Guaranteeing Reactive High-Level Behaviors for Robots with Complex Dynamics

Jonathan A. DeCastro and Hadas Kress-Gazit

***Abstract*— Applying correct-by-construction planning techniques to robots with complex nonlinear dynamics requires new formal analysis methods which guarantee that the requested behaviors can be achieved in the continuous space. In this paper, we construct low-level controllers that ensure the execution of a high-level mission plan. Controllers are generated using trajectory-based verification to produce a set of robust reach tubes which strictly guarantee that the required motions achieve the desired task specification. Reach tubes, computed here by solving a series of sum-of-squares optimization problems, are composed in such a way that all trajectories ensure correct high-level behaviors. We illustrate the new method using an input-limited unicycle robot satisfying task specifications expressed in linear temporal logic.**

I. Introduction

Growing attention in robot mission planning is being directed to addressing the problem of synthesizing controllers for a complex set of reactive tasks. Tools for correct-by-construction synthesis are therefore being developed to automatically synthesize hybrid controllers based on a set of user-defined instructions encoded formally as a specification. Recently, several researchers (e.g. [1]–[7]) have developed techniques which translate user-defined specifications expressed as temporal logic formulas into high-level controllers which guarantee fulfillment of the specification. One property of these methods is that they can guarantee fulfillment of a task as long as there exist low-level controllers to implement the actions requested by the hybrid controller. Another property is that many ([3], [7]) allow for *reactive* tasks, i.e. tasks that call for the robot's actions to change in response to real-time sensory information.

Motion planning for complex robotic platforms, such as robotic manipulators [8], personal assistants [9], self-driving vehicles [10], and unmanned air vehicles (UAVs) [4], has been the subject of intense research in recent years. While these algorithms deal well with specifications involving a fixed goal or sequence of goals, further work is required to generate motion controllers that guarantee the rich set of behaviors resulting from temporal logic planning. Temporal logic planners, on the other hand, have enjoyed success when applied to nonholonomic kinematic models [11] or piecewise-affine dynamical robot models [12]. For these classes of systems, atomic (low-level) feedback control strategies are devised to ensure that high-level specifications are satisfied. The primary drawback is that such methods do not extend generally to more complex systems.

The aim of this work is to introduce a method for the automatic synthesis of low-level controllers for arbitrary dynamical systems. Under this new framework, controllers can be generated to ensure the specification will be achieved either globally throughout the continuous domain or within a subset of it. This result is of key importance to guaranteeing the feasibility of the task in the continuous space. If the original specification is infeasible, it may be possible to either limit the domain or create alternative specifications which are more compatible with the robot dynamics.

The main contribution of this paper is an algorithm which implements reactive behaviors for a robot subjected to any possible environmental events in a known map. Our method assumes a discrete automaton synthesized from a high-level specification and generates motion primitives for complex systems implementing each of the automaton transitions. The algorithm will either result in the successful generation of a library of controllers, or failure if no controller is found which guarantees the execution of any portion of the automaton. If a controller is synthesizable, the subsets of the configuration space where guarantees hold will also be provided. We apply the method of invariant funnels [13] to perform construction of controllers and verification of the closed-loop system in the continuous domain, due to its applicability to a variety of types of models and treatment of different sources of uncertainty.

There has been considerable work on techniques which can generate controllers that preserve the reachability and safety of nonlinear systems, i.e. controllers which guarantee that desired goals may be reached while avoiding any "bad" portions of the state space. In [14], the authors offer an approach to directly synthesize controllers based on a game-theoretic criterion. Reachable sets of the closed-loop dynamics and controllers are selected based on whether or not these reachable sets intersect with obstacles. In [15], nonlinear models are abstracted symbolically to enable the construction of a hybrid control system. By virtue of the type of abstaction used, the method does not require the exact computation of reachable sets. Other safety verification methods include barrier certificates [16], and polyhedral methods based on state space partitioning [17].

The strategy we propose takes inspiration primarily from the work of [13], [18]. In [18], the authors propose a method which translates the desired high-level behaviors into continuous controller specifications. The high-level controller, taking the form of a hybrid automaton, is synthesized

This work was supported under NSF Expeditions in Computer Augmented Program Engineering (ExCAPE).

The authors are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA, {jad455,hadaskg}@cornell.edu

from a linear temporal logic (LTL) specification, and low-level controllers are then automatically constructed from the synthesized automaton. The invariant funnels method in [13] composes a reachable set based on locally-derived neighborhoods (funnels) computed about a set of sample trajectories. These funnels, computed based on the system dynamics, guarantee that all trajectories starting within the funnel remain within the funnel over a finite time interval. The method is extended in [19] to include the effects of bounded disturbances for on-line motion planning.

Our approach differs from these works in that we seek explicit guarantees in the continuous space for a given set of controllers acting on a nonlinear robot model operating *reactively* in a dynamic environment [7]. For example, if a housekeeping robot senses a fire, a reasonable reactive task would be for it to abort its current tasks and proceed to the living room to call for help. Although the work of [19] permits replanning in the presence of obstacles and disturbances, it does not address scenarios where goals may change as a result of events in the environment. The algorithm we introduce adheres to a property which we call *reactive composition* which enforces the requirement that, along any trajectory implementing any one automaton transition, there exist trajectories implementing all remaining transitions. Our approach, moreover, allows the robot model to assume the form of a high-order nonlinear model, while the work in [18] considers a fully-actuated robot.

The paper introduces the problem through a motivating example in Section II. In Section III, the concept of LTL-based controller synthesis is introduced as it relates to atomic controller design. Section IV covers the trajectory-based verification technique used for construction of local controllers. The algorithm for the design of a library of controllers which satisfy the specification is presented in Section V. Two illustrative examples are given in Section VI for an input-limited robotic unicycle. Finally, the paper concludes in Section VII with a summary and future work.

II. Motivating Example

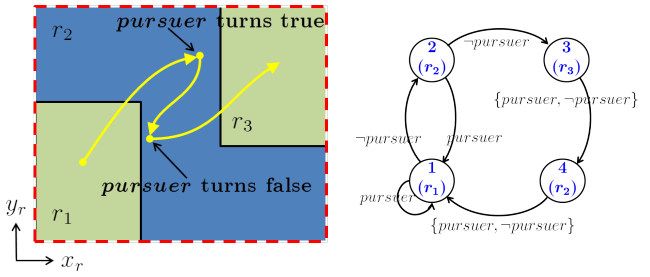
To motivate this work, we provide a simple example, and briefly discuss its implications.

Example 1. *Consider a balancing unicycle robot moving in the environment shown in Fig. 1(a). 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 (home). The specification is implemented by the discrete automaton shown in Fig. 1(b). In our model of the unicycle, the forward velocity is held fixed and the angular velocity is constrained to an interval.*

To model the unicycle robot, we consider the following kinematic model:

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

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



(a) (b)

Fig. 1. Workspace and controller automaton for Example 1. In (a), a 2-D workspace is shown, along with a sample trajectory. In (b), the number at the top of each circle is the state, while the values in parentheses denote the region associated with that state. The transitions between each state are each indicated with the truth value of the *pursuer* sensor needed to make the transition.

forward and angular velocity inputs to the system. For 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. The details of the construction of this controller are discussed in Section IV.

Our objective is to construct feedback controllers which guarantee the region transitions in the automaton; in this case, the automaton (shown in Fig. 1(b)) requires motion to occur between r_1, r_2, and r_3. One of the key challenges in synthesizing these controllers arises from the presence of *reactive* behaviors: the robot may change its goals based on the value of the *pursuer* sensor. This is illustrated more clearly by the segment of the trajectory pictured in Fig. 1(a). With *pursuer* set to `False`, the robot begins in r_1 (state 1 in Fig. 1(b)), then moves to r_2 (state 2), to be followed by r_3 (state 3). The *pursuer* sensor then turns `True` as the robot is implementing the r_2–r_3 transition. The new goal is now state 1, forcing the robot to move to r_1. Before exiting r_2, *pursuer* again becomes `False`, and the robot once again resumes towards r_3 (state 3).

If the robot had entered r_3 on the r_2–r_1 transition or r_1 on the r_2–r_3 transition, the automaton in Fig. 1(b) would be violated, and the robot would fail under this control strategy. In general, *pursuer* may turn `True` at any point in the robot's continuous trajectory, and so the challenge is finding low-level controllers which guarantee region transitions for every possible behavior of the environment.

III. Preliminaries

Presented in this section are key concepts from the LTL controller synthesis method in [7], definitions, and the assumptions on the class of systems treated in this paper.

A. Task Specifications in LTL

Linear temporal logic (LTL) extends propositional logic by introducing temporal operators, allowing the specification of desired system behaviors in response to the environment

in which the robot operates. Here, the term *system* refers to the set of user specifications which are ascribed to the robot (e.g. the specification "visit r_1 and r_3"). The term *environment* refers to the behavior of events external to the robot, as perceived by its sensor inputs (e.g. "expect a pursuer only in regions r_1 and r_2"). LTL formulas allow users to describe behaviors such as *liveness*, which occur infinitely often, and *safety*, which must always hold. The interplay between system and environment may be specified by *reactive* tasks which depend on events detected in the environment. We refer the reader to [20] for details regarding the syntax and semantics of LTL.

B. Discrete Abstraction and Automaton

As a necessary step in the synthesis process, we start with a discrete abstraction to the continuous system. Here, the continuous configuration space, $X \subseteq \mathbb{R}^n$, is partitioned into a set of discrete regions which, in our case, are 2D polygons (not necessarily convex). In this discrete abstraction, sensors and actions are defined as Boolean propositions. Sensors may be regarded, for example, as a thresholded value of a continuous sensed quantity (e.g. noise detection based on a microphone's signal intensity). Actions refer to discrete robot functions (e.g. stand up, sit down) which, along with locomotion commands, define the robot's abstracted behaviors.

An automaton may be synthesized from high-level task specifications using, for example, the method explained in [7]. Formally, a finite-state automaton A is defined as a tuple $A = (\mathcal{X}, \mathcal{Y}, Q, Q_0, \delta)$, where:

- $\mathcal{X}$ is a set of environment propositions.
- $\mathcal{Y}$ is a set of system propositions.
- $Q \subset \mathbb{N}$ 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.

Among the set of system propositions $\mathcal{Y}$, we distinguish those which correspond to regions as $\mathcal{R} \subseteq \mathcal{Y}$. We 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 $\gamma_{\mathcal{R}}(q)$. In the case of a 2-D nonholonomic robot, we have a 3-D configuration space (x_r, y_r, and θ) and hence 3-D polytopes. The set of edges in A is defined as $\Delta = \{(q, q') \in Q^2 | \exists z \in 2^{\mathcal{X}} \ . \ \delta(q, z) = q'\}$.

C. Continuous Dynamics

Before discussing the execution of the controller, we briefly outline the continuous dynamics. Consider the general description of a nonlinear system,

$$\dot{x} = f(t, x), \quad x(0) \in S$$

where $x \in X \subseteq \mathbb{R}^n$ is the state vector. The initial states are bounded by some start set S. Throughout, f is considered to be a smooth, continuous vector field within its domain X. The interpretation of the system model is that it represents the closed-loop dynamics of a nonlinear robot model, evolving according to some prescribed feedback control system. The details of the construction of these low-level controllers will be discussed in Section IV.

D. Continuous Execution of the High-Level Controller

A necessary condition for realization of the discrete abstraction in the continuous domain is for the closed-loop system to implement each of the region transitions in A. This requirement is met trivially if a controller exists for which each configuration in a region can be sent to at least one configuration in each adjacent region. In robots with kinematic models or fully-actuated dynamics, region transitions can be guaranteed through standard motion planners based on potential functions [21] or vector fields [12].

For more general systems, we wish to define the continuous-domain specifications that controllers must meet in order to guarantee region transitions in A. For a given transition from q_i to q_j not necessarily distinct, we have a start set $S_{ij} \subseteq R(q_i)$, goal set $G_{ij} \subseteq R(q_j)$, and invariant $Inv_{ij} \subseteq R(q_i) \cup R(q_j)$ defining where it is necessary for trajectories to remain as progress is made from one region to another. Any such controller satisfying a given region transition is referred to as an *atomic* controller for the transition.

Denote a reach tube $\mathcal{L}_{ij}$ as the set of states in which the controlled system remains on $t \in [0, T_{ij}]$ for the transition (q_i, q_j). Also, define $I^i_{out} = \{k \in \mathbb{N} | (q, q_k) \in \Delta, q \in q_i\}$ as the index set of all successor states for state q_i (e.g. for state 2 in Fig. 1(b), $I^2_{out} = \{1, 3\}$), and let $I_{out} = \cup_i I^i_{out}$.

As defined by [22], for a given set of controllers to be *sequentially composable*, we require goal sets to be contained within the domain of successor reach tubes. If we let $\mathcal{L}_{ij}(t)$ denote the slice of $\mathcal{L}_{ij}$ at time t, then for $\mathcal{L}_{ij}$ to be sequential composable for each edge in Δ implies that $\mathcal{L}_{ij}(T_{ij}) \subseteq \mathcal{L}_{jk}$ for all $k \in I^j_{out}$. The shortfall of the sequential composition approach for temporal logic planning is that reach tubes only need be connected at their boundaries and hence the liveness and safety conditions in the specification may no longer be guaranteed. To illustrate this, recall the r_2–r_3 portion of the trajectory in Fig. 1(a) with $pursuer$ turns `False`. When $pursuer$ turns `True`, the robot must already be in a state where it may resume motion towards r_1 *without* first entering r_3 (a possible safety violation). With sequentially-composed controllers, there could be states along the segment where the system will inevitably enter r_3 as the environment changes. To prevent such behaviors, we introduce the notion of reactive composition.

Definition 1 (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 $(q_i, q_j) \in \Delta$, all points on the state trajectories $\mathbf{x} \in \mathcal{L}_{ij}$ also belong to $\bar{X}_i \cup \bar{X}_j$.*

Reactive composability requires that the continuous trajectories associated with a transition out of one state in

A lie in the subset of the state space where there exist valid trajectories for the other transitions out of that state. The objective of the algorithm in Section V is to generate controllers which satisfy the reactive composition property.

IV. Controller Synthesis and Verification

We adopt the Invariant Funnels method of [13] to build controllers based on a library of trajectories. Associated with each motion plan is a funnel (robust neighborhood) for which the property holds that trajectories starting within the funnel will remain within it for a finite time interval. Constructing funnels happens in two steps: controller generation and funnel computation. The first step is controller generation, where a locally-valid linear quadratic regulator (LQR) control law is adopted to minimize excursions from a sample trajectory. The second step is to construct funnels which characterize the domain within which convergence holds and the continuous-domain specifications are upheld. We present only the salient information on the Invariant Funnels method here. For a more comprehensive treatment on the method, the reader is referred to [13], [19].

Let us denote m as the index of a simulated trajectory connecting regions $\gamma_{\mathcal{R}}(q_i)$ and $\gamma_{\mathcal{R}}(q_j)$. The goal is to compute a set of funnels ℓ_{ij}^m and associated set of controllers $\mathbf{c}_{ij}^m$ defined within those funnels.

A. Motion Planning and Feedback Control

Atomic controllers consist of a collection of control laws each defined within its own funnel. An important prerequisite to creating a control law is a sample trajectory which drives the system towards the goal region G_{ij} from other points in the state space while keeping it within Inv_{ij}. While any number of techniques can be applied for trajectory generation, we adopt a feedback linearization technique to construct the continuous sample trajectories [23]. Once generated, these trajectories are recorded as a time history $\mathbf{t}_{ij}^m$, a trajectory of states $\mathbf{x}_{ij}^m$, and a trajectory of control inputs $u(t) \in \mathbf{u}_{ij}^m$ over $t \in [0, T_{ij}^m]$. For systems which are not feedback linearizable, it is possible to use nonlinear trajectory optimization methods [24].

Next, local controllers are constructed using the LQR design approach [25] to drive the system from any neighboring initial conditions towards the sample trajectory. The system is first linearized at discrete points about the trajectory, and a Riccati equation is then solved at each time instant, producing a time-varying state feedback control gain $K(t) \in \mathbf{K}_{ij}^m$. Together, $\mathbf{u}_{ij}^m$ and $\mathbf{K}_{ij}^m$ are stored in a controller library $\mathbf{c}_{ij}^m$. Our goal is now to find level sets $\rho_{ij}^m(t) \in \mathbb{R}$ of a quadratic Lyapunov function $V_{ij}^m(x,t)$ which define the local region of invariance of the dynamic system.

B. Invariant Funnels

Given the mth trajectory $(\mathbf{t}_{ij}^m, \mathbf{u}_{ij}^m, \mathbf{x}_{ij}^m)$, funnel computation proceeds by computing the level sets of these Lyapunov functions, $\ell_{ij}^m(t) = \{x | V_{ij}^m(x,t) \leq \rho_{ij}^m(t)\}$, representing the regions of the state space within which trajectories remain for $t \in [0, T_{ij}^m]$. A reach tube for the ijth region transition (denoted $\mathcal{L}_{ij}$) is defined in this paper as the union of all funnels $\ell_{ij}^m(t)$ associated with that transition. We modify the constraints in the objective presented in [19] by including the goal and invariant sets directly in our search for a maximal $\rho_{ij}^m(t)$:

$$\max \rho_{ij}^m(t), \qquad t \in [0, T_{ij}^m] \tag{1}$$

$$\text{s.t.} \quad \dot{V}_{ij}^m(x,t) \leq \dot{\rho}_{ij}^m(t), \qquad \forall t \in [0, T_{ij}^m], \ \forall x \in \{x | V_{ij}^m(x,t) = \rho_{ij}^m(t)\}, \tag{2}$$

$$\rho_{ij}^m(t) \geq 0, \qquad \forall t \in [0, T_{ij}^m], \tag{3}$$

$$\ell_{ij}^m(t) = \{x | V_{ij}^m(x,t) \leq \rho_{ij}^m(t)\} \subseteq Inv_{ij}, \qquad \forall t \in [0, T_{ij}^m], \tag{4}$$

$$\ell_{ij}^m(T_{ij}^m) = \{x | V_{ij}^m(x, T_{ij}^m) \leq \rho_{ij}^m(T_{ij}^m)\} \subseteq G_{ij} \tag{5}$$

Inequalities (2) and (3) follow directly from [19], enforcing trajectory invariance to the funnel and positive-semidefiniteness of the level set. We include the remaining equalities, (4) and (5), to ensure that level sets are bounded to start and remain within the invariant $Inv_{ij} \subset X$ and end in a goal set $G_{ij} \subset X$ for the current transition.

V. Atomic Controller Synthesis Algorithm

The main contribution of this paper is an algorithm which takes as its input a finite-state automaton and returns a library of atomic controllers that guarantee *reactive* execution of the automaton. Funnels are computed iteratively until either all possible configurations within each region are enclosed (to within a desired metric) by funnels or until it is determined that coverage is not possible, i.e. there does not exist a $\mathcal{L}_{ij}$ for some $(q_i, q_j) \in \Delta$. Associated with each funnel ℓ_{ij}^m is a control law $\mathbf{c}_{ij}^m$; both are stored in a library for later use at runtime.

A. Algorithm Description

1) Overview: The algorithm for constructing atomic controllers is given in Algorithm 1. The algorithm begins by calling `AutomTransitions` which extracts the set of all unique edges Δ from automaton A. The algorithm next computes reach tubes for each element in Δ. We remark that our algorithm operates on automaton *states* rather than workspace regions to reduce conservatism in finding valid reach tubes. This is because each workspace region is associated with more than one state with possibly several transitions, with each transition imposing a unique constraint on the reach tube. The algorithm terminates successfully if reach tubes are found for each edge. If not, then the reach tube computations are revised to ensure they are reactively composable in the sense of Definition 1. Reactive composibility is illustrated in Fig. 2(d), where the reach tubes exiting q_1 (region a) and entering q_2 and q_3 (resp. b and c) are contained completely within the larger dashed region.

We introduce two types of reach tubes to assist with constructing this set: those which invoke a transition between adjacent regions, $\mathcal{L}_{ij}$, called *transition* reach tubes, and those which are confined to a given region, $\mathcal{L}_i^c$, labeled *inward* reach tubes. The purpose of including inward reach tubes is to maximize coverage of the state space all regions of successor states are accessible.

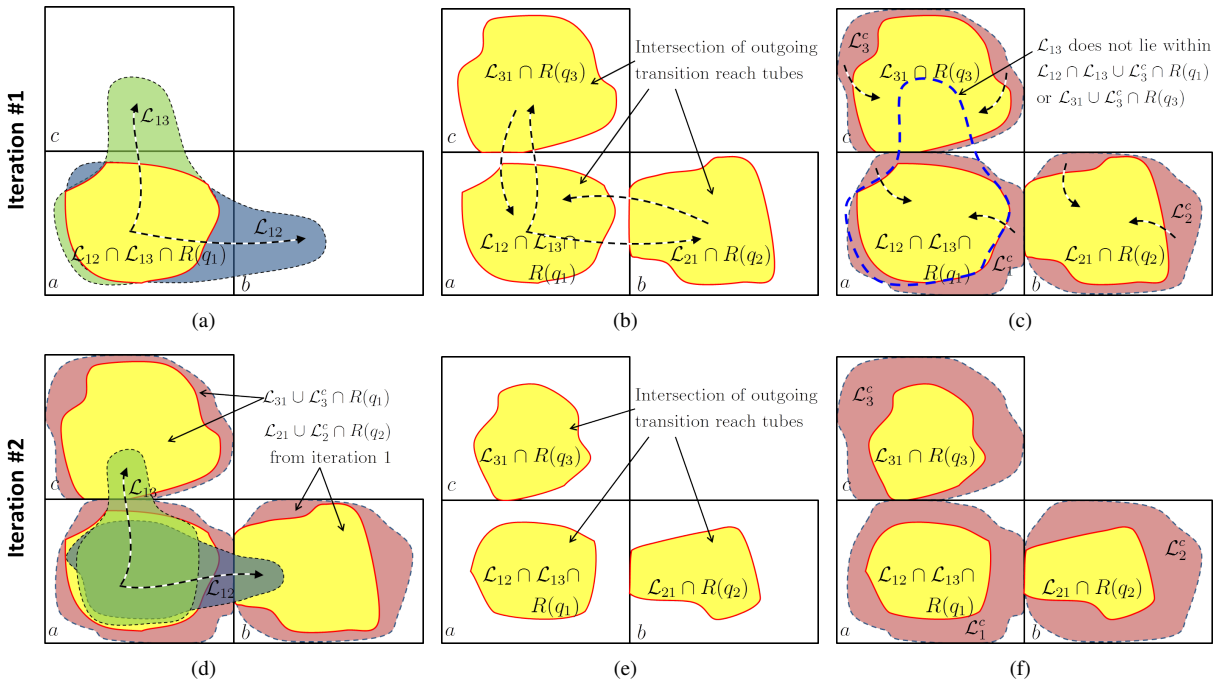


Fig. 2. Illustration of the reach tube computation steps, assuming two-way transitions between each adjoining region. For q_1, a pair of transition reach tubes $\mathcal{L}_{12}$ and $\mathcal{L}_{13}$ are computed in (a), the intersection of which (yellow) defines the new start set for the next iteration (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 $\left(\mathcal{L}_{12} \cap \mathcal{L}_{13} \cup \mathcal{L}_1^c\right) \cap R(q_1)$).

2) Computing $\mathcal{L}_{ij}$: The major steps of the algorithm are illustrated in Fig. 2. 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 initially taken 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. 2(a), 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). This intersection (see Fig. 2(b)) defines the set of states from which any region of successor states can be reached (by calling either $\mathcal{C}_{12}$ 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) Computing $\mathcal{L}_i^c$: In order to expand the size of reactively composable regions, lines 14–17 construct inward reach tubes for each region that will drive the robot to a configuration from which it can take a transition. The initial set S for $\mathcal{L}_i^c$, defined 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. 2(b)). In Fig. 2(c), the red regions enclosed by the dashed lines, $\mathcal{L}_i^c$, denote where controllers may be found which drive the system into the yellow region. Thus, if the transition reach tube is contained within the union of the red and yellow regions in Fig. 2(c), the reach tube is reactively composable in q_i.

4) Further Iterations: If, after one iteration, the sequentially-composable transition reach tubes constructed in the steps above are not also reactively composable, the process of finding $\mathcal{L}_{ij}$ and $\mathcal{L}_i^c$ must continue until they are. 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 1, by requiring that transition reach tubes must either lie within an inward reach tube or the sets where any successor state is reachable. This additional step is shown pictorially in the bottom row of Fig. 2. In this iteration, the sets S_{ij}, G_{ij}, and Inv_{ij} for the ijth edge are defined by $\mathcal{L}_{ij}$ and $\mathcal{L}_i^c$. In Fig. 2(d), lines 7–13 are once again repeated, and new transition reach tubes for a are computed ($\mathcal{L}_{12}$ and $\mathcal{L}_{13}$) which are constrained to remain within the red and yellow regions for q_1, q_2, and q_3. After the intersections are taken (yellow region in Fig. 2(e)), the reach tubes from the previous iteration are removed and a new set of inward reach tubes is computed in lines 14–17. Fig. 2(f) 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. Upon successful termination, the algorithm returns a library of funnels $\mathcal{L}$ along with a library of controllers $\mathcal{C}$ in lines 23–19.

5) Computing Reach Tubes: `GetReachTube` is iterated up to N times, with the following steps:

1) Pick a random initial point in the start region S
2) Pick a final point inside the goal set G, seeking the centroid of the region if G is a polygon and a random point if G is defined by reach tubes
3) Generate a feasible trajectory connecting the initial and final configurations

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

$(\Delta, I_{out}) \leftarrow$ `AutomTransitions`(A)
for $(q_i, q_j) \in \Delta$ **do**
  $\mathcal{L}_{ij} \leftarrow \mathbb{R}^n$
end
$\mathcal{L}_i^c \leftarrow \varnothing$, $\mathcal{C}_i^c \leftarrow \varnothing \ \forall q_i \in Q$
while `True` **do** `// Repeat until all reach tubes are reactively composable or until failure`
  for $(q_i, q_j) \in \Delta$ **do**
    $S \leftarrow \cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i)$
    $G \leftarrow \cap_{k \in I_{out}^j} \mathcal{L}_{jk} \cap R(q_j) \cup \mathcal{L}_j^c$
    $(\mathcal{L}_{ij}, \mathcal{C}_{ij}) \leftarrow$ `GetReachTube`$(S, G, S \cup \mathcal{L}_i^c \cup G, f, \epsilon, N)$
    if $\mathcal{L}_{ij} = \varnothing$ **then**
      return $\varnothing$ `// No controller exists`
    end
    $S \leftarrow R(q_i) \backslash \left(\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c \right)$
    $G \leftarrow \cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i)$
    $(\mathcal{L}_i^c, \mathcal{C}_i^c) \leftarrow$ `GetReachTube`$(S, G, R(q_i), f, \epsilon, N)$
  end
  if $\forall (q_i, q_j) \in \Delta$: $\left[(\mathcal{L}_{ij} \cap R(q_i)) \subseteq \left(\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cap R(q_i) \cup \mathcal{L}_i^c \right) \right] \wedge \left[(\mathcal{L}_{ij} \cap R(q_j)) \subseteq \left(\cap_{k \in I_{out}^j} \mathcal{L}_{jk} \cap R(q_j) \cup \mathcal{L}_j^c \right) \right]$ **then** `// All are reactively composable`
    $\mathcal{L} \leftarrow (\cup_{i,j} \mathcal{L}_{ij} \cup_i \mathcal{L}_i^c)$, $\mathcal{C} \leftarrow (\cup_{i,j} \mathcal{C}_{ij} \cup_i \mathcal{C}_i^c)$
    return $\mathcal{L}$, $\mathcal{C}$
  end
end

4) For any feasible trajectory, compute a funnel according to the procedure in Section IV
5) If feasible, append 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 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 the termination criterion. The former condition asserts that coverage terminates if the reach tube $\mathcal{L}$ covers a significant enough portion 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. Note that we are free to select a sufficiently large N for the sake of probabilistic completeness [13], however achieving good enough coverage depends in large part on the dimension of the state space.

One difficulty with implementing funnels as reach tubes is that coverage degenerates at the boundaries of polyhedral invariants due to the curvature arising from the ellipsoidal level sets, making it impossible for the algorithm to generate reach tubes spanning across regions. We work around this issue by relaxing the interface between neighboring regions by some fixed tolerance value d. 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.

B. Controller Execution

The controllers used to execute the motion plan associated with a discrete 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 containing 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, or (iii) an environment proposition changes. Priority is given to transition funnels $\mathcal{L}_{ij}$ over inward $\mathcal{L}_i^c$. If the robot is currently executing a funnel in $\mathcal{L}_i^c$ and it reaches a funnel in $\mathcal{L}_{ij}$, with r_j as the goal for that transition, the motion controller is switched accordingly. In example 1, consider the r_1–r_2 transition with $pursuer$ `False`. At the current time step, the robot is executing the reach tube $\mathcal{L}_{12}$ and has just reached r_2. The next goal (r_3) is implemented by switching to $\mathcal{L}_{23}$ if within that funnel. Otherwise, the planner will choose $\mathcal{L}_2^c$. To disambiguate between multiple funnel choices, one may be selected according to its ordering in the library.

VI. Examples

In this section, we demonstrate the application of the method developed in this paper to two examples. The model in Section II is adopted with the parameter settings $\omega_{min} = -3$, $\omega_{max} = 3$, and $v_{nom} = 2$.

A. Patrolling Two Regions

We address the case study in Example 1, whose $10m \times 9m$ workspace consists of the three regions arranged as shown in Fig. 1(a). The task specification is as follows: assuming a starting configuration in r_1, the robot must repeatedly visit r_1 and r_3. If a pursuer is sensed, the robot is to return immediately to r_1. We synthesize the specification as a high-level controller represented in Fig. 1(b).

A library of controllers is generated according to the algorithm in Section V, adopting sum-of-squares (SoS) programming [26] to solve (1)–(5). For this example, we set the coverage metric $\epsilon = 0.2$, the number of iterations $N = 100$, and the interface relaxation $d = 0.2m$. The computation took

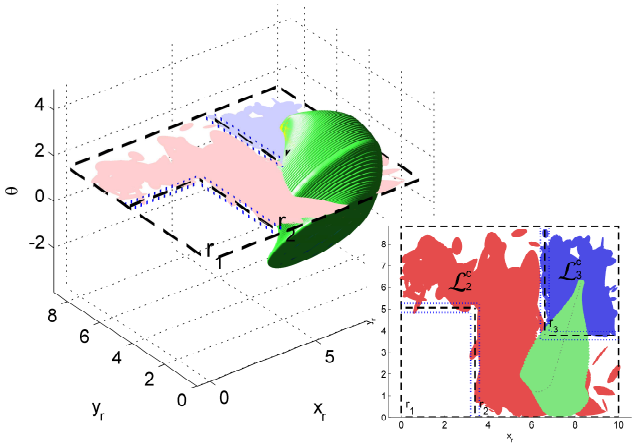


Fig. 3. A transition funnel and a slice of the inward reach tubes at $\theta = 1.46$ for the transition from r_2 to r_3 after the first iteration of Algorithm 1. The inset shows a 2-D view of the slice. In this iteration, the funnel is not reactively composable.

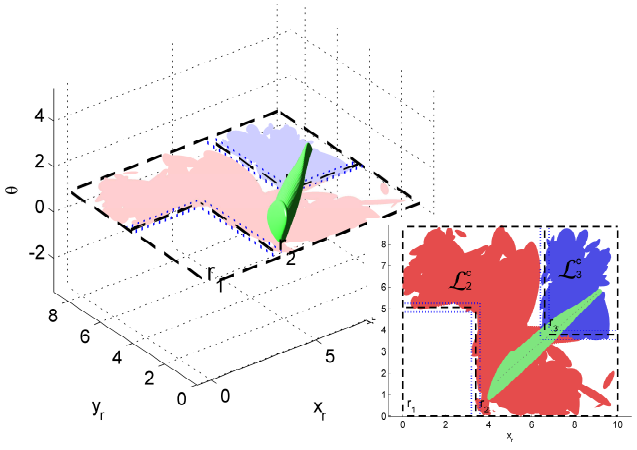


Fig. 4. A transition funnel and a slice of the inward reach tubes at $\theta = -0.41$ for the transition from r_2 to r_3 after the second iteration of Algorithm 1. The inset shows a 2-D view of the slice. The funnel is now reactively composable because it is now completely enclosed by the set of inward-facing funnels.

approximately 340 min. 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. 3 and 4. Also shown are the θ-slices of the inward reach tubes $\mathcal{L}_2^c$ and $\mathcal{L}_3^c$ (shown in red and blue). In the first iteration, the funnel spans a gap in the set of inward funnels and hence is not reactively composable. 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, respectively, 0.6018, 0.3388, 0.6507, and 0.3456 for q_1, q_2, q_3, and q_4. 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 $\cap_{k \in I_{out}^i} \mathcal{L}_{ik} \cup \mathcal{L}_i^c$.

Fig. 5 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

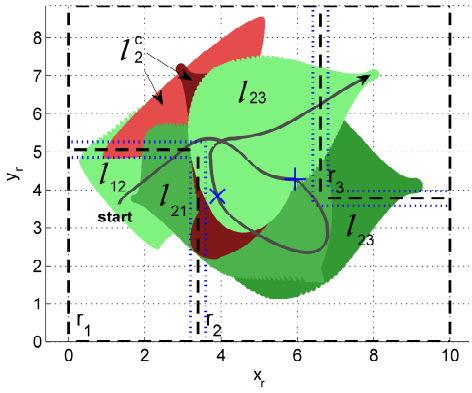


Fig. 5. Closed-loop trajectory generated from an initial state in r_1. A set of control laws are applied to implement the transitions (r_1, r_2) and (r_2, r_3), driving the robot from state 1 to state 2, then from state 2 to state 3 in Fig. 1(b). 2-D projections of the active funnels are also shown, 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 "×" location, and new control laws are used to resume the transition (r_2, r_3).

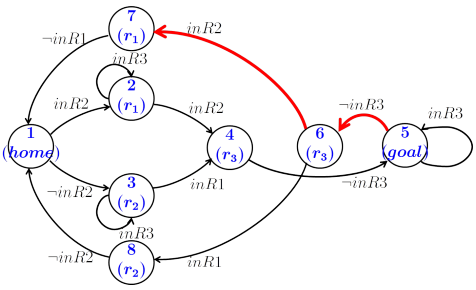


Fig. 6. Discrete automaton for the pursuit-evasion example.

funnels when making transitions between regions, even when reacting to the environment.

B. Pursuit-Evasion

In this example, the robot is engaged in a game where the robot must visit the home and goal regions in Fig. 7, while evading a pursuer which visits each of the three remaining regions infinitely often. Evasion is encoded by the requirement that the robot should always remain out of the region occupied by the pursuer. As a fairness condition for LTL synthesis, in the specification we assume also that the pursuer cannot occupy the same region as the robot. The high-level controller (Fig. 6) consists of eight states and 13 edges. The sensor values $inR1$, etc. correspond to the observed pursuer location among the three possible regions.

Reach tubes are constructed for each of the 13 transitions. In this example, the computation took approximately 660 min. A subset of these (the highlighted edges in Fig. 6) are shown in Fig. 7, 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 there are two possible transitions out of r_3 (r_1 and r_2) depending on the location of the pursuer. To the left of the goal region, the robot is easily able to toggle between the goals r_1 and r_2 as the pursuer

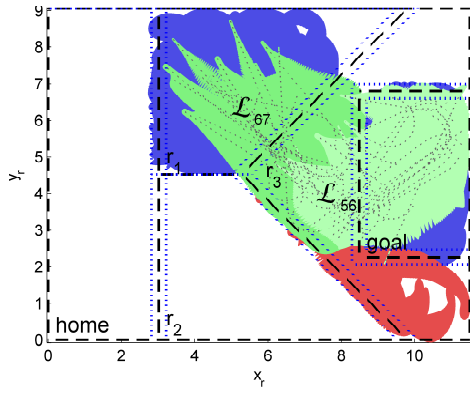


Fig. 7. Transition funnels for $\mathcal{L}_{56}$ and $\mathcal{L}_{67}$ (green). $\mathcal{L}_5^c$ and $\mathcal{L}_7^c$ are shaded red and $\mathcal{L}_6^c$ is shaded blue.

toggles between regions. Without reactive composition, a motion planner may deliver the robot above $goal$, where there may exist controllers which can deliver the robot to r_1 (when $inR2$ is `True`), but not r_2 (when $inR1$ is `True`). If $inR1$ remains `True` long enough, the robot may have no other option but to enter r_1 (because there are no funnels available to deliver it to r_2), resulting in a safety violation of the original specification.

VII. Conclusion

In this paper, a method is presented for synthesizing controllers in the continuous domain based on a discrete controller derived from temporal logic specifications. The central contribution of this paper is an algorithm that generates controllers guaranteeing every possible (reactive) execution of a discrete automaton for robots with nonlinear dynamics. In the future, our strategy will be extended as in [19] to deal with bounded disturbances in the continuous space.

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.

VIII. Acknowledgment

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

References

[1] M. Kloetzer and C. 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, (Berlin, Heidelberg), pp. 333–347, Springer-Verlag, 2006.

[2] S. G. Loizou and K. 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)*, pp. 153–158, 2004.

[3] T. Wongpiromsarn, U. Topcu, and R. 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.

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

[5] 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)*, pp. 2222–2229, 2009.

[6] A. Bhatia, L. Kavraki, and M. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation (ICRA 2010)*, pp. 2689–2696, IEEE, 2010.

[7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[8] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray, "Towards formal synthesis of reactive controllers for dexterous robotic manipulation," in *IEEE International Conference on Robotics and Automation (ICRA 2012)*, pp. 5183–5189, 2012.

[9] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE International Conference on Robotics and Automation (ICRA 2011)*, pp. 1470–1477, 2011.

[10] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the darpa urban challenge," in *IEEE Conference on Automation Science and Engineering*, (Washington D.C., USA), 2008.

[11] D. C. Conner, H. Choset, and A. A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," in *Robotics: Science and Systems*, 2006.

[12] C. Belta and L. Habets, "Constructing decidable hybrid systems with velocity bounds," in *Proc. of the 43rd IEEE Conf. on Decision and Control (CDC 2004)*, (Bahamas), pp. 467–472, 2004.

[13] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *I. J. Robotic Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[14] J. Ding, J. Gillula, H. Huang, M. P. Vitus, W. Zhang, and C. J. Tomlin, "Hybrid systems in robotics: Toward reachability-based controller design," *IEEE Robotics & Automation Magazine*, vol. 18, pp. 33 – 43, Sept. 2011.

[15] M. Zamani, G. Pola, M. Mazo, and P. Tabuada, "Symbolic models for nonlinear control systems without stability assumptions," *IEEE Trans. Automat. Contr.*, vol. 57, no. 7, pp. 1804–1809, 2012.

[16] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Proc. of the 4th Int. Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, pp. 477–492, 2004.

[17] G. Frehse, "Phaver: algorithmic verification of hybrid systems past hytech," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, pp. 263–279, May 2008.

[18] G. E. Fainekos, S. G. Loizou, and G. J. Pappas, "Translating temporal logic to controller specifications," in *Proc. of the 45th IEEE Conf. on Decision and Control (CDC 2006)*, pp. 899–904, 2006.

[19] A. Majumdar and R. Tedrake, "Robust online motion planning with regions of finite time invariance," in *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, 2012.

[20] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

[21] D. C. Conner, A. Rizzi, and H. 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)*, vol. 4, pp. 3546– 3551, IEEE, October 2003.

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

[23] G. Oriolo, A. De Luca, and M. Vendittelli, "Wmr control via dynamic feedback linearization: design, implementation, and experimental validation," *Control Systems Technology, IEEE Transactions on*, vol. 10, pp. 835–852, Nov. 2002.

[24] J. T. Betts, *Practical methods for optimal control using nonlinear programming*, vol. 3 of *Advances in Design and Control*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2001.

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

[26] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo, *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2004.