The synthesized trajectory is

Algorithm 2 Priority Assignment to the Robots

```
1: procedure assignPrioritiesToRobots($R$, InitObs, FinalObs, $Traj$)
2:     $\eta \leftarrow \emptyset$
3:     while $true$ do
4:         $C \leftarrow$ genConstraintsForPrio(InitObs, FinObs, $\eta$), [res, model, ucore] $\leftarrow$ solveSMTConstraints($C$)
5:         if res = SAT then
6:             $G \leftarrow$ generateRobotGroups(model), return $G$
7:         else
8:             $\eta \leftarrow$ findDependentRobots(ucore, $\eta$)
9:         end if
10:    end while
11: end procedure
```

Algorithm 3 Update Collision Avoidance Hash Table

```
1: procedure updateCollisionAvoidanceTable($R, PRIM, \rho, \Phi$)
2:     for $t = 0$ to length($\Phi$) $- 1$ do
3:         for $j = 1$ to $|R|$ do
4:             $X \leftarrow \phi_j(t).X,\ \ prim \leftarrow prim_j(t+1),\ W \leftarrow prim.W$
5:             for all $Z \in W$ do
6:                 $\rho(t) \leftarrow \rho(t) \cup (X + Z)$
7:             end for
8:         end for
9:     end for
10:    return $\rho$
11: end procedure
```

Algorithm 4 Synthesize Final Trajectories

```
1: procedure synthesizeFinalTrajectories($G$, $PRIM$, optGroupTraj)
2:     $\rho$ = NULL, max_len = 0
3:     for $i = 1$ to $|G|$ do
4:         traj $\leftarrow$ optGroupTraj[$i$], len $\leftarrow$ length(traj)
5:         while len $\leq$ len + max_len do
6:             status $\leftarrow$ checkTrajectoryFeasibility($\rho$, traj)
7:             if status = $false$ then
8:                 traj $\leftarrow$ addUnitDelay(traj)
9:             else
10:                finalTraj($G_i$) $\leftarrow$ traj
11:                $\rho \leftarrow$ updateCollisionAvoidanceTable($G_i, PRIM|G_i, \rho$, traj)
12:                max_len $\leftarrow$ max(max_len, length(traj))
13:            end if
14:        end while
15:    end for
16: end procedure
```

then used to update the collision avoidance hash table ρ using Algorithm 3.

Example 7: The final trajectories synthesized for the example in Figure 2(a) are shown in Figure 2(b). The numbers in the figures indicate the timestamp when the robot occupies the corresponding location. The execution time of the final phase for the example is 27.765s. Thus, the total time required to synthesize trajectories is 36min6.585s. The maximum trajectory length is 19 and the average trajectory length is 15. The total cost for all the trajectories if 64.64. We compare our result with the algorithm presented in [20]. The synthesis for the example using the algorithm requires 193min54s. The average length of the trajectories is 12 and the total cost of the trajectories is 71.30. Note that the increase in the trajectory lengths for our incremental algorithm is due to the fact that the robots of the lower priorities may need to wait at their initial locations before moving towards their destinations. However, as the robots follow their optimal paths in case of our incremental algorithm, the total cost of the trajectories obtained by our incremental algorithm is better than those obtained by the algorithm in [20].

IV. A Practical Incremental Motion Planning Algorithm

The trajectories generated by Algorithm 1 in Section III move the robots from their initial locations to their final locations using their optimal trajectories. The algorithm also attempts to minimize the delay to move the robots to their destinations. However, the major drawback of the algorithm is that it might synthesize trajectories for a large group of robots together due to their dependencies. This situation arises when

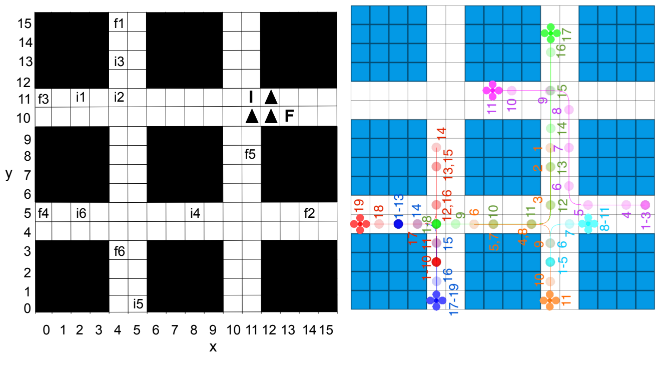

(a) (b)

Fig. 2. (a) Multi-robot motion planning: Example 1, (b) Trajectories generated by Algorithm 1.

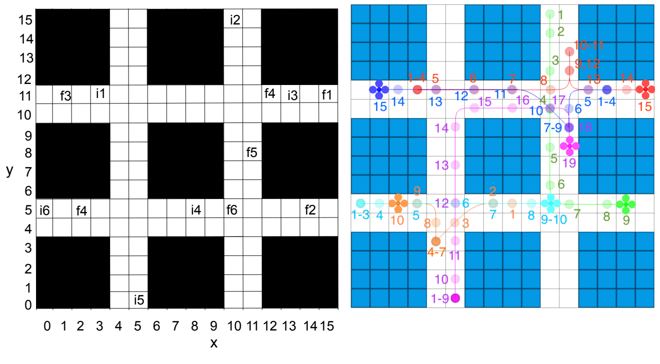

(a) (b)

Fig. 3. (a) Multi-robot motion planning: Example 2, (b) Trajectories generated by Algorithm 5.

the workspace is compact, or when we have a large number of robots in a small workspace. If we need to synthesize the trajectories for a large group of robots together, the benefit of using the incremental algorithm is minor.

In this section, we present a practical incremental motion planning algorithm that modifies Algorithm 1 to trade optimality for reducing computational effort. More specifically, we settle for suboptimal trajectories for individual robots to reduce the sizes of the groups of robots for which we need to synthesize trajectories together. To achieve this, we modify the first phase of Algorithm 1. The modified algorithm is shown-InitObs in Algorithm 5. For a robot R_i, the algorithm first synthesizes the optimal trajectory and computes $\texttt{InitObs}(R_i)$ and $\texttt{FinObs}(R_i)$. Next, the algorithm checks if the sets $\texttt{InitObs}(R_i)$ and $\texttt{FinObs}(R_i)$ are satisfactory by calling the function `isSatisfactory`, which can be implemented using different heuristics. For example, the simplest heuristic is to check if both the sets are empty. Another heuristic may be to check if a robot R_j is included both in $\texttt{InitObs}(R_i)$ and $\texttt{FinObs}(R_i)$, indicating obvious dependency between robot R_i and robot R_j.

If the sets $\texttt{InitObs}(R_i)$ and $\texttt{FinObs}(R_i)$ are not satisfactory, we attempt to synthesize the trajectory of robot R_i considering the initial and final locations of all other robots as static obstacles. Here our objective is to find a trajectory that is not blocked by the initial or the final location of any robot. We cannot expect optimality any more, however, we impose a bound on the length of the trajectory to ensure that the synthesized trajectory is not much longer from the optimal trajectory. We invoke the optimal trajectory planning algorithm in [20] with a bound on the length of $\texttt{length}(\texttt{optTraj}(R_i)) + \delta$ by invoking the function `findBoundedOptimalTrajectory`. If we find such a trajectory $\texttt{optTraj}'(R_i)$ for robot R_i, then we replace the optimal trajectory $\texttt{optTraj}(R_i)$ with $\texttt{optTraj}'(R_i)$ and the sets $\texttt{InitObs}(R_i)$ and $\texttt{FinObs}(R_i)$ are set to $\emptyset$. The trajectory is not optimal. Nevertheless, the sub optimality is bounded, and depends on δ. If the maximum cost of any primitive in $PRIM_i$ is $max\_prim\_cost_i$, then the cost of the trajectory cannot be $\delta \times max\_prim\_cost_i$ more than the cost of the optimal trajectory for robot R_i.

Example 8: We illustrate the benefit of our practical incremental motion planning algorithm on the example shown in Figure 3(a). Algorithm 1 generates the following group of robots: $G = [\{R_2\}, \{R_1, R_3\}, \{R_4, R_5, R_6\}]$. The synthesis of trajectories takes total $40\text{min}35\text{s}$. The total cost of the trajectories is 71.35. When we apply Algorithm 5, we get the following group of robots: $G = [\{R_2\}, \{R_4, R_6\}, \{R_5\}, \{R_1, R_3\}]$. The synthesis takes total $8\text{min}17\text{s}$. The total cost of the trajectories is 71.67. The synthesized trajectories by Algorithm 5 are shown in Figure 3(b). The computation time for the practical algorithm improves as the optimal path of robot R_5 passes through the initial location of robot R_4 and the final location of robot R_6. However, robot R_5 has another trajectory that does not pass through the initial or final location of any robot. Our practical algorithm can find such trajectories.

V. Robot Control Software Development Framework

In this section, we present a control software development framework for multi-robot systems. The framework, as shown in Figure 4, is based on our incremental motion plan generation tool Implan which incorporates the implementation of Algorithm 4 and Algorithm 5. Implan generates the motion plan for any robot in the multi-robot system as a sequence of pre-computed motion primitives which contain information regarding initial velocity, final velocity, offset of final pose with respect to the initial pose, and the free space required for the maneuver. The motion plan for each robot, in the form of a sequence of the motion primitives, is then passed to the *Primitive Handler* subsystem. The *Primitive Handler* subsystem then outputs the corresponding time parameterized trajectory for each of the motion primitives. The set of time parameterized trajectories is then passed to the *Robot Controller* subsystem, which is a PID (proportional-integral-derivative) controller [28] that generates the motor outputs for the quadrotors based on the current pose obtained from a localization sensor attached to the robot and the desired pose from the time parameterized trajectory. Thus, Implan together with *Primitive Handler* and *Robot Controller*, provides the control

Algorithm 5 A practical incremental multi-robot motion planning algorithm

```
1: procedure PracticalIncrementalMotionPlanning($\mathcal{P}$)
2:     for $i = 1$ to $|R|$ do
3:         $\mathcal{P}_i \leftarrow \langle \{R_i\}, I \mid \{R_i\}, F \mid \{R_i\}, PRIM_i, OBS \rangle$, optTraj($R_i$) $\leftarrow$ findOptimalTrajectory($\mathcal{P}_i$)
4:         InitObs($R_i$) $\leftarrow$ findInitialPositionObstacles(optTraj($R_i$), $I$)
5:         FinObs($R_i$) $\leftarrow$ findFinalPosisionObstacles(optTraj($R_i$), $F$)
6:         if (isSatisfactory(InitObs($R_i$), FinObs($R_i$)) $= FALSE$) then
7:             $OBS' \leftarrow OBS \cup I(R \setminus \{R_i\}) \cup F(R \setminus \{R_i\})$, $l \leftarrow$ length(optTraj($R_i$))
8:             $\mathcal{P}_i \leftarrow \langle \{R_i\}, I \mid \{R_i\}, F \mid \{R_i\}, PRIM_i, OBS' \rangle$, optTraj$'$($R_i$) $\leftarrow$ findBoundedOptimalTrajectory($\mathcal{P}_i, l + \delta$)
9:             if optTraj$'$($R_i$) $\neq \emptyset$ then
10:                InitObs($R_i$) $\leftarrow \emptyset$, FinObs($R_i$) $\leftarrow \emptyset$, optTraj($R_i$) $\leftarrow$ optTraj$'$($R_i$)
11:            end if
12:        end if
13:    end for
14:    $G \leftarrow$ assignPrioritiesToRobots($R$, InitObs, FinObs, optTraj)
15:    for $i = 1$ to $|G|$ do
16:        $G[i] \leftarrow \{R_{k_1}, \ldots, R_{k_{|G|}}\}$
17:        if $|G[i]| = 1$ then
18:            optGroupTraj[$i$] $\leftarrow$ optTraj($R_{k_1}$)
19:        else
20:            $\mathcal{P}_G \leftarrow \langle G[i], I \mid G[i], F \mid G[i], PRIM \mid G[i], OBS \rangle$, optGroupTraj[$i$] $\leftarrow$ findOptimalTrajectory($\mathcal{P}_G$)
21:        end if
22:    end for
23:    FinTraj $\leftarrow$ synthesizeFinalTrajectories($G$, optGroupTraj)
24: end procedure
```

software for a multi-robot system to satisfy the specification given in terms of the input problem.

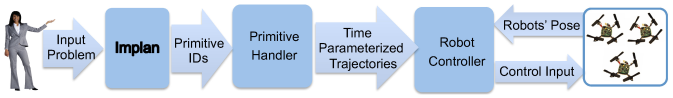


Fig. 4. The block diagram of the framework for developing multi-robot control software based on Implan

VI. Application to Quadrotors and Experimental Results

A. Experimental Setup

We implement our incremental trajectory planning algorithm using the SMT solver Z3 [2] as the backend solver. For our experiments, we consider the NanoQuad quadrotors from KMel Robotics [11]. We use ROS Simulator [18] to simulate the behavior of the quadrotors using the synthesized trajectories. The ROS simulator incorporates a faithful model of the NanoQuad quadrotor, thus a successful simulation guarantees that the generated trajectories can safely fly the quadrotors in real environment. In our experiments we use rectangular workspace where each block is a square with the length of the side to be 0.2m.

All quadrotors in our experiment share the same set of motion primitives, which has been generated using the algorithm described in [13]. For two dimensional spaces, the velocity profile has 9 configurations consisting of one hover state and a constant velocity in 8 uniform directions (E, W, N, S, NE, NW, SE, SW). For each quadrotor, we have 57 motion primitives, and each motion primitive is active for 1s.

In the following subsection, we present our experimental results. All experiments were run in a 64-bit Linux Ubuntu $12.04.3$ machine with an Intel(R) Core(TM) i7-3840QM CPU and 8GB RAM.

B. Results

To judge the scalability of our incremental motion planning algorithm, we carry out two sets of experiments. In the first set of experiments, we use a compact workspace where the presence of the obstacles renders the motion planning for the robots non-trivial. In the second set of experiments, we use a workspace free from obstacles. In both the cases, the size of the workspace is $15.2\text{m} \times 16.8\text{m}$.

We first attempt to use Algorithm 1 to synthesize trajectories for 25 robots in the compact workspace. We repeat the experiment 10 times. Among these 10 experiments, we found that the maximum robot group size computed in phase 2 of Algorithm 1 can be as large as 6. Trajectory synthesis for such a group of robots is computationally expensive. For a specific instance, the trajectory synthesis for 6 robots together could not be finished even in 6 hours.

For the rest of the experiments, we use Algorithm 5 to synthesize trajectories for the robots in the compact workspace and in the obstacle-free workspace. In the case of the compact workspace, we increase the number of robots from 5 to 25 with a step size of 5 robots. In the case of obstacle-free workspace, we increase the number of robots from 10 to 50 robots with a step size of 10 robots. The initial and the final locations of the robots are generated randomly ensuring that the initial locations (similarly, the final locations) are at least one block away from each other. The parameter δ has been chosen to be 4 for the compact workspace and to be 2 for the obstacle-free one.

$\|R\|$	Phase 1	Phase 2	Phase 3	Phase 4	Total
5	2min41s	0.016s	0.002s	4s	2min46s
10	1h22min31s	0.024s	0.005s	16s	1h22min48s
15	1h24min14s	0.029s	0.005s	3min55s	1h28min10s
20	2h43min28s	0.030s	0.005s	16min45s	3h0min14s
25	3h33min33s	0.072s	56min59s	38min39s	4h12min13s

Fig. 5. Time required in different phases for motion plan synthesis in a compact workspace for different number of robots

$\|R\|$	Phase 1	Phase 2	Phase 3	Phase 4	Total
10	30s	0.028s	0.004s	22s	53s
20	1min35s	0.034s	0.005s	57s	2min33s
30	2min35s	0.044s	0.005s	4min11s	6min47s
40	4min11s	0.073s	0.005s	10min47s	14min59s
50	5min56s	0.102s	0.005s	24min22s	30min19s

Fig. 6. Time required in different phases for motion plan synthesis in an obstacle-free workspace for different number of robots

The time required to execute different phases of Algorithm 5 is shown in Figure 5 for the experiments in the compact workspace and in Figure 6 for the experiments in the obstacle-free workspace. Each entry of the tables is average (and rounded to the nearest integer wherever applicable) of 10 runs. In the compact workspace, a significant amount of time is spent in synthesizing the independent optimal (or near-optimal) trajectories in phase 1. The priority assignment step requires less than 1s in phase 2. Synthesizing optimal trajectories for the robot groups in phase 3 requires negligible amount of time in most of the cases. This is because the practical algorithm has been able to keep the maximum size of the robot groups to be 1. Only for the case of 25 robots in compact workspace, we encountered a robot group of size 2 in a few instances.

The tables in Figure 7 and Figure 8 show the maximum and the average lengths of the trajectories synthesized in the first phase, the maximum and the average lengths of the trajectories synthesized in the final phase and maximum delay introduced in the final phase of Algorithm 5 for different number of robots in the compact workspace and in the obstacle-free workspace, respectively. The length of the final trajectory of a robot is obtained by adding the amount of delay introduced to the length of its optimal trajectory. The tables show the benefit of treating the higher-priority robots as dynamic obstacles. If the average length of an independent trajectory is L_a, and we send the robots one-by-one, then the maximum trajectory length for a robot can by $L_a \times |R|$. For example, for the 25 robot instance in the compact workspace, the length of the worst delay trajectory could be as much as $25 \times 15 = 375$. However, the maximum length trajectory for this instance is only 47.

An instance of motion planning for 25 robots in the compact workspace is shown in Figure 9(a), and for 50 robots in the obstacle-free workspace is shown in Figure 9(b).

For the examples with 6 robots, we carry out experiments in our laboratory using a set of nano-quadrotors developed by KMel Robotics [11]. The synthesized trajectories are translated to C programs and integrated with Robot Operating System (ROS) [18]. During our lab experiments, the states of the robots are tracked using a Vicon motion capture system [27]. The control inputs are computed on external computer using PID control theory [28]. Experimental results[1] confirm that the generated trajectories satisfy the desired specifications.

$\|R\|$	Independent		Final		
	Maximum Length	Average Length	Maximum Length	Average Length	Maximum Delay
5	20	13	20	13	0
10	24	11	24	12	3
15	28	14	34	16	13
20	29	17	39	21	17
25	30	15	47	22	26

Fig. 7. The maximum and the average lengths of the trajectories synthesized in the first phase, the maximum and the average lengths of the trajectories synthesized in the final phase and maximum delay introduced in the final phase of Algorithm 5 in a compact workspace for different number of robots

$\|R\|$	Independent		Final		
	Maximum Length	Average Length	Maximum Length	Average Length	Maximum Delay
10	20	12	20	12	0
20	19	12	19	12	1
30	19	11	20	12	4
40	19	11	20	12	8
50	21	12	26	13	10

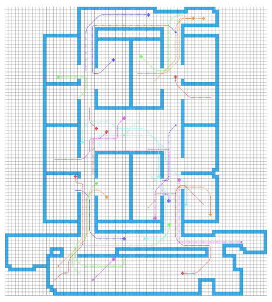
Fig. 8. The maximum and the average lengths of the trajectories synthesized in the first phase, the maximum and the average lengths of the trajectories synthesized in the final phase and maximum delay introduced in the final phase of Algorithm 5 in an obstacle-free workspace for different number of robots

VII. Related Work

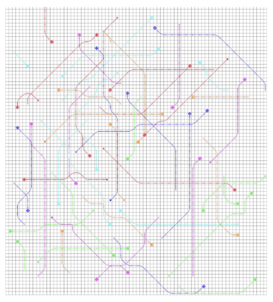
The path planning problem for a multi-agent system traditionally takes either a centralized or a decentralized approach. In the centralize approach, all agents are treated as a single agent with a high dimensional configuration space, and the state space becomes the cartesian product of the state space of the individual agents. The path planning problem then can be solved using sampling based method in the configuration space [10], [12], or using A^* search algorithm in the state space [7]. Both these approaches offer poor scalability due to the curse of dimensionality.

To alleviate the scalability issue with the centralized solutions, a number of decentralized methods have been proposed. Among these solutions, most popular is the *decoupled* planning [9], [15], [23], [22], [6], [16], where paths for individual agents are first synthesized independently, and then the agents interact online to ensure collision avoidance. Another decentralized approach is *priority-based* planning [3], [26], where priorities are pre-assigned to the individual agents to maintain an order among themselves. The combination of these planning algorithms has also been developed [4], where the collisions between the paths generated by the decoupled planning have been resolved based on the priorities of the agents. All these planning algorithms depend on some methods, such as navigation quad trees [21], visibility points [19], and meshes [24], to abstract a problem map into a search

[1]The videos of our experiments are available at the following link: `https://www.dropbox.com/s/ivymskxm3ntmuv4/iccps2016.mp4?dl=0`.



(a)



(b)

Fig. 9. Synthesized trajectories for (a) 25 robots in a compact workspace, (b) 50 robots in an obstacle-free workspace

graph. Our method, on the other hand, does not depend on any such abstraction technique. The possible motions considered in these algorithms are also very simple, generally one step advancement in any direction in unit time. Our technique can incorporate complex motions of the agents as captured by a rich set of motion primitives.

The optimal trajectory planning problem for multi-robot systems has been addressed in a few recent work. Jingjin and LaValle [29] address the optimal multi-robot path planning problem on graph based on a reduction to integer linear programming problem. Their optimization was based on the last arrival lime and the total distance covered by the robots. Their solution, though scalable, does not use motion primitives for planning, thus is not suitable for robots that have complex dynamics. Recently, Turpin et al. [25] proposed an algorithm for simultaneous planning and goal assignment for interchangeable robots and demonstrated their algorithm to work on a group of quadrotors. However, our objective is to solve the planning problem where the goals are pre-assigned to the robots.

SMT solvers have been used in motion planning with rectangular obstacles [8] and in synthesizing integrated task and motion plans from plan outlines [14]. We introduce the use of an SMT solver in solving the multi-robot motion planning problem in a scalable way, and show how different capabilities of SMT solvers, for example, finding a satisfiable solution and finding an unsatisfiable core can be effectively applied to solve motion planning problems.

VIII. Conclusion and Future Directions

We have developed an SMT-based incremental motion planning framework and showed that our method can synthesize trajectories for tens of robots with complex dynamics. We have demonstrated that our synthesized software is capable of controlling as many as 25 quadrotors in a compact workspace and as many as 50 quadrotors in obstacle free workspace. In our current work, we consider the robot specifications of the form of reaching goals from initial locations while avoiding obstacles. In our future research, we aim to extend our framework to support specification in any arbitrary Linear Temporal Logic (LTL) [17] formula.

References

[1] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.
[2] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
[3] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *ICRA*, volume 3, pages 1419–1424, 1986.
[4] A. Geramifard, P. Chubak, and V. Bulitko. Biased cost pathfinding. In *AIIDE*, pages 112–114, 2006.
[5] H. C. Glenn Wagner. M*: A complete multirobot path planning algorithm with performance bounds. In *IROS*, pages 3260–3267, 2011.
[6] Y. Guo and L. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *ICRA*, pages 2612–2619, 2002.
[7] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transaction on Systems Science and Cybernetics*, 4(2):100–107, 1968.
[8] W. N. N. Hung, X. Song, J. Tan, X. Li, J. Zhang, R. Wang, and P. Gao. Motion planning with Satisfiability Modulo Theroes. In *ICRA*, pages 113–118, 2014.
[9] K. Kant and S. Zucker. Towards efficient trajectory planning: The path velocity decomposition. *International Journal of Robotics Research*, 5(3):72–89, 1986.
[10] L. E. Kavraki, P. Ŝvestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
[11] KMel robotics. http://kmelrobotics.com/.
[12] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.
[13] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *ICRA*, pages 2520–2525, 2011.
[14] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki. SMT-based synthesis of integrated task and motion plans from plan outlines. In *ICRA*, pages 655–662, 2014.
[15] P. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *ICRA*, pages 484–489, 1989.
[16] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, 24(4):295–310, 2005.
[17] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
[18] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, 2009.
[19] S. Rabin. A* speed optimizations. In M. Deloura, editor, *Game Programming Gems*, pages 272–287. Charles River Media, 2000.
[20] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IROS*, pages 1525–1532, 2014.
[21] H. Samet. An overview of quadtrees, octrees, and related hierarchical data structures. In *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI Series, pages 51–68. Springer-Verlag, 1988.
[22] T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics and Automation*, 18(1):42–49, 2002.
[23] P. Ŝvestka and M. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998.
[24] P. Tozour. Building a near-optimal navigation mesh. In S. Rabin, editor, *Game Programming Wisdom*, pages 171–185. Charles River Media, 2002.
[25] M. Turpin, K. Mohta, N. Michael, and V. Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4):401–415, 2014.
[26] J. van den Berg and M. Overmars. Prioritized motion planning for multiple robots. In *IROS*, pages 430–435, 2005.
[27] Vicon motion capture system. http://www.vicon.com/.
[28] A. Visioli. *Practical PID Control*. Springer, 2006.
[29] J. Yu and S. M. LaValle. Planning optimal paths for multiple robots on graphs. In *ICRA*, pages 3612–3617, 2013.