A Framework for Multi-Robot Motion Planning from Temporal Logic Specifications

Koo, T. John; Li, Rongqing; Quottrup, Michael Melholt; Clifton, Charles; Izadi-Zamanabadi, Roozbeh; Bak, Thomas

Published in:
Science in China. Series F: Information Sciences

DOI (link to publication from Publisher):
10.1007/s11432-012-4605-8

Publication date:
2012

Document Version
Early version, also known as pre-print

SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

month year Vol. xx No. xx: 1–19
doi:

A Framework for Multi-Robot Motion Planning from Temporal Logic Specifications

T. John Koo[1]*, Rongqing Li[1], Michael M. Quottrup[2], Charles A. Clifton[3], Roozbeh Izadi-Zamanabadi[2] & Thomas Bak[2]

[1] *Center for Embedded Software Systems, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen* 518055, *China*
[2] *Department of Automation and Control, Aalborg University, Aalborg Oest* 9220, *Denmark*
[3] *Department of Electrical Engineering and Computer Science, College of Engineering, Vanderbilt University, Nashville, Tennessee* 37235, *USA*

Received xx, xx, xxxx; accepted xx xx, xxxx

Abstract We propose a framework for the coordination of a network of robots with respect to formal requirement specifications expressed in temporal logics. A regular tessellation is used to partition the space of interest into a union of disjoint regular and equal cells with finite facets, and each cell can only be occupied by a robot or an obstacle. Each robot is assumed to be equipped with a finite collection of continuous-time nonlinear closed-loop dynamics to be operated in. The robot is then modeled as a hybrid automaton for capturing the finitely many modes of operation for either staying within the current cell or reaching an adjacent cell through the corresponding facet. By taking the motion capabilities into account, a bisimilar discrete abstraction of the hybrid automaton can be constructed. Having the two systems bisimilar, all properties that are expressible in temporal logics such as Linear-time Temporal Logic, Computation Tree Logic, and μ-calculus can be preserved. Motion planning can then be performed at a discrete level by considering the parallel composition of discrete abstractions of the robots with a requirement specification given in a suitable temporal logic. The bisimilarity ensures that the discrete planning solutions are executable by the robots. For demonstration purpose, a finite automaton is used as the abstraction and the requirement specification is expressed in Computation Tree Logic. The model checker Cadence SMV is used to generate coordinated verified motion planning solutions. Two autonomous aerial robots are used to demonstrate how the proposed framework may be applied to solve coordinated motion planning problems.

Keywords Motion planning, Multi-robot systems, Temporal logic, Hybrid automata, Discrete abstraction

Citation T. John Koo, Rongqing Li, Michael M. Quottrup, Charles A. Clifton, Roozbeh Izadi-Zamanabadi, Thomas Bak. A Framework for Multi-Robot Motion Planning from Temporal Logic Specifications.

1 Introduction

The problem of controlling mobile multi-robot systems (MRS) in a coordinated manner has become an important research issue. By properly utilizing multiple robots, the robots can accomplish an assigned mission faster and more reliably than a single robot by performing the mission in a coordinated manner. Furthermore, multiple robots can often deal with tasks that are challenging, if not impossible, to be accomplished by a single robot in application domains such as container transshipment tasks in harbors, airports, and formation keeping and control in military

*Corresponding author (email: john.koo@siat.ac.cn)

applications [1, 2, 3]. In the context of MRS, one major challenge is the need to control, coordinate and synchronize the operations of several robots to perform some specified missions collectively, while satisfying their inter- and intra-robots dynamical constraints.

A number of different approaches have been taken in order to coordinate multi-robot systems. A formalism for the composition of concurrent robot behaviors, using threaded Petri nets, has been developed in [4]. In [5] multi-robot coordination is achieved by employing a plan-merging paradigm that guarantees coherent behavior of all robots in all situations. A distributed negotiation mechanism for multi-robot coordination is considered in [6]. A hybrid control approach to action coordination and collision avoidance was taken in [7, 8]. A formal hybrid approach to the modeling and analysis of coordinated multi-robot systems was taken in [9]. Bisimular abstraction of hybrid model for robots with *nonlinear* continuous-time dynamics was introduced in [10].

The use of temporal logic as a mechanism for requirement specification and controller synthesis in mobile robotic systems has been advocated as far back as [11]. Quottrup et. al. [12] has formulated the problem of coordination of networks of robots by using timed automata with motion specification expressed in Computation Tree Logic (CTL). Fainekos et al. [13] have considered the problem of motion planning for a single, fully actuated robot in a polygonal environment in order to satisfy formulae expressed in Linear-time Temporal Logic (LTL). In [14], abstractions are obtained for more complicated dynamics, such as affine systems in simplices, multi-affine in rectangles. Even more complicated dynamics (such as unicycles) can be handled if an extra (continuous) abstraction level is allowed. [15] considers the problem of controlling a planar robot in a polygon so that its trajectory satisfies an LTL formula. A fully automated framework for control of continuous-time *linear* control systems from specifications was provided by Kloetzer and Belta [16]. A single robot was used as an illustrative example. Kloetzer and Belta [17, 18, 19] have proposed a framework for motion planning in a partitioned environment. The robot is modeled as a transition system and algorithmic methods are used to generate motion plans for the robots that satisfy the task requirement specification. In [20], temporal logic constraints are used for optimal path planning of a robot for surveillance. In [21], the temporal logic motion planning problem for mobile robots that are modeled by continuous-time second order *linear* dynamics is investigated. However, the aforementioned temporal logic based approaches only applied to robots with either no or linear closed-loop dynamics.

In this paper we provide a framework for the coordination of a MRS by using temporal logic to formulate the mission specifications. The framework is extended from our previous works[10, 12] to consider a more general problem setting on the environment, the robot dynamics and the system composition. We assume a regular tessellation has been used to partition the space of interest into a union of disjoint regular and equal cells with finite facets, and each cell can only be occupied by a robot or an obstacle. Without imposing too much restriction on the robot dynamics, we assume that each robot has finitely many modes of operation that enable the robot either to stay within the current cell or to reach an adjacent cell through the corresponding facet. This framework provide the flexibility on allowing each robot being assumed to be equipped with a finite collection of continuous-time *nonlinear* closed-loop dynamics for defining the modes of operation. A hybrid automaton model can be used to capture the finite collection of robot dynamics. By considering the motion capabilities, a bisimilar discrete abstraction of the hybrid automaton can be constructed. Having the two systems bisimilar, all properties that are expressible in temporal logics such as LTL, CTL, and μ-calculus can be preserved. Therefore, on one hand, motion planning of robots can be performed at a discrete level by considering the parallel composition of discrete abstractions of the robots and a requirement specification expressed in some suitable temporal logic. On the other hand, the bisimilarity ensures that the discrete planning solutions are executable by the robots with continuous dynamics. The result is a framework which captures realistic robot dynamics in a discrete abstraction and allows the use of verification methods to generate motion plans for a MRS such that a requirement specification is met.

For demonstration purpose, a finite automaton is used as the discrete abstraction and the requirement specification is expressed in CTL. We use Cadence SMV [22, 23, 24] as a model checker for generating and verifying coordinated motion planning solutions. Furthermore, we are interested in the specification for having the robots reaching their goals eventually, while always avoiding collision. A feasible path for the robots can be generated as a counterexample to the *negation* of a given specification. Notice that the proposed framework does not impose any restrictions on the type of temporal logic or model checkers that should be used. Hence, depending on the nature of the problem, a suitable temporal logic along with a proper model-checker could be used. Two autonomous aerial robots will be used for illustrating the design challenges in motion planning and coordination of MRS.

This paper is organized as follows: In Section 2 the environment and MRS are modeled. The embedding of a generic hybrid automaton into a labeled transition system and the abstraction of the transition system are described in Section 3. Section 4 describes a system implementation of the networked finite automata along with requirement specification. In Section 5, experimental results are presented. Conclusion and discussion of the proposed framework are provided in Section 6.

2 System Modeling

In this section, environment model and robot model are introduced. Then, the assumptions made on the robot motions are presented.

2.1 Environment Model

Consider a continuous state space $Y \subseteq \mathbb{R}^m$. A family $\pi = \{Y_j\}$ of non-empty subsets of Y is called a partition of Y which satisfies the following two properties: $Y = \bigcup_j Y_j$ and $Y_i \cap Y_j = \emptyset,\ \forall i \neq j$. The partition π with a collection of Y_j cells induces an equivalence relation. In this context, the induced equivalence relation $\approx$ is called cell equivalence and is defined over the continuous state space Y. For any two positions $y', y'' \in Y,\ y' \approx y''$ iff there exists j such that $y', y'' \in Y_j$. The cell equivalence relation $\approx$ has finitely many equivalence classes, which are precisely the collection of cells Y_j.

We shall use a specific partition called regular tessellation. Regular tessellation is defined as a partition of space into the union of a set of disjoint regular and equal cells which can be regular polygons (in two dimensions), polyhedra (three dimensions), or polytopes (m dimensions). An m-dimensional polytope is bounded by a number of (m-1)-dimensional facets. These facets are themselves polytopes. In two dimensions, there are only three possible regular tessellations, squares, equilateral triangles, or regular hexagons. In three dimensions, a polyhedron which is capable of tessellating space is called a space-filling polyhedron. Examples include the cube, rhombic dodecahedron, and truncated octahedron.

Here, we assume that the set Y can be decomposed by a regular tessellation $\pi = \{Y_j\}$ of M non-empty cells of Y and N facets for each Y_j. The N adjacent cells of Y_j are labelled by $\{Y_k\}$ with $k \in \mathcal{I}_j$, where $\mathcal{I}_j$ is the set of indexes of $\{Y_j\}$'s adjacent cells . The regular tessellation can be chosen such that (i) the size of each cell can have at least one robot occupied while having an obstacle occupying one or more cells depending on its shape and size; (ii) the shape of the cell should be chosen to conform to the motion capabilities of the robots such that a robot can reach an adjacent cell via the corresponding facet in finite time. Furthermore, there is a trade-off between granularity and problem complexity which should be considered in deciding the tessellation.

2.2 Robot Model

For each robot, we assume that there are $N+1$ modes of operation labelled by q_i with $i = 0, 1, \ldots, N$ and the continuous dynamics associated with the modes are specified in the form of $\dot{y} = f(q_i, y)$ for describing the motion capabilities of the robot designed for a) staying within the current cell in mode q_0 and b) reaching the ith neighboring adjacent cell through the corresponding facet in mode q_i with $i \in \{1, \ldots, N\}$. Hence, if initially the robot is in cell Y_j, it can stay in the same cell by using mode q_0 or move to the ith adjacent cell by using mode q_i.

In order to quantitatively define the motion capabilities of the robots, we consider the following temporal operators $\Diamond$, $\Box$, $\Diamond\Box$ and the universal path quantifier A as defined in [25, 26]. Consider an initial set F_s and a set F with $F_s, F \subseteq Y$. We define the properties $A\Diamond F$, $A\Box F$, $A\Diamond\Box F$ with respect to the trajectory $y(t)$ of $\dot{y} = f(q_i, y)$ with initial conditions specified by $F_s \subseteq F$ as follows: (i)$A\Diamond F$ is true iff $\forall y(0) \in F_s\ \exists t \in [0, \infty),\ y(t) \in F$; (ii)$A\Box F$ is true iff $\forall y(0) \in F_s\ \forall t \in [0, \infty),\ y(t) \in F$; and (iii)$A\Diamond\Box F$ is true iff $\forall y(0) \in F_s\ \exists t_0 \in [0, \infty),\ \forall t \geqslant t_0,\ y(t) \in F$.

By using the properties, we can then specify the assumption made on the motion capabilities of the robots.

Assumption 1. Consider a robot with a collection of modes of operation $\{q_0, q_1, \ldots, q_N\}$, a cell Y_j also as the initial set, the robot satisfies the followings:

1. $A\Box Y_j$ is true for q_0;
2. $A\Diamond\Box Y_k \bigwedge A\Box(Y_j \cup Y_k)$ is true for $q_i \in \{q_1, \ldots, q_N\}$ and $k \in \mathcal{I}_j$;

where given the cell index j and the mode index i, the next cell index k can be determined.

The first condition is to ensure that in mode q_0 the continuous state y is kept positively invariant in Y_j. The second condition is to enable that when for any adjacent cell Y_k of Y_j with $k \in \mathcal{I}_j$, by using mode q_i the continuous state starting from any point in Y_j can eventually reach somewhere in Y_k and then keep staying in Y_k, furthermore the state is always within Y_j and Y_k. In Section 5, we will show the implementation of a controller design for the aerial robots which can satisfy the aforementioned assumption with experimental results.

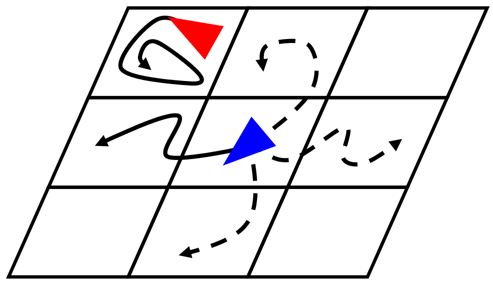


Figure 1: Each robot can have its distinct collection of modes of operation for staying within the current cell or for moving to an adjacent cell.

Given the modes of operation, a hybrid automaton can be used to model the motion of the robot among the cell Y_j and its adjacent cells $\{Y_k\}$ with $k \in \mathcal{I}_j$ by the following input σ. The hybrid automaton is defined as $H = (Q, Y, \Sigma, \mathit{Init}, f, D, G)$, where

- $Q = \{q_0, q_1, \cdots, q_N\}$ is the set of discrete states,
- $Y \subseteq \mathbb{R}^m$ is the continuous state space,
- $\Sigma = \{\sigma_1, \cdots, \sigma_N\}$ is the alphabet for input $\sigma \in \Sigma$,
- $\mathit{Init} = \{q_0\} \times Y_j$ is the initial set,
- f is the vector field defined by $\dot{y} = f(q, y)$ for $q \in Q$,
- D is the domain defined by

$$D(q) = \begin{cases} \{\sigma_0\} \times Y_j & \text{if } q = q_0, \\ \{\sigma_i\} \times (Y_j \cup Y_k) & \text{if } q = q_i. \end{cases}$$

- G is the guard relation defined by

$$G(q, q') = \begin{cases} \sigma_i & \text{if } q = q_0 \text{ and } q' = q_i, \\ \sigma_0 & \text{if } q = q_i \text{ and } q' = q_0. \end{cases}$$

The hybrid automaton H starts in the hybrid state $\mathit{Init} = \{q_0\} \times Y_j$. Hence, the robot can start in the discrete state q_0 at an arbitrary position within the continuous space Y_j. Each discrete state q, which has a continuous dynamics embedded, is treated as a mode of operation of the robot for reaching a specific cell among the N adjacent cells. In the q_0 mode, the robot stays within the cell Y_j with dynamics specified by $\dot{y} = f(q_0, y)$. In other modes q_i with $i = 1, \ldots, N$, the robot transits to the adjacent cell Y_k according to the continuous dynamics specified by $\dot{y} = f(q_i, y)$.

In q_0 the hybrid automaton H can accept any input from the set of events $\Sigma \setminus \{\sigma_0\}$ as defined by the guard relation G. If the input is σ_i, the guard $G(q_0, q_i)$ is enabled but the domain $D(q_0)$ is violated and hence the hybrid automaton H takes immediate transition to q_i. In $q_i \in Q \setminus \{q_0\}$ the hybrid automaton H accepts only the input σ_0 and takes the transition back to q_0.

2.3 Robot Motions

Here, we are interested in the reachability of the robots in the environment. In [10] the time-abstract transitions for describing the continuous transitions of a hybrid automaton have been introduced.

Time-abstract transition is defined as the type of continuous transition associated with hybrid automaton H. Time-abstract transition is essential in the process of embedding the hybrid automaton H into the class of labeled transition systems and subsequently for obtaining a finite quotient transition system. Define $\phi(t, q, y_0)$ as the solution of the differential equation $\dot{y} = f(q, y)$ with $y(0) = y_0$ for $t \geqslant 0$.

Definition 1 (σ-labeled transition)**.** Consider $y', y'' \in Y$ and $\sigma \in \Sigma$, the σ-labeled transition is defined as

$$y' \xrightarrow{\sigma} y'' \text{ iff } \exists \delta \in \mathbb{R}_{\geqslant 0}\ y'' = \phi(\delta, q, y').$$

This transition is defined for some period of time δ and it describes the continuous transition in discrete state $q \in Q$ with input $\sigma \in \Sigma$. The introduction of time-abstract transitions allows us to define cyclic transitions.

Definition 2 (σ-labeled cyclic transition)**.** Consider y', y'''
$\in Y$ and $\sigma_i \in \Sigma \setminus \{\sigma_0\}$, the σ_i-labeled cyclic transition is defined as

$$y' \overset{\sigma_i}{\Longrightarrow} y''' \text{ iff } \exists y'' \in Y\ y' \xrightarrow{\sigma_i} y'' \xrightarrow{\sigma_0} y'''.$$

Notice that we further assume that after each σ_i-labeled cyclic transition occurs, the continuous part of the domain in q_0, $D_x(q_0)$, is redefined as the reached adjacent cell. The hybrid automaton H can operate continuously by taking the cyclic transitions.

Given the partition π, due to the definition of adjacent cells, there are at most N possible adjacent cells for each cell. However, for the cells at the boundary of the partition π, we assume that there is at least one adjacent cell that can be reached. For each adjacent cell, there exists a mode of operation that can make the robot move towards the adjacent cell. Due to the properties of the system modeled by the hybrid automata H satisfying Assumption 1, one can easily show that a robot could start anywhere within the cell and then reach somewhere inside the adjacent cell in finite time and the robot can keep staying in the reached adjacent cell while without leaving the cell and its adjacent one at any time. Hence, we have the following result.

Theorem 1. Consider a hybrid automaton H with $(q_0, y') \in \mathit{Init}$, a finite partition of the continuous state space $Y \subseteq \mathbb{R}^m$ defined by $\pi = \{Y_j\}_{j=1}^M$. Given a cell $Y_j \in \pi$ and an adjacent cell Y_k, if H satisfies the properties defined in Assumption 1, there exists $\sigma_i \in \Sigma \setminus \{\sigma_0\}$ such that for all $y' \in Y_j$ there exists $y''' \in Y_k$ with $y' \overset{\sigma_i}{\Longrightarrow} y'''$.

Proof. For the given adjacent cell Y_k of Y_j, there is an input $\sigma_i \in \Sigma \setminus \{\sigma_0\}$ associated with Y_k. When $q(t') = q_0$ and $y(t') = y' \in Y_j$ at time t', σ_i is applied to H and hence σ_i-labeled transition occurs. Due to the property 2 of Assumption 1, $\exists t'' \geqslant t'$, $y'' = \phi(t'' - t', q_i, y') \in Y_k$. Then, when $q(t'') = q_i$ and $y(t'') = y'' \in Y_k$ at time t'', σ_1 is applied and hence σ_1-labeled transition occurs. Since the property 1 of Assumption 1 is satisfied, $\forall t''' \geqslant t''$, $y''' = \phi(t''' - t'', q_1, y'') \in Y_k$ and $q(t''') = q_1$. Hence the result.

3 Discrete Abstraction

The hybrid automaton H is embedded into the class of labeled transition systems with observations. Next a bisimular abstraction into a quotient transition system is obtained. Hence, the reachability properties of the labeled transition system can be preserved by a discrete abstraction. This bisimilar abstraction can be captured by a finite automaton and MRS problems can be represented by a network of interacting finite automata.

3.1 Embedding the Hybrid Automaton

In order to indicate the occupancy of the cells, we introduce a finite set of observations O associated with the finite set of cells defined by the partition $\pi = \{Y_j\}_{j=1}^{M}$ of the continuous state space Y. The labeled transition system associated with hybrid automaton H is defined as $T_h = (Q_h, \Sigma_h, \Longrightarrow_h, O, \Upsilon_h)$, where

- $Q_h = Y$ is the set of states,
- $\Sigma_h = \Sigma \setminus \{\sigma_0\}$ is the set of labels,
- $\Longrightarrow_h \subseteq Y \times \Sigma_h \times Y$ is the transition relation defined by $y \overset{\sigma}{\Longrightarrow}_h y'$ if $y, y' \in Y$,
- $O = \mathbb{B}^M$ is the set of observations, where $\mathbb{B} = \{0, 1\}$,
- $\Upsilon_h : Q_h \to O$ is the observation map defined as

$$\Upsilon_h(y) = \begin{bmatrix}\Upsilon_{h_1}(y) & \Upsilon_{h_2}(y) & \ldots & \Upsilon_{h_M}(y)\end{bmatrix}^{\mathrm{T}},$$

  where $\Upsilon_{h_j} : Y \to \mathbb{B} = \{0, 1\}$, for $j = 1, \ldots, M$ is defined as

$$\Upsilon_{h_j}(y) = \begin{cases} 1 & \text{if } y \in Y_j, \\ 0 & \text{otherwise.} \end{cases}$$

The transition system T_h is infinite since the set of states Q_h is defined as the continuous state space Y. However, the set of observations O is finite since the partition π is finite.

3.2 Constructing the Abstraction

The set of all equivalence classes Y_j in Y is called the quotient space $Y/\!\approx$ of Y induced by the cell equivalence relation $\approx$. The quotient space $Y/\!\approx$ is defined as $Y/\!\approx\, = \pi$, that is the set consisting of all equivalence classes Y_j of cell equivalence relation $\approx$. Given the cell equivalence relation $\approx$ there is a canonical projection map $\Psi_h : Y \to Y/\!\approx$ defined as $\Psi_h(y) = Y_i$ if $y \in Y_i$, which sends each $y \in Y$ to its equivalence class Y_i. The quotient transition system obtained from the labeled transition system T_h is defined as $T_h/\!\approx\, = (Q_h/\!\approx, \Sigma_h, \Longrightarrow_h /\!\approx, O, \Upsilon_h/\!\approx)$, where

- $Q_h/\!\approx\, = Y/\!\approx$ is the set of states,
- $\Longrightarrow_h /\!\approx\, \subseteq Q_h/\!\approx \times \Sigma_h \times Q_h/\!\approx$ is the transition relation defined by $Y_i \overset{\sigma}{\Longrightarrow}_h /\!\approx Y_j$ iff there exists $y \in Y_i$ and $y' \in Y_j$ such that $y \overset{\sigma}{\Longrightarrow}_h y'$ in T_h,
- $\Upsilon_h/\!\approx\, : Q_h/\!\approx\, \to O$ is the observation map defined by $\Upsilon_h\!\approx\!(\Psi_h(y)) = \Upsilon_h(y)$.

The labeled quotient transition system $T_h/\!\approx$ is finite since $Q_h/\!\approx$, Σ_h and O are finite. Note, that Σ_h and O are inherited from T_h.

In order to show that the cell equivalence relation $\approx$ is a bisimulation of transition system T_h associated with hybrid automaton H, we can use the following *Characterization* Proposition:

Proposition 1. [27] Consider transition T_h, and *observation-preserving* partition $\approx$ with quotient map $\Psi_h : Y \to Y/\!\approx$. Then $\approx$ is a bisimulation of T_h if and only if for all the states $y \in Q_h$ and for all $\sigma \in \Sigma_h$, we have

$$\Psi_h(Post_\sigma(\Psi_h^{-1}(\Psi_h(y)))) = \Psi_h(Post_\sigma(y))$$

where $Post_\sigma(P) = \{y' \in Q_h | \exists y \in P \text{ with } y \overset{\sigma}{\Longrightarrow}_h y'\}$.

By constructing the observation map, Υ_h, according to the partition as shown above, it can be concluded that the cell equivalence relation $\approx$ is *observation preserving* [27], *i.e.* if $y' \approx y''$ then $\Upsilon_h(y') = \Upsilon_h(y'')$. Using the result in [27] it can be show that $\approx$ is a bisimulation of T_h. By construction, $T_h/\!\approx$ automatically simulates T_h. The bisimulation property then ensures that T_h also simulates $T_h/\!\approx$, hence T and $T/\!\approx$ are bisimilar. As they are bisimilar, T and $T/\!\approx$ have equivalent reachability properties. Hence, checking any property expressible by a temporal logic formula for $T/\!\approx$, which is discrete and finite, can be performed equivalently on the bisimilar system T_h.

3.3 System Composition

Given the bisimilar discrete abstraction, a finite automaton can be constructed for preserving the reachability properties of the robot model composition of robots now formally be defined within the framework defined for finite automata. Finite automaton A_r is used as a template for defining robot processes $A_{r_1}\|\ldots\|A_{r_N}$, where $\|$ denotes parallel composition. We will consider both settings in which this composition is synchronous or asynchronous. To allow the robot processes to move concurrently in the environment a robot controller A_{c_i} for $i = 1,\ldots,N$ is associated with each robot. A set of static obstacles $A_{o_1}\|\ldots\|A_{o_M}$ are created from an automaton template.

For the resulting network of interacting finite automata $(A_{r_1}\|\ldots\|A_{r_N})\|(A_{c_1}\|\ldots\|A_{c_N})\|(A_{o_1}\|\ldots\|A_{o_M})$, model checkers for finite automata can be used to generate and verify coordinated motion planning solutions for the network of robots, given a requirement specification for the network in some suitable temporal logics. The sequence of input synchronization actions, generated by the model checker, can subsequently be used to control the network of robots $H_1,\ldots,H_N$ such that the requirement specification is satisfied.

4 System Implementation

In this section, a 2-dimensional case study is used to demonstrate how the framework can be implemented to solve the coordinated motion planning problem of robots in a partitioned environment. As shown in Figure 2, the partition π is constructed by placing a two-dimensional grid over the continuous state space $Y \subseteq \mathbb{R}^2$. The obtained partition is composed of identical square cells with length $\epsilon > 0$. The local motion of a robot is specified by the hybrid automaton H which restricts the robot movement from the current cell to only one of the adjacent cells.

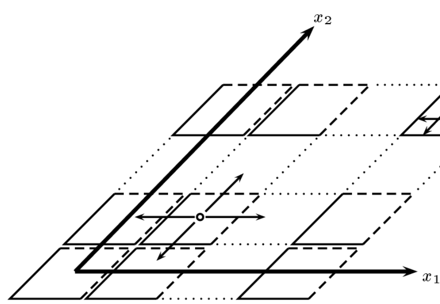


Figure 2: Partition π of the continuous space Y into a finite number of cells.

As described above, the environment, the robots and controllers are modelled by finite automata.

Definition 3 (Finite Automaton)**.** A Finite Automaton is a tuple (L, l_0, A, E) , where:

- L is a set of states,
- $l_0 \in L$ is the initial state,
- A is a finite alphabet,
- $E \subseteq L \times A \times L$ is the set of edges between states with a command.

For demonstration purpose, we are interested in having the requirement specification expressed in Computation Tree Logic (CTL) [22] and using Cadence SMV [22, 23, 24] as the model checker for generating and verifying coordinated motion planning solutions. The model checker Cadence SMV is designed to check CTL formulae against a finite automaton model.

CTL formulae can be defined inductively via a Backus Naur form as following:

$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi\wedge\phi) \mid (\phi\vee\phi) \mid (\phi\rightsquigarrow\phi) \mid AX\phi \mid EX\phi \mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi U\phi] \mid E[\phi U\phi]$

where p ranges over a set of atomic formulae.

In CTL formulae, a temporal connective is a pair of symbols. The first of the pair is A or E. A means 'along All paths' and E means 'along at least (there Exists) one path'. The second one of the pair is X, F, G, or U, meaning 'neXt state', 'some Future state', 'all future states' and Until, respectively.

To check a model, we should first construct an abstract model in the SMV input language, and specify properties using CTL. Both the system model and property specifications can be represented by binary decision diagrams (BDD). The SMV system uses the BDD-based symbolic model checking algorithm to determine whether specifications expressed in CTL are satisfied or not. If satisfied, SMV will give the result of truth, or else, report unsatisfied and give a counterexample.

The time complexity in model checking CTL formulae ϕ is $O(|S| \cdot |\phi|)$ [23] , where $|S|$ is the size of state space of the system model and $|\phi|$ is the length of formula ϕ. The time complexity of checking CTL formulae is linear in the state space of system model and the length of formula.

4.1 Modeling the Environment

A set of discrete states Z is represented in an occupancy table which is modeled as a two-dimensional integer array `Z: array 0..Z_1 of array of Z_2 of 0..1` in SMV, where `Z_1`, `Z_2` $\in \mathbb{Z}$ define the size of the array in the x_1 and x_2 direction, respectively. Thus, elements of the array represent discrete positions, where each discrete position can be assigned the value 0 (free) or 1 (occupied). A particular element $(1, 2)$ of the array `Z` is marked occupied by the assignment `Z[1][2] := 1`. By default all elements of the array `Z` are initialized to zero. Static obstacles may be present in the environment where the robots are moving. A static obstacle is modeled as an automaton $A_o = (L, l_0, E)$, where

- $L = \{l_0, l_1\}$ is the set of states,
- $l_0 \in L$ is the initial state,
- $E \subseteq L \times L$ is the set of edges, where an edge contains a source and a target state. The edges consist of $e_{00} = (l_0, l_0)$ and $e_{01} = (l_0, l_1)$.

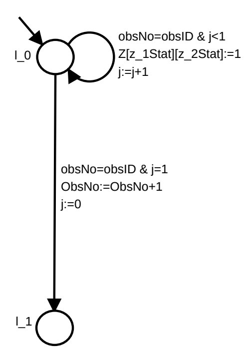


Figure 3: Finite automaton for one static obstacle.

Automaton A_o modeling one static obstacle is graphically shown in Figure 3. In order to encode the occupancy of a cell by the automaton A_o, the guard of A_o is augmented with additional conditions. Automaton A_o starts in the initial state l_0, when the guard `obsNo = obsId and j < 1` is enabled, the assignment `Z[z_1Stat][z_2Stat] := 1` is performed and the index variable `j` is incremented. The edge from l_0 to l_1 will then become fired since the guard `obsNo = obsID and j = 1` is satisfied, resulting in an update of index variable `j` to zero, an increment of `obsNo`. This automaton is used as a module for declaring static obstacles processes. Processes modeling the static obstacle module can be declared with the parameters specified in Table 1.

Table 1: Template parameters for one static obstacle.

Parameter	Type	Description
`obsID`	int	Unique identifier for static obstacle
`z_1Stat`	int	Static position of obstacle in x_1 direction
`z_2Stat`	int	Static position of obstacle in x_2 direction

4.2 Finite Automaton Model of Robot

Recall, that the goal is to generate a set of collision-free paths for the network of robots which satisfy a formal requirement specification in CTL and which enable the network of robots to eventually reach their goal positions. In CTL, the *"eventually reach goal position"* property is specified as a reachability property whereas the *"collision-avoidance"* property is specified as a safety property. This problem is solved in two steps: (i) The collision-avoidance property is guaranteed by using a correct-by-construction principle where the collision-avoidance property is embedded in the finite automaton modeling each robot. (ii) The eventually reaching goal position property is ensured for each robot in the network by using the model checker.

An SMV module is now constructed from the finite transition system T_t. The finite automaton associated with T_t is defined as $A_r = (L, l_0, A, E,)$, where

- $L = \{l_0, l_1, l_2, l_3, l_4, l_5\}$ is the set of states,
- $l_0 \in L$ is the initial state,
- $A = \{\texttt{sigma\_2}, \texttt{sigma\_3}, \texttt{sigma\_4}, \texttt{sigma\_5}, \texttt{TRUE}\}$ is the set of input commands,
- $E \subseteq L \times A \times L$ is the set of edges, where an edge contains a source state, a guard to be satisfied, a command to be received, and a target state. The edges consist of $e_{00} = (l_0, \texttt{TRUE}, l_0)$, $e_{01} = (l_0, \texttt{TRUE}, l_1)$, $e_{i1} = (l_i, \texttt{TRUE}, l_1)$ for $i = 2, \ldots, 5$, and $e_{1j} = (l_1, \texttt{sigma\_j}, l_j)$, for $\texttt{j} = 2, \ldots, 5$, where e_{ij} denotes the edge from location l_i to l_j.

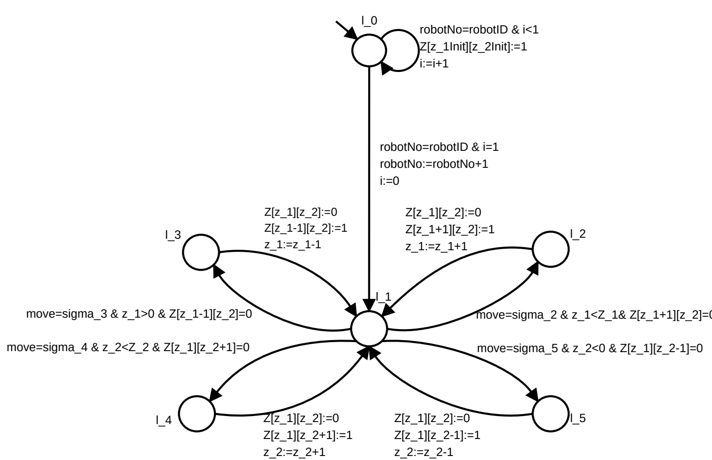


Figure 4: Finite automaton for one robot.

The finite automaton A_r starts in state l_0. In this state the robot is placed on its initial discrete position as specified by `z_1Init` and `z_2Init`. An enumeration typed state variable `move` is used to store the command `sigma_i` for `i` $= 2, \ldots, 5$. Thus, when a transition of A_r is taken, the guard `move=sigma_i` for `i` $= 2, \ldots, 5$ could be evaluated first. In state l_1 the robot can move from the initial cell to an adjacent cell in the partition given that one of the edges are fired and the associated controller sends the corresponding command. In state l_1 the finite automaton is ready to receive a command `sigma_i` for `i` $= 2, \dots, 5$ from the associated controller. However, in order to avoid the robot from moving to an occupied cell, the guard is augmented with additional conditions. If the edge e_{12} is fired and the command `sigma_2` is received the finite automaton fires the edge e_{12} to state l_2. Note that the edge e_{12} is only fired if the adjacent cell is free `Z[z_1+1][z_2] = 0` and within the defined partition, i.e. `z_1 < Z_1`. The adjacent cell is marked occupied `Z[z_1+1][z_2] := 1` when the edge e_{12} is fired. In state l_2 the movement towards the adjacent cell is performed for a fixed step. Then, the edge e_{21} is fired and a transition is taken back to state l_1. When the edge e_{21} is fired, the previous cell is marked free `Z[Z_1][Z_2] := 0` and the discrete position of the robot is updated `z_1 := z_1 + 1`. Parameters for declaring the robot module are specified in Table 2. The augmented finite automaton A_r modeling one robot is graphically illustrated in Figure 4.

Table 2: Template parameters for one robot and one robot controller.

Parameter	Type	Description
`robotID`	int	Unique identifier for robot
`z_1Init`	int	Initial position of robot in x_1-direction
`z_2Init`	int	Initial position of robot in x_2-direction
`sigma_2`	bool	Command to move robot in x_1-direction
`sigma_3`	bool	Command to move robot in $-x_1$-direction
`sigma_4`	bool	Command to move robot in x_2-direction
`sigma_5`	bool	Command to move robot in $-x_2$-direction

4.3 Finite Automaton Model of Controller

A controller is associated with each finite automaton modeling a robot. A controller for each robot is needed as the system consists of a network of concurrent robots moving in the environment. The robot controller is modeled as an automaton $A_c = (L, l_0, A, E)$, where

- $L = \{l_0\}$ is the set of states,
- $l_0 \in L$ is the initial state,
- $A = \{\texttt{sigma\_2}, \texttt{sigma\_3}, \texttt{sigma\_4}, \texttt{sigma\_5}, \texttt{TRUE}\}$ is the set of output commands,
- $E \subseteq L \times A \times L$ is the finite set of edges, where an edge contains a source state, and a target state. The edges consist of $e_i = (l_0, \texttt{sigma\_i}, l_0)$ for `i` $= 2, \dots, 5$.

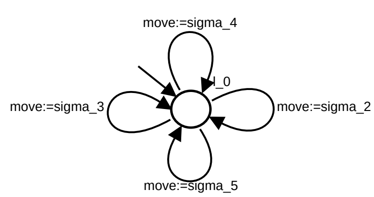


Figure 5: Finite automaton for one robot controller.

The robot controller automaton is shown in Figure 5. The automaton A_c starts in state l_0. In this state the automaton can send an output command `sigma_i` $\in A$ to the finite automaton A_r modeling a robot. The set of output command A represent all possible movements of a robot. The process of sending commands is implemented by using `move:=sigma_i` for `i` $= 2, \dots, 5$. Thus, if the automaton A_c takes the edge $e_2 = (l_0, \texttt{sigma\_2}, l_0)$ then finite automaton A_r will take the corresponding edge $e_{12} = (l_1, \texttt{sigma\_2}, l_2)$. The automaton A_c is used as a module for declaring control process instances. Furthermore, notice that it takes transitions in a nondeterministic manner in order to enable the generation of all possible movements.

4.4 Requirement Specification

Consider a network of two robots R_1 and R_2 with initial and goal position as shown in Figure 6. They have to move from initial to goal positions while avoiding collision with each other and the static obstacles. The system to be model checked consists of the following processes: Robots R_1 and R_2, robot controllers C_1 and C_2, and static obstacles $O_1, \ldots, O_{32}$.

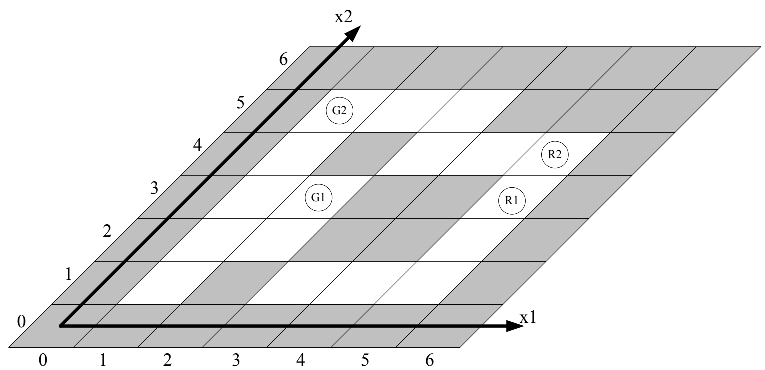


Figure 6: Setup for network of two robots R_1 and R_2 where G_1 and G_2 represent goal positions.

4.4.1 Reachability Properties

The reachability properties are used for the generation of a feasible motion plan, that will move the robots from their initial to goal positions, while avoiding collision among robots and obstacles.

Property 1 (Reachability)

Does a location trajectory exist where the robots R_1 and R_2 eventually reach their goal positions G_1 and G_2?

```
EF (contr1.rob.z_1 = 2  & contr1.rob.z_2 = 3
& contr2.rob.z_1 = 1  & contr2.rob.z_2 = 5)
```

Property 2 (Reachability with step requirement)

Does a location trajectory exist where the robots R_1 and R_2 eventually reach their goal positions G_1 and G_2 in less than 15 steps?

```
EF (contr1.rob.z_1 = 2 & contr1.rob.z_2 = 3
& contr2.rob.z_1 = 1 & contr2.rob.z_2 = 5
& contr1.rob.step < 15 & contr2.rob.step < 15)
```

Property 1 expresses the behavior that the robots eventually will reach their goal positions, whereas Property 2 expresses the behavior that the robots eventually will reach their goal positions within a given step constraints; here specified by 15 steps.

4.4.2 Safety Properties

The safety properties are used to check if collision avoidance is achieved among the robots when moving and static obstacles and also that the robots will move within the environment.

Property 3 (Collision avoidance)

Does all location trajectories the robots R_1 and R_2 take ensure they never collide after they both start to move, i.e. for step > 0?

```
AG ! ((contr1.rob.z_1 = contr2.rob.z_1)
```

```
& (contr1.rob.z_2 = contr2.rob.z_2)
& contr1.rob.step > 0  & contr2.rob.step > 0)
```

Property 4 (Bounded movement)

Does location trajectories the robots R_1 and R_2 ensure that they move within the boundaries of the environment?

```
AG (contr1.rob.z_1 >= 0  & contr1.rob.z_1 <= Z_1
& contr1.rob.z_2 >= 0  & contr1.rob.z_2 <= Z_2
& contr2.rob.z_1 >= 0  & contr2.rob.z_1 <= Z_1
& contr2.rob.z_2 >= 0  & contr2.rob.z_2 <= Z_2)
```

Property 3 expresses the requirement that collision avoidance is achieved among the robots once they start to move. Further, the requirement that the robots always move within the boundaries of the partition is expressed in Property 4.

Table 3: Comparison of synchronous composition and asynchronous composition.

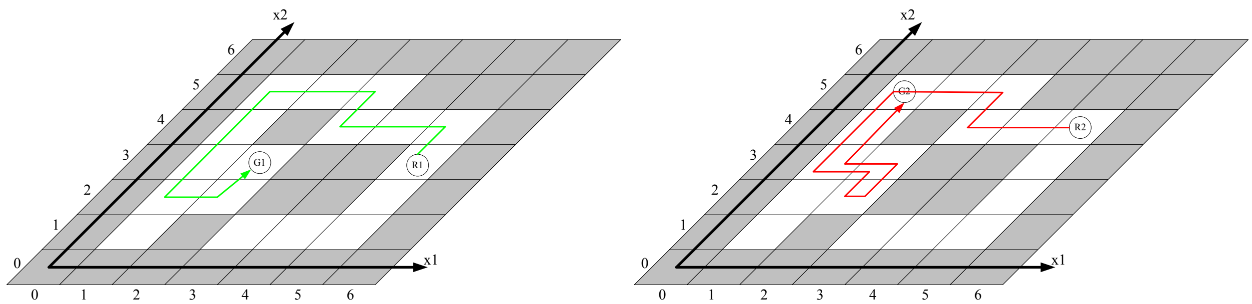
Synchronous Composition			Asynchronous Composition		
Step	Robot1	Robot2	Step	Robot1	Robot2
0	(5, 3)	(5, 4)	0	(5, 3)	(5, 4)
1	(5, 3)	(4, 4)	1	*	(4, 4)
2	(5, 4)	(3, 4)	2	*	(3, 4)
3	(4, 4)	(3, 5)	3	*	(3, 5)
4	(3, 4)	(2, 5)	4	*	(2, 5)
5	(3, 5)	(1, 5)	5	(5, 4)	*
6	(2, 5)	(1, 4)	6	(4, 4)	*
7	(1, 5)	(1, 3)	7	(3, 4)	*
8	(1, 4)	(2, 3)	8	(3, 5)	*
9	(1, 3)	(2, 2)	9	*	(1, 5)
10	(1, 2)	(2, 3)	10	(2, 5)	*
11	(2, 2)	(1, 3)	11	*	(1, 4)
12	(2, 2)	(1, 4)	12	*	(1, 3)
13	(2, 3)	(1, 5)	13	*	(1, 2)
14			14	(1, 5)	*
15			15	(1, 4)	*
16			16	(1, 3)	*
17			17	(2, 3)	*
18			18	*	(1, 3)
19			19	*	(1, 4)
20			20	*	(1, 5)

* means a robot stays in the same location.

In order to obtain paths of the network of two robots with both synchronous composition and asynchronous composition, we first assume it is not the case that both R_1 and R_2 reach their respective goal positions G_1 and G_2 by property specification with a CTL formula:

```
AG ! (contr1.rob.z_1 = 2  & contr1.rob.z_2 = 3
& contr2.rob.z_1 = 1  & contr2.rob.z_2 = 5).
```

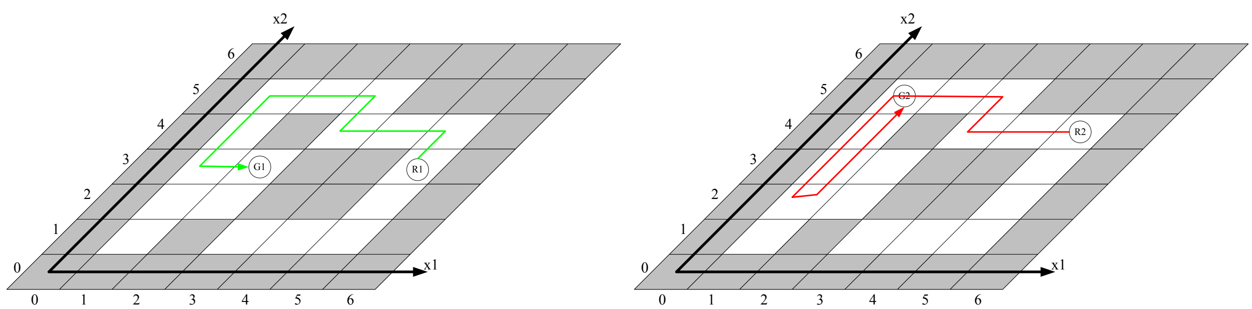
When checking this property, the results are false with counterexamples to illustrate paths of R_1 and R_2 from initial position to the designated goals without collision shown in Table 3. By default, modules in SMV are composed synchronously, it means that modules executes in parallel. However, by using the key word *process*, modules can be composed asynchronously, that is, modules run at different speeds with interleaving manner. For the asynchronous composition in Table 3, the items marked as star * indicate the robots keep in the same location. According to the results in Table 3, the demonstration of paths of both synchronous composition and asynchronous composition are shown in Figure 7 and Figure 8, respectively. By using the synchronous composition, one can observe that less time steps are needed for both robots to accomplish the required specification. While in the other case, by using the asynchronous composition, even though the required specification is accomplished with more time steps, less transitions are taken by the robots. This could be explained by the fact that the robots move in an interleaving manner and hence less conflicts are introduced among robots.



(a) Path of robot R_1 from initial to goal position G_1.

(b) Path of robot R_2 from initial to goal position G_2.

Figure 7: Paths for the network with two robots R_1 and R_2 with synchronous composition.



(a) Path of robot R_1 from initial to goal position G_1.

(b) Path of robot R_2 from initial to goal position G_2.

Figure 8: Paths for the network with two robots R_1 and R_2 with asynchronous composition.

A summary of model checking the reachability and safety properties of both synchronous composition and asynchronous composition are given in Table 4. We have tested the cases from 2 robots in 6×6 mapsize up to 3 robots in 15×15 mapsize. The environment for testing is Linux with 2.2G Hz CPU and 32GB Memory, and the model checker is Cadence SMV. As shown in Table 4 all the properties are satisfied.

Table 4: Results from model checking reachability properties 1 and 2 and safety properties 3 and 4 with synchronous and asynchronous composition.

		BDD Nodes		Time[s]	
		Asynch	Synch	Asych	Synch
Reachbility	2r6m	117642	179233	1.41	1.08
	3r6m	902945	652421	15.94	10.09
	2r10m	1437255	1109173	61.03	25.88
	2r15m	4633615	4508088	252.78	223.9
	3r10m	7875544	7327447	202.65	130.88
	3r15m	59566953	47008200	2855.34	1461.24
Reachability(Step)	2r6m	1063504	281992	85.8	3.72
	3r6m	28031275	645226	476.65	16.11
	2r10m	1487836	1264218	69.92	40.98
	2r15m	6198876	4508229	328.54	216.1
	3r10m	45967858	36187637	2294.17	1707.56
	3r15m	165029664	111473393	6965.57	6822.95
Collision Avoidance	2r6m	10635004	278866	81.98	3.38
	3r6m	25642975	610225	271.43	9.24
	2r10m	1449886	1170607	79.5	39.45
	2r15m	6390886	4508229	244.01	206.91
	3r10m	8967858	6687570	494.61	303.08
	3r15m	165039363	32704530	4444.24	3124.23
Bounded Movement	2r6m	177642	179233	1.08	0.95
	3r6m	584109	636246	5.27	8.1
	2r10m	1469870	1109173	66.96	21.69
	2r15m	4633615	4508088	197.84	205.32
	3r10m	5008749	4129897	99.03	92.9
	3r15m	18402783	21856316	1059.8	934.74

Here 2r6m means 2 robots moving in 6×6 mapsize, other cases could be comprehended in same manner.

5 Experimental Results

In this section we focus on demonstrating the system implementation of the proposed framework by using a quadrotor-based aerial robots testbed. The system is setup as follows. Two quadrotor aerial robots are designed and implemented in the testbed. Each robot is equipped with sensors that are detected by a motion tracker. The motion tracker continuously sends the 3-D position of all the sensors to the analysis and visualization programs running on a visualization workstation. These programs calculate the position and rotation of the vehicle with respect to a pre-configured reference frame and pass this information to the controllers running on the real-time computing nodes. These real-time computing nodes get this data, derive the appropriate control data based on the desired position and orientation, and send the control signal to the robots via radio transmitters.

In order to implement the framework, the key idea is to determine a partition such that the constraints are respected and furthermore the properties specified in Assumption 1 can be satisfied. For determining the partition, the closed-loop dynamics should be taken into consideration. Here, the dynamical model and the controller developed in [28] for the quadrotor-based aerial robot are presented. The dynamics of the robot is modeled as an outer-inner model and the controller is constructed by cascading an outer controller with an inner controller.

In the model, the outer system is continuous-time and can be expressed by using the motion equations for a rigid body, and the discrete-time inner system dynamics can be obtained by performing system identification over discrete data sets. The dynamics of the robot can be described as:

$$\Sigma : \begin{cases} y(t) &= h(x_O(t)) \\ \dot{x}_O(t) &= f_O(x_O(t), x_I(t)) \\ x_I(t+\Delta) &= f_I(x_I(t), u(t)) \end{cases}$$

where $y \in \mathbb{R}^3$ is the output vector, $x_O \in \mathbb{R}^9$ is the outer system state, $x_I \in \mathbb{R}^{n_I}$ is the inner state vector, the inner input vector $u \in \mathbb{R}^4$ and $\Delta > 0$ is the sampling time with $h : \mathbb{R}^9 \to \mathbb{R}^3$, $f_O : \mathbb{R}^9 \times \mathbb{R}^{n_I} \to \mathbb{R}^9$ and $f_I : \mathbb{R}^{n_I} \times \mathbb{R}^4 \to \mathbb{R}^{n_I}$. The dimension of the inner state, n_I, is determined in the system identification process. The output vector and the outer state vector can be specified as $y = p$ and $x_O = [p^T \; v^T \; \Theta^T]^T$, respectively, in which $p \in \mathbb{R}^3$ is the position vector, $v \in \mathbb{R}^3$ is the velocity vector, $\Theta = [\phi \; \theta \; \psi]^T \in \mathbb{S}^3$ are the ZYX Euler angles.

(a) t $\in$ [70s, 72.5s]

(b) t $\in$ [72s, 75s]

(c) t $\in$ [75s, 77.5s]

(d) t $\in$ [77.5s, 80s]

Figure 9: Motion of two robots R1 and R2 during $[70s, \; 80s]$.

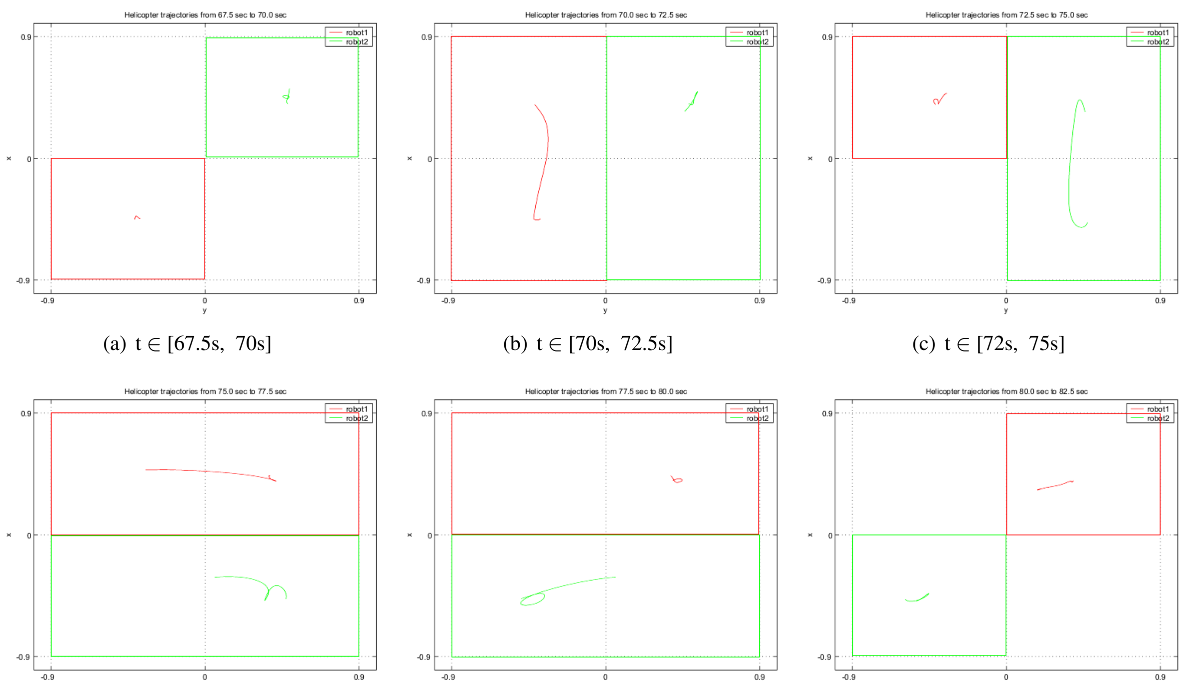
The controller is constructed by cascading a backstepping-based nonlinear outer controller and a robust linear inner controller together. They are designed to guarantee bounded output tracking and bounded state performance for bounded desired output trajectory in the presence of anticipated disturbance. In the following, we assume that all system state variables are properly initialized in order to satisfy the bounded tracking condition. Since backstepping is used in the outer controller design, the position dynamics can be written as:

$$\begin{aligned} \dot{p} &= v \\ &= \gamma_v(p_d) + (v - \gamma_v(p_d)). \end{aligned}$$

where $\gamma_v(p_d)$ is the desired virtual input of the position dynamics so that the position p can converge to a desired position p_d only if the velocity v converges to $\gamma_v(p_d)$. However, the velocity tracking performance can be only achieved with $\|v - \gamma_v(p_d)\|_2 \leqslant \delta_v$ for some $\delta_v > 0$ due to the presence of anticipated disturbance.

By considering the velocity tracking performance, we can characterize the position tracking performance. Consider that $\gamma_v(p_d)$ is designed for regulating the position at the desired position p_d by having $\gamma_v(p_d) = -K_a(p - p_d)$ with a diagonal matrix K_a. Define $z_p = p - p_d$. Consider the Lyapunov function $V_p = z_p^T P_p z_p$ with a positive definite matrix P_p and the Lyapunov equation $(-K_a)P_p + P_p(-K_a)^T = -I$ where I is an identity matrix and $P_p = \frac{1}{2}K_a^{-1}$. Hence, $\dot{V}_p = -z_p^T z_p + 2z_p^T P_p \delta_v \leqslant -\|z_p\|_2^2 + 2\overline{\sigma}(P_p)\delta_v\|z_p\|_2$ where $\overline{\sigma}(P_p)$ is the largest eigenvalue of P_p. Thus, $-\dot{V}_p$ is positive definite whenever $\|z_p\|_2 > 2\overline{\sigma}(P_p)\delta_v$. Define $W(p_d) = \{p \in \mathbb{R}^3 | \ \|p - p_d\|_2 \leqslant 2\overline{\sigma}(P_p)\delta_v\}$. Therefore, if $p(0) \in W(p_d)$, then $\forall t \geqslant 0\ p(t) \in W(p_d)$; otherwise, $\exists t \geqslant 0\ p(t) \in W(p_d)$ due to the fact that $-\dot{V}$ is positive definite outside $W(p_d)$. In other words, any state starting from $W(p_d)$ will stay within $W(p_d)$ and all the points outside $W(p_d)$ can reach $W(p_d)$ eventually. Therefore, the set $W(p_d)$ can be said to be both attractive and positive invariant.

For each cell Y_j, the center of the cell is defined as the desired position labelled by p_{dj}. If Y_j is large enough to contain $W(p_{dj})$, then Y_j is a positive invariant set. Furthermore, for the adjacent cell of Y_j, say Y_k, if p_{dk} is chosen to be the center of Y_k, since all the cells in a partition are the same, the set $Y_j \cup Y_k$ is also a positive invariant set and also all the point in Y_j move eventually to the set $W(p_{dk})$ inside Y_k. Hence, both properties specified in Assumption 1 can be satisfied by properly choosing the set point corresponding to the discrete input symbol.



(a) t ∈ [67.5s, 70s] (b) t ∈ [70s, 72.5s] (c) t ∈ [72s, 75s]

(d) t ∈ [75s, 77.5s] (e) t ∈ [77.5s, 80s] (f) t ∈ [80s, 82.5s]

Figure 10: Progression of flight trajectory and acceptable cell occupation.

Given the set $W_1(\cdot)$, the continuous state space Y is partitioned as $\pi = \{Y_j\}_{j=1}^M$ such that every cell respects the ball and the physical dimensions of the aerial robots. Given the cell partition, the time range $\tau = [0, \tau_2]$ gives bound on the time needed to go from one cell to the center of a neighboring cell. In the implementation, the dimensions of each helicopter are 75cm$\times$$75$cm but due to the limited space each cell is restricted to be a 100cm$\times$$100$cm square and there are only 4 cells considered. The radius of $W_1(\cdot)$, δ_r, is determined to be 12.5cm experimentally. This means that given a fixed reference point, the center of the helicopter will stay within a ball of radius 12.5cm centered at that reference point. The time range $\tau = [1.5s,\ 4.5s]$ is determined experimentally and a time range of $\tau' = [1s,\ 5s]$ is used for constructing the finite automaton model of a given robot as a more conservative approximation.

Consider a mission involving two quadrotor aerial robots, robot ($R1$) and robot ($R2$). The objective is to coordinate the robot motion so that they eventually reach their target (discrete) locations while satisfying imposed dynamical and static constraints. The following reachability and safety properties of the multi-robot system are verified on the network of finite automata using SMV, given the initial locations $P_1 = (1, 1)$ and $P_1 = (2, 2)$ for $R1$ and $R2$, respectively:

Property 1 (Reachability)

Does a location trajectory exist where the robots R_1 and R_2 eventually reach their goal positions G_1 and G_2?

```
EF (contr1.rob.z_1=2 & contr1.rob.z_2=2
& contr2.rob.z_1=1 & contr2.rob.z_2=1)
```

Property 3 (Collision avoidance)

Does all location trajectories the robots R_1 and R_2 take, ensure they never collide after they start to move, i.e. for step > 0?

```
AG ! ((contr1.rob.z_1=contr2.rob.z_1
& contr1.rob.z_2=contr2.rob.z_2)
& contr1.rob.step>0 & contr2.rob.step>0)
```

In this implementation, the robots are designed to be synchronized to their own robot controllers and hence the sequence generated by the model checker. The flight results in pictures are shown in Figure 9 and the position trajectories of the robots are shown in Figure 10. Two robots $R1$ and $R2$ are initially located diagonally. The reachability property is checked such that the robot $R1$ will move from one corner (discrete) location to another corner (discrete) location and robot $R2$ starts and finishes adjacent to robot $R1$. We can see that the robots reach the target locations, remain in the boundary and avoid colliding.

In order to understand the significance of the verification and the validity of the bisimulation, we compare the verification result and the experimental result. In SMV, the wider time range $\tau' = [1s,\ 5s]$ is used. Since the range τ' covers the range τ, the bisimulation still holds and furthermore SMV provides a more conservative result. By observing the trace file of the verification results by using the range $\tau' = [1s,\ 5s]$ in SMV, we observe that the first transition (for both vehicles) should take place in the range $[70s,\ 75s]$, the second transition during $[72s, 80s]$, and the third transition between $[73s,\ 85s]$. In the experiment, transition times (to get to an adjacent cell) for $R1$ are 72.25s, 76.15s, and 80s, and for $R2$ are 74.2s, 77.9s, and 82.6s. This can be seen in Figure 10. The time intervals provided by the model checker for the finite automata cover the transition times taken for the robots in implementation. These results demonstrate the effectiveness of the abstraction technique.

6 Conclusion

A framework for the coordination of a network of robots with respect to formal requirement specifications in temporal logic has been proposed. In this framework, a regular tessellation is used for partitioning the space of interest into a union of disjoint regular and equal cells with finite facets. Each cell can only be occupied by a robot or an obstacle. Each robot is assumed to be equipped with a finite collection of continuous-time nonlinear closed-loop dynamics to be operated in. Robots are modeled as hybrid automata capturing finite modes of operations for either staying within the current cell or reaching an adjacent cell through the corresponding facet. By taking the motion capabilities into account, a bisimilar discrete abstraction of the hybrid automaton can be constructed. Having the two systems bisimilar, all properties that are expressible in temporal logics such as LTL, CTL, and μ-calculus can be preserved. Therefore, on one hand, motion planning of robots can be performed at a discrete level by considering the parallel composition of discrete abstractions of the robots and a requirement specification expressed in some suitable temporal logic. On the other hand, the bisimilarity ensures that the discrete planning solutions are executable by the robots with continuous dynamics. A 2-dimensional case study is used to demonstrate how the framework can be implemented and solve the coordinated motion planning problem of robots in a partitioned environment. Finite automata are used as the abstraction of the robot model and the requirement specification is expressed in CTL with Cadence SMV as the model checker for generating and verifying coordinated motion planning solutions. The quadrotor-based aerial robots testbed has been used to demonstrate the implementation of the proposed framework with two aerial robots. Experimental results have shown the effectiveness of the proposed framework for the coordination of a network of robots by using temporal logic to formulate the mission specifications for a network of robots.

The results presented here assumes an infra-structure of the robots with feedback controllers that constrain the motion capabilities of the individual robots. Although in the system implementation the controllers are implemented for regulating the robots at some predefined desired positions for demonstration purpose, the framework does allow less coupling in the selection of the type of partition and the motion capabilities of the robots. Hence, the robots can be designed with hierarchical dynamical behaviors with various levels of trajectory granularity so long as the assumption made on the motion capabilities within and among cells can be satisfied. For example, in a search-and-rescue mission the assumption can be interpreted in such a way that the robots can be asked to "stay" within the current cells with some continuous trajectories for performing some rescuing tasks or to "move" from the current cells to the adjacent cells with some other continuous trajectories for covering some specific search areas. On the other hand, as shown in the examples, the robots are designed to be synchronized to their own robot controllers and hence the sequence generated by the model checker. By introducing additional finite automata into the network for modeling communication protocol between robots, various forms of centralization and synchronization can be incorporated. With these flexibilities, this framework can be extended to incorporate heterogeneous robots even with asymmetric motion capabilities to accomplish a given mission collectively. However, as in many model checking based approaches, the computational complexity of model checking the system increases as the number of robots in the network and the size of the occupancy table increases. The complexity of checking a CTL formula in Cadence SMV is linear in the state space of the system and the length of the formula. In order to make the proposed framework applicable for large networks of robots an extensive search of the state space should be avoided or substantially reduced.

Acknowledgements

This work is supported by the Shenzhen Science Fund for Distinguished Young Scholars, Shenzhen Science and Technology Foundation, Grant No. JC201005270259A; the Shenzhen Institutes of Advanced Technology, the Chinese Academy of Sciences, Grant No. O9G003; and the Faculty Research Funds from Aalborg University, Denmark, and Vanderbilt University, USA.

References

1 T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics & Automation*, 14(6):926–939, December 1998.

2 R. Fierro, A. K. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 157–162. IEEE, 2001.

3 R. A. Zachery, S. S. Sastry, and V. Kumar. *Proceedings of the IEEE, Special Issue on Swarming in Natural and Engineered Systems*, 99(9), 2011.

4 E. Klavins and D. E. Koditschek. A formalism for the composition of concurrent robot behaviors. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 3395–3402. IEEE, 2000.

5 R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the MARTHA project. *IEEE Robotics & Automation Magazine*, 5(1):36–47, March 1998.

6 B. P. Gerkey and M. J. Mataric. Sold: Auction methods for multirobot coordination. *IEEE Transactions on Robotics & Automation*, 18(5):758–768, October 2002.

7 M. Egerstedt and X. Hu. A hybrid control approach to action coordination for mobile robots. *Automatica*, 38(1):125–130, January 2002.

8 M. Egerstedt and C. F. Martin. Conflict resolution for autonomous vehicles: A case study in hierarchical control design. *International Journal of Hybrid Systems*, 2(3):221–234, September 2002.

9 R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *Proceedings of Formal Methods, World Congress on Formal Methods in the Development of Computing Systems 1999, Lecture Notes in Computer Science 1708*, pages 212–232. Springer, 1999.

10 T. J. Koo and S. Sastry. Bisimulation based hierarchical system architecture for single-agent multi-modal systems. In *Proceedings of 2002 International Workshop on Hybrid Systems: Computation and Control, Lecture Notes in Computer Science 2289*, pages 281–293. Springer, 2002.

11 M. Antoniotti and B. Mishra. Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers. In *Proceedings of the 1995 IEEE Conference on Robotics & Automation*, pages 1441–1446. IEEE, 1995.

12 M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi. Multi-robot planning: A timed automata approach. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, pages 4417–4422. IEEE, 2004.

13 G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.

14 C. Belta, V. Isler, and G. J. Pappas. Discrete abstractions for robot planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–875, 2005.

15 G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning : A temporal logic approach. In *Proceedings of the 2005 IEEE Conference on Decision and Control and the European Control Conference*, pages 4885–4890. IEEE, 2005.

16 M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from ltl specifications. In *Proceedings of the 2006 International Workshop on Hybrid Systems: Computation and Control, Lecture Notes in Computer Science 3927*, pages 333–347. Springer, 2006.

17 M. Kloetzer and C. Belta. LTL planning for groups of robots. In *Proceedings of the 2006 IEEE International Conference on Networking, Sensing, and Control*, pages 578–583. IEEE, 2006.

18 M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.

19 Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. A formal approach to deployment of robotic teams in an urban-like environment. *10th International Symposium on Distributed Autonomous Robotics Systems (DARS), Lausanne, Switzerland*, 2010.

20 C. Belta S. L. Smith, J. Tumova and D. Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.

21 G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

22 E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

23 K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, Massachusetts, 1993.

24 K. L. McMillan. Getting started with SMV: User's manual. *Cadence Berkeley Laboratories, USA*, 1998.

25 A. Chutinan and B. H. Krogh. Verification of infinite-state dynamic systems using approximate quotient transition systems. *IEEE Transactions on Automatic Control*, 46(9):1401–1410, 2001.

26 A. Podelski and S. Wagner. Model checking of hybrid systems: From reachability towards stability. In *Proceedings of the 2006 International Workshop on Hybrid Systems: Computation and Control, Lecture Notes in Computer Science 3927*, pages 507–521. Springer, 2006.

27 G. J. Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, December 2003.

28 T. J. Koo, C. A. Clifton, and G. Hemingway. Casacded control design for a quadrotor aerial robot. In *Proceedings of the 2006 Asian Control Conference*. IEEE, 2006.