2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
September 14-18, 2014. Chicago, IL, USA,

Automated Composition of Motion Primitives for Multi-Robot Systems from Safe LTL Specifications*

Indranil Saha[1,2], Rattanachai Ramaithitima[2], Vijay Kumar[2], George J. Pappas[2] and Sanjit A. Seshia[1]

***Abstract*— We present a compositional motion planning framework for multi-robot systems based on an encoding to satisfiability modulo theories (SMT). In our framework, the desired behavior of a group of robots is specified using a set of safe linear temporal logic (LTL) properties. Our method relies on a library of motion primitives, each of which corresponds to a controller that ensures a particular trajectory in a given configuration. Using the closed-loop behavior of the robots under the action of different controllers, we formulate the motion planning problem as an SMT solving problem and use an off-the-shelf SMT solver to generate trajectories for the robots. Our approach can also be extended to synthesize optimal cost trajectories where optimality is defined with respect to the available motion primitives. Experimental results show that our framework can efficiently solve complex motion planning problems in the context of multi-robot systems.**

I. Introduction

Numerous applications, such as monitoring, surveillance and disaster response, involve tasks that are performed better by a team of robots rather than by a single robot. Collision-free motion planning for such systems is a fundamental problem in robot motion planning. Any generic solution to the problem finds application in different domains, including assembly planning [1], evacuation [2], search and rescue [3], localization [4], object transportation [5], and formation control [6].

In this paper, we address the motion planning problem for multi-robot systems, where the robots have complex dynamics, and complex specification of the system is given in terms of a set of *safe linear temporal logic (LTL)* properties [7]. Safe LTL is useful in capturing numerous requirements related to multi-robot systems, for example, maintaining a formation during the flight of the group of robots, maintaining a precedence relationship between the robots, maintaining a minimum distance between the robots and so on.

The motion planning problem where the specification is given in terms of some temporal logic has been addressed in a number of recent works [8], [9], [10], [11], [12], [13], [14]. In these works, a finite model for the robot dynamics is first generated using an abstraction process based on discretization of the configuration space [15], and then game theoretic synthesis techniques [8], [13] are used to generate high level motion plans and low level control policies on the abstract model. However, the abstraction algorithm and the synthesis algorithm both scale exponentially with the dimension of the configuration space, thus limiting the application of this approach to simple systems with lower dimensions.

To solve the safe LTL motion planning problem for a multi-robot system where the robots have complex dynamics, we need to deal with a significantly higher-dimensional system. We deal with the complexity of the problem by decomposing it into two subproblems. First, we design a set of controllers to control different aspects of the members of the multi-robot system. For example, for a UAV, one may have different controllers for moving it to different positions and orientations. A controller together with the corresponding closed-loop trajectory of the robot is termed a *motion primitive*. Second, we utilize these motion primitives to build a system of constraints where the decision variables encode the choice of motion primitives used at any discrete-time point on the trajectory. For the specifications considered in this paper, the system of constraints involves a Boolean combination of linear constraints. We leverage the power of an off-the-shelf satisfiability modulo theories (SMT) solver [16] to solve the system of constraints. To render the constraint solving problem easier, we use a simple over-approximation of the trajectories to simplify the set of constraints that ensure collision avoidance amongst the robots. Moreover, we show how an SMT solver can be used to synthesize an optimal trajectory for each robot, where optimality is defined with respect to the available motion primitives.

Our motion planning technique can be viewed as a compositional (modular) synthesis technique, where we use an SMT solver to compose a set of motion primitives to generate trajectories for a group of robots. Several compositional frameworks for robot motion planning have been proposed in the past - motion description language [17], [18], [19], the maneuver automata [20], sequential composition of closed-loop behavior in the absence of noise [21], [22] and in the presence of noise [23]. The motion planning problem has also been reduced to constrained dynamical simulation, where the trajectories are computed iteratively by solving a set of constraints on the system [24], [25], [26]. Recently, SMT solvers have been used in motion planning with rectangular obstacles [27] and in synthesizing integrated task and motion plans from plan outlines [28]. However, composing a

*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).

[1] Indranil Saha and Sanjit A. Seshia are with Department of Electrical Engineering and Computer Science, University of California Berkeley {indranil,sseshia}@eecs.berkeley.edu

[2] Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar and George J. Pappas are with GRASP Lab, University of Pennsylvania {isaha,ramar,kumar,pappasg}@seas.upenn.edu

set of motion primitives to synthesize trajectories for a group of robot using an SMT solver has not been attempted before.

In our experiments, we synthesize a sequence of primitives to generate optimal trajectories for a group of 4 UAVs in a $4m \times 4m$ workspace that contains a few obstacles. We use these primitives to fly 4 nano quadrotors successfully in our lab space. Our results show that the proposed approach can efficiently solve complex motion planning problems in the context of multi-robot systems.

II. Problem

A. Preliminaries

1) Workspace: The workspace is represented as a 3D occupancy grid map, where we decompose the workspace into 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 $(0, 0, 0)$. If the identifier of a block is $ID = (I_x, I_y, I_z)$ where I_x, I_y and I_z 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. The unit of distance is 1 block in the workspace. The distance between two blocks with identifier $ID_1 = (I_{x_1}, I_{y_1}, I_{z_1})$ and $ID_1 = (I_{x_2}, I_{y_2}, I_{z_2})$ is given by $dist(ID_1, ID_2) = (I_{x_2} - I_{x_1}, I_{y_2} - I_{y_1}, I_{z_2} - I_{z_1})$.

Each block may either be free, or may be occupied by an obstacle. We denote by OBS the set of all the identifiers corresponding to the blocks that are occupied by an obstacle.

2) Motion Primitives: Motion primitives capture a set of precomputed control laws that can be used to avoid solving complex dynamical system in real-time. A motion primitive consists of a precomputed control law that regulates the outputs of the system as a function of time, the closed loop trajectory of the robot under the action of the controller, and the cost of executing the controller for a predefined duration of time. Let U denote the set of available control laws. A control law can be applied to the robot if it is in a particular *velocity configuration*. A velocity configuration represents a velocity with a constant magnitude and a direction. Let us denote the set of all possible velocity configurations by $\mathbb{V}$.

Definition 2.1 (Motion Primitive): A motion primitive is formally defined as a 7-tuple: $\langle u, \tau, q_i, q_f, X_{rf}, W, cost \rangle$. The symbol $u \in \mathbb{U}$ denotes a precomputed control input. The symbol τ denotes the duration for which the control input is applied to the robot. 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 $X_{rf} \in \mathbb{R}^3$ denotes the distance of the final position of the robot from the position where the motion primitive is applied. The symbol W denotes the set of relative blocks through which the robot passes to move from its initial location to its final location. The symbol $cost \in \mathbb{R}+$ denotes an estimated cost for executing the precomputed control law for τ duration in free space.

3) Specification Language – Safe LTL_f: We express the behavioral specification of a multi-robot system using linear temporal logic on finite traces [29], denoted by LTL_f. Let Π denote the set of atomic propositions. From the atomic propositions $\pi \in \Pi$, any LTL_f formula can be formulated according to the following grammar:

$$\phi ::= \pi \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \, \mathcal{U} \, \phi$$

Given the above grammar, we can define $false$ and $true$ in the following way: $false = \phi \wedge \neg\phi$ and $true = \neg false$. Given negation ($\neg$) and conjunction ($\wedge$), we can define disjunction ($\vee$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$) in the standard way. Moreover, given the temporal operators next ($\bigcirc$) and until ($\mathcal{U}$), we can derive additional temporal operators, for example, eventually ($\Diamond$) and always ($\Box$). These operators are derived as $\Diamond\phi = true\mathcal{U}\phi$ and $\Box\phi = \neg\Diamond\neg\phi$.

The semantics of an LTL_f formula is defined over a finite sequence σ of the truth assignment to the propositions used in the formula. Let us denote the length of the sequence σ by $length(\sigma)$. Let $\sigma(i)$ demote the set of atomic propositions that are true at the i-th position of σ, where $0 \leq i \leq length(\sigma) - 1$. For an LTL_f formula ϕ, we denote by $\sigma, i \models \phi$ the fact that the sequence σ satisfies the LTL_f formula ϕ at location i, $0 \leq i \leq length(\sigma) - 1$, and is recursively defined as follows:

$\sigma, i \models \pi$	iff $\pi \in \sigma(i)$
$\sigma, i \models \neg\phi$	iff $\sigma, i \not\models \phi$
$\sigma, i \models \phi_1 \wedge \phi_2$	iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$
$\sigma, i \models \bigcirc\phi$	iff $i < length(\sigma) - 1$ and $\sigma, i+1 \models \phi$
$\sigma, i \models \phi_1 \mathcal{U} \phi_2$	iff there exists $i \leq k \leq length(\sigma) - 1$ such that $\sigma, k \models \phi_2$ and for all $i \leq j < k, \sigma, j \models \phi_1$

The sequence σ satisfies a formula ϕ, if $\sigma, 0 \models \phi$.

In this paper, we consider only safety properties [7]. Hence, we use a subset of LTL_f called safe LTL_f which is sufficient to express safety properties on finite traces.

Definition 2.2 (Safe LTL_f): An LTL_f formula is called a safe LTL_f formula if it can be represented using the temporal operators next ($\bigcirc$) and always ($\Box$).
Intuitively, the formula $\bigcirc\phi$ at any location i in the sequence is true, if $i < length(\sigma) - 1$ and the formula ϕ is true at the $i+1$-th location of that sequence. The formula $\Box\phi$ holds for a sequence, if ϕ is true at every location till the end of the sequence.

B. Problem Definition

In this subsection, we define our problem formally.

Definition 2.3 (State of a robot): The *state of a robot* is a pair $\langle q, X \rangle$, where $q \in \mathbb{V}$ is the velocity configuration of the robot and $X \in \mathbb{R}^3$ denotes its position.

Definition 2.4 (State of a Multi-Robot System): The *state of a multi-robot system* with N robots is denoted by $\Phi = [\phi_1, \ldots, \phi_N]$, where ϕ_i denotes the state of the i-th robot.

Definition 2.5 (Input motion planning problem instance): An *input motion planning problem instance* is given by a seven tuple $\mathcal{P} = \langle R, I, F, PRIM, OBS, L, \Psi \rangle$, where

- $R = \{R_1, \ldots, R_N\}$ - The set of robots
- I - The set of initial states for the group of robots
- F - The set of final states for the group of robots

- $PRIM$ - A vector $[PRIM_1, \ldots, PRIM_N]$, where $PRIM_i$ denotes the set of motion primitives available for the i-th robot
- OBS - The set of blocks in the workspace, that are occupied by obstacles
- L - The total number of hops in the trajectory
- Ψ - A set of safe LTL_f properties that should always be satisfied by the group of robots. The properties can be classified into two groups:
  1) Safety properties:
     - *Obstacle Avoidance:* No agent faces a collision with an obstacle
     - *Collision Avoidance:* The agents do not collide with each other
  2) Behavioral properties:
     The behavioral properties are provided in terms of a safe LTL_f formula, and is denoted by ξ. The propositions used in ξ are defined using the position component of the states of the robots.

The runtime behavior of a multi-robot system is given by a discrete-time transition system $\mathcal{T}$, where the state transitions are defined in the following way.

Definition 2.6 (Transition): Let $\Phi_1 = [\phi_{11}, \ldots, \phi_{1N}]$ and $\Phi_2 = [\phi_{21}, \ldots, \phi_{2N}]$ be two states of the multi-robot system, and $Prim = [prim_1, \ldots, prim_N]$, where $prim_i \in PRIM_i$, be a vector containing as elements the primitives applied to individual robot in state Φ_1 to bring them to state Φ_2. The transition from Φ_1 to Φ_2 is given by the following rule:

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

iff $\forall i \in \{1, \ldots, N\}$:

- $\phi_{1i}.q = prim_i.q_i$
- $\phi_{2i}.q = prim_i.q_f$
- $\phi_{2i}.X = \phi_{1i}.X + prim_i.X_{rf}$
- *obstacle_avoidance*$(\Phi_1, \Phi_2, Prim, OBS)$
- *collision_avoidance*$(\Phi_1, \Phi_2, Prim)$

The inputs to the predicate *obstacle_avoidance* are Φ_1 and Φ_2, the states before and after the transition, $Prim$, the vector containing the primitives applied to individual robot in state Φ_1, and OBS, the set of obstacles. This predicate is $true$ if no robot trajectory between the states Φ_1 and Φ_2 overlaps with the obstacles. Similarly, the inputs to the predicate *collision_avoidance* are Φ_1 and Φ_2 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 2.7 (Trajectory): A *trajectory* of a multi-robot system for an input problem instance $\mathcal{P} = \langle R, I, F, PRIM, OBS, L, \Psi \rangle$ is defined as a sequence of states $\Phi = (\Phi(0), \Phi(1), \ldots, \Phi(L))$ such that $\Phi(0) \in I$ and $\Phi(L) \in F$ and the states are related by the transitions in the following way:

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

With the sequence of states Φ, we associate another sequence $\sigma = (s_0, \ldots, s_L)$ of length $L+1$, where $s_i, i \in \{0, \ldots, L\}$, captures the truth assignment to the propositions used in ξ in state $\Phi(i)$.

Definition 2.8 (Valid trajectory): A trajectory Φ, with σ to be the sequence of truth assignments to the propositions used in ξ in the corresponding states in Φ, is called a *valid trajectory*, if σ satisfies the formula ξ, i.e., $\sigma, 0 \models \xi$.

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

Definition 2.9 (Motion Planning Problem): Given an input problem instance $\mathcal{P}$ with the number of hops in the trajectory for each robot to be L, synthesize a valid trajectory of length L.

C. Example

In this section, we present an illustrative example of motion planning for a group of quadrotors. A quadrotor is a nonlinear underactuated dynamical system which can be described in twelve-dimensional space (3D-position and orientation and the corresponding time derivatives). The control input for the system is net body force for each rotor. The space of all possible control inputs is denoted by $\mathbb{U} \in \mathbb{R}+^4$. This system is known to be differentially flat [30]. This implies that full state space and control inputs can be written as a function of the flat outputs and their derivatives. For the quadrotor, the set of flat outputs are the 3D position and yaw angle in an inertial frame, $y = [r_x, r_y, r_z, \psi]$. Pivtoraiko et al. [31] has provided the method of offline computation of motion primitives for a micro-UAV. We adapt their method to compute the motion primitives.

Figure 1 shows the top view of a three dimensional workspace. The black rectangular regions denote the region occupied by some obstacles. The set of obstacles OBS is given by $OBS = \{(5,0,h), (6,0,h), (7,0,h), (8,0,h), (5,1,h), \ldots\}$. We assume that the quadrotors maintain the same height h during their flight, and thus we are only interested in the obstacles occupying the blocks with h in the z component of the identifiers.

In this example, the group consists of four quadrotors. The circles labeled with $I1$, $I2$, $I3$ and $I4$ denote the initial positions of the quadrotors. Our objective is to fly the group of quadrotors from the initial location to a specified final location while avoiding the obstacles.

We want to impose the following invariant properties on the quadrotors during their flight:

- *Maintaining formation:* During the flight, the quadrotors have to be either in a straight-line in x, y or z direction, or they should maintain a rectangular formation.
- *Maintaining minimum distance:* The distance between the quadrotors may be different at different time instances, but they have to maintain a specified minimum distance between each other.
- *Maintaining precedence:* The quadrotors should maintain relative positions with respect to each other. For example, during the flight the x coordinate of the quadrotors at the location $I1$ and $I2$ should always

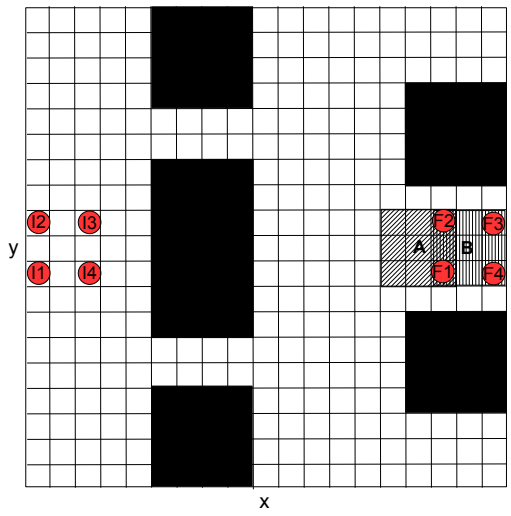


Fig. 1. A 19×19 workspace with a few obstacles.

be less than the x coordinates of the quadrotors at the location $I3$ and $I4$.

We aim to synthesize trajectories for a group of 4 quadrotors satisfying the following specification:

Spec 1: The quadrotors starting from location $I1$, $I2$, $I3$ and $I4$ should reach location $F1$, $F2$, $F3$ and $F4$ respectively, maintaining the above specified properties on formation, minimum distance and precedence.

Spec 2: The quadrotors at the location $I1$ and $I2$ should occupy any block in the rectangular region B and the other two quadrotors should occupy any block in the rectangular region A. We also want the quadrotors to maintain the properties on formation and minimum distance.

Note that we do not include the property on maintaining precedence in the second specification. This is due to the fact that the final state in the second specification does not satisfy the precedence constraint.

III. Constraint Based Motion Composition

In this section, we describe the system of constraints that model the motion planning problem introduced in Section II. Given an input problem instance $\mathcal{P} = \langle N, I, F, PRIM, OBS, L, \Psi \rangle$, our objective is to generate a system of constraints where the primitive at each state is considered to be the decision variable. For each robot $R_j \in R$, at each time instant $t \in \{0, \dots L\}$, the state of robot R_j is denoted by $\Phi_j(t)$. For each robot $R_j \in R$, at each time instant $t \in \{0, \dots L-1\}$, the primitive applied to robot R_j is denoted by $Prim_j(t+1)$. The constraints can be classified into two categories:

- **General Constraints.** These constraints are common to any motion planning problem for a set of robots. In Section III-A, we describe the general constraints related to the motion planning problem for a group of robots.
- **Property Specific Constraints.** These constraints depend on the safe LTL_f properties we wish to impose on the runtime behavior of the robots. In Section III-B we illustrate how we generate constraints to impose some runtime properties on the group of robots.

The system of constraints that we solve is conjunction of all these constraints.

A. General Constraints

Here we present the constraints that are common to any motion planning problem for a set of robots.

Initial state: The state of the group of robots at time $t = 0$ is an element of the specified set of initial states I.

$$\Phi(0) \in I \tag{III.1}$$

Final state: The state of the group of robots at the discrete time point $t = L$ is equal to an element of the set of specified final states F.

$$\Phi(L) \in F \tag{III.2}$$

Obstacle position: We use the predicate $obstacle$ to indicate whether a position is blocked by an obstacle. The input to the predicate is a position X in the workspace. The predicate is $true$ if there is an obstacle at that position. Otherwise, the predicate is $false$.

$$\begin{aligned} &\forall X \in OBS,\ obstacle(X) = true \wedge \\ &\forall Y \notin OBS,\ obstacle(Y) = false \end{aligned} \tag{III.3}$$

Primitive selection: For each robot $R_j \in R$, the primitive $Prim_j(t+1)$ at the time instance $t \in \{0, \dots, L-1\}$ is chosen from the set of primitives $PRIM_j$ for the corresponding robot.

$$\forall R_j \in R,\ \forall t \in \{0, \dots, L-1\} : Prim_j(t+1) \in PRIM_j \tag{III.4}$$

Ensuring continuity of trajectories: For each robot $R_j \in R$, at each time instant t, the position $\Phi_j(t).X$ is equal to the vector sum of its position $\Phi_j(t-1).X$ at time $(t-1)$ and the relative position associated with the primitive chosen at the time instant $(t-1)$.

$$\begin{aligned} &\forall R_j \in R,\ \forall t \in \{1, \dots, L\} : \\ &\qquad \Phi_j(t).X = \Phi_j(t-1).X + Prim_j(t).X_{rf} \end{aligned} \tag{III.5}$$

Ensuring conformance between two consecutive motion primitives: For each robot $R_j \in R$, the initial configuration of the primitive applied at each time instant t is equal to the final configuration of the primitive applied at the previous discrete time instant $(t-1)$.

$$\begin{aligned} &\forall R_j \in R,\ \forall t \in \{1, \dots, L-1\} : \\ &\qquad Prim_j(t+1).q_i = Prim_j(t).q_f \end{aligned} \tag{III.6}$$

Obstacle avoidance: This set of constraints ensures that the robots do not collide with an obstacle when they move from one point to another point. These constraints capture the predicate $obstacle_avoidance$ in Section II-B.

$$\begin{aligned} &\forall R_j \in R,\ \forall t \in \{0, \dots, L-1\},\ \forall w \in Prim_j(t+1).W : \\ &\qquad obstacle(\Phi_j.X + w) = 0 \end{aligned} \tag{III.7}$$

Collision avoidance: This set of constraints ensures that the robots do not collide with each other. These constraints capture the predicate $collision_avoidance$ in Section II-B.

$$\begin{aligned} &\forall t \in \{0, \dots, L-1\}, \forall R_i \in R, \forall R_j \in R \backslash \{R_i\} \\ &\forall w_i \in Prim_i(t+1).W, \forall w_j \in Prim_j(t+1).W : \\ &\qquad \Phi_i(t).X + w_i \neq \Phi_j(t).X + w_j \end{aligned} \tag{III.8}$$

B. Constraints Capturing Behavioral Requirements

In this section we illustrate how we capture safe LTL specification in the system of constraints. Let ξ denote the behavioral property of the multi-robot system and $\sigma = [s_0, \ldots, s_L]$ denote the sequence that captures the truth assignment to the propositions used in ξ at discrete time instants $0, \ldots, L$. For an atomic proposition π, we denote $\pi(s_i)$ as the truth assignment of π in s_i. We denote σ_i as the suffix of the sequence σ that starts from index i. If α and β are two subsequences of σ, we denote by $\alpha;\beta$ the concatenation of the two subsequences. Now, we define the encoding $\mathcal{E}$ of the safe LTL property ξ recursively as the following:

$$\begin{array}{lll} \mathcal{E}(\sigma,\xi) & = true & \text{for } \sigma = [\,] \\ & = \pi(s_0) & \text{for } \sigma = s_0;\sigma_1 \text{ and } \xi = \pi \\ & = \mathcal{E}(\sigma,\xi_1) \wedge \mathcal{E}(\sigma,\xi_2) & \text{for } \xi = \xi_1 \wedge \xi_2 \\ & = \neg\, \mathcal{E}(\sigma,\xi') & \text{for } \xi = \neg\xi' \\ & = \bigwedge_{i\in\{0,\ldots,L\}} \mathcal{E}(\sigma_i,\xi') & \text{for } \xi = \Box\xi' \\ & = \mathcal{E}(\sigma_1,\xi') & \text{for } \sigma = s_0;\sigma_1 \text{ and } \xi = \bigcirc\xi' \end{array}$$

We now illustrate the encoding above on the example invariant properties introduced in Section II-C. The properties are of the form $\xi = \Box\xi'$, where ξ' is the Boolean combination of the propositions using the position components of the states of the robots.

1) Maintaining formation: We assume that the number of robots N is a square of a natural number. Our objective is to keep the robots always either in a straight line in x, y or z direction, or in a rectangular formation. This is achieved using the following set of constraints. The variable gx_{tij} (gy_{tij}, gz_{tiz}) is set to 1 if and only if at the discrete time instance t, the x (respectively, y, z)-coordinates of robot R_i and robot R_j are the same, and either y (resp. z, x) co-ordinates or z (resp. x, y) co-ordinates of the two robots are not the same.

$$\begin{array}{l} \forall t \in \{0,\ldots,L\},\ \forall R_i \in R, \forall R_j \in R\backslash\{R_i\} : \\ (gx_{tij} = 1) \Leftrightarrow ((\Phi_i(t).X.x = \Phi_j(t).X.x) \wedge \\ ((\Phi_i(t).X.y \neq \Phi_j(t).X.y) \vee (\Phi_i(t).X.z \neq \Phi_j(t).X.z))) \\ (gy_{tij} = 1) \Leftrightarrow ((\Phi_i(t).X.y = \Phi_j(t).X.y) \wedge \\ ((\Phi_i(t).X.z \neq \Phi_j(t).X.z) \vee (\Phi_i(t).X.x \neq \Phi_j(t).X.x))) \\ (gz_{tij} = 1) \Leftrightarrow ((\Phi_i(t).X.z = \Phi_j(t).X.z) \wedge \\ ((\Phi_i(t).X.x \neq \Phi_j(t).X.x) \vee (\Phi_i(t).X.y \neq \Phi_j(t).X.y))) \end{array} \quad \text{(III.9)}$$

Now the following constraints ensure that for each discrete time instance t, the value of the sum of gx_{tij} for all robots R_i and R_j correspond to values specific to a straight line in x, y or z direction, or a rectangular formation.

$$\begin{array}{l} \forall t \in \{0,\ldots,L\} : \\ sum_gx_t = \sum_{i\in\{1,\ldots,N\}} \sum_{j\in\{1,\ldots,N\}\backslash\{i\}} gx_{tij} \ \wedge \\ sum_gy_t = \sum_{i\in\{1,\ldots,N\}} \sum_{j\in\{1,\ldots,N\}\backslash\{i\}} gy_{tij} \ \wedge \\ sum_gz_t = \sum_{i\in\{1,\ldots,N\}} \sum_{j\in\{1,\ldots,N\}\backslash\{i\}} gz_{tij} \ \wedge \\ ((sum_gx_t = N \times (N-1) \wedge sum_gy_t = 0 \ \wedge \ sum_gz_t = 0) \vee \\ (sum_gx_t = 0 \ \wedge \ sum_gy_t = N \times (N-1) \wedge sum_gz_t = 0) \vee \\ (sum_gx_t = 0 \wedge sum_gy_t = 0 \ \wedge sum_gz_t = N \times (N-1)) \vee \\ (sum_gx_t = N \times (\sqrt{N}-1) \ \wedge sum_gy_t = N \times (\sqrt{N}-1) \wedge \\ sum_gz_t = N \times (\sqrt{N}-1))) \end{array} \quad \text{(III.10)}$$

2) Maintaining minimum distance: At each discrete time instant, the distance between any two quadrotors is at least one block.

$$\begin{array}{l} \forall t \in \{0,\ldots,L\}, \forall R_i \in R, \forall R_j \in R\backslash\{R_i\} : \\ \quad | \ \Phi_i(t).X.x - \Phi_j(t).X.x \ | > 1 \ \vee \\ \quad | \ \Phi_i(t).X.y - \Phi_j(t).X.y \ | > 1 \ \vee \\ \quad | \ \Phi_i(t).X.z - \Phi_j(t).X.z \ | > 1 \end{array} \quad \text{(III.11)}$$

3) Maintaining precedence: The robots have to maintain relative position with each other. For N = 4, assuming that the robots always maintain the same height, the constraints maintaining precedence are given below:

$$\begin{array}{l} \forall t \in \{0,\ldots,L\} \\ \Phi_1(t).X.x \leq \Phi_3(t).X.x \wedge \Phi_1(t).X.x \leq \Phi_4(t).X.x \ \wedge \\ \Phi_2(t).X.x \leq \Phi_3(t).X.x \wedge \Phi_2(t).X.x \leq \Phi_4(t).X.x \ \wedge \\ \Phi_1(t).X.y \leq \Phi_2(t).X.y \wedge \Phi_1(t).X.y \leq \Phi_3(t).X.y \ \wedge \\ \Phi_4(t).X.y \leq \Phi_2(t).X.y \ \wedge \Phi_4(t).X.y \leq \Phi_3(t).X.y \end{array} \quad \text{(III.12)}$$

The following theorem presents the completeness of our method.

Theorem 3.1: **Completeness of Primitive-Based Path Planning.** Given an input motion planning problem instance $\mathcal{P}$ with the set of primitives for the robots given in $PRIM$ and a desired length of trajectory L, if the system of constraints is not satisfiable, there does not exist a trajectory of length L, that can be synthesized using the primitives in $PRIM$.

C. Rectangular Abstraction for Collision avoidance

In Equation III.8, the constraints ensure that the trajectory of one robot does not intersect the trajectory of another robot. This is performed by comparing each block on the trajectory of one robot with each block on the trajectory of another robot and ensuring that there is no common block on these two trajectories. However, block-wise ensuring collision avoidance becomes a bottleneck in solving the constraints due to the large number of constraints that are generated in this manner. To alleviate this inefficiency, we rather abstract a trajectory with a rectangular region. The rectangular region is an over-approximation of the set of blocks contained in the trajectory. Figure 2(a) and Figure 2(c) show the blocks covered by trajectories corresponding two two different primitives, and Figure 2(b) and Figure 2(d) show the corresponding rectangular over-approximation. In Figure 2(a) and Figure 2(c), the circle with a label 'I' ('F') denotes the initial (final) point on the trajectory, and the intermediate triangles denote the blocks through which the trajectory passes. To ensure collision avoidance, we can enforce that the rectangular region over-approximating the trajectories of any two robots do not intersect with each other. We introduce two new fields in the tuple representing a primitive: ll and ur. The field ll (ur) captures the relative position of the lower left (upper-right) corner of the rectangle over-approximating the area covering the relative positions of the blocks on the trajectory corresponding to the primitive. Now the following constraints ensure collision avoidance between two robots.

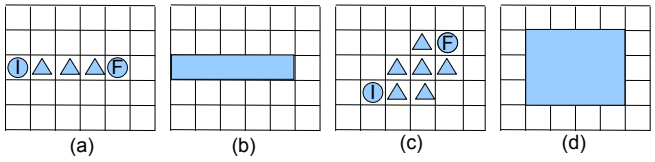


Fig. 2. Abstraction for checking collision avoidance

$$\begin{aligned}&\forall t \in \{0,\ldots,L-1\}, \forall R_i \in R, \forall R_j \in R \backslash \{R_i\} : \\ &\quad (\Phi_i(t).X.x + Prim_i(t+1).ll.x > \\ &\qquad \Phi_j(t).X.x + Prim_j(t+1).ur.x) \vee \\ &\quad (\Phi_i(t).X.x + Prim_i(t+1).ur.x < \\ &\qquad \Phi_j(t).X.x + Prim_j(t+1).ll.x) \vee \ldots \\ &\quad (\Phi_i(t).X.z + Prim_i(t+1).ur.z < \\ &\qquad \Phi_j(t).X.z + Prim_j(t+1).ll.z)\end{aligned} \tag{III.13}$$

We comment that the abstraction of the trajectories, though makes solving the motion planning problem easier, causes the loss of completeness.

D. Generating Optimal Trajectory

While generating a trajectory, an auxiliary goal is to minimize the cost for all robots for traversing through the generated trajectory. In our framework, this goal is achieved in two steps. In the first step, we find the minimum value for L such that a trajectory exists for that L. The algorithm for finding the minimum value of L is given in Algorithm III.1. The algorithm starts with guessing a suitable value for L, given by L_0. It uses two variable lb and ub to bound the search space for the optimal value of L. Initially, the values of lb and ub are set to 0 and ∞, respectively. In the first while loop, we gradually increase the value of lb by a step size of L_0. The loop terminates when we get a value for L that can generate a trajectory. When this while loop terminates, the difference between ub and lb is L_0. Now, in the second while loop we perform a binary search in the range $[lb, ub]$ to get L_{opt}, the optimal value for L. The search terminates when the difference between ub and lb becomes 1.

Once we find the value of L_{opt}, we want to find the minimal-cost trajectory of length L_{opt}. Each primitive has an associated cost. For robot R_j, the cost for using a primitive at the time instant t is denoted by $Prim_j(t+1).cost$, where $Prim_j(t+1) \in PRIM_j$. Let $total_cost$ denote the cost incurred by all the robots to move from their initial location to the final location. Thus, $total_cost$ is given by

$$total_cost = \sum_{j \in \{1,\ldots,N\}} \sum_{t \in \{0,\ldots,L-1\}} Prim_j(t+1).cost. \tag{III.14}$$

In our framework we can handle additional specification that the total cost incurred by all the robots is bounded by a specified cost C by having additional constraint

$$total_cost < C. \tag{III.15}$$

The trajectory with minimal cost can be found by solving an optimization problem, where we minimize $total_cost$ subject to the constraints described in Section III-A and Section III-B. Let C_{opt} denote the minimum value of C, such that the constraint in (III.15) together with the constraints described in Section III-A and Section III-B are satisfiable To find the value of C_{opt}, we perform a binary search in the range $[C_{min}, C_{max}]$, where C_{min} and C_{max} are given by

$$C_{min} = N \times L \times \left(\min_{k \in \{1,\ldots,N\}} \left(\min_{pr \in PRIM_k} pr.cost \right) \right) \tag{III.16}$$

and

$$C_{max} = N \times L \times \left(\max_{k \in \{1,\ldots,N\}} \left(\max_{pr \in PRIM_k} pr.cost \right) \right). \tag{III.17}$$

We comment that the method presented here finds the optimal trajectory with shortest possible length, which may not be the optimal trajectory in general. There may exist a longer trajectory with a smaller cost.

Algorithm III.1: Computation of optimal trajectory length.

Input: $\mathcal{P}$, the input motion planning problem instance

Output: L_{opt}, the minimum value for L for which a trajectory exists

```
function findOptimalTrajectoryLength(P)
begin
    L := L_0; lb := 0; ub := ∞;
    while true do
        κ := generate_constraints(P, L);
        result := solve_constraints(κ);
        if result = SAT then
            ub := L;
            break;
        else
            L := L + L_0;  lb := L;
        end
    end
    while (ub − lb > 1) do
        L = ⌈(lb + ub)/2⌉;
        κ := generate_constraints(P, L);
        result := solve_constraints(κ);
        if result = SAT then
            ub = L;  L_opt = L;
        else
            lb = L;
        end
    end
    return L_opt;
end
```

IV. Application to Quadrotor and Experimental Results

A. Tool Implementation

We implement our motion planning algorithm in a tool called Complan (for COmpositional Motion PLANner). The architecture of Complan is shown in Figure 3. Complan has two main components: (1) a tool that takes as input a problem instance $\mathcal{P}$, and generates a system of constraints in the input language of an SMT solver, and (2) an SMT solver that solves the constraints and generates a trajectory in terms of primitives to be used for all the robots at the intermediate points on the trajectory. We use Z3 [32] as the backend SMT solver to solve the system of constraints.

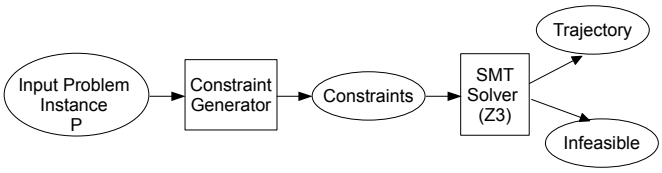
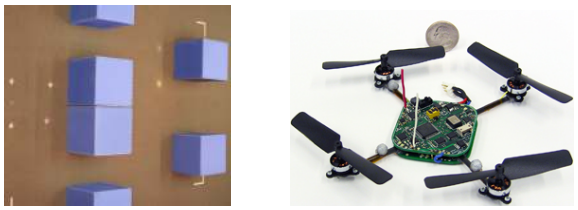


Fig. 3. Tool Architecture for Complan



(a) (b)

Fig. 4. (a) The workspace, (b) KMel Robotics NanoQuad quadrotor.

B. Workspace, Quadrotor and Motion Primitives

Figure 4(a) shows the workspace where we carry out our experiments. Figure 4(b) shows the test vehicle used in our experiment. The test vehicles are NanoQuad quadrotors from KMel Robotics [33], each one has weight of 84g. All quadrotors in our experiment share the same set of motion primitives, which has been generated using the algorithm as described in [34]. For two dimensional spaces, the velocity profile has 9 configurations consisting of one hover state and constant velocity in 8 uniform directions (N, E, S, W, NE, SE, SW, NW). To reduce the complexity in synthesis, we choose the same duration for all motion primitives. Using the duration of $1.2sec$, the algorithm in [34] yields a set of 33 motion primitives. To generate the primitives we use a cost function containing two components. The first component is the estimated energy consumption which can be derived from the control inputs. The second, the trajectory smoothness, is the weighted sum of the snap of the trajectory which penalizes the trajectories that require abrupt change in acceleration and jerk. The cost corresponding to each primitive is in the range of $[0.298, 1.201]$.

C. Results

We report two case studies based on the two specifications mentioned in Section II-C. In both case studies, the quadrotors move in a two dimensional plane. Thus, we ignore the z-coordinate when referring to their positions. In both case studies, the quadrotors initially located in the blocks with ID $(0, 8)$, $(0, 10)$, $(2, 10)$ and $(2, 8)$. In the first case study, the robots fly to the blocks with ID $(16, 8)$, $(16, 10)$, $(18, 10)$ and $(18, 8)$ respectively, satisfying the constraints on maintaining the specified formation, minimum distance and the precedence. In the second case study, we want the robots initially on the blocks with ID $(0, 8)$ and $(0, 10)$ to fly in the region marked by B in Figure 1, and the other two quadrotors to fly in the region marked by A, maintaining specified formation and minimum distance. Our attempt to generate a trajectory in the second case study reveals that it is also not possible to generate a trajectory of reasonably small length by maintaining the formation constraints. We attempt to synthesize a trajectory until $L = 20$, but no such trajectory exists. We then eliminate the constraints on maintaining formation, and are able to generate a trajectory that only satisfies the safe LTL specification on maintaining the minimum distance between any two quadrotors at each discrete time step.

Table I shows the experimental results for two case studies. Our 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. For both the case studies, we carry out our experiments with and without the rectangular abstraction for collision avoidance as described in Section III. In each case, we find the optimal value for the length L for which a solution exists, the optimal cost, the total number of calls to the SMT solver to compute the optimal trajectory (this includes steps to compute L_{opt} and the optimal cost), and the average amount of time the SMT solver took each time it was invoked. In our search for the value of L_{opt} in Algorithm III.1, we start with $L_0 = 5$. As evident from the results of both cases, our tool synthesizes the trajectories with equal length and cost whether the rectangular abstraction is used or not. However, the synthesis time reduces significantly when we use the rectangular abstraction in generating the constraints.

Figure 5(a) and Figure 5(b) show the synthesized optimal plans for the two case studies. The four quadrotors are shown using four different colors. The location of the quadrotors are time stamped at each discrete time instants. Figure 5(c) shows a non-optimal trajectory for the second case study, though the trajectories have an equal number of intermediate points to the optimal trajectories shown in Figure 5(b).

In our experiments, robots states are tracked using a Vicon motion capture system. The control inputs are computed on external computer using PID control. All codes are written in C++ and ROS. A video capturing our experimental results is available at `http://youtu.be/pSjGwhH29Zs`. Experimental results confirm that the generated trajectories satisfy the desired specifications.

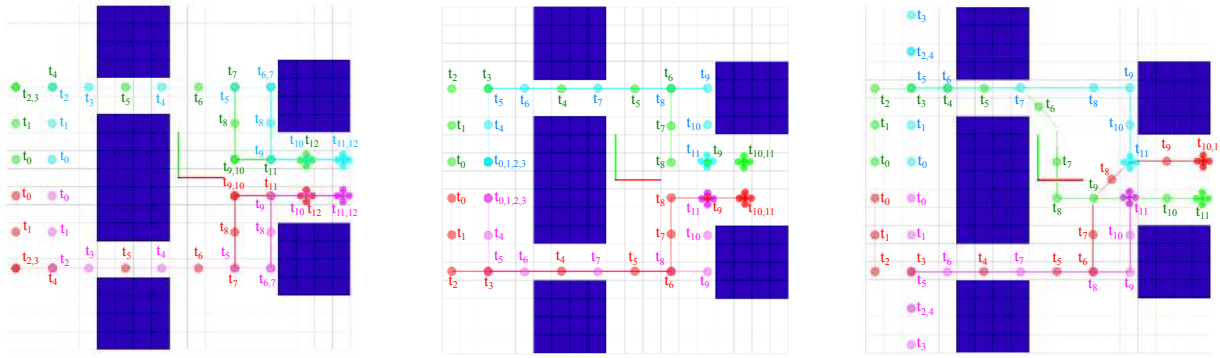
V. Discussion

We introduce a compositional multi-robot motion planning framework that uses precomputed motion primitives for a group of robots and employs an SMT solver to synthesize trajectories for the individual robots. Our planning technique is offline, and while it is neither complete nor optimal in general, we achieve completeness and optimality with respect to the given set of motion primitives. One strength of our motion planning framework is that we can synthesize plans so as to meet complex behavioral requirements specified in safe LTL. Additionally, our encoding to SMT means that our framework will directly benefit from enhancements to the expressiveness and efficiency of SMT solvers, an active research topic in recent years.

Destination Specification	Without rectangular abstraction				With rectangular abstraction			
	L_{opt}	opt. cost	# steps	avg. time	L_{opt}	opt. cost	# steps	avg. time
Spec 1	13	39.36	12	2m57s	13	39.36	12	1m18s
Spec 2	12	34.85	12	5m1s	12	34.85	12	2m9s

TABLE I

EXPERIMENTAL RESULTS ON TWO CASE STUDIES.



(a) Plan for Spec 1 (Optimal) (b) Plan for Spec 2 (Optimal) (c) Plan for Spec 2 (Sub-optimal)

Fig. 5. The trajectories for two case studies: (a) An optimal plan for specification 1, (b) An optimal plan for specification 2, and (c) A sub-optimal plan for specification 2

REFERENCES

[1] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," in *Annual Symposium on Computational Geometry*, 1998, pp. 9–18.

[2] S. Rodríguez and N. M. Amato, "Behavior-based evacuation planning," in *ICRA*, 2010, pp. 350–355.

[3] J. Jennings, G. Whelan, and W. Evans, "Cooperative search and rescue with a team of mobile robots," in *ICRA*, 1997.

[4] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots*, vol. 8, no. 3, pp. 325–344, 2000.

[5] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *IROS*, 1995, pp. 235–242.

[6] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transaction on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[7] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," in *CAV*, 1999, pp. 172–183.

[8] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *ICRA*, 2007, pp. 3116–3121.

[9] ——, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[10] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *CDC*, 2009, pp. 2222–2229.

[11] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Motion planning with hybrid dynamics and temporal goals," in *CDC*, 2010, pp. 1108–1115.

[12] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Automat. Contr.*, vol. 57, no. 11, pp. 2817–2830, 2012.

[13] Y. Chen, J. Tumova, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," in *ICRA*, 2012, pp. 5177–5182.

[14] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotic Res.*, vol. 32, no. 8, pp. 889–911, 2013.

[15] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[16] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 4, ch. 8.

[17] R. W. Brockett, "Formal languages for motion description and map making," *Robotics*, vol. 41, pp. 181–193, 1990.

[18] V. Manikonda, P. S. Krishnaprasad, and J. Hendler, "Languages, behaviors, hybrid architectures and motion control," *Mathematical Control Theory*, pp. 199–226, 1998.

[19] W. Zhang and H. G. Tanner, "Composition of motion description languages," in *HSCC*, 2008, pp. 570–583.

[20] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transaction on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[21] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[22] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-Trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[23] J. L. Ny and G. J. Pappas, "Sequential composition of robust controller specifications," in *ICRA*, 2012, pp. 5190–5195.

[24] M. Garber and M. Lin, "Constraint-based motion planning using voronoi diagrams," in *Fifth International Workshop on Algorithmic Foundations of Robotics*, 2002.

[25] W. Moss, M. C. Lin, and D. Manocha, "Constraint-based motion synthesis for deformable models," *Journal of Visualization and Computer Animation*, vol. 19, no. 3-4, pp. 421–431, 2008.

[26] R. Gayle, W. Moss, M. C. Lin, and D. Manocha, "Multi-robot coordination using generalized social potential fields," in *ICRA*, 2009, pp. 106–113.

[27] 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*, 2014, pp. 113–118.

[28] 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*, 2014, pp. 655–662.

[29] G. D. Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI*, 2013.

[30] M. J. V. Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control*, vol. 8, no. 11, pp. 995–1020, 1998.

[31] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," in *ICRA*, 2013, pp. 2452–2458.

[32] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008, pp. 337–340.

[33] "KMel robotics." [Online]. Available: http://kmelrobotics.com/

[34] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *ICRA*, 2011, pp. 2520–2525.