Synthesis of Nonlinear Continuous Controllers for Verifiably-Correct High-Level, Reactive Behaviors

Jonathan A. DeCastro and Hadas Kress-Gazit *

November 18, 2013

Abstract

Planning robotic missions in environments shared by humans involves designing controllers which are *reactive* to the environment yet able to fulfill a complex high-level task. This paper introduces a new method for designing low-level controllers for nonlinear robotic platforms based on a discrete-state high-level controller which encodes the behaviors of a reactive task specification. We build our method upon a new type of trajectory constraint which we introduce in this paper – reactive composition – to provide the guarantee that any high-level reactive behavior may be fulfilled at any moment during the continuous execution. We generate pre-computed motion controllers in a piecewise manner by adopting a sample-based synthesis method which associates a certificate of invariance with each controller in the sample set. As a demonstration of our approach, we simulate different robotic platforms executing complex tasks in a variety of environments.

1 Introduction

As robots become more sophisticated, we see increasing potential for them to perform complex tasks. Sensor-rich platforms such as self-driving cars, robotic manipulators, humanoids, and unmanned air vehicles enable capabilities ranging from human-assistance to search-and-rescue to exploration. To operate effectively in the real world, a robot must be able to perform tasks in a way that avoids causing harm to itself or the people it encounters, by appropriately *reacting* to the environment. It is therefore imperative that the controllers we provide to these robots produce behaviors which are guaranteed to satisfy all task instructions provided to the robot given our knowledge of the environment. Building upon existing concepts from the controller verification and motion planning communities, we present an automated methodology which computes (if possible) a set of verified controllers which are able to fulfill the reactive behaviors of a high-level task.

In the controller synthesis literature, recent activity has focused on solving the "reach-avoid" problem for nonlinear systems. Methods based on a game-theoretic representation of the problem have been solved [DGH11], as have game-theoretic solutions involving a sequence of goals

*This work was supported by the NSF Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering [grant number CCF-1138996]. The authors are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA, {jad455,hadaskg}@cornell.edu

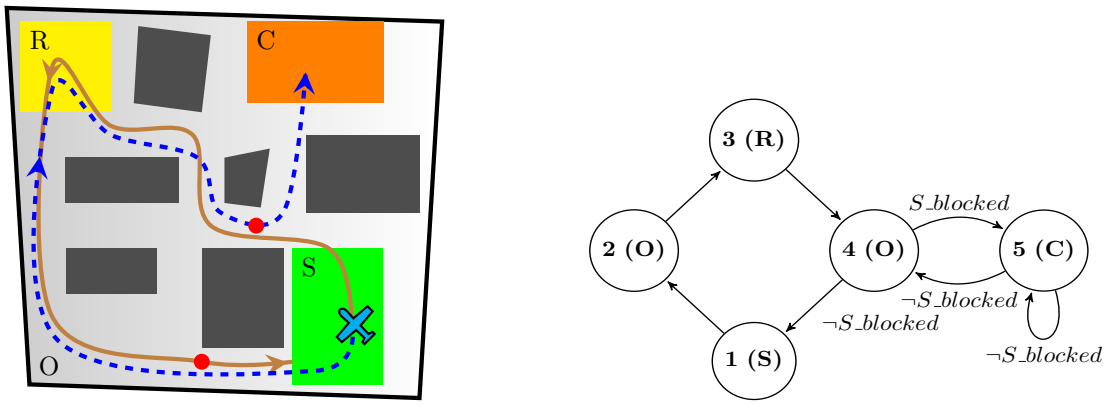
[THDT12]. Rather than working directly with the nonlinear system, other researchers compute symbolic abstractions of the underlying system model [ZPMT12] for which efficient algorithms exist for the exact computation of reachable sets. The particular formulation in [ZPMT12] enables the synthesis of hybrid controllers for goal satisfaction with static obstacles [MDT10].

Concepts from nonlinear controller verification have been adopted in the motion planning domain. For example, libraries of verified motion plans have been generated for hybrid systems [JFA07] as well as for nonlinear polynomial systems [TMTR10]. Both use a sampling-based strategy and employ solution methods which are efficient enough to allow on-line re-planning based on runtime sensor information [MTT12]. The latter method ([TMTR10, MTT12]) builds upon the controller sequencing work of [BRK99] by combining sampling-based motion planning with local feedback controllers to define a robust neighborhood (funnels) computed about one of the sample trajectories. Controller sequencing has been applied to specific robotic applications; for example, in [CCR06], local verified motion controllers have been devised for robots with nonholonomic kinematic models. While each of these methods provide the framework for motion planning in the context of safety and reachability, they are not immediately equipped to handle automatic synthesis of controllers for *reactive* tasks with possibly infinite duration.

Ongoing work in the robotics community has been devoted to developing provably correct synthesis algorithms for complex robot task specifications. A variety of techniques exist for synthesizing controllers for non-reactive tasks [KB06, LK04, Fra01, KF09, BKV10], as well as those for reactive tasks [WTM10, KGFP09]. The typical workflow is a three-step process: abstract the system model and the robot's workspace; perform synthesis based on the abstraction and task specification; and design low-level controllers fulfilling each of the behaviors in the high-level controller. Low-level controller design can be performed using potential functions [CRC03], vector fields [BH04], or rapidly-exploring random trees (RRTs) [LaV06], to name a few. While such strategies often suffice for fully-actuated robots, correctness guarantees usually do not extend to robots with more complicated dynamics. Some researchers have extended provably correct controller synthesis to nonlinear systems [LOTM13, WTM13]. These methods, however, rely on finding suitable discrete abstractions for the nonlinear system model, if any such abstractions exist. Others have introduced a multi-layered synthesis strategy in which the high-level controller is designed off-line, while the motion planning layer processes the high-level requests by accounting for nonlinear dynamics and any encountered workspace obstacles [BKV10, MLK13]. However, the tasks that are considered in those works are non-reactive and the task fulfillment guarantees are obtained at runtime, rather than at the time of synthesis. In the work of [FLP06], an automated framework is introduced which translates the behaviors of a high-level controller into low-level continuous controller specifications. The possibility exists to use such specifications in the design of nonlinear motion planners.

In this paper, we seek to answer the following two questions. Given a high-level reactive mission plan, a robot model, and workspace, can we design a set of low-level controllers which guarantee satisfaction of the task in a dynamic environment? If so, what is the set of allowable robot configurations for which these guarantees hold?

The main contributions of this work are as follows. First, building from the notion of sequential composition introduced in [BRK99], we define a new composition property which we call *reactive composition* to assure that the continuous trajectories preserve every possible behavior in the high-level controller in the reactive setting. Given a high-level controller encoded as a finite-state automaton, we use reactive composition to construct continuous specifications which the low-level controller must satisfy. This part of the work is inspired by the general approach in [FLP06], how-



(a) (b)

Figure 1: Workspace and high-level controller automaton for Example 1. In (a), a 2-D environment is shown, along with a set of trajectories: one which does not satisfy the automaton (solid) and one which does (dashed). The •'s indicate when $S\_blocked$ turns from `False` to `True`. In (b), the number designates the state; the name in parenthesis denotes the region associated with that state. For each transition, the truth values for the $S\_blocked$ sensor are given; unlabeled transitions imply that $S\_blocked$ can take on any value.

ever, in our work we extend controller specifications to encompass nonlinear robot models satisfying high-level reactive tasks. Another contribution of our work is an algorithm which synthesizes controllers that adhere to reactive composition properties for the given task. The algorithm takes as an input a high-level controller with reactive behaviors, and tries to synthesize a library of low-level (atomic) controllers which are guaranteed to satisfy these behaviors. The algorithm is, to the authors' knowledge, the first attempt at designing verified controllers for reactive tasks. Finally, we contribute a sample-based method for computing controllers which guarantee reactive composition and avoid behaviors prohibited by the high-level controller. We adapt the invariant funnels method in [TMTR10, MT12], which takes advantage of powerful numerical optimization techniques to produce a set of verifiably-safe atomic controllers. We introduce a new set of numerical constraints in order to enforce reactive composition while preventing any robot behaviors prohibited by the high-level controller.

In the authors' preliminary version of this work, [DKG13], we introduced the concept of reactive composition, developed a constructive algorithm to ensure this property, and presented preliminary simulation results. This paper serves to build upon that work by detailing the precise technical conditions upon which reactively composable reachable sets should be constructed. Also, we explain the numerical method we use to compute atomic controllers, in particular our approach for incorporating the necessary invariance conditions for the reactive setting. We furthermore provide more illustrative examples in a more complex environment setting.

We motivate the work through an example.

Example 1. *Consider an autonomous delivery robot, modeled as fixed-wing aircraft, operating in a dense urban environment in Fig. 1a. The robot must continually deliver items between the store (S) and the restaurant (R) by visiting them infinitely often. If the plane is in O and en route to S but it senses that store is blocked, then it must re-route to the charging station C (without entering S). The discrete automaton shown in Fig. 1b is guaranteed to fulfill the specification.*

Our goal is to construct feedback controllers which guarantee the sequence of motions by the controller automaton like the one in Fig. 1b using, in this case, a fixed wing airplane model and the workspace in Fig. 1a. This task is a reactive one because the behavior changes depending on

whether or not the store is "blocked." If the robot starts in S and $S\_blocked$ stays `False` forever, the robot should follow the sequence of regions $SOROS\ldots$ indefinitely; if $S\_blocked$ becomes `True`, then it may follow the sequence $SOROCC\ldots$, i.e. the robot must go to C and stay there once it has visited R. In the continuous domain, there may be instances where the robot may fail this task. To see this, consider the solid-line trajectory pictured in Fig. 1a, where the plane moves counterclockwise starting with $S\_blocked =$ `False`. If the robot senses $S\_blocked =$ `True` at the •, it may be unable to avoid hitting S, and the task would fail. On the other hand, a clockwise trajectory (the dotted line in Fig. 1a), is likely to succeed in this workspace. In general, it is possible that $S\_blocked$ may toggle between `True` and `False` at any point in the robot's continuous trajectory, and so the robot must always be in a configuration where it can make any legal transition. If, for a different workspace, such controllers are found not to exist for the given platform, then the specified task may may not be suitable for that robot.

In the next section, we introduce the reader to relevant background concepts for our atomic controller design approach. In Section 3, we introduce our approach to solving the problem and outline an algorithm for automatic controller synthesis. The trajectory-based approach used for computing verified controllers is presented in Section 4, and the execution of the controllers in Section 5. We then present simulations tasks performed by different robots in Section 6. The paper concludes in Section 7 with a summary and discussion.

2 Preliminaries

2.1 Robot and Environment Abstractions

Similar to the implementations in [KGFP09], we define an abstraction as a partitioning of the robot system and the environment in which it is operating. In particular, we partition the continuous map $\mathcal{M} \subset \mathbb{R}^2$ into M disjoint proposition-preserving map regions $\mathcal{M}_i$, each associated with a label r_i. We also assume that the robot can travel between any adjacent regions. In this paper, map regions are assumed to be 2-D polygons, possibly with holes.

In addition to the workspace, sensor and action values are discretized such that they may be effectively replaced by a set of Boolean propositions. Here sensors refer to events external to the robot (detection of a person, non-detection of a person), whereas actions refer to discrete robot functions (stand up, sit down). We define $\mathcal{X}$ as a set of *environment propositions*, collecting sensor propositions, and $\mathcal{Y}$ as a set of *system propositions*, collecting robot action propositions and region propositions. Define the set $\mathcal{R} = \cup_i r_i \subseteq \mathcal{Y}$ as the subset of $\mathcal{Y}$ corresponding to regions.

2.2 Controller Automaton

High-level controllers synthesized from task specifications and abstractions [WTM10, KGFP09] are assumed to be given in this work. The synthesized controller takes the form of a finite-state automaton A, defined as a tuple $A = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta)$, where:

- $\mathcal{X}$ and $\mathcal{Y}$ are proposition sets as defined in Section 2.1.
- $Q \subset \mathbb{N}$ is a set of discrete states.
- $Q_0 \subseteq Q$ is a set of initial states.

- $\delta : Q \times 2^{\mathcal{X}} \to Q$ is a deterministic transition relation, mapping states and subset of environment propositions to successor states.

We introduce the following additional definitions. Define $\gamma_{\mathcal{R}} : Q \to \mathcal{R}$ as a state labeling function assigning to each state the region label for that state, r_i. Define the operator $R : Q \to \mathbb{R}^n$ as a mapping that associates with each $q \in Q$ the subset $X_q = R(q)$ of the free configuration space X, where X_q corresponds to an n-D polytope labeled with $\gamma_{\mathcal{R}}(q)$. For a nonholonomic planar mobile robot with $X \subset \mathbb{R}^3$, each X_i is a 3-D polytope. We also define $\Delta = \{(q, q') \in Q^2 | \exists z \in 2^{\mathcal{X}} \, . \, \delta(q, z) = q'\}$. In this paper, we assume that actions other than locomotion are instantaneous.

For some sequence $w(0)w(1)w(2)\ldots, w(i) \in \mathcal{X}, i = 0, 1, 2, \ldots$, denote a *run* of A as $q(0)q(1)q(2)\ldots$, with $q(i+1) = \delta(q(i), w(i))$. We call the sequence $w(0)\gamma_{\mathcal{R}}(q(0))w(1)\gamma_{\mathcal{R}}(q(1))\ldots$ an *execution trace of* A.

2.3 Continuous Dynamics and Operations

In this paper, we consider systems of the form

$$\dot{x} = f(x, u, d), \quad x(0) \in S, \quad d \in D, \quad u \in U \tag{1}$$

where we are given a state vector $x \in X \subseteq \mathbb{R}^n$, a control vector $u \in U \subset \mathbb{R}^m$, a vector of unknown disturbances $d \in D \subseteq \mathbb{R}^{n_d}$, and a set of initial states S. f is considered to be a smooth, continuous vector field with respect to its arguments.

Under the action of a full-state feedback control law $u(t) = \kappa(x(t), t)$, let $\dot{x} = \hat{f}(x, d)$ represent the closed-loop system for (1) under $\kappa(x(t), t)$. Given a concrete start state $x(0) \in S$ and a time horizon $T > 0$, denote $\xi_T : [0, T] \to X$ as a continuous finite-time trajectory of states under $\hat{f}$, $\mu_T : [0, T] \to U$ as a trajectory of control inputs, and $\xi_T^d : [0, T] \to D$ as a trajectory of disturbances. Say we are given a sequence of time indices, $\mathbf{t} \in \{0, \ldots, T\}$; then, we can represent the the continuous trajectory as a sequence of states $\mathbf{x} = \{\xi(\tau)\}_{\tau \in \mathbf{t}}$, and a sequence of control inputs $\mathbf{u} = \{\mu(\tau)\}_{\tau \in \mathbf{t}}$.

Given a smooth function $V(x, t)$, and some $\rho(t) > 0$, we define the *ρ-sub-level set* as

$$\ell(\rho(t), t) = \{x | V(x, t) \leq \rho(t)\}$$

Let $X_i, X_j \in X$ be regions in the configuration space where $X_i \cap X_j \neq \varnothing$. Define an *atomic controller* κ_{ij} as a controller that, when applied to $f(x, u, d)$ on $t \in [0, T]$, given $\xi_T(0) \in X_i$ there exists $\xi_T(T) \in X_j$ such that $\xi_T(t) \in X_i \cup X_j, t \in [0, T]$ for all possible $\xi_T^d(t)$. We refer to trajectories driven by the action of an atomic controller as *atomic trajectories*.

Lastly, given $X_i, X_j \in X$, define a *reach tube* $\mathcal{L}_{ij}$ as the set of trajectories in which the controlled system remains for $t \in [0, T_{ij}]$. Formally, suppose that we define some initial set S_{ij} and some atomic control law κ_{ij}, then $\mathcal{L}_{ij} = \{\xi_T(t) | \xi_T(0) \in S_{ij}, \forall \xi_T^d(t), t \in [0, T]\}$.

3 Controller Synthesis Approach

To lay the foundation for our approach, in this section we define the composition strategies we make use of in our problem setting. We then introduce different classes of atomic controllers and the technical conditions that allow us to guarantee composition and avoid prohibited behaviors in the high-level automaton. Lastly, we describe an algorithm which takes as its input a finite-state automaton and returns a library of atomic controllers that guarantee the continuous executions of the automaton at runtime.

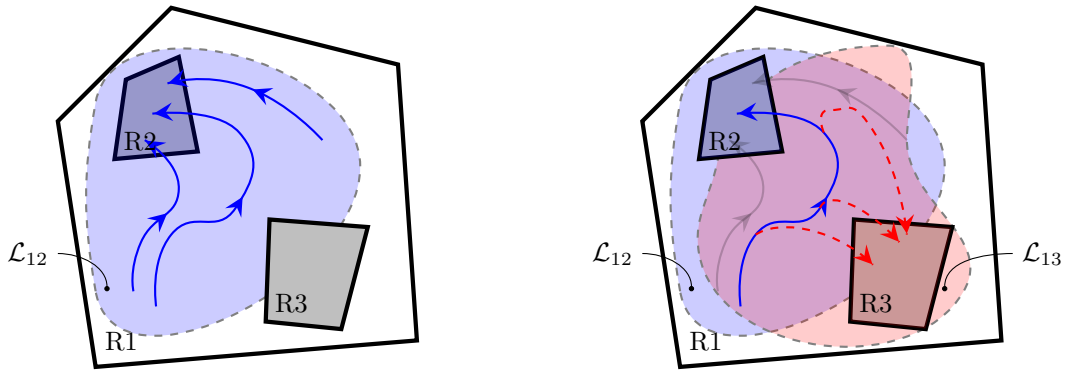


Figure 2: Illustration of reactive composition. (a) shows a reach tube from R1 to R2 and possible trajectories. A reach tube from R1 to R3 shown in (b). Along any given $\mathcal{L}_{12}$ trajectory (blue) in $\mathcal{L}_{12} \cap \mathcal{L}_{13}$, there exist $\mathcal{L}_{13}$ trajectories (red dashed) leading to R3.

3.1 Composition Strategies

Define $I^i_{out} = \{k \in \mathbb{N} | (q_i, q_k) \in \Delta\}$ as the index set of all successor states for state q_i (e.g. for state 4 in Fig. 1b, $I^4_{out} = \{1, 5\}$), and let $I_{out} = \cup_i I^i_{out}$. As defined by [BRK99], for a given set of atomic controllers to be *sequentially composable*, we require goal sets of the reach tubes to be contained within the domain of successor reach tubes. Formally:

Definition 1 (Sequential Composition [BRK99])**.** *Let $\mathcal{L}_{ij}(t)$ denote the slice of $\mathcal{L}_{ij}$ at time t. For $\mathcal{L}_{ij}$ to be sequential composable for each pair in Δ implies that $\mathcal{L}_{ij}(T_{ij}) \subseteq \mathcal{L}_{jk}$ for all $k \in I^j_{out}$.*

For reactive tasks, the underlying conditions for sequential composition must be strengthened. The reason for this can be seen by returning to Example 1 and the two trajectories pictured in Fig. 1a. When the robot is in O with $S\_blocked =$ `False`, and moving towards S, both trajectories may in fact satisfy the sequential composition property if atomic controllers exist for transitions between R and O and O and S. However, if $S\_blocked$ turns `True` during the execution, the robot must already be in a state where it may access C *without* first entering S (a violation of the specification). The configurations where S is reachable from O must also be the set of configurations where C is reachable from O. We introduce a constraint called *reactive composition* to deal with co-reachability of successor regions.

Definition 2 (Reactive Composition)**.** *Let $\bar{X}_i \subset X$ denote the set of states such that, for all $q_i \in Q$, there exists a trajectory from q_i to any q_k, $k \in I^i_{out}$, i.e. $\bar{X}_i = \cap_{k \in I^\alpha_{out}} \mathcal{L}_{ik}$. A given reach tube $\mathcal{L}_{ij}$ is* reactively composable *with respect to A if, for $(q_i, q_j) \in \Delta$, for all state trajectories $\xi_T \in \mathcal{L}_{ij} \Rightarrow \xi_T \in \bar{X}_i \cup \bar{X}_j$.*

Reactive composability is illustrated in the 2-D scenario in Fig. 2, where the solid blue trajectory shown exiting region $R1$ (corresponding to state 1) and entering $R2$ (state 2) is both contained completely within its own reach tube (shaded blue) and the reach tube (shaded red) for trajectories leading to $R3$ (state 3).

3.2 Classes of Atomic Controllers

We now introduce two types of atomic controllers for constructing reactively composable controllers, and define their respective reach tubes. *Transition* controllers κ_{ij} refer to those which

invoke a transition between adjacent regions. *Inward-facing* controllers κ_i^c are used to maximize coverage of the region. Respectively, the reach tubes for κ_{ij} and κ_i^c are $\mathcal{L}_{ij}$ and $\mathcal{L}_i^c$.

3.2.1 Transition Reach Tubes, $\mathcal{L}_{ij}$

Consider a pair $(q_i, q_j) \in \Delta$. When we construct reach tubes which realize this transition, we do not want to simply implement a region transition as if it were isolated from the remaining transitions in the high-level controller. Rather, we want to compute reach tubes that satisfy the current transition in the context of all possible subsequent transitions. To this end, we introduce the following three conditions.

1. Trajectories must be *atomic* with respect to the reactively composable set $\bar{X}_i \cup \bar{X}_j$: we want trajectories to start in the set $R(q_i)$ and progress to $R(q_j)$ while remaining in an invariant Inv_{ij} (defined in Section 3.2.2). For now, we apply definition 2 by choosing $Inv_{ij} = \bar{X}_i \cup \bar{X}_j$.

2. Trajectories must reach a goal set $G_{ij} \subseteq X_j$; formally, $\xi_T(T) \in G_{ij}$. We require G_{ij}, whose precise definition will become clear in Section 3.2.2, in order for the reach tube to be sequentially composable with reach tubes for successor states from q_j. For now, we will assume G_{ij} to be $\bar{X}_j$.

3. To prevent the robot from re-entering a region X_i once it has entered X_j for $X_i \neq X_j$, the trajectories need to be invariant in finite time to the set X_j. That is, for some $\tau \in [0, T]$,

$$\xi_T(\tau) \in \partial X_j \Rightarrow \xi_T(t) \in X_j, \;\; \tau < t \leq T$$

   where we denote ∂X_j to mean the boundary of the set X_j.

To devise a certificate for the third condition, we can draw from region-of-attraction analysis [Kha02], as follows. Let $V : \mathbb{R}^n \times [0, T] \to \mathbb{R}$ be a smooth differentiable function with $V(x_0(t), t) = 0$ and $V(x, t) > 0, x \neq x_0(t)$. We want to ensure that the level set $\partial\ell(\rho(t), t) = \{x | V(x, t) = \rho(t)\}$ satisfies $\dot{V}(x, t) = \frac{d}{dt}V(x, t) < 0$ on $\{x | x \in \partial\ell(\rho(t), t) \cap X_j\}$. Put in terms of the closed-loop system $\hat{f}(\cdot, \cdot)$, the third condition can be reduced to an easier problem of restricting $\rho(t)$, the size of the attraction region, such that it satisfies:

$$\dot{V}(x,t) = \frac{\partial}{\partial x}V(x = \xi_T(t), t)\ \hat{f}(\xi_T(t), d) + \frac{\partial}{\partial t}V(\xi_T(t), t) < 0, \quad \xi_T(t) \in \partial X_j \cup \ell(\rho(t), t) \neq \varnothing, \quad \forall d \in D.$$

Intuitively, this statement requires that the system flow toward the successor region only on that segment of the region boundary which is also in the computed region of attraction.

We can satisfy these conditions by simultaneously imposing constraints on the construction of $\mathcal{L}_{ij}(t)$. Respectively, these constraints are:

$$\mathcal{L}_{ij}(0) \cap S_{ij} \neq \varnothing, \tag{2}$$

$$\mathcal{L}_{ij}(t) \subseteq Inv_{ij}, \quad \forall t \in [0, T], \tag{3}$$

$$\mathcal{L}_{ij}(T) \subseteq G_{ij}, \tag{4}$$

$$\dot{V}(x, t) < 0, \quad \forall x \in \mathcal{L}_{ij}(t) \cap \partial X_j \neq \varnothing, \quad \forall d \in D, \quad \forall t \in [0, T]. \tag{5}$$

where condition (2) assures that $\mathcal{L}_{ij}$ has a nonempty intersection with the start set. Note that when $X_i = X_j$, condition (5) can be dropped, and condition (3), the set inclusion Inv_{ij} is merely $\bar{X}_i$.

3.2.2 Inward-Facing Reach Tubes, $\mathcal{L}_i^c$

Checking whether or not whole trajectories lie within reactively composable sets is more restrictive than checking whether or not trajectories are sequentially composable. If these sets are too small or are spatially disconnected, it may be impossible to find transition reach tubes which satisfy (3). To increase the size of the reactively composable set, we introduce inward-facing reach tubes for each state. Consider a state $q_i \in Q$. Our goal is to generate atomic controllers that admit finite-time trajectories which satisfy the following two conditions:

1. Trajectories must be *invariant* with respect to the region X_i; that is, starting anywhere within X_i while staying within X_i.
2. Trajectories need to reach a goal set $G_i^c \subseteq X_i$; that is, $\xi_T(T) \in G_i^c$.

The above statements require that trajectories are both invariant to the region and are sequentially composable with respect to G_i^c. This leads immediately to the following reach tube conditions:

$$\mathcal{L}_i^c(t) \subseteq X_i, \quad \forall t \in [0, T], \tag{6}$$
$$\mathcal{L}_i^c(T) \subseteq G_i^c. \tag{7}$$

Notice that, by adding in the inward-facing reach tubes, we are able to expand the reactively-composable invariant in (3) (with a slight abuse of terminology) as $Inv_{ij} = (\bar{X}_i \cup \mathcal{L}_i^c) \cup (\bar{X}_j \cup \mathcal{L}_j^c)$, and likewise also expand the goal set in (4) as $G_{ij} = \bar{X}_j \cup \mathcal{L}_j^c$. This will help our iterative approach for synthesizing atomic controllers; as will be shown in the next section, the larger the sets are, the greater the likelihood of finding controllers which satisfy the specification.

3.3 Atomic Controller Synthesis Algorithm

The process for constructing atomic controllers is given in Algorithm 1. The basic procedure is as follows. First, the set of all transitions Δ are extracted from automaton A (`AutomTransitions` in line 1). Next, atomic controllers and their reach tubes are computed for each edge in Δ in lines 7–17. Reach tubes are computed iteratively until either all possible configurations within $R(q_i)$ for each q_i are enclosed (to within a desired tolerance) or until it is determined that coverage is not possible, i.e. it is not possible to compute $\mathcal{L}_{ij}$ for some $(q_i, q_j) \in \Delta$. The algorithm terminates successfully if reach tubes are found for each edge (line 20). If not, then the reach tube computations are revised by repeating lines 6–22 to ensure they are reactively composable in the sense of Definition 2. That is, each reach tube associated with either an incoming or outgoing transition from q_i is checked whether or not it lies within the set of states for which all successor regions of q_i are reachable.

3.3.1 Computing $\mathcal{L}_{ij}$

Fig. 3 illustrates, through an example, the computation steps in Algorithm 1. In the first iteration of lines 7–13, reach tubes are computed for each edge $(q_i, q_j) \in \Delta$. The set $\mathcal{L}_{ij}$ is initialized as the whole configuration space, while the goal set G_{ij} is the region $R(q_j)$ and the invariant Inv_{ij} is the region $R(q_i) \cup R(q_j)$. In Fig. 3a, reach tubes are computed for the two transitions (q_1, q_2) (blue region) and (q_1, q_3) (green region), and the intersection of the two is taken (yellow region). Intuitively, this intersection (see Fig. 3b) defines the set of states from which any region of successor

Algorithm 1:
$(\mathcal{L}, \mathcal{C}) \leftarrow$ `ConstructControllers`(A, R, f, ϵ, N)

Input: Synthesized automaton A with region mappings $R(\cdot)$, closed-loop robot dynamics $f(\cdot)$, coverage metric ϵ, and number of iterations N for coverage

Output: A set of funnels $\mathcal{L}$ and controllers $\mathcal{C}$ guaranteeing the execution of A

1 $(\Delta, I_{out}) \leftarrow$ `AutomTransitions`(A)
2 **for** $(q_i, q_j) \in \Delta$ **do**
3 $\quad \mathcal{L}_{ij} \leftarrow \mathbb{R}^n$
4 **end**
5 $\mathcal{L}_i^c \leftarrow \varnothing, \mathcal{C}_i^c \leftarrow \varnothing \; \forall q_i \in Q$
6 **while** `True` **do**
7 $\quad$ **for** $(q_i, q_j) \in \Delta$ **do**
8 $\quad\quad S \leftarrow \cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i)$
9 $\quad\quad G \leftarrow \cap_{k \in I_{out}^j} \mathcal{L}_{jk} \cap R(q_j) \cup \mathcal{L}_j^c$
10 $\quad\quad (\mathcal{L}_{ij}, \mathcal{C}_{ij}) \leftarrow$ `GetReachTube`$(S, G, S \cup \mathcal{L}_i^c \cup G, f, \epsilon, N)$
11 $\quad\quad$ **if** $\mathcal{L}_{ij} = \varnothing$ **then**
12 $\quad\quad\quad$ **return** $\varnothing$ `// No controller exists`
13 $\quad\quad$ **end**
14 $\quad\quad S \leftarrow R(q_i) \backslash \cap_{k \in I_{out}^i} \mathcal{L}_{ik}$
15 $\quad\quad G \leftarrow \cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i)$
16 $\quad\quad (\mathcal{L}_i^c, \mathcal{C}_i^c) \leftarrow$ `GetReachTube`$(S, G, R(q_i), f, \epsilon, N)$
17 $\quad$ **end**
18 $\quad$ **if** $\forall (q_i, q_j) \in \Delta : \left[(\mathcal{L}_{ij} \cap R(q_i)) \subseteq \left(\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i) \cup \mathcal{L}_i^c\right)\right] \wedge$
$\quad \left[(\mathcal{L}_{ij} \cap R(q_j)) \subseteq \left(\cap_{k \in I_{out}^j} \mathcal{L}_{jk} \cap R(q_j) \cup \mathcal{L}_j^c\right)\right]$ **then**
19 $\quad\quad \mathcal{L} \leftarrow (\cup_{i,j} \mathcal{L}_{ij} \cup_i \mathcal{L}_i^c), \mathcal{C} \leftarrow (\cup_{i,j} \mathcal{C}_{ij} \cup_i \mathcal{C}_i^c)$
20 $\quad\quad$ **return** $\mathcal{L}, \mathcal{C}$
21 $\quad$ **end**
22 **end**

states can be reached (by using either controller $\mathcal{C}_{12}$ or $\mathcal{C}_{13}$). The process repeats for the remaining edges in the automaton. The algorithm immediately returns failure if an edge is encountered where a reach tube cannot be constructed (lines 11–13).

3.3.2 Computing $\mathcal{L}_i^c$

In order to expand the size of reactively composable regions, we create inward reach tubes in lines 14–17 which steer the robot to a configuration from which it can take a transition. The collection of $\mathcal{L}_{ij}$ from the current iteration produce the start sets S_i^c and the sequentially composable goal sets G_i^c for each q_i. The set S_i^c in line 14 is the set $R(q_j)$ minus the intersection of all transition reach tubes from that region (the white portions in Fig. 3b), while the set G_i^c in line 15 is defined by applying Definition 1 (the yellow portions in Fig. 3b).

In Fig. 3c, the red regions enclosed by the dashed lines, $\mathcal{L}_i^c$, denote where controllers were found which drive the system into the yellow region. We seek transition reach tubes which are contained within the union of the red and yellow regions in Fig. 3c.

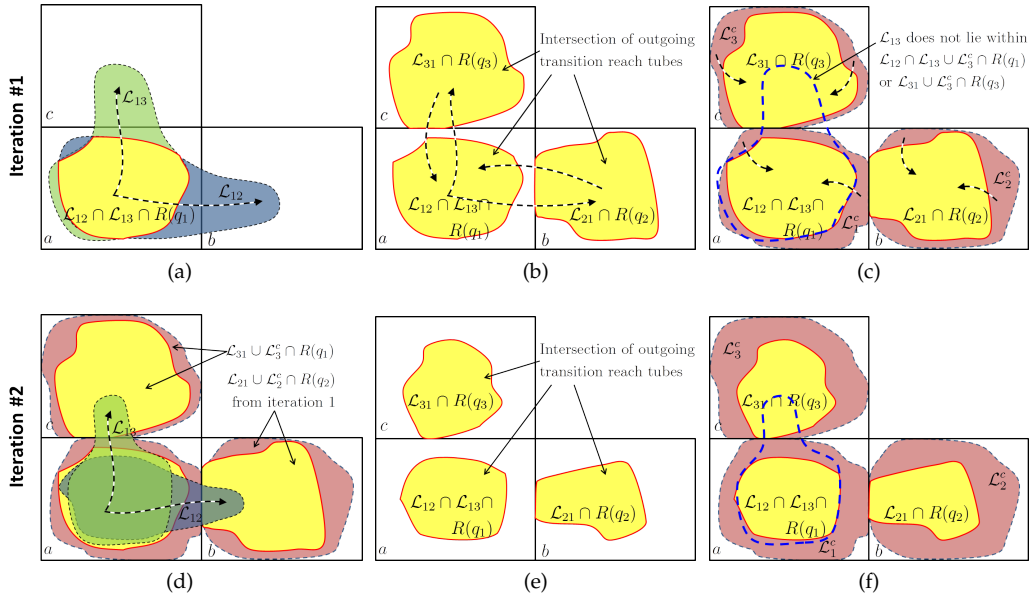


Figure 3: Illustration of the reach tube computation steps, assuming symmetric transitions between each adjoining region. For q_1, a pair of transition reach tubes $\mathcal{L}_{12}$ and $\mathcal{L}_{13}$ are computed in (a), the intersection of which (yellow) defines the new start set for the next iteration (see lines 7–13 in Algorithm 1). The same is done for the remaining states q_2 and q_3, (b). Next, inward reach tubes $\mathcal{L}_i^c$ (red) are generated for each region, (c) (see lines 14–17 in Algorithm 1). This expanded region defines the invariant for the next iteration. The process in lines 7–17 is again repeated for the new start sets and invariants, (d) - (f), and terminates at (f) since all reach tubes lie inside the regions bounded by the dotted borders (e.g. for q_1 this is $(\mathcal{L}_{12} \cap \mathcal{L}_{13} \cup \mathcal{L}_1^c) \cap R(q_1)$).

3.3.3 Further Iterations

After a single iteration, if the sequentially-composable transition reach tubes are not reactively composable, the algorithm continues alternately computing $\mathcal{L}_{ij}$ and $\mathcal{L}_i^c$ until they are reactively composable for all $\mathcal{L}_{ij}$. To test if $\mathcal{L}_{ij}$ is reactively composable, we need to determine if $\mathcal{L}_{ij}$ is contained within a subset of states where outgoing transitions from q_i or q_j are possible, i.e. satisfies $(\mathcal{L}_{ij} \cap R(q_\alpha)) \subseteq \left(\cap_{k \in I_{out}^\alpha} \mathcal{L}_{\alpha k} \cap R(q_\alpha) \cup \mathcal{L}_\alpha^c\right)$ for $\alpha \in \{i, j\}$. As such, the termination criterion in line 18 enforces Definition 2, by requiring that transition reach tubes must either lie within an inward reach tube or the sets where any successor state is reachable. An additional iteration of the algorithm is shown pictorially in the bottom row of Fig. 3.

In any given iteration, the sets S_{ij}, G_{ij}, and Inv_{ij} for the ijth edge are updated by the $\mathcal{L}_{ij}$ and $\mathcal{L}_i^c$ from the previous iteration. Fig. 3d shows the second iteration of lines 7–13, where new transition reach tubes for a are computed ($\mathcal{L}_{12}$ and $\mathcal{L}_{13}$), constrained to stay within the red and yellow regions for q_1, q_2, and q_3. After intersections are taken (yellow regions in Fig. 3e), the reach tubes from the previous iteration are replaced with a new set of inward reach tubes computed in lines 14–17. Fig. 3f illustrates this last step, and is an example of a situation where the algorithm successfully terminates because the reactive composability criterion in line 18 is fulfilled. If the

algorithm terminates successfully, a library of reach tubes $\mathcal{L}$ is returned in lines 19–20 along with a library of controllers $\mathcal{C}$.

4 Computing Atomic Controllers

We now present an implementation for constructing reach tubes in possibly cluttered workspaces. Probabilistic planning approaches, such as rapidly-exploring random trees [LaV06], have gained widespread use in various path planning applications; in such methods, exploration of the configuration space is probabilistically complete. That is, the probability of solving a motion planning problem at a given initial configuration goes to one as the number of samples goes to infinity. Recent techniques such as the Invariant Funnels technique in [TMTR10] uses sampled trajectories to compute verified controllers in a randomized tree structure. We adopt this framework to construct, in a piecewise manner, the reach tubes in Algorithm 1. For each computed sample trajectory, we also compute a region of invariance (funnel) about the sample trajectory. The workflow for constructing each funnel is as follows [TMTR10,MT12]: (1) generate a nominal trajectory connecting a given starting configuration and goal configuration, (2) design a local feedback controller to stabilize about the trajectory, and (3) solve a sum-of-squares program for the trajectory/controller pair to find the maximally-permissive funnel for this trajectory.

Throughout this section, let us denote m as the index of a sample trajectory associated with some reach tube $\mathcal{L}$. Let ℓ^m and $\mathbf{c}^m$ denote, respectively, a funnel and controller associated with this trajectory. The reach tube $\mathcal{L}$ are constructed from a collection of funnels such that $\mathcal{L} = \cup_m \ell^m$ and, likewise, the atomic controller $\mathcal{C}$ is constructed from a collection of local controllers such that $\mathcal{C} = \cup_m \mathbf{c}^m$.

4.1 Trajectory Generation

In our work, we solve a two-point planning problem to generate each sample trajectory, where we attempt to connect a point in S with a goal point in G. For differentially-flat platforms such as nonholonomic wheeled mobile robots, we apply feedback linearization [OLLV02]: a nonlinear transformation on the inputs which produces new pseudo-inputs that are derivatives of the robot's Cartesian coordinates. For static feedback linearization the pseudo-inputs are $u_{pseudo} = [\dot{x}, \dot{y}]^T$. We can then generate an instantaneous steering command by choosing this command to be the Cartesian vector displacement between the current robot configuration and the desired goal configuration, i.e. $u_{pseudo} = [x - x_{goal}, y - y_{goal}]^T$. Using this strategy, we make use of standard ODE solvers to solve the planning problem.

We discard those trajectories which do not satisfy the constraints for the current reach tube (as detailed in Section 3.2); those that are accepted are represented by the pair (ξ_T^m, μ_T^m). For systems which are not feedback linearizable (e.g. 3-D Cartesian robot arms), trajectories can still be generated using nonlinear trajectory optimization methods [Bet09], or any number of motion planning tools.

4.2 Trajectory-Stabilizing Controllers

Local controllers are constructed for the purpose of correcting for deviations from the nominal sample trajectory due to disturbances or initialization. In this work, we apply a linear quadratic regulator (LQR) control design approach [Kir76] to a linearized version of the system (1) based on

the mth trajectory (ξ_T^m, μ_T^m). LQR controllers are generated using the metric quantities $\bar{x}(t) = x(t) - \xi_T^m(t)$ and $\bar{u}(t) = u(t) - \mu_T^m(t)$ using a cost function of the form $\int_0^T (\bar{x}^T \mathcal{Q}_1 \bar{x} + \bar{u}^T \mathcal{Q}_2 \bar{u}) dt + \bar{x}^T \mathcal{S}_T \bar{x}$. The matrices $\mathcal{Q}_1$, $\mathcal{Q}_2$, and $\mathcal{S}_T$ are design parameters that can be adjusted to tailor the shape of the funnels. In our case, we are interested in funnels with wide mouths (initial sets) and small tails (goal sets), so we typically choose $\mathcal{S}_T$ one order of magnitude larger than $\mathcal{Q}_2$, while the values in $\mathcal{Q}_1$ are chosen to remain within the same relative order as $\mathcal{Q}_2$. The control gain $K^m(t)$ is computed based on the matrix solution $\mathcal{S}^m(t)$ to a Riccati equation.

4.3 Invariant Funnels

Given the nominal trajectory (ξ_T^m, μ_T^m) and controller $K^m(t)$, the task is to find a maximally-permissive funnel which satisfies the conditions in Section 3.2. The matrix solution to the Riccati equation from the controller generation step immediately parameterizes quadratic Lyapunov functions $V^m(x,t) = \bar{x}^T \mathcal{S}^m(t) \bar{x}$. Working with quadratic functions averts the problem of searching exhaustively over all classes of Lyapunov function candidates. Given that quadratic representations in general carry only local guarantees, the task is equivalent to finding a maximal $\rho^m(t)$ such that the $\rho^m(t)$-sub-level sets of these Lyapunov functions $\ell^m(\rho^m(t), t)$ (represented as ellipsoids) satisfy the finite-time invariance conditions explained in [TMTR10], while also adhering to the conditions of Section 3.2.

To explain our approach in the context of [TMTR10, MT12], we present the most general problem statement for solving for transition reach tubes using the system (1), then point out which parts of the problem are changed when adapting the problem for inward-facing reach tubes. The full objective is as follows:

$$\max \rho^m(t), \qquad t \in [0, T^m] \tag{8}$$

$$\text{s.t.} \quad \rho^m(t) \geq 0, \qquad \forall t \in [0, T^m], \tag{9}$$

$$\dot{V}^m(x,t) \leq \dot{\rho}^m(t), \qquad \forall t \in [0, T^m], \forall x \in \{x | V^m(x,t) = \rho^m(t)\}, \forall d \in D \tag{10}$$

$$\dot{V}^m(x,t) \leq 0, \qquad \forall t \in [0, T^m], \forall x \in \{x | V^m(x,t) = \rho^m(t)\} \cap X_j, \tag{11}$$

$$\ell^m(\rho^m(t), t) = \{x | V^m(x,t) \leq \rho^m(t)\} \subseteq Inv, \qquad \forall t \in [0, T^m], \tag{12}$$

$$\ell^m(\rho^m(T^m), T^m) = \{x | V^m(x, T^m) \leq \rho^m(T^m)\} \subseteq G \tag{13}$$

where X_j is the polyhedral set corresponding to the successor state q_j. Constraints (9) and (10) enforce trajectory invariance to the funnel and positive-semidefiniteness of the level set. The constraint (11) is a re-statement of (5) in the funnel setting. Likewise, (12) and (13) are, respectively, re-statements of the invariance (3) and goal (4) conditions. Condition (2) is not included here because satisfaction of the start set is implicitly satisfied by sampling the initial state of the trajectory from within S. It is worth pointing out that the difference between our formulation and the one in [MT12] is precisely the addition of conditions (11)–(13).

Often, there are unknown disturbances affecting the robot (wind gusts pushing the robot in one direction or another) which cause unexpected motions and possible collision with obstacles. When we have such disturbances affecting the dynamics, we require that the invariance condition in (9) to be true for *all* $d \in D$.

When computing funnels for inward reach tubes, the inequality (11) is removed and the inclusion (12) is replaced with the following:

$$\ell^m(\rho^m(t), t) = \{x | V^m(x,t) \leq \rho^m(t)\} \subseteq X_i$$

to reflect the condition in (6).

When solving the optimization problem in (8) – (13) using numerical methods, we replace the trajectory (ξ_T, κ_T) with its discrete sequence $(\mathbf{t}^m, \mathbf{c}^m, \mathbf{x}^m)$. We then express the closed-loop system $\hat{f}(x,d)$ under the action of $\mathbf{c}^m$ as being *polynomial* in its arguments x and d. To simplify our implementation, we make sure to choose from a single ρ^m regardless of the time index, so that $\dot{\rho}^m = 0$. We then transform the problem (8)–(13) into a sum-of-squares program replacing the intervals $[0, T^m]$ with $\mathbf{t}^m$ and by making the following substitutions:

Eq. (10) **and** (11)**:**

$$\begin{cases} -\Big(\frac{\partial}{\partial x}V^m(x,t)\hat{f}(t,x,0) + \frac{\partial}{\partial t}V^m(x,t) + \lambda_1(x,t)\left(\rho^m - V^m(x,t)\right) + \lambda_2(x,t)P_j(x)\Big) + \lambda_3(d,t)P_d(d) \\ \qquad\qquad \text{is s.o.s.}, \quad t \in \mathbf{t}^m, \\ \lambda_1(x,t), \;\; \lambda_2(x,t), \;\; \lambda_3(x,t) \text{ are s.o.s.}, \quad t \in \mathbf{t}^m, \end{cases} \tag{14}$$

Eq. (12)**:** $$\begin{cases} (V^m(x,t) - \rho^m) - \lambda_4(x,t)P_{inv}(x) \text{ is s.o.s.}, \quad t \in \mathbf{t}^m \setminus T^m, \\ \lambda_4(x,t) \text{ is s.o.s.}, \quad t \in \mathbf{t}^m \setminus T^m, \end{cases} \tag{15}$$

Eq. (13)**:** $$\begin{cases} (V^m(x,T^m) - \rho^m) - \lambda_5(x)P_G(x) \text{ is s.o.s.}, \\ \lambda_5(x) \text{ is s.o.s.}. \end{cases} \tag{16}$$

Where $\lambda_i(x,t)$, $i = 1, \ldots, 5$ are positive-definite polynomial multipliers and $P_j(x)$, $P_{inv}(x)$,$P_G(x)$, and $P_d(d)$ are all polynomials which define the sets X_j, Inv, G, and D by parameterizing them as the zero-sub-level sets. In particular,

$$X_j = \{x | P_j(x) \leq 0\}, \quad Inv = \{x | P_{inv}(x) \leq 0\},$$
$$G = \{x | P_G(x) \leq 0\}, \quad D = \{d | P_d(d) \leq 0\}.$$

The sum-of-squares program is solved via the MATLAB toolboxes SPOT [Meg] and Ellipsoids [KV06].

4.4 Algorithm

The steps for generating reach tubes are encapsulated in an algorithm `GetReachTube`, outlined in Algorithm 2. The process is iterated up to N times. `GetInit` on line 4 randomly picks an initial point in the S. The function `GetFinal` on line 5 picks a final point inside G, seeking the centroid of the region if the goal set is a polygon and randomly if it is defined by reach tubes. With the boundary conditions defined, `SimTrajectory` is invoked to generate a feasible trajectory and a controller based on LQR. Given a $\mathbf{t}^m$, the function returns a controller library $\mathbf{c}^m$ as the sequence $(\boldsymbol{\mu}^m, \mathbf{K}^m)$, where $\boldsymbol{\mu}^m = \{\mu^m(t)\}_{t \in \mathbf{t}^m}$ and $\mathbf{K}^m = \{K^m(t)\}_{t \in \mathbf{t}^m}$. A trajectory is deemed infeasible if it ever leaves the invariant for the current transition. If this happens, new final points and trajectories are generated until it is unobstructed. If a feasible trajectory is found, `ComputeFunnels` computes the funnels according to the procedure in Section 4.3. Lines 9–12 check if a feasible funnel is found. If so, it is appended to the existing library of funnels.

We address some important implementation details relating to funnel coverage. Due to the regional constraints imposed by boundaries and neighboring regions, the problem of covering the

Algorithm 2:
$(L, C) \leftarrow$ `GetReachTube`$(S, G, Inv, f, \epsilon, N)$

Input: A start set S, a goal set G, an invariant set Inv, along with f, ϵ, N
Output: A set of reach tubes L and controllers C

$m \leftarrow 0$, $L \leftarrow \varnothing$, $C \leftarrow \varnothing$
while $Vol(L \cap S) < (1-\epsilon)Vol(S) \wedge m < N$ **do**
  $m \leftarrow m + 1$
  $x^i \leftarrow$ `GetInit`(S)
  $x^f \leftarrow$ `GetFinal`(G)
  $(\mathbf{t}^m, \mathbf{x}^m, \mathbf{c}^m) \leftarrow$ `SimTrajectory`(f, x^i, x^f, Inv)
  if $\mathbf{x}^m \neq \varnothing$ **then**
    $\ell^m \leftarrow$ `ComputeFunnel`$(\mathbf{t}^m, \mathbf{x}^m, \mathbf{c}^m, G, Inv)$
    if $\ell^m \neq \varnothing$ **then**
      $L \leftarrow L \cup \ell^m$
      $C \leftarrow C \cup (\mathbf{t}^m, \mathbf{x}^m, \mathbf{c}^m)$
    end
  end
end
return L, C

set of configurations in state q_i for a transition from q_i to q_j may not terminate. The algorithm allows for this by introducing a metric $\epsilon \in [0, 1]$ allowing the coverage loop to terminate without actually achieving full coverage. We declare the space covered if $Vol(\mathcal{L} \cap S) \geq (1-\epsilon)Vol(S)$ or if $m = N$ where Vol is the volume of a particular set defined in $\mathbb{R}^n$, m is the current funnel iterate, and N is an integer. The former condition asserts that coverage terminates if the volume of the reach tube $\mathcal{L}$ within the start set is a significant enough fraction of start set. If the coverage is not achieved before N iterations, then there may be transitions from region $\gamma_{\mathcal{R}}(q_i)$ which are not reachable from some parts of the state space $R(q_i)$ for that region. Hence, the method is sound, but not complete.

Another implementation issue arises from the curvature of the ellipsoidal level sets. At the boundaries of polyhedral sets, coverage degenerates where the ellipsoid level sets meet the polyhedral region boundaries, causing difficulty in generating transition reach tubes. We work around this issue by relaxing the interface between neighboring regions by some fixed tolerance value ϵ_{reg}. This parameter allows regions to share territory by an amount defined by this fixed distance. To more strictly enforce one of the two region boundaries, one can adjust the shared boundary in the direction of one region or another. Although not explored in this work, it is also possible to remove this shared boundary by adopting an approach which allows ellipsoidal level sets to be expanded beyond region boundaries as long as the boundary itself is certified as an invariant.

5 Controller Execution

The controllers used to execute the motion plan associated with the high-level controller automaton are selected at runtime according to the current state of the robot and the current values of the sensor propositions. The planner executes the controller associated with the funnel associated

with the current state (e.g. $\mathbf{c}_{ij}^m$ if within $\ell_{ij}^m(t)$). As the continuous trajectory evolves, a new funnel is selected if one of three events occur: (i) the end of a funnel is reached, (ii) a region transition is made and either an inward funnel or a transition funnel for the next transition is reached, or (iii) an environment proposition changes. Priority is always given to transition controllers κ_{ij} over inward ones κ_i^c. That is, if the robot is currently executing a controller in κ_i^c and the trajectory reaches a funnel in $\mathcal{L}_{ij}$, with r_j as the goal for that transition, the motion controller is switched from κ_i^c to κ_{ij}. In Example 1, consider the case when the robot is in state q_4 with $S\_blocked = \texttt{False}$. At the current time step, the robot is executing the controller κ_{34} and has just reached O. The next goal (S) is implemented by switching controllers to κ_{41} if within the associated funnel. Otherwise, the planner will choose κ_4^c. If the trajectory is in more than one funnel, the execution paradigm operates deterministically (the first funnel encountered in the library is always selected).

We now show that the algorithm, when executed using the runtime implementation described above, produces trajectories which remain consistent with respect to the behaviors in the original controller automaton. Define $\omega : \mathbb{R}_+ \to \mathcal{X}$ as a possible environment proposition trajectory initialized at $\omega(0)$. Let $q(0)$ be the initial state, and $\xi^{q(0)}$ be a disturbance-free continuous trajectory in $X_{q(0)}$, with the robot's configuration initially $\xi^{q(0)}(0)$. Let $q(k)$, $k > 0$ be defined as follows. Initializing $k = 0$, $\tau_0 = 0$, we increment k either when a time $t = \tau_k > 0$ is reached for which $\xi^{q(k)}(\tau_k) \in \partial X_{q(k)}$ (the trajectory reaches a boundary) or when $\delta(q(k), \omega(\tau_k)) = q(k)$ (a self-transition is already satisfied at time $t = \tau_k$ under the environment input $\omega(\tau_k)$). Each time k is incremented, let $\xi^{q(k-1)}(\tau_{k-1}) = \xi^{q(k)}(\tau_{k-1})$ to ensure continuity of the trajectories and build a trajectory segment $\xi^{q(k)}(t)$, $t \in [\tau_{k-1}, \tau_k)$. To avoid livelock, we assume that the environment does not change too fast by restricting $\omega(t)$, $t \in [\tau_k, \tau_{k+1})$ to only those input traces for which livelocking does not occur for all $k \in \mathbb{N}$. Given any such ω, we call the sequence $\sigma = \omega(0)\mathcal{R}(q(0))\omega(\tau_1)\mathcal{R}(q(1))\ldots$ a *reactive execution trace* associated with the continuous trajectory.

In the following proposition, we show that the continuous trajectories obtained by executing the atomic controllers synthesized from Algorithm 1 yield reactive execution traces which enforce the behaviors of the high-level controller.

Proposition 1. *Consider a robot initialized within $\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c$, $q_i \in Q_0 \wedge (q_i, \cdot) \in \Delta$. For all ω, executing Algorithm 1 produces a reactive execution trace σ of the high-level controller A.*

Proof. To show that σ produced by the execution is in fact an execution trace of A, we remark that reach tubes are constructed from edges in Δ. For any $(q_i, q_j) \in \Delta$, the robot will either be in $\mathcal{L}_{ik}$ for some $k \in I_{out}^i$ where conditions (3) and (4) hold true, or will be in $\mathcal{L}_i^c$, where (6) and (7) hold true. Once in X_j, the robot will not re-enter X_i as a consequence of (5). Therefore, the robot will not exhibit additional behaviors not in A.

To show that the execution traces of A are complete with respect to σ, it is only necessary to show that, for any state in $\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c$, we can take any transition from q_i. By construction, the set $\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c$ is reactively composable, thus proving completeness of the executions of A. □

Note that in general possible successor regions can be far away from each other and so, we cannot guarantee the behaviors of A for any arbitrary trajectory of environment propositions. For example, consider again state q_4 in Example 1. The environment is allowed to toggle $S\_blocked$ between `True` and `False` indefinitely when the robot is within O. In the continuous setting, this would cause the robot to get trapped forever in q_4, while in the discrete automaton, there is no such livelock, since this toggling does not appear in the discrete execution. This is due to the physical

setting in which robots operate and is a limitation of the abstraction, rather than the low-level controller synthesis approach.

6 Simulations

In this section, we demonstrate the application of the method developed in this paper to three examples. For these examples, we make use of a unicycle robot model consisting of three states, governed by the kinematic relationship:

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix},$$

where x_r and y_r are the Cartesian coordinates of the robot, θ is the orientation angle, and v and ω are, respectively, the forward and angular velocity inputs to the system. In this work, we augment the model by limiting ω such that

$$\omega = \begin{cases} \omega_{max} & \text{if } u \geq \omega_{max} \\ \omega_{min} & \text{if } u \leq \omega_{min} \\ u & \text{otherwise} \end{cases}$$

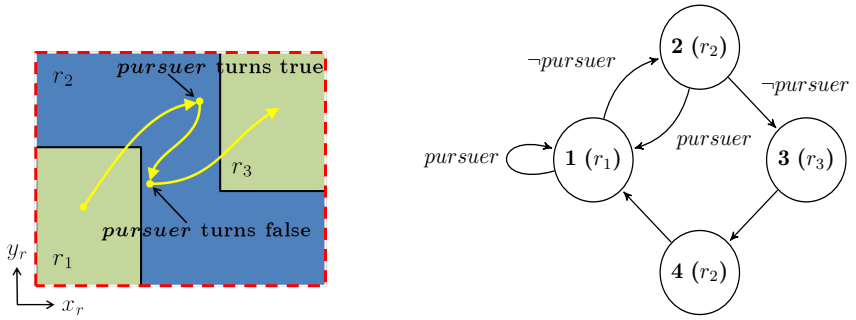
and $v = v_{nom}$. The input u is governed by a feedback controller which steers the robot from some initial configuration to the desired configuration. We adopt the parameter settings $\omega_{min} = -3$, $\omega_{max} = 3$, and $v_{nom} = 2$.

For each example, we perform computations using MATLAB on a standard 64-bit Windows desktop machine, with an Intel Core i7 processor clocked at 3.40 GHz and 8 GB of RAM.

6.1 Patrolling Two Regions

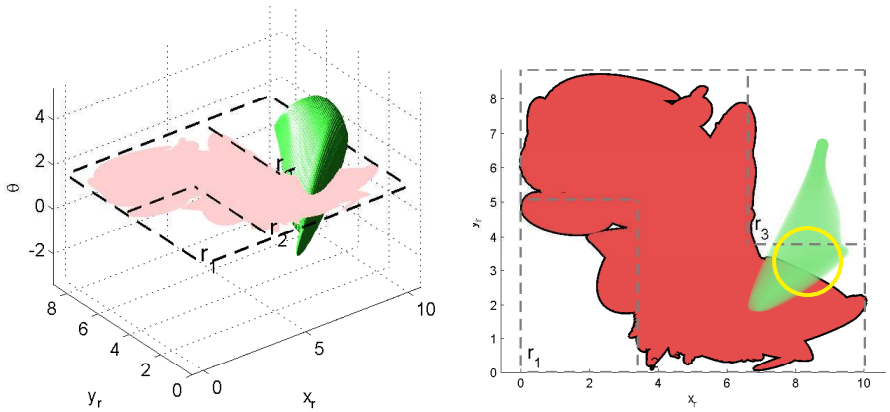
Consider a unicycle robot moving in the $10m \times 9m$ workspace shown in Figure 4a. The robot is initially in r_1 and must continually patrol r_3 and r_1. If the robot senses a pursuer, then it must return to r_1 (safe zone). The specification is implemented by the discrete automaton shown in Figure 4b.

A library of controllers is generated which are reactively composable. For this example, we set the coverage metric $\epsilon = 0.2$, the number of iterations $N = 100$, and the interface relaxation $\epsilon_{reg} = 0.2m$. Reach tubes $\mathcal{L}_{23}$ are generated in the first and second iterations for (r_2, r_3) and sample funnels are shown in Figs. 5 and 6. Also shown are the θ-slices of the union of the inward reach tube and transition reach tube $\mathcal{L}_2^c \cup \mathcal{L}_{21}$ (shown in red). In the first iteration, it is apparent that the funnel spans a gap in the set of inward funnels and hence the controllers are not reactively composable. If the controller drives the robot to this portion of the configuration space, the robot cannot change direction if it senses a pursuer. We thus continue with another iteration of Algorithm 1. After the second iteration, the revised funnel is reactively composable for all transitions and no further iterations are necessary. The volumetric region coverage for each of the four states are shown in Table 1. Here, volume fraction is defined as the ratio of the actual volume of the region polytope $R(q_i)$ in (x_r, y_r, θ) and the subset of that polytope which contains $\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c$.



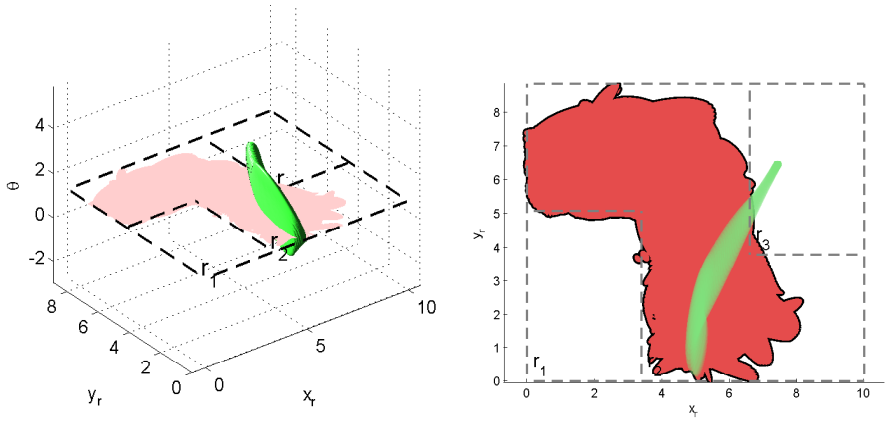
(a) (b)

Figure 4: Controller automaton for the "patrolling two regions" example. The transitions are labeled with the truth value of the $pursuer$ sensor; unlabeled transitions imply that $pursuer$ can take on any value.



(a) (b)

Figure 5: (a) shows a transition funnel for the transition from r_2 to r_3 and a slice of the set where r_1 is reachable from r_2 at $\theta = 1.35$ (defined by $\mathcal{L}_2^e \cup \mathcal{L}_{21}$) after the *first* iteration of Algorithm 1. (b) shows a 2-D view of the slice. In this iteration, the funnel is not reactively composable because the portion of it which lies in r_2 is not contained in the red set. If the robot is outside of the red set when enroute to r_3, there are no controllers which can deliver it to r_1 if it sees a pursuer.



(a) (b)

Figure 6: (a) shows a transition funnel for the transition from r_2 to r_3 and a slice of the set where r_1 is reachable from r_2 at $\theta = 1.05$ (defined by $\mathcal{L}_2^c \cup \mathcal{L}_{21}$) after the *second* iteration of Algorithm 1. (b) shows a 2-D view of the slice. The funnel is now reactively composable because the portion of it which lies in r_2 is now completely enclosed by the red set. The robot is therefore able to move to r_1 if it senses a pursuer anywhere along its path to r_3.

Table 1: Fraction of (x_r,y_r,θ) coverage for each X_i associated with the automaton.

q_1	q_2	q_3	q_4
(r_1)	(r_2)	(r_3)	(r_2)
0.5846	0.5976	0.5295	0.6171

Fig. 7 shows a sample trajectory of a robot starting in r_1, in which a controller in $\mathcal{C}_{12}$ is applied, followed by a controller in $\mathcal{C}_2^c$ and one in $\mathcal{C}_{23}$. Part way through its motion to r_3, the $pursuer$ sensor turns `True`, invoking the sequence $\mathcal{C}_2^c$, $\mathcal{C}_{21}$ to take the robot to r_1. $pursuer$ once again becomes `False` prompting activation of a controller in $\mathcal{C}_2^c$ followed by one in $\mathcal{C}_{23}$. As can be seen, the robot remains within the funnels and is able to satisfy the specification regardless of the environment as it moves between regions.

6.2 Pursuit-Evasion Game

Consider a game being played between a robot and an adversary, where the robot must repeatedly visit the home and goal regions pictured in Fig. 9, while evading an adversary which visits each of the three remaining regions infinitely often. Evasion is encoded by the requirement that the robot should never enter the same region as the enemy. By assuming the enemy patrols its three regions, the robot cannot get trapped forever in the goal region. As a fairness condition, in the specification we assume also that the enemy cannot enter the region the robot is currently in. The high-level controller (Fig. 8) consists of eight states and 13 transitions. The sensor values $inR1$, $inR2$ and $inR3$ correspond to the sensed location of the adversary.

Reach tubes are constructed for each of the 13 transitions. A subset of these (the highlighted edges in Fig. 8) are shown in Fig. 9, showing the possible trajectories that the robot may follow when transitioning between $goal$ and r_3 (denoted green), and when transitioning between r_3 and r_1. Note that the $goal$–r_3 funnels all deliver the robot to the left of $goal$. The reason for this is that

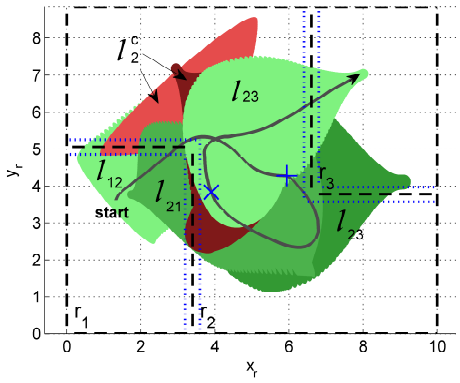


Figure 7: Closed-loop trajectory generated from an initial state in r_1. A set of control laws are applied to implement the transitions (r_1, r_2) and (r_2, r_3), driving the robot from state 1 to state 2, then from state 2 to state 3 in Fig. 4b. 2-D projections of the active funnels are also shown, the red corresponds to inward and green corresponds to transition. $pursuer$ turns `True` when at the location marked by the "$+$" sign. At this instant, another controller is invoked to make the transition (r_2, r_1). $pursuer$ turns `False` at the "$\times$" location, and new control laws are used to resume the transition (r_2, r_3).

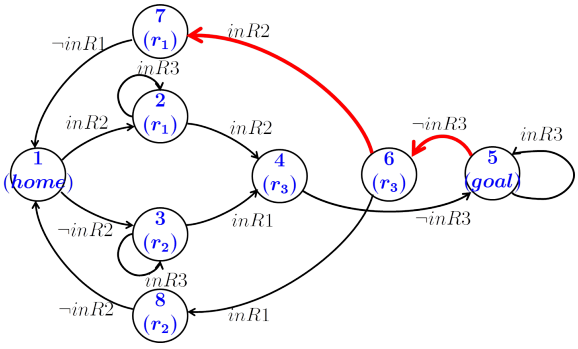


Figure 8: Discrete automaton for the pursuit-evasion example. For simplicity, we label each edge by the value the sensor proposition must take, and exclude labels for which the remaining input labels can be inferred based on the assumptions on the enemy's behavior. For example, the transition (q_6, q_7) is labeled $inR2$; by mutual exclusion of the regions the enemy can occupy, when $inR2$ is `True`, $inR1$ and $inR3$ are `False`.

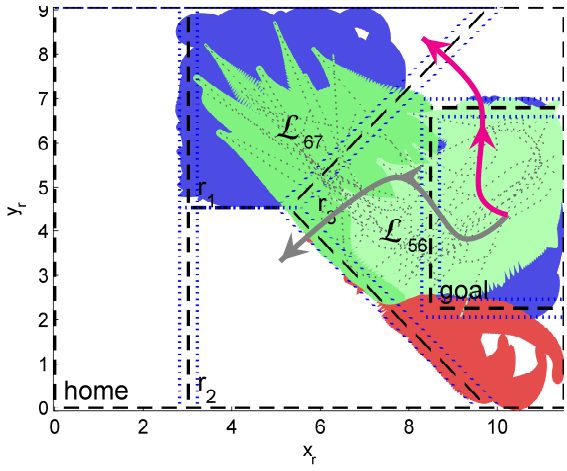


Figure 9: Transition funnels for $\mathcal{L}_{56}$ and $\mathcal{L}_{67}$ (green) in the pursuit-evasion example. $\mathcal{L}_5^c$ and $\mathcal{L}_7^c$ are shaded blue and $\mathcal{L}_6^c$ is shaded red. Two hypothetical trajectories are shown; the one exiting the top of goal is not reactively composable and hence always enters r_4 regardless of the environment when in r_3, while the one exiting the left face of goal is reactively composable allowing the robot to choose between entering r_1 or r_2.

there are two possible transitions out of r_3 (r_1 and r_2) depending on the location of the enemy. If the robot invokes these reach tubes, it always exits the left-hand facet of the goal region, where the robot is easily able to toggle between the goals r_1 and r_2 as the enemy toggles between those regions. The gray trajectory in Fig. 9 illustrates this for the case when $inR1$ remains `True`. Without reactive composition, a motion planner may cause the robot to exit $goal$ via the top or bottom facets. If the robot follows the hypothetical magenta trajectory in Fig. 9, if if $inR1$ stays `True` when in r_3, the robot will have no other option but to enter r_1 (violating the specification) because there are neither any transition funnels $\mathcal{L}_{68}$ nor any inward funnels $\mathcal{L}_6^c$ in the area above the goal region.

6.3 Delivery in a Cluttered Environment

We next return to the example in Example 1. The task, once again, is to repeatedly deliver supplies between region S and R, connected via O, and return to C if the store S is blocked. We apply two models to fulfill this task: a model of the under-actuated unicycle and a model of a non-holonomic car. The non-holonomic car is represented by the four-dimensional system:

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{\ell_r}\tan\phi \\ \omega \end{bmatrix},$$

where x_r, y_r, θ, v and ω are the same as for the unicycle, ϕ is the steering angle and ℓ_r is the axle spacing, which we take to be $\ell_r = 0.5$ m. Unlike with the unicycle, we impose no limits on the control inputs v and ϕ.

We apply Algorithm 1 to this problem for a workspace similar to the one shown in Figure 1a. Because the central region (O) consists of a large area interspersed with several obstacles, we forego attempting to cover the entirety of the region and instead seed inward controllers at configurations where transition controllers already reside. We do this to aid composition; for a given number of funnels, concentrating funnels together reduces gaps in in the reactively- and sequentially-composable sets. Rather than keeping the goal region the same for all inward funnels, we attempt to increase coverage depth by permitting inward funnels to be sequentially composed with one another. We do this by redefining the goal region to include the most recent inward funnel each time one is computed. We limit the number of transition controllers to 20 and the number of inward controllers to 100. With these parameters, each major iteration in the while loop (lines 6–22) the MATLAB implementation took approximately 780 minutes to complete for the unicycle robot and 1470 minutes for the car. For both models, we terminate after two major loop iterations. We note that the method we introduce is intended as an offline verification and controller synthesis approach, and therefore the code and computer hardware implementations were not optimized in this work.

As constructed, the controllers are able to handle infinite executions of the high-level automaton in Figure 1b. A partial sample trajectory for the unicycle robot starting in region S is shown in Figure 10. The controller coverage (projection of the inward and transition reach tubes onto x_r, y_r) is indicated by the shaded regions in the map. In the first iteration of the synthesis algorithm, transition funnels initially cover a large portion of region O. After the first iteration, sampling of the funnel trajectories becomes more concentrated only in areas where reactive composition can be met. In our case, this is in the central part of the region. A side benefit to the approach is that the resulting controllers tend to yield efficient (non-circuitous) trajectories in the free configuration space. Figure 11 shows the same scenario as Figure 10, except using the car robot. Similar to the unicycle, the car robot is able to satisfy the high-level automaton. One of the differences is that the coverage area for the car is slightly smaller than the unicycle, which is primarily due to the dynamics being restricted by nonholonomic constraints. For both robot models, the trajectory remains within the verified reach tube for most of the trajectory, but in a few cases the trajectory momentarily exits the reach tube. This is attributable to the fact that we construct reach tubes based on an a polynomial approximation of the true dynamics.

7 Conclusion

In this paper, a method is presented for synthesizing controllers in the continuous domain based on a discrete controller derived from a high-level reactive specification. The central contribution of this paper is an algorithm that generates controllers guaranteeing every possible execution of a discrete automaton for robots with nonlinear dynamics.

Since a large number of computations are required to compute trajectories and funnels satisfying ellipsoidal constraints, the trade-off between completeness and complexity will need to be explored further. In contrast to the approach taken in this paper, one could devise a depth-first strategy which seeks to generate atomic controllers in concentrated parts of the configuration space. While there is much to be gained in terms of computational efficiency (there would be fewer funnels in the database), this would be at the expense of completeness, since the vast majority of possible configurations would not be tied into the funnel libraries.

If a set of atomic controllers cannot be synthesized, a natural question might be to ask: which parts of the specification, when modified, would allow the synthesis of controllers? Such feedback

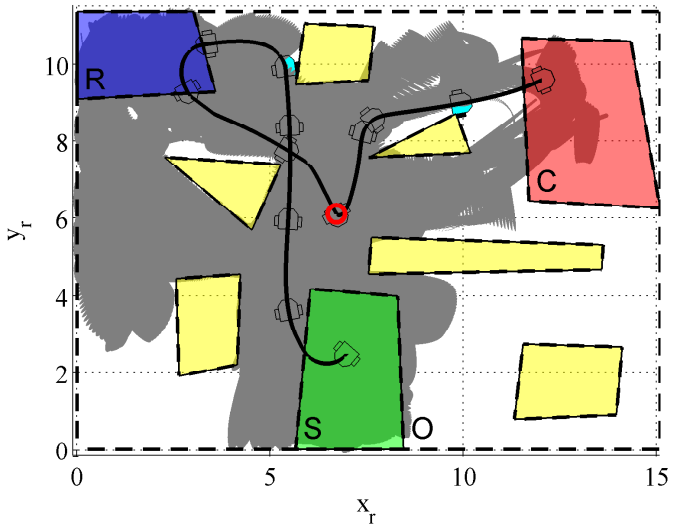


Figure 10: A partial sample trajectory for the delivery scenario in Example 1 using the unicycle robot. The robot pose is shown at uniform time intervals, and the projection of the funnels appears as shaded fill areas. The red ∘ indicates the rising edge of the $S\_blocked$ sensor.

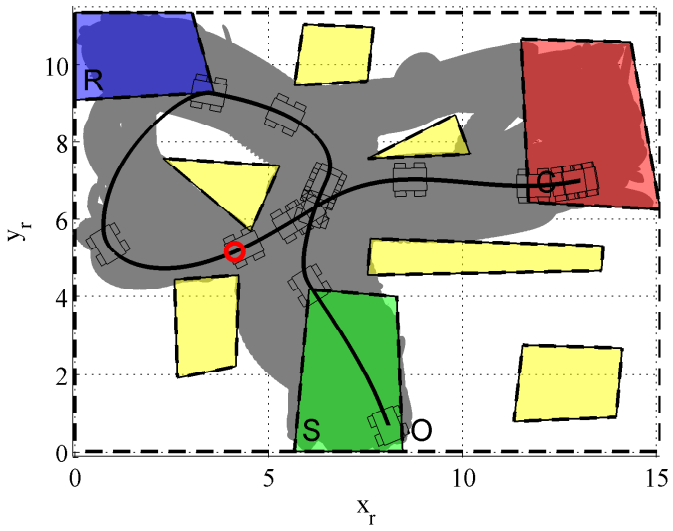


Figure 11: A partial sample trajectory for the delivery scenario in Example 1 using the non-holonomic car. The robot pose is shown at uniform time intervals, and the projection of the funnels appears as shaded fill areas. The red ∘ indicates the rising edge of the $S\_blocked$ sensor.

would allow users to revise the specification in such a way to allow for the execution of a modified task. We intend to explore this topic further as future work.

Acknowledgment

The authors are grateful to Russ Tedrake and Anirudha Majumdar for insightful discussions and advice with the technical implementations in this paper.

References

[Bet09] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[BH04] Calin Belta and L.C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *Proc. of the 43rd IEEE Conf. on Decision and Control (CDC 2004)*, pages 467–472, Bahamas, 2004.

[BKV10] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA 2010)*, pages 2689–2696. IEEE, 2010.

[BRK99] R.R. Burridge, A.A. Rizzi, and D.E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18:534–555, 1999.

[CCR06] David C. Conner, Howie Choset, and Alfred A. Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Robotics: Science and Systems*, 2006.

[CRC03] David C Conner, Alfred Rizzi, and Howie Choset. Composition of local potential functions for global robot control and navigation. In *Proc. of 2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2003)*, volume 4, pages 3546– 3551. IEEE, October 2003.

[DGH11] Jerry Ding, Jeremy Gillula, Haomiao Huang, Michael P. Vitus, Wei Zhang, and Claire J. Tomlin. Hybrid systems in robotics: Toward reachability-based controller design. *IEEE Robotics & Automation Magazine*, 18(3):33 – 43, Sept. 2011.

[DKG13] Jonathan A. DeCastro and Hadas Kress-Gazit. Guaranteeing reactive high-level behaviors for robots with complex dynamics. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2013)*, 2013.

[FLP06] Georgios E. Fainekos, Savvas G. Loizou, and George J. Pappas. Translating temporal logic to controller specifications. In *Proc. of the 45th IEEE Conf. on Decision and Control (CDC 2006)*, pages 899–904, 2006.

[Fra01] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.

[JFA07] A. Agung Julius, Georgios E. Fainekos, Madhukar Anand, Insup Lee, and George J. Pappas. Robust test generation and coverage for hybrid systems. In *Proc. of the 10th int. conf. on Hybrid systems: computation and control (HSCC'07)*, HSCC'07, pages 329–342, Berlin, Heidelberg, 2007. Springer-Verlag.

[KB06] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from ltl specifications. In *Proc. of the 9th Int. Conf. on Hybrid Systems: Computation and Control*, HSCC'06, pages 333–347, Berlin, Heidelberg, 2006. Springer-Verlag.

[KF09] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proc. of the 48th IEEE Conf. on Decision and Control (CDC 2009)*, pages 2222–2229, 2009.

[KGFP09] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[Kha02] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 3rd edition, 2002.

[Kir76] D.E. Kirk. *Optimal Control Theory: An Introduction*. Prentice-Hall, 1976.

[KV06] Alex A. Kurzhanskiy and Pravin Varaiya. Ellipsoidal toolbox. Technical Report UCB/EECS-2006-46, EECS Department, University of California, Berkeley, May 2006.

[LaV06] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[LK04] Savvas G. Loizou and Kostas J. Kyriakopoulos. Automatic synthesis of multiagent motion tasks based on ltl specifications. In *Proc. of the 43rd IEEE Conf. on Decision and Control (CDC 2004)*, pages 153–158, 2004.

[LOTM13] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Automat. Contr.*, 58(7):1771–1785, 2013.

[MDT10] Manuel Mazo, Anna Davitian, and Paulo Tabuada. Pessoa: A tool for embedded controller synthesis. In *22nd International Conference on Computer Aided Verification (CAV 2010)*, pages 566–569, 2010.

[Meg] A. Megretski. Systems polynomial optimization tools (spot). http://web.mit.edu/ameg/www/.

[MLK13] MR Maly, M Lahijanian, Lydia E. Kavraki, H. Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 353–362, Philadelphia, PA, USA, 08/04/2013 2013. ACM, ACM.

[MT12] Anirudha Majumdar and Russ Tedrake. Robust online motion planning with regions of finite time invariance. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, 2012.

[MTT12] Anirudha Majumdar, Mark Tobenkin, and Russ Tedrake. Algebraic verification for parameterized motion planning libraries. In *Proc. of the 2012 American Control Conference (ACC)*, 2012.

[OLLV02] Giuseppe Oriolo, Alessandro De Luca, Ro De Luca, and Marilena Vendittelli. Wmr control via dynamic feedback linearization: Design, implementation, and experimental validation. *Control Systems Technology, IEEE Transactions on*, 10(6):835–852, November 2002.

[THDT12] Ryo Takei, Haomiao Huang, Jerry Ding, and Claire J. Tomlin. Time-optimal multi-stage motion planning with guaranteed collision avoidance via an open-loop game formulation. In *IEEE International Conference on Robotics and Automation (ICRA 2012)*, pages 323–329, 2012.

[TMTR10] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *I. J. Robotic Res.*, 29(8):1038–1052, 2010.

[WTM10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Proc. of the 13th Int. Conf. on Hybrid Systems: Computation and Control (HSCC'10)*, 2010.

[WTM13] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2013)*, 2013.

[ZPMT12] Majid Zamani, Giordano Pola, Manuel Mazo, and Paulo Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Automat. Contr.*, 57(7):1804–1809, 2012.