

# Temporal Logic Motion Planning for Dynamic Robots

Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas

Technical Report MS-CIS-07-02

Department of Computer and Information Science, Univ. of Pennsylvania.

Last update : June 13, 2007

## Abstract

In this paper, we address the temporal logic motion planning problem for point robots that are modeled by second order dynamics. Temporal logic specifications can capture the usual control specifications such as reachability and invariance as well as more complex specifications like sequencing and obstacle avoidance. In order to solve this problem, we follow a hierarchical approach that enables the control of the second order system by designing control laws for a fully actuated kinematic model. Our approach consists of three basic steps. First, we design a control law that enables the dynamic model to track a simpler kinematic model with a globally bounded error. Second, we built a robust temporal logic specification that takes into account the tracking errors of the first step. Finally, we solve the new robust temporal logic path planning problem for the kinematic model using automata theory and simple local vector fields. The resulting continuous time trajectory is provably guaranteed to satisfy the initial user specification.

## Index Terms

Motion Planning, Temporal Logic, Robustness, Hybrid Systems, Hierarchical Control.

## I. INTRODUCTION

One of the main challenges in robotics is the development of mathematical frameworks that formally and verifiably integrate high level planning with continuous control primitives. Traditionally, the path planning problem for mobile robots has considered reachability specifications of the form “move from the Initial position  $I$  to the Goal position  $G$  while staying within region  $R$ ”. The solutions to this well-studied problem span a wide variety of methods, from continuous (like potential or navigation functions [1, §4]) to discrete (like Canny’s algorithm, Voronoi diagrams, cell decompositions and probabilistic road maps [1], [2]).

This research is partially supported by NSF EHS 0311123, NSF ITR 0121431 and ARO MURI DAAD 19-02-01-0383.

Georgios E. Fainekos, Hadas Kress-Gazit and George J. Pappas are with the GRASP Laboratory at the University of Pennsylvania. E-mail: {fainekos, hadaskg, pappasg}@grasp.upenn.edu

Antoine Girard is with the Laboratoire de Modélisation et Calcul at the Université Joseph Fourier. E-mail: antoine.girard@imag.fr

Whereas these methods solve the basic path planning problem, they do not address high level planning issues that arise when one considers a number of goals or a particular ordering of them. In order to manage such constraints, one should either employ an existing high level planning method [3] or attack the problem using optimization techniques like mixed integer linear programming [4]. Even though the aforementioned methods can handle partial ordering of goals, they cannot deal with temporally extended goals. For such specifications, planning techniques [5], [6] that are based on model checking [7] seem to be a more natural choice. Using temporally extended goals, one would sacrifice some of the efficiency of the standard planning methods for expressiveness in the specifications. Temporal logics such as the Linear Temporal Logic (LTL) [8] and its continuous time version propositional temporal logic over the reals (RTL) [9] have the expressive power to describe a conditional sequencing of goals under a well defined formal framework. Such a formal framework can provide us with the tools for automated controller synthesis and code generation. Beyond the provably correct synthesis of hybrid controllers for path planning from high level specifications, temporal logics have one more potential advantage when compared to other formalisms, e.g. regular languages [10]. That is to say, temporal logics were designed to bear a resemblance to natural language [11, §2.5]. Moreover, one can develop computational interfaces between natural language and temporal logics [12].

In our previous work [13], [14], we have combined such planning frameworks with local controllers defined over convex cells [15], [16] in order to perform temporal logic motion planning for a first order model of a robot. However, in certain cases a kinematics model is not enough, necessitating thus the development of a framework that can handle a dynamics model. In this paper, we provide a tractable solution to the RTL motion planning problem for dynamics models.

In order to solve this problem, we take a hierarchical approach. A hierarchical control system consists of two layers. The first layer consists of a coarse and simple model of the plant (the kinematics model in our case). A controller is designed so that this abstraction meets the specification of the problem. The control law is then refined to the second layer, which in this paper consists of the dynamics model. Architectures of hierarchical controllers based on the notion of simulation relation have been proposed in [17], [18]. More recently, it has been claimed that approaches based on the notion of approximate simulation relations [19] would provide more robust control laws while allowing to consider simpler discrete [20] or continuous [21] abstractions for control synthesis.

Following [21], we first abstract the system to a kinematic model. An interface is designed so that the system is able to track the trajectories of its abstraction with a given guaranteed precision  $\delta$ . The control objective  $\phi$ , which is provided by the user in RTL, is then modified and replaced by a more robust specification also expressed as an RTL formula  $\phi'$ . The formula  $\phi'$  is such that given a trajectory satisfying  $\phi'$ , any trajectory remaining within distance  $\delta$  satisfies  $\phi$ . It then remains to design a controller for the abstraction such that all its trajectories satisfy the robust specification  $\phi'$ . This is achieved by first lifting the problem to the discrete level by partitioning the environment into a finite number of equivalence classes. A variety of partitions are applicable [1], [2], but we focus on triangular decompositions [22] and general convex decompositions [23] as these have been successfully applied to the basic path planning problem in [15] and [16] respectively. The partition results in a natural discrete abstraction of the robot motion which is used then for planning with automata theoretic methods [5]. The resulting discrete plan acts

as supervisory controller that guides the composition of local vector fields [15], [16] which generate the desired motion. Finally using the hierarchical control architecture, we lift the hybrid controller  $H'_{\phi'}$ , which guarantees the generation of trajectories for the kinematics model that satisfy  $\phi'$ , to a hybrid controller  $H_{\phi}$  which now generates trajectories that satisfy the original specification  $\phi$ .

## II. PROBLEM DESCRIPTION

We consider a mobile robot which is modeled by a second order system  $\Sigma$  of the form

$$\ddot{x}(t) = u(t), \quad x(t) \in X, x(0) \in X_0, u(t) \in U \quad (1)$$

where  $x(t) \in X$  is the position of the robot in the plane,  $X \subseteq \mathbb{R}^2$  is the free workspace of the robot,  $X_0 \subseteq X$  is the set of initial positions. Here, we assume that initially the robot is at rest, i.e.  $\dot{x}(0) = 0$  and that  $U = \{u \in \mathbb{R}^2 \mid \|u\| \leq \mu\}$  where  $\mu \in \mathbb{R}_{>0}$  models the constraints on the control input and  $\|\cdot\|$  is the Euclidean norm. The goal of this paper is to construct a hybrid controller that generates control inputs  $u(t)$  for system  $\Sigma$  so that for the set of initial states  $X_0$ , the resulting motion  $x(t)$  satisfies a formula-specification  $\phi$  in the propositional temporal logic over the reals [9]. Following [24], we refer to this logic as RTL.

For the high level planning problem, we consider the existence of a number of regions of interest to the user. Such regions could be rooms and corridors in an indoor environment or areas to be surveyed in an outdoor environment. Let  $\Pi = \{\pi_0, \pi_1, \dots, \pi_n\}$  be a finite set of symbols that label these areas. The denotation  $\llbracket \cdot \rrbracket : \Pi \rightarrow \mathcal{P}(X)$  of each symbol in  $\Pi$  represents a subset of  $X$ , i.e. for any  $\pi \in \Pi$  it is  $\llbracket \pi \rrbracket \subseteq X$ . Here,  $\mathcal{P}(I)$  denotes the powerset of a set  $I$ . We reserve the symbol  $\pi_0$  to model the free workspace of the robot, i.e.  $\llbracket \pi_0 \rrbracket = X$ .

In order to make apparent the use of RTL for the composition of motion planning specifications, we first give an informal description of the traditional and temporal operators. The formal syntax and semantics of RTL are presented in Section III. RTL formulas are built over a set of propositions, the set  $\Pi$  in our case, using combinations of the traditional and temporal operators. Traditional logic operators are the *conjunction* ( $\wedge$ ), *disjunction* ( $\vee$ ) and *negation* ( $\neg$ ). Some of the temporal operators are *eventually* ( $\diamond$ ), *always* ( $\square$ ), *until* ( $\mathcal{U}$ ) and *release* ( $\mathcal{R}$ ). The propositional temporal logic over the reals can describe the usual properties of interest for control problems, i.e. *reachability* ( $\diamond\pi$ ) and *safety*: ( $\square\pi$  or  $\square\neg\pi$ ). Beyond the usual properties, RTL can capture sequences of events and infinite behaviours. For example:

- **Reachability while avoiding regions:** The formula  $\neg(\pi_1 \vee \pi_2 \vee \dots \vee \pi_n) \mathcal{U} \pi_{n+1}$  expresses the property that eventually  $\pi_{n+1}$  will be true, and until  $\llbracket \pi_{n+1} \rrbracket$  is reached, we must avoid all unsafe sets  $\llbracket \pi_i \rrbracket$ ,  $i = 1, \dots, n$ .
- **Sequencing:** The requirement that we must visit  $\llbracket \pi_1 \rrbracket$ ,  $\llbracket \pi_2 \rrbracket$  and  $\llbracket \pi_3 \rrbracket$  in that order is naturally captured by the formula  $\diamond(\pi_1 \wedge \diamond(\pi_2 \wedge \diamond\pi_3))$ .
- **Coverage:** Formula  $\diamond\pi_1 \wedge \diamond\pi_2 \wedge \dots \wedge \diamond\pi_m$  reads as the system will eventually reach  $\llbracket \pi_1 \rrbracket$  and eventually  $\llbracket \pi_2 \rrbracket$  and ... eventually  $\llbracket \pi_m \rrbracket$ , requiring the system to eventually visit all regions of interest without imposing any ordering.

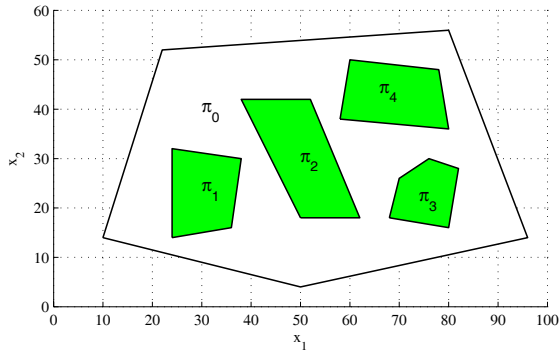


Fig. 1. The simple environment of Example 1. The four regions of interest  $\pi_1, \pi_2, \pi_3, \pi_4$  are enclosed by the polygonal region labeled by  $\pi_0$ .

- **Recurrence (Liveness):** The formula  $\Box(\Diamond\pi_1 \wedge \Diamond\pi_2 \wedge \dots \wedge \Diamond\pi_m)$  requires that the trajectory does whatever the coverage does and, in addition, will force the system to repeat the desired objective infinitely often.

More complicated specifications can be composed from the basic specifications using the logic operators. In order to better explain the different steps in our framework, we consider throughout this paper the following example.

*Example 1:* Consider a robot that is moving in a convex polygonal environment  $\pi_0$  with four areas of interest denoted by  $\pi_1, \pi_2, \pi_3, \pi_4$  (see Fig. 1). Initially, the robot is placed somewhere in the region labeled by  $\pi_1$  and its velocity is set to zero. The robot must accomplish the following task : “Visit area  $[\pi_2]$ , then area  $[\pi_3]$ , then area  $[\pi_4]$  and, finally, return to and stay in region  $[\pi_1]$  while avoiding areas  $[\pi_2]$  and  $[\pi_3]$ ”. Also, it is implied that the robot should always remain inside the free workspace  $X$ , i.e. region  $[\pi_0]$ , and that  $X_0 = [\pi_1]$ .

In this paper, for such spatio-temporal specifications, we provide a computational solution to the following problem.

*Problem 1:* Given a system  $\Sigma$  and an RTL formula  $\phi$ , construct a hybrid controller  $H_\phi$  for  $\Sigma$  such that the trajectories of the closed-loop system satisfy formula  $\phi$ .

We propose a hierarchical synthesis approach which consists of three ingredients : tracking control using approximate simulation relations [21], robust satisfaction of RTL formulas [25] and hybrid control for motion planning [13], [14]. First,  $\Sigma$  is abstracted to a first order fully actuated system  $\Sigma'$ :

$$\dot{z}(t) = v(t), z(t) \in Z, z(0) \in Z_0, v(t) \in V \quad (2)$$

where  $z(t) \in Z$  is the position of the kinematic model of the robot,  $Z \subseteq \mathbb{R}^2$  is a modified free workspace,  $Z_0 = X_0$  is the set of possible initial positions and  $V = \{v \in \mathbb{R}^2 \mid \|v\| \leq \nu\}$  for some  $\nu \in \mathbb{R}_{>0}$  is the set of control input values. Using the notion of approximate simulation relation, we evaluate the precision  $\delta$  with which the system  $\Sigma$  is able to track the trajectories of the abstraction  $\Sigma'$  and design a continuous tracking controller that we call *interface*. Secondly, from the RTL formula  $\phi$  and the precision  $\delta$ , we derive a more robust formula  $\phi'$  such that if a trajectory  $z$  satisfies  $\phi'$ , then any trajectory  $x$  remaining at time  $t$  within distance  $\delta$  from  $z(t)$  satisfies the formula  $\phi$ . Thirdly, we design a hybrid controller  $H'_{\phi'}$  for the abstraction  $\Sigma'$ , so that the trajectories of the closed loop

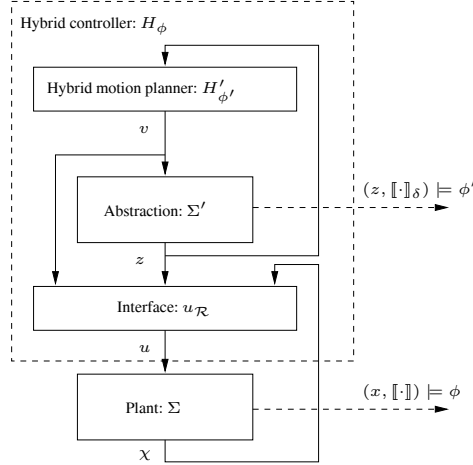


Fig. 2. Hierarchical architecture of the hybrid controller  $H_\phi$ .

system satisfy the formula  $\phi'$ . Finally, by putting these three ingredients together, as shown in Fig. 2, we design a hybrid controller  $H_\phi$  solving Problem 1. In the following, we detail each step of our approach.

### III. PROPOSITIONAL TEMPORAL LOGIC OVER THE REALS

Physical processes, such as the motion of a robot, evolve in continuous time. As such, it is more intuitive for the user to state the desired robotic behavior using temporal logics with underlying continuous time semantics [9], [24] instead of discrete [8]. In this paper, we advocate the applicability of the propositional temporal logic over the reals with the until connective (RTL) [9], [24] as a natural formalism for a motion planning specification language. RTL has the same temporal connectives as the Linear Temporal Logic (LTL) [8], but now the underlying time line is the positive real line instead of the natural numbers.

First, we introduce the syntax of RTL formulas in Negation Normal Form (NNF). In NNF, we push the negations inside the subformulas such that the only allowed negation operators appear in front of propositions. In this paper, as opposed to [13], [14], we use temporal logic in NNF since our “robustification” procedure in Section V cannot be applied to the negation of arbitrary formulas, but only to the negation of atomic propositions.

*Definition 1 (RTL Syntax in NNF):* For  $\pi \in \Pi$ , the set  $\Phi_\Pi$  of all well formed RTL formulas over  $\Pi$  in Negation Normal Form is constructed using the grammar

$$\phi ::= \pi \mid \neg\pi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{R} \phi$$

As usual, the boolean constants  $\top$  (true) and  $\perp$  (false) are defined as  $\top = \pi \vee \neg\pi$  and  $\perp = \pi \wedge \neg\pi$  respectively.

Formally, the semantics of RTL formulas is defined over continuous time boolean signals. Here, we instantiate the definitions of the semantics over abstractions of the trajectories of the system  $\Sigma$  with respect to the sets  $\llbracket \pi \rrbracket$  for all  $\pi \in \Pi$ . Let  $(x, \llbracket \cdot \rrbracket) \models \phi$  to denote the satisfaction of the RTL formula  $\phi$  over the output trajectory  $x$  starting at time 0 with respect to the proposition mapping  $\llbracket \cdot \rrbracket$ . If all the trajectories  $x$  of the system  $\Sigma$  driven by a controller

$H_\phi$  and associated to an initial state in  $X_0$  are such that  $(x, \llbracket \cdot \rrbracket) \models \phi$ , then we write  $(\llbracket \Sigma, H_\phi \rrbracket, \llbracket \cdot \rrbracket) \models \phi$  and we say that  $\llbracket \Sigma, H_\phi \rrbracket$  satisfies  $\phi$ . In the following, given any function  $f$  from some time domain  $\mathbb{T}$  (i.e.  $\mathbb{R}$  or  $\mathbb{N}$ ) to some set  $\mathbb{X}$ , we define  $f|_t$  for  $t \in \mathbb{T}$  to be the  $t$  time shift of  $f$  with definition  $f|_t(s) = f(t+s)$  for  $s \in \mathbb{T}$ .

*Definition 2 (RTL Semantics):* Let  $x$  be a trajectory of  $\Sigma$  and  $\Pi$  be the set of propositions. For  $t, s \in \mathbb{R}_{\geq 0}$ , the semantics of any formula  $\phi \in \Phi_\Pi$  can be recursively defined as:

$$(x, \llbracket \cdot \rrbracket) \models \pi \text{ iff } x(0) \in \llbracket \pi \rrbracket$$

$$(x, \llbracket \cdot \rrbracket) \models \neg\pi \text{ iff } x(0) \notin \llbracket \pi \rrbracket$$

$$(x, \llbracket \cdot \rrbracket) \models \phi_1 \vee \phi_2 \text{ if } (x, \llbracket \cdot \rrbracket) \models \phi_1 \text{ or } (x, \llbracket \cdot \rrbracket) \models \phi_2$$

$$(x, \llbracket \cdot \rrbracket) \models \phi_1 \wedge \phi_2 \text{ if } (x, \llbracket \cdot \rrbracket) \models \phi_1 \text{ and } (x, \llbracket \cdot \rrbracket) \models \phi_2$$

$$(x, \llbracket \cdot \rrbracket) \models \phi_1 \mathcal{U} \phi_2 \text{ if exists } t \geq 0 \text{ such that } (x|_t, \llbracket \cdot \rrbracket) \models \phi_2 \text{ and for all } s \text{ with } 0 \leq s < t \text{ we have } (x|_s, \llbracket \cdot \rrbracket) \models \phi_1$$

$$(x, \llbracket \cdot \rrbracket) \models \phi_1 \mathcal{R} \phi_2 \text{ if for all } t \geq 0 \text{ we have } (x|_t, \llbracket \cdot \rrbracket) \models \phi_2 \text{ or there exists } s \text{ such that } 0 \leq s < t \text{ and } (x|_s, \llbracket \cdot \rrbracket) \models \phi_1$$

Therefore, the formula  $\phi_1 \mathcal{U} \phi_2$  intuitively expresses the property that over the trajectory  $x$ ,  $\phi_1$  is true until  $\phi_2$  becomes true. The release operator  $\phi_1 \mathcal{R} \phi_2$  states that  $\phi_2$  should always hold, a requirement which is released when  $\phi_1$  becomes true. Furthermore, we can derive additional temporal operators such as *eventually*  $\diamond\phi = \top \mathcal{U} \phi$  and *always*  $\square\phi = \perp \mathcal{R} \phi$ . The formula  $\diamond\phi$  indicates that over the trajectory  $x$  the subformula  $\phi$  becomes eventually true, whereas  $\square\phi$  indicates that  $\phi$  is always true over  $x$ .

Note that due to the definition of the negation operator, the duality property of the logic holds and, thus, by using RTL in NNF we do not loose in expressive power. To see that notice that  $x(t) \in \llbracket \pi \rrbracket$  iff  $x(t) \notin \overline{\llbracket \pi \rrbracket}$ . Here,  $\overline{\llbracket \pi \rrbracket}$  denotes the complement of  $\llbracket \pi \rrbracket$ .

*Example 2:* Going back to Example 1, we can now formally define the specification using an RTL formula:

$$\square\pi_0 \wedge \diamond(\pi_2 \wedge \diamond(\pi_3 \wedge \diamond(\pi_4 \wedge (\neg\pi_2 \wedge \neg\pi_3) \mathcal{U} \square\pi_1))) \quad (3)$$

One important assumption which we need to make when we write specifications for physical processes is that the trajectories must satisfy the property of *finite variability* [26]. The finite variability property requires that within a finite amount of time there cannot be an infinite number of changes in the satisfaction of the propositions with respect to the trajectory. In other words, we should not consider Zeno trajectories [27]. We address this issue in the design of our hybrid controllers in Section VI-C.

#### IV. TRACKING USING APPROXIMATE SIMULATION

In this section, we present a framework for tracking control with guaranteed error bounds. It allows us to design an interface between the dynamics model  $\Sigma$  and its kinematic abstraction  $\Sigma'$  so that  $\Sigma$  is able to track the trajectories of  $\Sigma'$  with a given precision. It is based on the notion of approximate simulation relation [19]. Whereas exact simulation relations require the observations, i.e.  $x(t)$  and  $z(t)$ , of two systems to be identical, approximate simulation relations allow them to be different provided their distance remains bounded by some parameter.

Let us first rewrite the 2nd order model  $\Sigma$  as a system of 1st order differential equations.

$$\Sigma : \begin{cases} \dot{x}(t) = y(t), & x(t) \in X, x(0) \in X_0 \\ \dot{y}(t) = u(t), & y(t) \in \mathbb{R}^2, y(0) = [0 \ 0]^T \end{cases}$$

or if we let  $\chi = [x^T \ y^T]^T$  and  $\chi(0) \in \mathcal{X}_0$  such that  $x(0) \in X_0$  and  $y(0) = [0 \ 0]^T$ , then

$$\dot{\chi} = A\chi + Bu \quad \text{and} \quad x = C_x\chi, \ y = C_y\chi$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad C_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad C_y = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Then, the approximate simulation relation is defined as follows.

*Definition 3 (Simulation Relation):* A relation  $\mathcal{W} \subseteq \mathbb{R}^2 \times \mathbb{R}^4$  is an approximate simulation relation of precision  $\delta$  of  $\Sigma'$  by  $\Sigma$  if for all  $(z_0, \chi_0) \in \mathcal{W}$ ,

- 1)  $\|z_0 - C_x\chi_0\| \leq \delta$
- 2) For all state trajectories  $z$  of  $\Sigma'$  such that  $z(0) = z_0$  there exists a state trajectory  $\chi$  of  $\Sigma$  such that  $\chi(0) = \chi_0$  and satisfying  $\forall t \geq 0, (z(t), \chi(t)) \in \mathcal{W}$ .

Let us remark that for  $\delta = 0$ , we recover the notion of exact simulation relation as defined in [28], [29]. An interface associated to the approximate simulation relation  $\mathcal{W}$  allows to choose the input of  $\Sigma$  so that the states of  $\Sigma'$  and  $\Sigma$  remain in  $\mathcal{W}$ .

*Definition 4 (Interface):* A continuous function  $u_{\mathcal{W}} : V \times \mathcal{W} \rightarrow U$  is an interface associated with an approximate simulation relation  $\mathcal{W}$ , if for all  $(z_0, \chi_0) \in \mathcal{W}$ , for all trajectories  $z$  of  $\Sigma'$  associated with input  $v$  and such that  $z(0) = z_0$ , the trajectory  $\chi$  of  $\Sigma$  starting at  $\chi(0) = \chi_0$  given by

$$\dot{\chi}(t) = A\chi(t) + Bu_{\mathcal{W}}(v(t), z(t), \chi(t)) \quad (4)$$

satisfies for all  $t \geq 0, (z(t), \chi(t)) \in \mathcal{W}$ .

Thus, by interconnecting  $\Sigma$  and  $\Sigma'$  through the interface  $u_{\mathcal{W}}$  as shown on Fig. 2, the system  $\Sigma$  tracks the trajectories of the abstraction  $\Sigma'$  with precision  $\delta$ .

*Proposition 1:* Let  $\chi_0 \in \mathcal{X}_0, z_0 = C_x\chi_0 \in Z_0$  such that  $(z_0, \chi_0) \in \mathcal{W}$ , then for all trajectories  $z$  of  $\Sigma'$  associated with input  $v$  and initial state  $z_0$ , the trajectory  $\chi$  of  $\Sigma$  given by (4) for  $\chi(0) = \chi_0$ , satisfies for all  $t \geq 0, \|C_x\chi(t) - z(t)\| \leq \delta$ .

*Proof:* From Definition 4, we have that for all  $t \geq 0, (z(t), \chi(t)) \in \mathcal{W}$ . Then, from Definition 3, for all  $t \geq 0, \|C_x\chi(t) - z(t)\| \leq \delta$ . ■

Let us remark that the choice of the initial state  $z_0$  of the abstraction  $\Sigma'$  is not independent of the initial state  $\chi_0$  of the system  $\Sigma$  ( $z_0 = C_x\chi_0$ ).

*Remark 1:* Usual hierarchical control approaches assume that the plant  $\Sigma$  is simulated by its abstraction  $\Sigma'$ . In this paper, the contrary is assumed. The abstraction  $\Sigma'$  is (approximately) simulated by the plant  $\Sigma$ : the approximate simulation relation is used as a tool for tracking controller design.

The construction of approximate simulation relations can be done effectively using a class of functions called simulation functions [19]. Essentially, a simulation function of  $\Sigma'$  by  $\Sigma$  is a positive function bounding the distance between the observations and non-increasing under the parallel evolution of the systems.

*Definition 5 (Simulation Function):* Let  $\mathcal{V} : \mathbb{R}^2 \times \mathbb{R}^4 \rightarrow \mathbb{R}_{\geq 0}$  be a continuous and piecewise differentiable function. Let  $u_{\mathcal{V}} : V \times \mathbb{R}^2 \times \mathbb{R}^4 \rightarrow \mathbb{R}^2$  be a continuous function.  $\mathcal{V}$  is a simulation function of  $\Sigma'$  by  $\Sigma$ , and  $u_{\mathcal{V}}$  is an associated interface if for all  $(z, \chi) \in \mathbb{R}^2 \times \mathbb{R}^4$ , the following two inequalities hold

$$\mathcal{V}(z, \chi) \geq \|z - C_x \chi\|^2 \quad (5)$$

$$\sup_{v \in V} \left( \frac{\partial \mathcal{V}(z, \chi)}{\partial z} v + \frac{\partial \mathcal{V}(z, \chi)}{\partial \chi} (A\chi + B u_{\mathcal{V}}(v, z, \chi)) \right) \leq 0 \quad (6)$$

Then, approximate simulation relations can be defined as level sets of the simulation function.

*Theorem 1:* Let the relation  $\mathcal{W} \subseteq \mathbb{R}^2 \times \mathbb{R}^4$  be given by

$$\mathcal{W} = \{(z, \chi) \mid \mathcal{V}(z, \chi) \leq \delta^2\}.$$

If for all  $v \in V$ , for all  $(z, \chi) \in \mathcal{W}$ , we have  $u_{\mathcal{V}}(v, z, \chi) \in U$ , then  $\mathcal{W}$  is an approximate simulation relation of precision  $\delta$  of  $\Sigma'$  by  $\Sigma$  and  $u_{\mathcal{W}} : V \times \mathcal{W} \rightarrow U$  given by  $u_{\mathcal{W}}(v, z, \chi) = u_{\mathcal{V}}(v, z, \chi)$  is an associated interface.

*Proof:* Let  $(z, \chi) \in \mathcal{W}$ , then from (5),

$$\|z - C_x \chi\| \leq \sqrt{\mathcal{V}(z, \chi)} \leq \delta.$$

Let  $z$  be a trajectory of  $\Sigma'$  associated to input  $v$  and such that  $z(0) = z$ . Let  $\chi$  starting at  $\chi(0) = \chi$  be given by

$$\dot{\chi}(t) = A\chi(t) + B u_{\mathcal{V}}(v(t), z(t), \chi(t)).$$

From equation (6), we have that  $d\mathcal{V}(z(t), \chi(t))/dt \leq 0$ . Therefore, for all  $t \geq 0$ ,  $(z(t), \chi(t)) \in \mathcal{W}$ . Furthermore, it implies that for all  $t \geq 0$ ,  $u_{\mathcal{V}}(v(t), z(t), \chi(t)) \in U$ . Thus,  $\chi$  is a trajectory of  $\Sigma$  which allows to conclude. ■

Now we are in position to state the result that will enable us to perform tracking control.

*Proposition 2:* Assume that for the systems  $\Sigma$  and  $\Sigma'$  the constraints  $\mu$  and  $\nu$  satisfy the inequality

$$\frac{\nu}{2} (1 + |1 - 1/\alpha| + 2/\sqrt{\alpha}) \leq \mu \quad (7)$$

for some  $\alpha > 0$ . Then,  $\mathcal{W} = \{(z, \chi) \mid \mathcal{V}(z, \chi) \leq 4\nu^2\}$  where  $\mathcal{V}(z, \chi) = \max(Q(z, \chi), 4\nu^2)$  with

$$Q(z, \chi) = \|C_x \chi - z\|^2 + \alpha \|C_x \chi - z + 2C_y \chi\|^2$$

is an approximate simulation relation of precision  $\delta = 2\nu$  of  $\Sigma'$  by  $\Sigma$  and an associated interface is

$$u_{\mathcal{W}}(v, z, \chi) = \frac{v}{2} + \frac{-1 - \alpha}{4\alpha} (C_x \chi - z) - C_y \chi.$$

*Proof:* First, let us remark that (5) clearly holds. Now, consider the interface  $u_{\mathcal{V}}(v, z, \chi) = u_{\mathcal{W}}(v, z, \chi)$ . If  $Q(z, \chi) \leq 4\nu^2$ , then it is clear that (6) holds. If  $Q(z, \chi) \geq 4\nu^2$ , then we can show that

$$\begin{aligned} & \frac{\partial \mathcal{V}(z, \chi)}{\partial z} v + \frac{\partial \mathcal{V}(z, \chi)}{\partial \chi} (A\chi + Bu_{\mathcal{V}}(v, z, \chi)) = \\ & = -2(C_x \chi - z)^T v - 2\alpha(C_x \chi - z + 2C_y \chi)^T v + 2(C_x \chi - z)^T C_x (A\chi + Bu_{\mathcal{V}}) + \\ & \quad + 2\alpha(C_x \chi - z + 2C_y \chi)^T (C_x + 2C_y) (A\chi + Bu_{\mathcal{V}}) \\ & = -2(C_x \chi - z)^T v - \alpha(C_x \chi - z + 2C_y \chi)^T (2v - 2(C_x + 2C_y)(A\chi + Bu_{\mathcal{V}})) + 2(C_x \chi - z)^T C_x (A\chi + Bu_{\mathcal{V}}) \end{aligned}$$

Also, we have

$$\begin{aligned} C_x(A\chi + Bu_{\mathcal{V}}) &= C_x A\chi + C_x Bu_{\mathcal{V}} \\ &= C_y \chi + 0 = C_y \chi \end{aligned}$$

and

$$\begin{aligned} (C_x + 2C_y)(A\chi + Bu_{\mathcal{V}}) &= C_x A\chi + C_x Bu_{\mathcal{V}} + 2C_y A\chi + 2C_y Bu_{\mathcal{V}} \\ &= C_y \chi + 0 + 0 + 2u_{\mathcal{V}} \\ &= C_y \chi + v + \frac{-1 - \alpha}{2\alpha} (C_x \chi - z) - 2C_y \chi \\ &= v + \frac{-1 - \alpha}{2\alpha} (C_x \chi - z) - C_y \chi. \end{aligned}$$

Hence,

$$\begin{aligned} & \frac{\partial \mathcal{V}(z, \chi)}{\partial z} v + \frac{\partial \mathcal{V}(z, \chi)}{\partial \chi} (A\chi + Bu_{\mathcal{V}}(v, z, \chi)) = \\ & = -2(C_x \chi - z)^T v - \alpha(C_x \chi - z + 2C_y \chi)^T (2v - 2v + \frac{1 + \alpha}{\alpha} (C_x \chi - z) + 2C_y \chi) + 2(C_x \chi - z)^T C_y \chi \\ & = -2(C_x \chi - z)^T v - \alpha(C_x \chi - z + 2C_y \chi)^T (C_x \chi - z + 2C_y \chi) - \\ & \quad - \alpha(C_x \chi - z + 2C_y \chi)^T \frac{1}{\alpha} (C_x \chi - z) + 2(C_x \chi - z)^T C_y \chi \\ & = -2(C_x \chi - z)^T v - \alpha \|C_x \chi - z + 2C_y \chi\|^2 - (C_x \chi - z)^T (C_x \chi - z) - \\ & \quad - 2(C_y \chi)^T (C_x \chi - z) + 2(C_x \chi - z)^T C_y \chi \\ & = -Q(z, \chi) - 2(C_x \chi - z) \cdot v \leq -Q(z, \chi) + 2\nu \|C_x \chi - z\| \end{aligned}$$

because

$$-2(C_x \chi - z) \cdot v \leq |2(C_x \chi - z) \cdot v| \leq 2\|(C_x \chi - z)\| \|v\| \leq 2\nu \|(C_x \chi - z)\|.$$

Since  $\|C_x \chi - z\|^2 \leq Q(z, \chi)$ , we have

$$\begin{aligned} & \frac{\partial \mathcal{V}(z, \chi)}{\partial z} v + \frac{\partial \mathcal{V}(z, \chi)}{\partial \chi} (A\chi + Bu_{\mathcal{V}}(v, z, \chi)) \leq \\ & -Q(z, \chi) + 2\nu \sqrt{Q(z, \chi)} \leq \sqrt{Q(z, \chi)} (2\nu - \sqrt{Q(z, \chi)}). \end{aligned}$$

Since  $Q(z, \chi) \geq 4\nu^2$ , equation (6) holds and  $\mathcal{V}$  is a simulation function of  $\Sigma'$  by  $\Sigma$ , and  $u_{\mathcal{V}}$  is an associated interface.

Moreover, for all  $v \in V$ ,  $(z, \chi) \in \mathcal{W}$ , the interface  $u_{\mathcal{V}}(v, z, \chi)$  satisfies the velocity constraints of  $\Sigma$ :

$$\begin{aligned} \|u_{\mathcal{V}}\| &= \|u_{\mathcal{W}}\| = \left\| \frac{v}{2} + \frac{-1 + \alpha - 2\alpha}{4\alpha}(C_x\chi - z) - C_y\chi \right\| \\ &= \left\| \frac{v}{2} + \frac{-1 + \alpha}{4\alpha}(C_x\chi - z) - \frac{1}{2}(C_x\chi - z + 2C_y\chi) \right\| \\ &\leq \left\| \frac{v}{2} \right\| + \left| \frac{-1 + \alpha}{4\alpha} \right| \|C_x\chi - z\| + \frac{1}{2} \|C_x\chi - z + 2C_y\chi\| \\ &\leq \frac{\nu}{2} + \frac{|-1 + \alpha|}{4\alpha} \sqrt{\mathcal{V}(z, \chi)} + \frac{1}{2} \sqrt{\frac{\mathcal{V}(z, \chi)}{\alpha}} \end{aligned}$$

since  $\|C_x\chi - z\|^2 \leq \mathcal{V}(z, \chi)$  and  $\alpha\|C_x\chi - z + 2C_y\chi\|^2 \leq \mathcal{V}(z, \chi)$ . But,  $(z, \chi) \in \mathcal{W}$ , that is  $\mathcal{V}(z, \chi) \leq 4\nu^2$ , and, hence, using (7):

$$\begin{aligned} u_{\mathcal{V}} &\leq \frac{\nu}{2} + \frac{|-1 + \alpha|}{4\alpha} 2\nu + \frac{1}{2} \frac{2\nu}{\sqrt{\alpha}} \\ &\leq \frac{\nu}{2} (1 + |1 - 1/\alpha| + 2/\sqrt{\alpha}) \\ &\leq \mu. \end{aligned}$$

Therefore, Theorem 1 applies and  $\mathcal{W}$  is an approximate simulation relation of precision  $2\nu$  of  $\Sigma'$  by  $\Sigma$  and an associated interface is given by  $u_{\mathcal{W}}(v, z, \chi) = u_{\mathcal{V}}(v, z, \chi)$ .  $\blacksquare$

The importance of Proposition 2 is the following. Assume that the initial state of the abstraction  $\Sigma'$  is chosen so that  $z(0) = C_x\chi(0)$  and that  $\Sigma'$  and  $\Sigma$  are interconnected through the interface  $u_{\mathcal{W}}$ . Then, from Theorem 1, the observed trajectories  $x(t)$  of system  $\Sigma$  track the trajectories  $z(t)$  of  $\Sigma'$  with precision  $2\nu$ .

## V. $\delta$ -ROBUST RTL FORMULAS

In the previous section, we designed a control interface which enables the dynamic model  $\Sigma$  to track its abstract kinematic model  $\Sigma'$  with accuracy  $\delta = 2\nu$ . In this section, we modify the initial RTL specification  $\phi$  to a new more robust specification  $\mathbf{rob}(\phi)$  that takes into account the bound  $\delta$  of the tracking error. The new modified robust formula  $\phi' = \mathbf{rob}(\phi)$  will be used in the following sections for the design of a hybrid controller  $H'_{\phi'}$  for the system  $\Sigma'$ .

Similar to [30], we introduce the notion of  $\delta$ -expansion and  $\delta$ -contraction for sets in order to define our notion of robustness.

*Definition 6 ( $\delta$ -Contraction,  $\delta$ -Expansion):* Given a radius  $\delta \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$  and a point  $a$  in a normed space  $A$ , the  $\delta$ -ball centered at  $a$  is defined as  $B_{\delta}(a) = \{b \in A \mid \|a - b\| \leq \delta\}$ . If  $\Gamma \subseteq A$ , then  $C_{\delta}(\Gamma) = \{a \in A \mid B_{\delta}(a) \subseteq \Gamma\}$  is the  $\delta$ -contraction and  $B_{\delta}(\Gamma) = \{a \in A \mid B_{\delta}(a) \cap \Gamma \neq \emptyset\}$  is the  $\delta$ -expansion of the set  $\Gamma$ .

Consider now a new set of propositions  $\Xi_{\Pi}$  such that  $\Xi_{\Pi} = \{\xi_{\psi} \mid \psi = \pi \text{ or } \psi = \neg\pi \text{ for } \pi \in \Pi\}$ . For a given

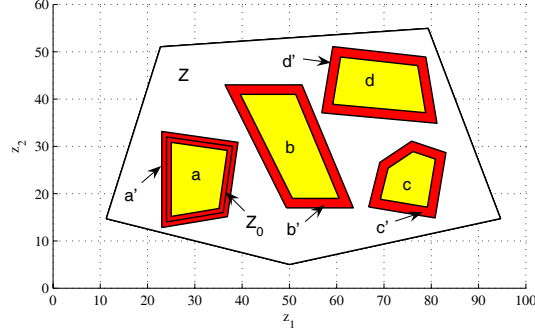


Fig. 3. Modified workspace  $Z$  and modified regions  $\Xi_{\Pi}$  for  $\delta = 1$ . Any point  $z$  in the region  $a$  is labeled by the set of propositions  $h_{\delta}(z) = \{\xi_{\pi_0}, \xi_{\pi_1}, \xi_{\neg\pi_2}, \xi_{\neg\pi_3}, \xi_{\neg\pi_4}\}$ , while any point  $z$  in the annulus region  $a'$  (dark gray) is labeled by the set  $h_{\delta}(z) = \{\xi_{\pi_0}, \xi_{\neg\pi_2}, \xi_{\neg\pi_3}, \xi_{\neg\pi_4}\}$ . Notice that  $Z = \llbracket \xi_{\pi_0} \rrbracket_{\delta}$  and that  $Z_0 = X_0 = \llbracket \pi_1 \rrbracket$ .

$\delta \in \mathbb{R}_{\geq 0}$ , we define a new map  $\llbracket \cdot \rrbracket_{\delta} : \Xi_{\Pi} \rightarrow \mathcal{P}(\mathbb{R}^2)$  based on the map  $\llbracket \cdot \rrbracket$  as follows:

$$\forall \xi \in \Xi_{\Pi}, \quad \llbracket \xi \rrbracket_{\delta} = \begin{cases} C_{\delta}(\llbracket \pi \rrbracket) \cap Z & \text{if } \xi = \xi_{\neg\pi} \\ C_{\delta}(\llbracket \pi \rrbracket) \cap Z & \text{if } \xi = \xi_{\pi} \end{cases}$$

where  $Z = C_{\delta}(X)$  is the free workspace of  $\Sigma'$ .

For clarity of the presentation, we define a translation algorithm  $\mathbf{rob} : \Phi_{\Pi} \rightarrow \Phi_{\Xi_{\Pi}}$  which takes as input a RTL formula  $\phi$  in NNF and it returns a formula  $\mathbf{rob}(\phi)$  where the occurrences of propositions  $\pi$  and  $\neg\pi$  have been replaced by the members  $\xi_{\pi}$  and  $\xi_{\neg\pi}$  of  $\Xi_{\Pi}$  respectively. Note that after the translation no negation operator appears in the formula  $\mathbf{rob}(\phi)$ . Moreover for the purposes of the following discussion, we define the map  $h_{\delta} : Z \rightarrow \mathcal{P}(\Xi_{\Pi})$  such that for any  $z \in Z$  we have  $h_{\delta}(z) = \{\xi \in \Xi_{\Pi} \mid z \in \llbracket \xi \rrbracket_{\delta}\}$ .

*Example 3:* Revisiting Example 1, we can now apply the expansion and contraction operators on the regions of interest labeled by  $\Pi$  and the free workspace  $X$  and derive the modified regions labeled by  $\Xi_{\Pi}$  and the modified workspace  $Z$  (see Fig. 3). Note that the  $\delta$ -contraction (or  $\delta$ -expansion) of a polyhedral set is not always a polyhedral set. In order to maintain a polyhedral description for all the sets, we under-approximate the  $\delta$ -contraction by the inward  $\delta$ -offset. Informally, the  $\delta$ -offset of a polyhedral set is the inward  $\delta$ -displacement of its facets along the corresponding normal directions. Since the  $\delta$ -offset is an under-approximation of the  $\delta$ -contraction, Theorem 2 still holds.

The following theorem is the connecting link between the specifications satisfied by the abstraction  $\Sigma'$  and the concrete system  $\Sigma$ . Informally, it states that given  $\delta > 0$  if we  $\delta$ -expand the sets that must be avoided and  $\delta$ -contract the sets that must be reached, then we will obtain a  $\delta$ -robust specification. The latter implies that if a trajectory satisfies the  $\delta$ -robust specification, then any other trajectory that remains  $\delta$ -close to the initial one will also satisfy the same non-robust initial specification.

*Theorem 2:* Consider a formula  $\phi \in \Phi_{\Pi}$ , a map  $\llbracket \cdot \rrbracket : \Pi \rightarrow \mathcal{P}(\mathbb{R}^2)$ , and a number  $\delta \in \mathbb{R}_{>0}$ , then for all functions  $x, z : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^2$  such that for all  $t \geq 0$ ,  $\|z(t) - x(t)\| \leq \delta$ , the following holds :

$$(z, [\cdot]_\delta) \models \mathbf{rob}(\phi) \implies (x, [\cdot]) \models \phi.$$

*Proof:* The proof is by induction on the structure of the formula  $\mathbf{rob}(\phi)$ . Recall that there is no negation operator.

**Case  $\mathbf{rob}(\phi) = \xi \in \Xi_\Pi$ :** We have two sub-cases

- $\xi = \xi_\pi$  with  $\pi \in \Pi$ , then  $(z(0), [\cdot]_\delta) \models \xi$  iff  $z(0) \in [\xi]_\delta = C_\delta([\pi])$  which implies that  $B_\delta(z(0)) \subseteq [\pi]$ . Since  $\|z(0) - x(0)\| \leq \delta$ , we immediately get that  $x(0) \in [\pi]$ . Therefore,  $(x(0), [\cdot]) \models \pi$ .
- $\xi = \xi_{\neg\pi}$  with  $\pi \in \Pi$ , then  $(z(0), [\cdot]_\delta) \models \xi$  iff  $z(0) \in [\xi]_\delta = \overline{B_\delta([\pi])}$  which implies that  $B_\delta(z(0)) \subseteq \overline{[\pi]}$ . Since  $\|z(0) - x(0)\| \leq \delta$ , we get that  $x(0) \in \overline{[\pi]}$  iff  $x(0) \notin [\pi]$ . Therefore,  $(x(0), [\cdot]) \not\models \pi$  or  $(x(0), [\cdot]) \models \neg\pi$ .

**Case  $\phi = \phi_1 \vee \phi_2$ :**  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi)$  or by definition of  $\mathbf{rob}(\phi)$  we get that  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1) \vee \mathbf{rob}(\phi_2)$  iff  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1)$  or  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_2)$ . Using the induction hypothesis, we get that  $(x, [\cdot]) \models \phi_1$  or  $(x, [\cdot]) \models \phi_2$ . Therefore,  $(x, [\cdot]) \models \phi_1 \vee \phi_2$ .

**Case  $\phi = \phi_1 \wedge \phi_2$ :**  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi)$  or by definition of  $\mathbf{rob}(\phi)$  we get that  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1) \wedge \mathbf{rob}(\phi_2)$  iff  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1)$  and  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_2)$ . Using the induction hypothesis, we get that  $(x, [\cdot]) \models \phi_1$  and  $(x, [\cdot]) \models \phi_2$ . Therefore,  $(x, [\cdot]) \models \phi_1 \wedge \phi_2$ .

**Case  $\phi = \phi_1 \mathcal{U} \phi_2$ :**  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi)$  or by definition of  $\mathbf{rob}(\phi)$  we get that  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1) \mathcal{U} \mathbf{rob}(\phi_2)$  iff by definition there exists  $t \geq 0$  such that  $(z|_t, [\cdot]_\delta) \models \mathbf{rob}(\phi_2)$  and for all  $s \in [0, t)$  we have  $(z|_s, [\cdot]_\delta) \models \mathbf{rob}(\phi_1)$ . Hence, by the induction hypothesis, we get that  $(x|_t, [\cdot]) \models \phi_2$  and that for all  $s \in [0, t)$ ,  $(x|_s, [\cdot]) \models \phi_1$ . Therefore,  $(x, [\cdot]) \models \phi_1 \mathcal{U} \phi_2$ .

**Case  $\phi = \phi_1 \mathcal{R} \phi_2$ :**  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi)$  or by definition of  $\mathbf{rob}(\phi)$  we get that  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi_1) \mathcal{R} \mathbf{rob}(\phi_2)$  iff for all  $t \geq 0$ ,  $(z|_t, [\cdot]_\delta) \models \mathbf{rob}(\phi_2)$  or there exists  $s \in [0, t)$  such that  $(z|_s, [\cdot]_\delta) \models \mathbf{rob}(\phi_1)$ . Hence, by the induction hypothesis, we get that for all  $t \geq 0$ ,  $(x|_t, [\cdot]) \models \phi_2$  or there exists  $s \in [0, t)$  such that  $(x|_s, [\cdot]) \models \phi_1$ . We conclude by the definition of release that  $(x, [\cdot]) \models \phi_1 \mathcal{R} \phi_2$ . ■

*Remark 2:* When  $(z, [\cdot]_\delta) \not\models \mathbf{rob}(\phi)$  we cannot conclude that  $(x, [\cdot]) \not\models \phi$ . The only conclusion we can make in this case is that  $z(\tau)$  is not a  $\delta$ -robust trajectory in the sense of [31]. Potentially, we can gain more information if we define 3-valued semantics [32] for the satisfaction relation of  $(z, [\cdot]_\delta) \models \mathbf{rob}(\phi)$ , but for the scope of this paper Theorem 2 is sufficient.

*Remark 3:* Theorem 2 does not particularly refer to the output trajectories of systems  $\Sigma$  and  $\Sigma'$ . If we consider both functions  $z$  and  $x$  to be trajectories of  $\Sigma'$ , then Theorem 2 classifies which trajectories of  $\Sigma'$  are  $\delta$ -robust.

## VI. TEMPORAL LOGIC MOTION PLANNING

Having presented the procedure for deriving the modified RTL formula  $\phi' = \mathbf{rob}(\phi)$  for the kinematics model  $\Sigma'$ , we proceed to solve the temporal logic motion planning problem for  $\Sigma'$ . Formally, we solve the following general problem.

*Problem 2:* Given the robot model  $\Sigma'$ , a set of initial conditions  $Z_0 \subseteq Z$ , an RTL formula  $\phi'$  over  $\Xi_\Pi$  and a map  $[\cdot]_\delta$ , construct a hybrid controller  $H'_{\phi'}$ , so that the resulting robot's trajectories  $z(t)$  satisfy formula  $\phi'$ .

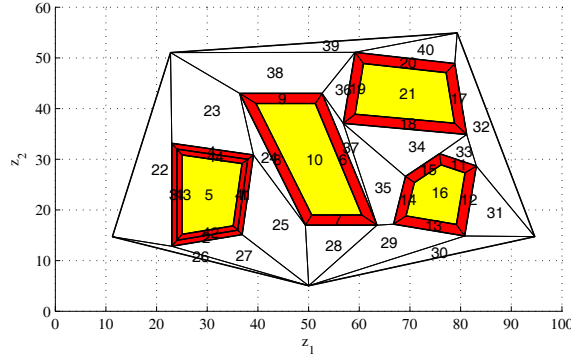


Fig. 4. Convex cell decomposition (Example 4).

Our solution to generating continuous robot trajectories satisfying RTL formulas comprise the following three steps:

- 1) *Discrete Abstraction of Robot Motion*: Decompose the environment  $Z$  into a finite number of equivalence classes resulting in a finite state model of robot motion.
- 2) *Temporal Logic Planning*: Construct plans for the discrete robot motion that satisfy the desired specification using automata theory.
- 3) *Continuous Implementation of Discrete Plan*: Implement the discrete plan at the continuous level while preserving the satisfaction of the temporal formula.

*Remark 4*: Our proposed solution to Problem 2 does not only hold for the particular  $\phi' = \mathbf{rob}(\phi)$  and corresponding map  $\llbracket \cdot \rrbracket_\delta$ . It actually holds for any RTL formula  $\phi$  in NNF and any map  $\llbracket \cdot \rrbracket$  as long as  $\Sigma'$  is given by eq. (2) – see also [13], [14].

#### A. Discrete Abstraction of Robot Motion

In order to use discrete logics to reason about continuous systems, we need to construct a finite partition of the continuous state space  $Z$  with respect to the map  $\llbracket \cdot \rrbracket_\delta$ . For that purpose, we can use many efficient cell decomposition methods for polygonal environments [1], [2]. The choice of a particular decomposition method depends on the type of controllers that we wish to use. The affine controllers presented in [15] require a triangular decomposition [22], while the potential field controllers of [16] can be applied to any convex cell that can be represented by a polygon [23]. Note that by employing any workspace decomposition method we can actually construct a *topological graph* (or *roadmap* as it is sometimes referred to [2]). A topological graph describes which cells are topologically adjacent, i.e. each node in the graph represents a cell and each edge in the graph implies topological adjacency of the cells. No matter what decomposition algorithm is employed, the workspace's decomposition must be *proposition preserving* with respect to the mapping  $\llbracket \cdot \rrbracket_\delta$ . In other words, we require that all the points which belong in the same cell must be labeled by the same set of propositions.

*Example 4*: To better explain the notion of a proposition preserving cell decomposition, let us consider the convex

decomposition of the workspace of Example 3 which appears in Fig. 4. The topological graph contains 40 nodes and 73 edges. Notice that all the points in the interior of each cell satisfy the same set of propositions.

On a more formal note, we can partition the workspace  $Z$  with respect to the map  $\llbracket \cdot \rrbracket_\delta$  and the set of initial conditions  $Z_0$  into a number of equivalence classes, that is, into a number of sets of points which satisfy the same property (in this case the same set of propositions). Note that mathematically the set of equivalence classes consists of the interior of the cells, the edges and the vertices. In the following, we define  $\mathcal{Q} = \{q_1, \dots, q_n\}$  to be the set of all equivalence classes. Let us introduce the map  $T : Z \rightarrow \mathcal{Q}$  which sends each state  $z \in Z$  to one equivalence class in  $\mathcal{Q}$ . Then, we can formally state the proposition preserving property as follows:

$$\forall z_i, z_j \in Z . T(z_i) = T(z_j) \implies h_\delta(z_i) = h_\delta(z_j)$$

Recall that  $h_\delta(z) = \{\xi \in \Xi_\Pi \mid z \in \llbracket \xi \rrbracket_\delta\}$ . Moreover, we define the “inverse” map  $T^{-1} : \mathcal{Q} \rightarrow \mathcal{P}(Z)$  of  $T$  such that  $T^{-1}(q)$  maps to all states  $z \in Z$  which are contained in the equivalence class  $q$ .

In the following paragraphs, we present how the topological graph resulting from the decomposition of  $Z$  with respect to  $\llbracket \cdot \rrbracket_\delta$  can be converted into a Finite Transition System (FTS) that serves as an abstract model of the robot motion. Posing the topological graph as an FTS allows us to use standard automata theoretic techniques [7] in order to solve the high level planning problem. In the following,  $Q \subseteq \mathcal{Q}$  denotes the equivalence classes that represent the interior of the cells.

*Definition 7 (FTS):* A Finite Transition System is a tuple  $D = (Q, Q_0, \rightarrow_D, h_D, \Xi_\Pi)$  where:

- $Q_0 \subseteq Q$  is the set of possible initial cells. Here,  $Q_0$  satisfies  $\cup_{q_0 \in Q_0} cl(T^{-1}(q_0)) = Z_0$ .
- $\rightarrow_D \subseteq Q \times Q$  captures the topological relationship between the cells. There is a transition from cell  $q_i$  to cell  $q_j$  written as  $q_i \rightarrow_D q_j$  if the cells labelled by  $q_i, q_j$  are adjacent, i.e.  $cl(T^{-1}(q_i))$  and  $cl(T^{-1}(q_j))$  share a common edge. Finally, for all  $q \in Q$  we add a self-loop, i.e.  $q \rightarrow_D q$ .
- $h_D : Q \rightarrow \mathcal{P}(\Xi_\Pi)$  is a map defined as  $h_D(q) = \{\xi \in \Xi_\Pi \mid T^{-1}(q) \subseteq \llbracket \xi \rrbracket_\delta\}$ .

The operator  $cl(\Gamma)$  denotes the closure of a set  $\Gamma$ . We define a *path* on the FTS to be a sequence of states (cells) and a *trace* to be the corresponding sequence of sets of propositions. Formally, a path is a function  $p : \mathbb{N} \rightarrow Q$  such that for each  $i \in \mathbb{N}$  we have  $p(i) \rightarrow_D p(i+1)$  and the corresponding trace is the function composition  $\bar{p} = h_D \circ p : \mathbb{N} \rightarrow \mathcal{P}(\Xi_\Pi)$ . The paths generated by  $D$  are non-terminating.

Note that in the above definition we have ignored the equivalence classes that represent the vertices of the triangles and the corresponding edges. The vertices do not correspond to robust engineering solutions, therefore we can safely ignore them. The edges of each triangle form the boundaries of the halfspaces that act as guards for each location in the final hybrid controller  $H'_\phi$ , that governs the behavior of  $\Sigma'$ . Moreover, when  $\delta > 0$  we do not care whether an edge or a point of a cell belongs to the set representing  $\pi$  or its negation  $\neg\pi$ . As such, the edges dictate the behavior of the system only at run time and, hence, they do not play any role in the initial high level planning step. Finally, the addition of a self-loop in each state is actually a trick that will enable us to design controllers that require the robot to stay forever in a region of interest.

## B. Linear Temporal Logic Planning

The transition system  $D$ , which was constructed in the previous section, will serve as an abstract model of the robot's motion. We must now lift our problem formulation from the continuous to the discrete domain. For that purpose we introduce and use the Linear Temporal Logic (LTL) [8] which has exactly the same syntax as RTL, but its semantics is interpreted over discrete paths generated by a finite transition system. In the next section, we will show that under the finite variability assumption if a formula  $\phi'$  is satisfiable on  $D$ , then we can construct a continuous trajectory  $z(t)$  that will satisfy the RTL formula  $\phi'$ .

In the following, we let  $\bar{p} \models \phi$  to denote the satisfiability of an LTL formula  $\phi$  over a trace  $\bar{p}$  starting at 0.

*Definition 8 (Discrete LTL Semantics):* For  $i, j \in \mathbb{N}$ , the semantics of any LTL formula  $\phi'$  in NNF built on the set of propositions  $\Xi_{\Pi}$  can be recursively defined as:

$$\bar{p} \models \xi \text{ iff } \xi \in \bar{p}(0) = h_D(p(0))$$

$$\bar{p} \models \phi'_1 \wedge \phi'_2 \text{ if } \bar{p} \models \phi'_1 \text{ and } \bar{p} \models \phi'_2$$

$$\bar{p} \models \phi'_1 \vee \phi'_2 \text{ if } \bar{p} \models \phi'_1 \text{ or } \bar{p} \models \phi'_2$$

$$\bar{p} \models \phi'_1 \mathcal{U} \phi'_2 \text{ if exists } i \geq 0 \text{ such that } \bar{p}|_i \models \phi'_2 \text{ and for all } j \text{ with } 0 \leq j < i \text{ we have } \bar{p}|_j \models \phi'_1$$

$$\bar{p} \models \phi'_1 \mathcal{R} \phi'_2 \text{ if for all } i \geq 0 \text{ we have } \bar{p}|_i \models \phi'_2 \text{ or there exists } j \in [0, i) \text{ such that } \bar{p}|_j \models \phi'_1$$

Even though theoretically the decision problem for both LTL and RTL is PSPACE-complete [8], [33], only LTL has the mathematical properties that can lead to practical decision methods [8]. In particular, in this work we are interested in the construction of automata that accept the traces of  $D$  which satisfy the LTL formula  $\phi'$  [34]. Such automata (which we will refer to as Büchi automata) differ from the classic finite automata in that they accept infinite strings (traces of  $D$  in our case). In the following, we assume that we are given the Büchi automaton that accepts the traces which satisfy an LTL specification and we refer the reader to [7, §9.4] for the theoretical details behind this translation.

*Definition 9:* A Büchi automaton that accepts traces that satisfy a Linear Temporal Logic formula  $\phi'$  is a tuple  $B_{\phi'} = (S, s_0, \mathcal{P}(\Xi_{\Pi}), \lambda_{B_{\phi'}}, F_{B_{\phi'}})$  where:

- $S$  is a finite set of states and  $s_0$  is the initial state.
- $\mathcal{P}(\Xi_{\Pi})$  is the input alphabet.
- $\lambda_{B_{\phi'}} : S \times \mathcal{P}(\Xi_{\Pi}) \rightarrow \mathcal{P}(S)$  is a transition relation.
- $F_{B_{\phi'}} \subseteq S$  is the set of accepting states.

In order to define what it means for a Büchi automaton to accept a trace, we must first introduce some terminology. A *run*  $r$  of  $B_{\phi'}$  is the sequence of states  $r : \mathbb{N} \rightarrow S$  that occurs under an input trace  $\bar{p}$ , that is for  $i = 0$  we have  $r(0) = s_0$  and for all  $i \geq 0$  we have  $r(i+1) \in \lambda_{B_{\phi'}}(r(i), \bar{p}(i))$ . Let  $\lim(\cdot)$  be the function that returns the set of states that are encountered infinitely often in the run  $r$  of  $B_{\phi'}$ .

*Definition 10 (Büchi acceptance):* A Büchi automaton  $B_{\phi'}$  accepts an infinite trace  $\bar{p}$  if and only if the corresponding run  $r$  satisfies the relationship  $\lim(r) \cap F_{B_{\phi'}} \neq \emptyset$ .

Informally, what the above definition states is that the traces  $\bar{p}$  that are accepted by  $B_{\phi'}$  have corresponding runs  $r$  which go through accepting states infinitely often, i.e.  $r(i) \in F$  for infinitely many  $i$ 's. We define the language of  $B_{\phi'}$ , i.e.  $\mathcal{L}(B_{\phi'})$ , to consist of all the traces of  $D$  that have a run that is accepted by  $B_{\phi'}$ . Finding the existence of accepting runs is an easy problem [7]. First, we convert the Büchi automaton to a directed graph and, then, we find the strongly connected components (SCC) in the graph [35]. Second, we use Breadth or Depth First search [35] starting from  $s_0$  in order to find the reachable vertices (and hence states) in the graph. If at least one SCC that contains a final state is reachable from  $s_0$ , then the language  $\mathcal{L}(B_{\phi'})$  is non-empty (for more details see [7]). The theoretical running time for solving emptiness problem is  $O(|Q| + |\rightarrow_D|)$  since both the BFS or DFS and the algorithm for generating the SCCs have running time  $O(|Q| + |\rightarrow_D|)$  [35]. Here,  $|Q|$  denotes the number of states in  $D$  and  $|\rightarrow_D|$  the number of transitions.

We can now use the abstract representation of robot's motion, that is the FTS, in order to reason about the desired motion of the robot. First, we convert the FTS  $D$  into a Büchi automaton  $D'$ . The translation from  $D$  to  $D'$  enables us to use standard tools and techniques from automata theory [7, §9] alleviating, thus, the need for developing new theories.

Translating an FTS into an automaton is a standard procedure which can be found in any formal verification textbook such as [7, §9.2]. The procedure consists of adding a dummy initial state  $q_d \notin Q$  that has one transition to each state  $q_0$  in  $Q_0$  and by moving the label from each state  $q$  to all of its incoming transitions.

*Definition 11 (FTS to Automaton):* The Büchi automaton  $D'$  which corresponds to the FTS  $D$  is a tuple  $D' = (Q', q_d, \mathcal{P}(\Xi_{\Pi}), \lambda_{D'}, F_{D'})$  where:

- $Q' = Q \cup \{q_d\}$  for  $q_d \notin Q$ .
- $\mathcal{P}(\Xi_{\Pi})$  is the input alphabet.
- $\lambda_{D'} : Q' \times \mathcal{P}(\Xi_{\Pi}) \rightarrow \mathcal{P}(Q')$  is a transition relation defined as:  $q_j \in \lambda_{D'}(q_i, l)$  iff  $q_i \rightarrow_D q_j$  and  $l = h_D(q_j)$  and  $q_0 \in \lambda_{D'}(q_d, l)$  iff  $q_0 \in Q_0$  and  $l = h_D(q_0)$ .
- $F_{D'} = Q'$  is the set of accepting states.

Similar to  $B_{\phi'}$ , we define the language  $\mathcal{L}(D')$  of  $D'$  to be the set of all possible traces that are accepted by  $D'$ . Note that any path generated by  $D$  has a trace that belongs to  $\mathcal{L}(D')$ .

Now that all the related terminology is defined, we can give an overview of the basic steps involved in the temporal logic planning [5]. Our goal in this section is to generate paths on  $D$  that satisfy the specification  $\phi'$ . In automata theoretic terms, we want to find the subset of the language  $\mathcal{L}(D')$  which also belongs to the language  $\mathcal{L}(B_{\phi'})$ . This subset is simply the intersection of the two languages  $\mathcal{L}(D') \cap \mathcal{L}(B_{\phi'})$  and it can be constructed by taking the product  $D' \times B_{\phi'}$  of the automaton  $D'$  and the Büchi automaton  $B_{\phi'}$ . Informally, the Büchi automaton  $B_{\phi'}$  restricts the behaviour of the system  $D'$  by permitting only certain acceptable transitions. Then, given an initial state in the FTS  $D$ , which is the abstraction of the actual initial position of the robot, we can choose a particular path from  $\mathcal{L}(D) \cap \mathcal{L}(B_{\phi'})$  according to a preferred criterion. In the following, we present the details of this construction.

Consider the automaton  $A$  that derives from the product of the automaton  $D'$  that describes the topology of our system and the automaton  $B_{\phi'}$  that represents the temporal logic specification (formula  $\phi'$ ). The design of a hybrid

controller for motion planning using temporal logic specifications reduces to the problem of finding the accepting executions of the automaton  $A$ .

*Definition 12:* The product automaton  $A = D' \times B_{\phi'}$  is a tuple  $A = (S_A, s_{A0}, \mathcal{P}(\Xi_{\Pi}), \lambda_A, F_A)$  where:

- $S_A = Q' \times S$  and  $s_{A0} = \{(q_d, s_0)\}$ .
- $\mathcal{P}(\Xi_{\Pi})$  is the input alphabet.
- $\lambda_A : S_A \times \mathcal{P}(\Xi_{\Pi}) \rightarrow \mathcal{P}(S_A)$  such that  $(q_j, s_j) \in \lambda_A((q_i, s_i), l)$  iff  $q_j \in \lambda_{D'}(q_i, l)$  and  $s_j \in \lambda_{B_{\phi'}}(s_i, l)$ .
- $F_A = Q' \times F$  is the set of accepting states.

Informally, the automaton  $A$  permits a transition to a state  $(q_j, s_j)$  if both the automaton  $D'$  and the automaton  $B_{\phi'}$  have a transition with an input symbol that is actually the set of propositions at the state  $q_j$  of  $D$ . That is, we allow a transition if and only if we can observe an acceptable proposition  $h_D(q_j)$  at the next state.

By construction, the following theorem is satisfied (recall that  $\bar{p}$  is a trace of  $D$  if and only if  $\bar{p}$  is accepted by  $D'$ ).

*Theorem 3 (Adapted from [5]):* A trace  $\bar{p}$  of  $D$  that satisfies the specification  $\phi'$  exists iff the language of  $A$  is non-empty, i.e.  $\mathcal{L}(A) = \mathcal{L}(D' \times B_{\phi'}) = \mathcal{L}(D') \cap \mathcal{L}(B_{\phi'}) \neq \emptyset$ .

Implementation-wise, we solve the non-emptiness problem of the language  $\mathcal{L}(A)$  as follows. First, we find all the shortest sequences of states from  $s_{A0}$  to all the accepting states in  $F_A$  using Breadth First Search (BFS) [35]. The running time of BFS is linear in the number of states and the number of transitions in  $A$ . Then, from each reachable accepting state  $q_a \in F_A$  we initiate a new BFS in order to find the shortest sequence of states that leads back to  $q_a$ . The rationale behind this construction is that any infinite path on a finite graph must visit at least one node of the graph infinitely often (see [36, §8] or [7, §9.3]). If no accepting state is reachable from  $s_{A0}$  or no infinite loop can be found, then the language  $\mathcal{L}(A)$  is empty and, hence, the temporal logic planning problem does not have a solution. Note that if  $\mathcal{L}(A) \neq \emptyset$ , then this algorithm can potentially return a set  $R$  of accepting runs  $r$  each leading to a different accepting state in  $F_A$ . From each such run  $r$ , we can easily derive the corresponding path  $p$  of  $D$  which satisfies the specification  $\phi'$ .

*Proposition 3:* Let  $pr : S_A \rightarrow Q$  be a projection function such that  $pr(q, s) = q$ . If  $r$  is an accepting run of  $A$ , then  $p = (pr \circ r)|_1$  is a path on  $D$  such that  $\bar{p} \models \phi'$ .

*Proof:* Let  $r$  be an accepting run of  $A$ . Then by definition,  $pr \circ r$  is an accepting run of  $D'$ . By construction of  $D'$ , the input trace  $\bar{p}$  that corresponds to the run  $pr \circ r$  is  $\bar{p} = h_D \circ (pr \circ r)|_1$ . Therefore,  $\bar{p}$  is a trace of  $D$  and  $p = (pr \circ r)|_1$  is a path on  $D$ . ■

Let  $P$  be the set of paths on  $D$  that correspond to accepting runs in  $R$ . The set  $P$  as computed above might not contain a path for every  $q_0 \in Q_0$  and in some cases the paths computed might not even be the shortest ones. To see this, consider the runs which start from  $s_{A0}$  and pass through different  $q_0 \in Q_0$ . It is possible that all these runs should converge to the same shortest sequence of states before reaching an accepting state. Since BFS constructs a tree, this implies that only one run would be the shortest and the rest would either reach an accepting state at longer distance or not reach an accepting state at all. One practical solution to this problem is to start a new BFS from each accepting state that belongs to an accepting cycle, i.e. the state  $q_a$  in the above algorithm, and

find which states in  $Q_0$  are backward reachable from it (note that now we are looking for runs that terminate at states in  $Q_0 \times S$ ). Then, we can repeat the procedure by starting from a different  $q_a \in F_A$  until all the states in  $Q_0$  have been covered.

Any path  $p \in P$  can be characterized by a pair of sequences of states  $(p^f, p^l)$ . Here,  $p^f = p_1^f p_2^f \dots p_{n_f}^f$  denotes the finite part of the path and  $p^l = p_1^l p_2^l \dots p_{n_l}^l$  the periodic part (infinite loop) such that  $p_{n_f}^f = p_1^l$ . Note that  $p$  is given by  $p(i) = p_{i+1}^f$  for  $0 \leq i \leq n_f - 2$  and  $p(i) = p_j^l$  with  $j = ((i - n_f + 1) \bmod n_l) + 1$  for  $i \geq n_f - 1$ .

*Remark 5:* Under the assumption that the initial workspace  $X$  is a connected space and due to the fact that the system  $\Sigma'$  models a fully actuated kinematic model of a robot, there can exist only two cases for which our planning method can fail to return a solution. First, when the workspace  $Z$  becomes disconnected due to the dynamics of the system  $\Sigma$ , and second, when there exist logical inconsistencies in the temporal logic formula  $\phi'$  with respect to the environment  $Z$  (see Example 7).

*Example 5:* The Büchi automaton that accepts the paths that satisfy (3) has 5 states (one accepting) and 13 transitions. For the conversion from LTL to Büchi automata, we use the python toolbox LTL2NBA by Fritz and Teegen, which is based on [37]. The product automaton  $A$  has 205 states. The shortest path on the topological graph starting from cell 5 is:  $(p^f, p^l) = (\{5, 41, 1, 25, 24, 8, 10, 6, 37, 35, 14, 16, 15, 34, 18, 21, 19, 36, 38, 23, 4, 44, 5\}, \{5\})$ . Using Fig. 4, the reader can verify that this sequence satisfies (3). A prototype MATLAB implementation of the planning part of our framework took 0.61 sec for this example.

### C. Continuous Implementation of Discrete Trajectory

Our next task is to utilize each discrete path  $p \in P$  in order to construct a hybrid control input  $v(t)$  for  $t \geq 0$  which will drive  $\Sigma'$  so as its trajectories  $z(t)$  satisfy the RTL formula  $\phi'$ . We achieve this desired goal by simulating (or implementing) at the continuous level each discrete transition of  $p$ . This means that if the discrete system  $D$  makes a transition  $q_i \rightarrow_D q_j$ , then the continuous system  $\Sigma'$  must match this discrete step by moving from any position in the cell  $cl(T^{-1}(q_i))$  to a position in the cell  $cl(T^{-1}(q_j))$ . Moreover, if the periodic part in the path  $p$  consists of just a single state  $q_l$ , then we have to guarantee that the position of the robot always remains in the invariant set  $T^{-1}(q_l)$ .

These basic control specifications imply that we need at least two types of continuous feedback control laws. We refer to these control laws as *reachability* and *cell invariant* controllers. Informally, a reachability controller drives each state inside a cell  $q$  to a predefined region on the cell's boundary, while the cell invariant controller guarantees that all the trajectories that start inside a cell  $q$  will always remain in that cell.

Let us assume that we are given or that we can construct a finite collection of continuous feedback control laws  $\{g_\kappa\}_{\kappa \in K}$  indexed by a control alphabet  $K$  such that for any  $\kappa \in K$  we have  $g_\kappa : Z_\kappa \rightarrow V$  with  $Z_\kappa \subseteq Z$ . In our setting, we make the following additional assumptions. First, we define the operational range of each controller to be one of the cells in the workspace of the robot, i.e. for any  $\kappa \in K$  there exists for some  $q \in Q$  such that  $Z_\kappa = cl(T^{-1}(q))$ . Second, if  $g_\kappa$  is a reachability controller, then we require that all the trajectories which start in  $Z_\kappa$  must converge on the same subset of the boundary of  $Z_\kappa$  within finite time while never exiting  $Z_\kappa$  before that

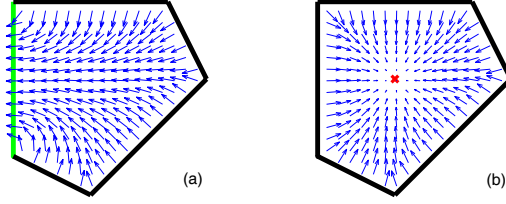


Fig. 5. (a) Reachability and (b) Cell invariant controller.

time. Finally, if  $g_\kappa$  is a cell invariant controller, then we require the all the trajectories which initiate from a point in  $Z_\kappa$  converge on the barycenter  $b_\kappa$  of  $Z_\kappa$ . Examples of such feedback control laws for  $\Sigma'$  appear in Fig. 5. A formal presentation of these types of controllers is beyond the scope of this paper and the interested reader can find further details in [15], [16], [38].

Formally, the assumptions on the set of controllers  $\{g_\kappa\}_{\kappa \in K}$  can be stated as follows. First, we assume that the application of the controller  $g_\kappa$  to  $\Sigma'$  gives rise to a Lipschitz continuous vector field  $f_\kappa : Z \rightarrow \mathbb{R}^2$ , therefore the autonomous system  $\dot{z}(t) = f_\kappa(z(t))$  with initial conditions  $z_0 = z(0) \in Z_\kappa$  has a unique solution. Let  $\vartheta_\kappa(t, z_0)$  for  $t \geq 0$  be the resulting continuous trajectory, i.e.  $\vartheta_\kappa(t, z_0) = z(t)$  when  $z(0) = z_0$ . The curve satisfies the property  $\vartheta_\kappa(t, \vartheta_\kappa(t_1, z_0)) = \vartheta_\kappa(t + t_1, z_0)$ . For convenience in the notation, we denote the set of all the trajectories of  $\dot{z}(t) = f_\kappa(z(t))$  starting in  $Z_\kappa$  by  $\mathcal{Z}_\kappa$ , i.e.  $\mathcal{Z}_\kappa = \{\vartheta_\kappa(\cdot, z_0) : \mathbb{R}_{\geq 0} \rightarrow Z \mid z_0 \in Z_\kappa\}$ . Also, let  $bd(A)$  denote the boundary of a set  $A$ . Then for any  $\kappa \in K$ , we define:

- *Invariant set:* The compact set  $Inv : K \rightarrow \mathcal{P}(Z)$  such that  $Inv(\kappa) = cl(Z_\kappa)$  defines the invariant set of the controller  $g_\kappa$ . That is, either for all the trajectories  $\vartheta_\kappa(\cdot, z_0) \in \mathcal{Z}_\kappa$  there exists a finite time  $\tau_\kappa(z_0)$ , which depends on the initial conditions, such that for all  $t \in [0, \tau_\kappa(z_0)]$  we have  $\vartheta_\kappa(t, z_0) \in Inv(\kappa)$  and for  $t > \tau_\kappa(z_0)$  we have  $\vartheta_\kappa(t, z_0) \notin Inv(\kappa)$  or all trajectories always stay in  $Inv(\kappa)$  and therefore  $\tau_\kappa(z_0) = \infty$ .
- *Goal set:* The goal set  $Gl : K \rightarrow \mathcal{P}(Inv(\kappa))$  is either the set of points on the boundary of  $Inv(\kappa)$  that each trajectory from  $\mathcal{Z}_\kappa$  intersects before it leaves the set  $Inv(\kappa)$  or an  $\omega$  limit set contained in  $Inv(\kappa)$ .

The way we can compose such controllers given the pair  $(p^f, p^l)$ , which characterizes a path  $p \in P$ , is as follows. First note that it is possible to get a finite repetition of states in the path  $p$ , for example there can exist some  $i \geq 0$  such that  $p(i) = p(i+1)$  but  $p(i+1) \neq p(i+2)$ . This situation might occur because we have introduced self-loops in the automaton  $D'$  in conjunction with possibility that the Büchi automaton  $B_{\phi'}$  might not be optimal (in the sense of number of states and transitions). Therefore, we first remove finite repetitions<sup>1</sup> of states from  $p$ .

Next, we define the control alphabet to be  $K = K^f \cup K^l \subseteq Q \times Q$  where  $K^f = \cup_{i=1}^{n_f-1} \{(p_i^f, p_{i+1}^f)\} \cup \{(p_{n_f}^f, p_1^l)\}$  and  $K^l = \cup_{i=1}^{n_l-1} \{(p_i^l, p_{i+1}^l)\} \cup \{(p_{n_l}^l, p_1^l)\}$  when  $n_l > 1$  or  $K^l = \emptyset$  otherwise. For any  $\kappa = (q_i, q_j) \in K \setminus \{(p_{n_f}^f, p_1^l)\}$ , we design  $g_\kappa$  to be a reachability controller that drives all initial states in  $Z_\kappa = cl(T^{-1}(q_i))$  to the goal set

<sup>1</sup>Removing such repeated states from  $p$  does not change the fact that  $\bar{p} \models \phi'$ . This is true because LTL formulas without the *next* time operator are *stutter invariant* (see [36, §6] or [7, §10]).

$Gl(\kappa) = cl(T^{-1}(q_i)) \cap cl(T^{-1}(q_j))$ . Finally for  $\kappa = (p_{n_f}^f, p_1^l)$ , we let  $g_\kappa$  be a cell invariant controller with goal set  $Gl(\kappa) = \{b_\kappa\}$ .

It is easy to see now how we can use each pair  $(p^f, p^l)$  in order to construct a hybrid controller  $H'_{\phi'}$ . Starting anywhere in the cell  $cl(T^{-1}(p_1^f))$ , we apply the control law  $g_{(p_1^f, p_2^f)}$  until the robot crosses the edge  $cl(T^{-1}(p_1^f)) \cap cl(T^{-1}(p_2^f))$ . At that point, we switch the control law to  $g_{(p_2^f, p_3^f)}$ . The above procedure is repeated until the last cell of the finite path  $p^f$  at which point we apply the cell invariant controller  $g_{(p_{n_f}^f, p_1^l)}$ . If the periodic part  $p^l$  of the path has only one state, i.e.  $n_l = 1$ , then this completes the construction of the hybrid controller  $H'_{\phi'}$ . If on the other hand  $n_l > 1$ , then we check whether the trajectory  $z(t)$  has entered an  $\varepsilon$ -neighborhood of the barycenter of the cell invariant controller. If so, we apply sequentially the controllers that correspond to the part  $p^l$  indefinitely while keeping the cell invariant controller in the loop. The cell invariant controller is necessary in order to avoid Zeno behavior [27].

Since there can only exist at most one Zeno cycle [39] in the final hybrid automaton and this cycle is guaranteed to not generate Zeno behaviors due to the existence of the cell invariant controller, the following proposition is immediate.

*Proposition 4:* The trajectories  $z$  of the system  $[\Sigma', H'_{\phi'}]$  satisfy the finite variability property.

By interconnecting  $\Sigma'$  with the hybrid controller  $H'_{\phi'}$ , we actually construct a hybrid automaton [40]. Let's assume first that  $P = \{p\}$ , i.e. the hybrid controller  $H'_{\phi'}$  corresponds to a single path  $p$ .

*Definition 13 (Hybrid Automaton):* The hybrid automaton  $\mathbf{H}$  that corresponds to the system  $[\Sigma', H'_{\phi'}]$  is a tuple  $\mathbf{H} = (\mathbb{R}^2, N, \mathcal{F}, \mathcal{I}, E, \mathcal{G})$  where:

- $\mathbb{R}^2$  is the continuous state space of  $\Sigma'$  and  $N \subseteq \mathbb{N}$  is a finite set of control locations such that  $N = \{1, 2, \dots, n_f\}$  if  $n_l = 1$  and  $N = \{1, 2, \dots, n_f + n_l\}$  otherwise. Moreover, we define  $\mathcal{H} = N \times \mathbb{R}^2$  to be the state space of  $\mathbf{H}$  and  $\mathcal{H}_0 \subseteq \mathcal{H}$  to be the set of initial conditions such that  $\mathcal{H}_0 = \{1\} \times \mathcal{I}(1)$ .
- $\mathcal{F} : N \rightarrow (Z \rightarrow \mathbb{R}^2)$  is the collection of vector fields such that

$$\forall i \in \{1, 2, \dots, n_f - 1\}. \mathcal{F}(i) = f_{(p_i^f, p_{i+1}^f)} \text{ and } \mathcal{F}(n_f) = f_{(p_{n_f}^f, p_1^l)}$$

$$\text{and when } n_l > 1 \text{ then } \forall i \in \{1, 2, \dots, n_l - 1\}. \mathcal{F}(n_f + i) = f_{(p_i^l, p_{i+1}^l)} \text{ and } \mathcal{F}(n_f + n_l) = f_{(p_{n_l}^l, p_1^l)}$$

- $\mathcal{I} : N \rightarrow \mathcal{P}(Z)$  is the invariant set of each control location with definition:

$$\forall i \in \{1, 2, \dots, n_f - 1\}. \mathcal{I}(i) = \text{Inv}(p_i^f, p_{i+1}^f) \text{ and } \mathcal{I}(n_f) = \text{Inv}(p_{n_f}^f, p_1^l)$$

$$\text{and when } n_l > 1 \text{ then } \forall i \in \{1, 2, \dots, n_l - 1\}. \mathcal{I}(n_f + i) = \text{Inv}(p_i^l, p_{i+1}^l) \text{ and } \mathcal{I}(n_f + n_l) = \text{Inv}(p_{n_l}^l, p_1^l)$$

- $E \subseteq N \times N$  is a relation that captures the switching between the control locations with definition:

$$E = \begin{cases} \{(i, i+1) \mid i \in \{1, 2, \dots, n_f - 1\}\} & \text{if } n_l = 1 \\ \{(i, i+1) \mid i \in \{1, 2, \dots, n_f + n_l - 1\}\} \cup \{(n_f + n_l, n_f)\} & \text{otherwise} \end{cases}$$

- $\mathcal{G} : E \rightarrow \mathcal{P}(Z)$  returns the conditions that permit the transition from one control location to the next:

$$\mathcal{G}(i, i+1) = \begin{cases} Gl(p_i^f, p_{i+1}^f) & \text{if } i \in \{1, 2, \dots, n_f - 1\} \\ B_\varepsilon(Gl(p_{n_f}^f, p_1^l)) \text{ such that } \varepsilon > 0 \text{ and } B_\varepsilon(Gl(p_{n_f}^f, p_1^l)) \subset \mathcal{I}(n_f) & \text{if } i = n_f \\ Gl(p_{i-n_f}^l, p_{i+1-n_f}^l) & \text{if } i \in \{n_f + 1, n_f + 2, \dots, n_f + n_l - 1\} \text{ and } n_l > 1 \\ Gl(p_{n_l}^l, p_1^l) & \text{if } i = n_f + n_l \text{ and } n_l > 1 \end{cases}$$

It is easy to see how the above construction can be extended when the set  $P$  contains multiple paths. The resulting global hybrid automaton is the union [41] of each simple hybrid automaton that corresponds to a single path. Obviously, the above construction is not minimal and further research is required toward that direction. Note however that the set  $K$  cannot be directly used as the set of control locations since the same controller can be used in several different places in the hybrid controller  $H'_{\phi'}$ . In other words, some kind of memory is required in the hybrid automaton.

Formally, the semantics of a hybrid automaton are given in terms of timed transition systems [41]. A generalized transition system is a tuple  $\mathcal{T}_H = (\mathcal{H}, \mathcal{H}_0, \rightarrow)$  where  $\rightarrow = (\cup_{e \in E} \xrightarrow{(e,0)}) \cup (\cup_{\delta \in \mathbb{R}_{\geq 0}} \xrightarrow{(\tau, \delta)})$  is a transition relation such that for  $(i, z_i), (j, z_j) \in \mathcal{H}$ :

- 1)  $\xrightarrow{(e,0)}$  is a *discrete transition*:  $(i, z_i) \xrightarrow{(e,0)} (j, z_j)$  for  $e = (i, j) \in E$  iff  $z_i \in \mathcal{G}(i, j)$ . Moreover, in this paper we consider the reset function to be the identity, i.e.  $z_i = z_j$ .
- 2)  $\xrightarrow{(\tau, \delta)}$  is a *continuous flow*:  $(i, z_i) \xrightarrow{(\tau, \delta)} (j, z_j)$  for  $\delta \in \mathbb{R}_{\geq 0}$  iff  $i = j$  and either
  - a)  $\delta = 0$  and  $z_i = z_j$ , or
  - b) there exists a continuously differentiable curve  $\vartheta_i(t, z_i)$  such that  $\vartheta_i(0, z_i) = z_i$ ,  $\vartheta_i(\delta, z_i) = z_j$  and for all  $t \in [0, \delta]$  we have  $\vartheta_i(t, z_i) \in \mathcal{I}(i)$  and  $\dot{\vartheta}_i(t, z_i) = \mathcal{F}(i)(\vartheta_i(t, z_i))$ .

The symbol  $\tau$  denotes the silent transition. We also write  $(i, z_i) \xrightarrow{(e, \delta)} (j, z_j)$  for the transitions  $(i, z_i) \xrightarrow{(e,0)} (j, z_i)$  and  $(j, z_i) \xrightarrow{(\tau, \delta)} (j, z_j)$ . Let  $E_\tau = E \cup \{\tau\}$ .

A *trajectory*  $\eta$  of the transition system  $\mathcal{T}_H$  starting from a state  $\eta_0 = (i_0, z_0) \in \mathcal{H}_0$  is an infinite sequence of states  $\eta = \eta_0 \eta_1 \eta_2 \dots$  such that for some interval  $I \subseteq \mathbb{N}$  and for all  $j \in I$  we have  $\eta_j = (i_j, z_j) \in \mathcal{H}$  and  $\eta_j \xrightarrow{(e_j, \delta_j)} \eta_{j+1}$  with  $(e_j, \delta_j) \in E_\tau \times \mathbb{R}_{\geq 0}$ . Therefore, we can consider the trajectories of the hybrid automaton  $\mathbf{H}$  or equivalently the trajectories of the system  $[\Sigma', H'_{\phi'}]$  to be functions from the positive real line  $\mathbb{R}_{\geq 0}$  to the set  $Z$  with definition  $z(t) = \vartheta_{i_j}(t - \sum_{k=0}^{j-1} \delta_k, z_{i_j})$ .

Assuming now that  $\Sigma'$  is controlled by the hybrid controller  $H'_{\phi'}$  which is constructed as described above, we can prove the following theorem.

**Theorem 4:** Let  $\phi' \in \Phi_{\Xi\Pi}$ ,  $P$  be a set of paths on  $D$  such that  $\forall p \in P. \bar{p} \models \phi'$  and  $H'_{\phi'}$  be the corresponding hybrid controller, then  $([\Sigma', H'_{\phi'}], \llbracket \cdot \rrbracket_\delta) \models \phi'$ .

*Proof:* For any  $p \in P$  we prove that if  $\bar{p} \models \phi'$ , then  $(z, \llbracket \cdot \rrbracket_\delta) \models \phi'$  for any trajectory  $z$  of the system  $[\Sigma', H'_{\phi'}]$  starting at any  $z(0) \in cl(T^{-1}(p(0)))$ . The proof uses induction on the structure of  $\phi'$ . Note that there is no finite repetition of states  $q$ , i.e. cells, in the path  $p$ . The only repetition of states that can occur is when the periodic part of the path consists of just one state which repeats indefinitely (i.e.  $n_l = 1$ ).

**Base case:** Let  $\bar{p} \models \xi$ . Since  $z(0) \in cl(T^{-1}(p(0)))$  we get that  $z(0) \in \llbracket \xi \rrbracket_\delta$  and, hence,  $(z, \llbracket \cdot \rrbracket_\delta) \models \xi$ .

**Conjunction:** Let  $\bar{p} \equiv \phi'_1 \wedge \phi'_2$ . By definition,  $\bar{p} \equiv \phi'_1$  and  $\bar{p} \equiv \phi'_2$ . Note that in this case  $H'_{\phi'_1} = H'_{\phi'_2}$  since  $p$  is the same for both subformulas. For any trajectory  $z$  starting at some  $z(0) \in cl(T^{-1}(p(0)))$  we get that  $(z, [\cdot]_\delta) \models \phi'_1$  and  $(z, [\cdot]_\delta) \models \phi'_2$  by the induction hypothesis. Therefore,  $(z, [\cdot]_\delta) \models \phi'_1 \wedge \phi'_2$ .

**Disjunction:** Let  $\bar{p} \equiv \phi'_1 \vee \phi'_2$ . By definition,  $\bar{p} \equiv \phi'_1$  or  $\bar{p} \equiv \phi'_2$ . Note that in this case  $H'_{\phi'_1} = H'_{\phi'_2}$  or  $H'_{\phi'_1} = H'_{\phi'_2}$ . For any trajectory  $z$  starting at some  $z(0) \in cl(T^{-1}(p(0)))$  we get that  $(z, [\cdot]_\delta) \models \phi'_1$  or  $(z, [\cdot]_\delta) \models \phi'_2$  by the induction hypothesis. Therefore,  $(z, [\cdot]_\delta) \models \phi'_1 \vee \phi'_2$ .

**Until:** Let  $\bar{p} \equiv \phi'_1 \mathcal{U} \phi'_2$ . Then there exists some  $i \geq 0$  such that  $\bar{p}|_i \equiv \phi'_2$  and for all  $j \in [0, i)$  we get that  $\bar{p}|_j \equiv \phi'_1$ . Consider now the trajectory  $z$  that is generated by  $\Sigma'$  using the controller  $H'_{\phi'}$  that corresponds to the path  $p$ . The initial condition for the trajectory  $z$  is any point in the initial cell, i.e.  $z(0) \in cl(T^{-1}(p(0)))$ . By construction, there exists a sequence of times  $\tau_0 \leq \tau_1 \leq \dots \leq \tau_{i-1}$ , where  $\tau_j$  for  $j \in [0, i)$  is the time that the trajectory  $z$  crosses the edge  $cl(T^{-1}(p(j))) \cap cl(T^{-1}(p(j+1)))$ . Consider any time instant  $s \in [\tau_j, \tau_{j+1}]$  for any  $j \in [0, i)$ . Then we know that  $z|_s(0) \in cl(T^{-1}(p|_j(0)))$ . Now the induction hypothesis applies and we get that  $(z|_s, [\cdot]_\delta) \models \phi'_1$ . The same holds for any time  $s$  in  $[0, \tau_0]$ . Therefore, for all  $s \in [0, \tau_{i-1}]$  we have  $(z|_s, [\cdot]_\delta) \models \phi'_1$ . Now, note that  $z(\tau_{i-1}) \in cl(T^{-1}(p(i-1))) \cap cl(T^{-1}(p(i)))$ , hence by the induction hypothesis we get that  $(z|_{\tau_{i-1}}, [\cdot]_\delta) \models \phi'_2$ . Thus, if we set  $s = \tau_{i-1}$ , then we are done and  $(z, [\cdot]_\delta) \models \phi'_1 \mathcal{U} \phi'_2$ . Note that if  $\bar{p} \equiv \phi'_2$ , then we are immediately done since the induction hypothesis applies directly.

**Release:** Let  $\bar{p} \equiv \phi'_1 \mathcal{R} \phi'_2$ . Then for all  $i \geq 0$  we have  $\bar{p}|_i \equiv \phi'_2$  or there exists  $j \in [0, i)$  such that  $\bar{p}|_j \equiv \phi'_1$ . Consider now the trajectory  $z$  that is generated by  $\Sigma'$  using the controller  $H'_{\phi'}$  that corresponds to the path  $p$ . The initial condition for the trajectory  $z$  is any point in the initial cell, i.e.  $z(0) \in cl(T^{-1}(p(0)))$ . By construction, there exists a sequence of times  $\tau_0 \leq \tau_1 \leq \tau_2 \leq \dots$ , where  $\tau_j$  for  $j \in [0, i)$  is the time that the trajectory  $z$  crosses the edge  $cl(T^{-1}(p(j))) \cap cl(T^{-1}(p(j+1)))$ . There exist two cases now.

First, assume that  $\bar{p}|_j \equiv \phi'_1$  is true for some  $j \in [0, i)$  and denote the corresponding controller by  $H'_1$ . Since there exists some time instant  $s \in [\tau_j, \tau_{j+1}]$  such that  $z(s) \in cl(T^{-1}(p(j)))$  we get by the induction hypothesis that  $(z|_s, [\cdot]_\delta) \models \phi'_1$ . Note that for all  $k \in [0, j]$  it must also be  $\bar{p}|_k \equiv \phi'_2$ . Consider any time instant  $t \in [\tau_k, \tau_{k+1}]$  for some  $k \in [0, j]$ . Then we know that  $z|_t(0) \in cl(T^{-1}(p|_k(0)))$ . Now the induction hypothesis applies and we get that  $(z|_t, [\cdot]_\delta) \models \phi'_2$ . The same holds for any time  $t$  in  $[0, \tau_0]$ . Therefore, for all  $t \in [0, \tau_j]$  we have  $(z|_t, [\cdot]_\delta) \models \phi'_2$ . We conclude that  $(z, [\cdot]_\delta) \models \phi'_1 \mathcal{R} \phi'_2$ .

Second, assume that the solution path  $p$  is given by the first part of the definition, that is by “for all  $i \geq 0$  we have  $\bar{p}|_i \equiv \phi'_2$ ”. When we generate the continuous trajectory using the corresponding hybrid controller  $H'_2$ , we get a total order on the times that the trajectory  $z$  crosses each edge, i.e.  $\tau_0 \leq \tau_1 \leq \tau_2 \leq \dots$ . Using the same reasoning as in the first case we get that for all  $t \geq 0$  we have  $(z|_t, [\cdot]_\delta) \models \phi'_2$ .

Finally, we need to consider the case for a path  $p$  whose unique periodic state repeats indefinitely. This case implies that the specification requires that for all  $i \geq 0$  we have  $\bar{p}|_i \equiv \phi'_2$ , which is part of the semantics of the release operator for a formula  $\phi' = \phi'_1 \mathcal{R} \phi'_2$ . Here,  $\phi'_2$  is a Boolean combination of propositions from  $\Xi_\Pi$ . The hybrid controller  $H'_{\phi'}$  realizes the “for all  $i \geq 0$  we have  $\bar{p}|_i \equiv \xi$ ” part of the semantics by composing a cell invariant controller. Therefore, for all  $t \geq 0$  we get that  $z(t) \in cl(T^{-1}(p(i)))$  for any  $i \geq 0$  and we conclude that



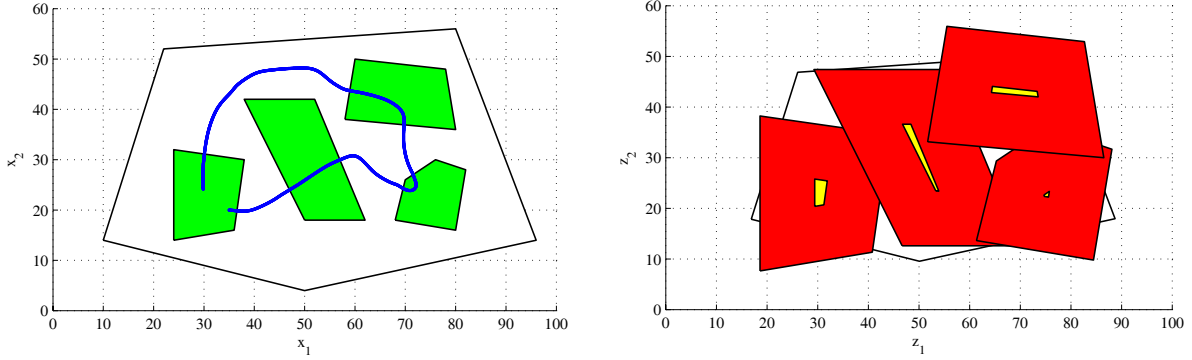


Fig. 7. The trajectory of system  $\Sigma$  which corresponds to the trajectory Fig. 8. Modified workspace for  $\nu = 2.7$ , i.e.  $\delta = 5.4$ . of system  $\Sigma'$  presented in Fig. 6.

number of iterations. Note that the more robust the system is with respect to the specification, the faster the robot can go.

Even though our framework regards as input the bound on acceleration  $\mu$  and then derives the velocity bound  $\nu$ , in the following examples we give as input the bound  $\nu$ . We believe that this makes the presentation of the examples clearer.

*Example 7:* The trajectory of system  $\Sigma$  which corresponds to the trajectory of Example 6 of system  $\Sigma'$  appears in Fig. 7. The parameters for this problem are  $\nu = 0.5$  and  $\alpha = 100$  which implies that  $\mu$  should at least be 0.5475. Notice that the two trajectories are almost identical since the velocity of  $\Sigma'$  is so low. The total computation time for this example in MATLAB - including the design of the controllers and generation of the trajectories - is about 10 sec. Figure 8 shows the modified environment for system  $\Sigma'$  when  $\nu = 2.7$ , i.e.  $\delta = 5.4$ . In this case, Problem 2 does not have a solution, i.e. the formula is unsatisfiable.

The next example considers larger velocity bounds than Example 7 and a non-terminating specification.

*Example 8:* Consider the environment in Fig. 9 and the RTL formula  $\phi = \square(\pi_0 \wedge \diamond(\pi_1 \wedge \diamond \pi_2))$ . This specification requires that the robot first visits  $\llbracket \pi_1 \rrbracket$  and then  $\llbracket \pi_2 \rrbracket$  repeatedly while always remaining in  $\llbracket \pi_0 \rrbracket$ . For this example, we use the controllers developed in [15] and for the triangulation of the environment we use the C library [42]. We consider  $\nu = 3$  and  $a = 100$ , therefore,  $\delta = 6$ . The resulting trajectories appear in Fig. 9 and 10. The dark colored region in the center of the workspace represents a static obstacle in the environment. Finally, in Fig. 11 we present the distance between the trajectories  $x(t)$  and  $z(t)$ . Notice that the distance is always bounded by 6 and that this bound is quite tight.

## VIII. RELATED RESEARCH AND DISCUSSION

There exist several related approaches to motion planning using hybrid or symbolic methods. For example, the maneuver automata in [43] generate trajectories for helicopters by composing simple dynamic maneuvers. The control quanta [44] solve the navigation problem for non-holonomic vehicles using quantized control. The motion

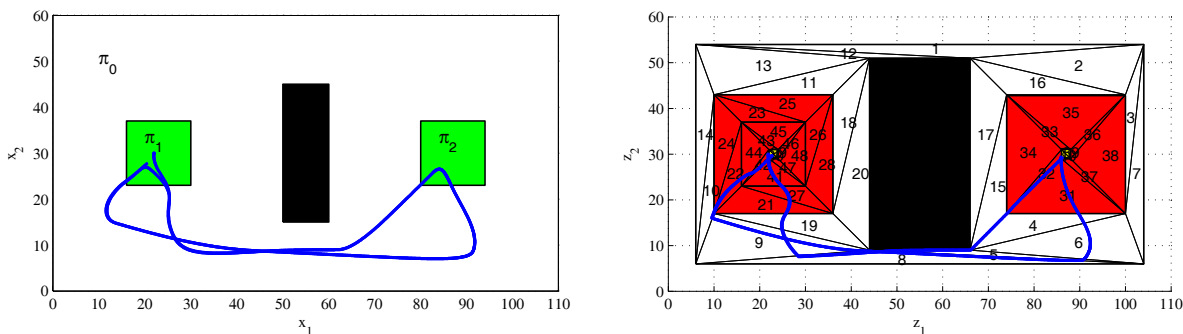


Fig. 9. The initial environment of Example 8 and the resulting Fig. 10. The modified environment of Example 8, its triangulation and the trajectory  $z(t)$  of the kinematic model  $\Sigma'$ .

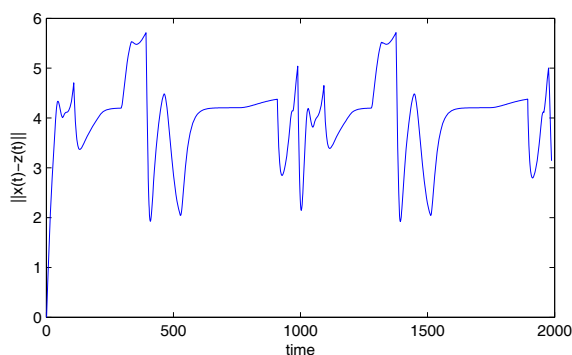


Fig. 11. The distance between the trajectories  $x(t)$  and  $z(t)$ .

description language [45] and the framework in [10] utilize regular languages in order to guide the construction of hybrid systems. In [46], the author presents a framework for the synthesis of distributed hybrid controllers for an assembly factory given basic controllers and descriptions of the tasks. Finally in [47], the authors present a hierarchical framework for the programming and coordination of robotic groups using the modelling language CHARON.

Our work fundamentally builds upon the concept of sequential composition of controllers [48], [49]. Particularly, we employ methodologies [15], [16], [38] that decompose the workspace or the state space of the robot into convex operational regions and then apply simple controllers in every such region. The advantage of these methods is that they solve the motion planning problem for point robots in complex maze-like environments. However, the deployment of controllers for second order systems [16] is not automatic and it requires user intervention.

The applicability of temporal logics in discrete event systems was advocated as far back as in 1983 [50]. Some of the first explicit applications in robotics appear in [51] and [52]. The first paper deals with the controller synthesis problem for locomotion, while the second with the synchronization of plans for multi-agent systems. In [30], the authors synthesize robust hybrid automata starting from specifications expressed in a modal logic. In [53], generators of models for LTL formulas (Büchi automata) have been utilized as supervisors of multi-robot navigation functions.

The UPPAAL model checking toolbox for timed automata has been applied to the multi-robot motion planning problem in [54], but without taking into account kinematic or dynamic models of the robots. The design of discrete time controllers that satisfy LTL specifications is addressed in [55]. In [56], controller specifications are derived from a fragment of RTL. These specifications are used to design simple motion controllers that solve the basic path planning problem : “*move from I to the goal G*”. When these controllers are composed sequentially, the desired motion is generated. More recently, the authors in [57] have demonstrated the applicability of LTL motion planning techniques for swarms of robots building upon their previous work [58].

The work that is the closest related to ours appears in [58]. The authors in [58] extend the framework presented in [14] in order to design hybrid automata with affine dynamics with drift using the controllers presented in [59]. The framework in [58] can also solve Problem 1, but we advocate that our approach has several clear advantages when one explicitly considers the motion planning problem. First, the hierarchical approach enables the design of control laws for a 2D system instead of a four dimensional one. Second, our approach avoids the state explosion problem introduced by (i) the fine partitioning of the state space with respect to the predicates, and (ii) the consequent tessellation<sup>2</sup> of the 4D space (see [58]). Finally, the freedom to choose a  $\delta$  greater than  $2\nu$  enables the design of hybrid controllers that can tolerate bounded inaccuracies in the system. For these reasons, we strongly believe that a hierarchical approach can provide a viable solution to a large class of control problems.

## IX. CONCLUSIONS AND FUTURE WORK

We have presented an automatic framework for the solution of the temporal logic motion planning problem for dynamic mobile robots. Our framework is based on hierarchical control, the notion of approximate bisimulation relations and a new definition of robustness for temporal logic formulas. In the process of building this new framework we have also derived two intermediate results. First, we presented a solution to Problem 2, i.e. an automatic framework for the solution of the temporal logic motion planning problem for kinematic models. Second, using Theorem 2, we can construct a more robust solution to Problem 2, which can account for bounded errors of measure  $\delta$  in the trajectories of the system. To the best of our knowledge, this paper presents the first computationally tractable approach to all the above problems.

Future research will concentrate on several directions. First, we are considering employing controllers for non-holonomic systems [60] at the low hierarchical level. Complementary to the first direction, we are investigating new interfaces that can take into account nonholonomic constraints. One also important direction is the extension of this framework to 3D motion planning with application to unmanned aerial vehicles [61]. Finally, we are currently working on converting our motion planning framework to a reactive system [62].

<sup>2</sup>In higher dimensions there do not exist exact space decomposition algorithms and, hence, one has to resort on such approximate partitioning techniques as the tessellation.

## ACKNOWLEDGMENT

The authors would like to thank Rajeev Alur for the useful discussions and David Conner for providing them with his implementation of the potential field controllers.

## REFERENCES

- [1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementations*. MIT Press, March 2005.
- [2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: <http://msl.cs.uiuc.edu/planning/>
- [3] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning*. Elsevier, 2004.
- [4] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 IEEE American Control Conference*, May 2002, pp. 1936–1941.
- [5] G. D. Giacomo and M. Y. Vardi, "Automata-theoretic approach to planning for temporally extended goals," in *European Conference on Planning*, ser. LNCS, vol. 1809. Springer, 1999, pp. 226–238.
- [6] F. Giunchiglia and P. Traverso, "Planning as model checking," in *European Conference on Planning*, ser. LNCS, vol. 1809, 1999, pp. 1–20.
- [7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.
- [8] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science: Formal Models and Semantics*, J. van Leeuwen, Ed. North-Holland Pub. Co./MIT Press, 1990, vol. B, pp. 995–1072.
- [9] J. P. Burgess and Y. Gurevich, "The decision problem for linear temporal logic," *Notre Dame Journal of Formal Logic*, vol. 26, no. 2, pp. 115–128, Apr. 1985.
- [10] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon, "Supervisory control of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1026 – 1049, July 2000.
- [11] P. Ohrstrom and P. F. V. Hasle, *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Springer, 1995.
- [12] T. Laureys, "From event-based semantics to linear temporal logic: The logical and computational aspects of a natural language interface for hardware verification," Master's thesis, University of Edinburgh, Nov. 1999.
- [13] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Apr. 2005, pp. 2032–2037.
- [14] —, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec. 2005, pp. 4885 – 4890.
- [15] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864– 874, October 2005.
- [16] D. C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 3546–3551.
- [17] G. J. Pappas, G. Lafferriere, and S. Sastry, "Hierarchically consistent control systems," *IEEE Trans. on Auto. Cont.*, vol. 45, no. 6, pp. 1144–1160, 2000.
- [18] P. Tabuada and G. J. Pappas, "Hierarchical trajectory generation for a class of nonlinear systems," *Automatica*, vol. 41, no. 4, pp. 701–708, 2005.
- [19] A. Girard and G. J. Pappas, "Approximation metrics for discrete and continuous systems," *IEEE Trans. Auto. Cont.*, vol. 52, no. 5, pp. 782–798, 2007.
- [20] P. Tabuada, "An approximate simulation approach to symbolic control," 2006, submitted.
- [21] A. Girard and G. J. Pappas, "Hierarchical control using approximate simulation relations," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006.
- [22] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag, 2000.
- [23] J. M. Keil, "Polygon decomposition," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. North-Holland, Amsterdam: Elsevier Science, 2000, pp. 491–518.

- [24] M. Reynolds, "Continuous temporal models," in *the 14th Australian Joint Conference on Artificial Intelligence*, ser. LNCS, vol. 2256. Springer, Dec. 2001, pp. 414–425.
- [25] G. E. Fainekos, A. Girard, and G. J. Pappas, "Hierarchical synthesis of hybrid controllers from temporal logic specifications," in *Hybrid Systems: Computation and Control*, ser. LNCS, no. 4416. Springer, 2007, pp. 203–216.
- [26] H. Barringer, R. Kuiper, and A. Pnueli, "A really abstract concurrent model and its temporal logic," in *POPL '86: Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. New York, NY, USA: ACM Press, 1986, pp. 173–183.
- [27] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. Sastry, "Dynamical properties of hybrid automata," *IEEE Transactions on Automatic Control*, vol. 48, pp. 2–17, 2003.
- [28] G. J. Pappas, "Bisimilar linear systems," *Automatica*, vol. 39, no. 12, pp. 2035–2047, December 2003.
- [29] A. van der Schaft, "Equivalence of dynamical systems by bisimulation," *IEEE Trans. Auto. Cont.*, vol. 49, pp. 2160–2172, 2004.
- [30] T. Moor and J. M. Davoren, "Robust controller synthesis for hybrid systems using modal logic," in *Proceedings of Hybrid Systems: Computation and Control*, ser. LNCS, vol. 2034. Springer, 2001, pp. 433–446.
- [31] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications," in *Proceedings of FATES/RV*, ser. LNCS, vol. 4262. Springer, 2006, pp. 178–192.
- [32] G. E. Fainekos, "An introduction to multi-valued model checking," Dept. of CIS, Univ. of Pennsylvania, Tech. Rep. MS-CIS-05-16, September 2005.
- [33] M. Reynolds, "The complexity of the temporal logic over the reals," [Online at: <http://www.csse.uwa.edu.au/~mark/research/pubs.html>].
- [34] P. Wolper, M. Vardi, and A. Sistla, "Reasoning about infinite computation paths," in *Proceedings of 24th IEEE Symposium on the Foundations of Computer Science*, 1983, pp. 185–194.
- [35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press/McGraw-Hill, Sept. 2001.
- [36] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.
- [37] C. Fritz, "Constructing Büchi automata from LTL using simulation relations for alternating Büchi automata," in *the 8th International Conference on Implementation and Application of Automata*, ser. LNCS, vol. 2759, 2003, pp. 35–48.
- [38] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec. 2005, pp. 3553–3559.
- [39] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, "Effective synthesis of switching controllers for linear systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1011–1024, 2000.
- [40] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 2, pp. 971–984, 2000.
- [41] T. A. Henzinger, "The theory of hybrid automata," in *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996, pp. 278–292.
- [42] A. Narkhede and D. Manocha, "Fast polygon triangulation based on seidel's algorithm," in *Graphics Gems V*, A. W. Paeth, Ed. Academic Press, 1995, ch. VII.5, pp. 394–397. [Online]. Available: <http://www.cs.unc.edu/~dm/CODE/GEM/chapter.html>
- [43] E. Frazzoli, "Robust hybrid control for autonomous vehicle motion planning," Ph.D. dissertation, Massachusetts Institute of Technology, May 2001.
- [44] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi, "Motion planning through symbols and lattices," in *Proceedings of the International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 3914–3919.
- [45] D. Hristu-Varsakelis, M. Egerstedt, and P. S. Krishnaprasad, "On the complexity of the motion description language MDLe," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, December 2003, pp. 3360–3365.
- [46] E. Klavins, "Automatic compilation of concurrent hybrid factories from product assembly specifications," in *Hybrid Systems: Computation and Control*, ser. LNCS, vol. 1790. Springer, 2000, pp. 174–187.
- [47] R. Alur, A. Das, J. Esposito, R. Fierro, Y. Hur, G. Grudic, V. Kumar, I. Lee, J. P. Ostrowski, G. J. Pappas, J. Southall, J. Spletzer, and C. Taylor, "A framework and architecture for multi-robot coordination," *International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 977–995, 2002.
- [48] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, June 1999.

- [49] A. Quaid and A. Rizzi, "Robust and efficient motion planning for a planar robot using hybrid control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, April 2000, pp. 4021 – 4026.
- [50] A. Fusaoka, H. Seki, and K. Takahashi, "A description and reasoning of plant controllers in temporal logic," in *Proceedings of 8th International joint Conference on Artificial Intelligence*, 1983, pp. 405–408.
- [51] M. Antonioti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1995, pp. 1441–1446.
- [52] F. Kabanza, "Synchronizing multiagent plans using temporal logic specifications," in *Proceedings of the First International Conference on Multi-Agent Systems*, V. Lesser, Ed. San Francisco, CA: MIT Press, 1995, pp. 217–224.
- [53] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Dec. 2004.
- [54] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot planning: A timed automata approach," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, April 2004, pp. 4417–4422.
- [55] P. Tabuada and G. J. Pappas, "From discrete specifications to hybrid control," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, December 2003.
- [56] G. E. Fainekos, S. G. Loizou, and G. J. Pappas, "Translating temporal logic to controller specifications," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec. 2006, pp. 899–904.
- [57] M. Kloetzer and C. Belta, "Hierarchical abstractions for robotic swarms," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006.
- [58] —, "A fully automated framework for control of linear systems from LTL specifications," in *Proceedings of Hybrid Systems: Computation and Control*, ser. LNCS, vol. 3927. Springer, 2006, pp. 333–347.
- [59] L. C. Habets and J. H. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.
- [60] D. C. Conner, H. Choset, and A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," in *Proceedings of Robotics: Science and Systems II*, Cambridge, USA, June 2006. [Online]. Available: <http://www.roboticsproceedings.org/rss02/p08.pdf>
- [61] S. Bayraktar, G. E. Fainekos, and G. J. Pappas, "Experimental cooperative control of fixed-wing unmanned aerial vehicles," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, December 2004, pp. 4292–4298.
- [62] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-Based Temporal Logic Motion Planning," in *Proceedings of the IEEE Conference on Robotics and Automation*, Apr. 2007, pp. 3116–3121.