Provably Correct Continuous Control for High-Level Robot Behaviors with Actions of Arbitrary Execution Durations

Vasumathi Raman[1], Nir Piterman[2] and Hadas Kress-Gazit[3]

Abstract— **Formal methods have recently been successfully applied to construct verifiable high-level robot control. Most approaches use a discrete abstraction of the underlying continuous domain, and make simplifying assumptions about the physical execution of actions given a discrete implementation. Relaxing these assumptions unearths a number of challenges in the continuous implementation of automatically-synthesized hybrid controllers. This paper describes a controller-synthesis framework that ensures correct continuous behaviors by explicitly modeling the activation and completion of continuous low-level controllers. The synthesized controllers exhibit desired properties like immediate reactiveness to sensor events and guaranteed safety of physical executions. The approach extends to any number of robot actions with arbitrary relative timings.**

I. Introduction

Formal methods tools have recently been applied to the construction of controllers for high-level autonomous robot behaviors such as search and rescue missions and autonomous vehicle control [1], [4], [5], [7], [13]. Traditionally, the high-level control for such tasks is hard-coded, combining low-level techniques in an ad hoc fashion. Such an implementation provides no guarantees of fulfilling the desired requirements, motivating the use of formal methods to construct controllers that do provide such guarantees.

One technique that has been successfully applied to high-level robot planning is synthesis, in which a correct-by-construction controller is automatically synthesized from a formal task specification [7], [13]. These synthesis-based approaches operate on a discrete abstraction of the robot workspace and a formal specification of the environment assumptions and desired robot behavior, and produce an automaton fulfilling the specification on this abstraction if one exists. This automaton is used to construct a hybrid controller that calls low-level continuous controllers to execute each discrete transition. Consequently, a single transition between discrete states may correspond to the simultaneous execution of several low-level controllers.

In general, a robot with multiple action capabilities will use low-level controllers that take varying amounts of time to complete. This gives rise to a number of challenges in the continuous implementation of the synthesized hybrid controller. Using the continuous execution paradigm described in [7], all controllers except motion between adjacent regions of the workspace (i.e. all "fast" actions) are assumed to have instantaneous execution. Given a discrete transition between two states with different locations, the motion controller for driving the robot between regions is activated first, and the remaining controllers are only activated (or deactivated) once the robot has crossed into the new region. However, as described in [11], there are artifacts of this continuous execution paradigm that are not modeled at the discrete level. These include a delayed response to events in the environment, and unsafe continuous executions even though the discrete automaton is provably correct.

If the execution paradigm is changed to activate the fast actions as soon as possible (before the slow actions complete), the synthesis algorithm must be changed to ensure safe continuous execution [11]. However, with this approach, the execution paradigm ignores changes in the environment once a transition has been started, until the destination state is reached. The resulting controllers are correct under the assumption that the environment does not change during the execution of a discrete transition; correctness is thus at the expense of being fully responsive to the environment.

This paper presents a new approach that ensures both immediate reactivity and continued responsiveness to the environment, for low-level controllers of unknown relative execution times. Like the approach in [11], which changed the original synthesis algorithm [9] to ensure that fast actions could complete safely before slow ones, the approach presented here changes the algorithm (albeit in an entirely different manner) to ensure safe executions while maintaining tractability. Additionally, each robot action is explicitly separated into two events: the activation of the corresponding low level controller, and the triggering of a new sensor that indicates completion of its execution. This approach is shown to solve both the aforementioned problems.

This is one of the first works to consider the safety and correctness of continuous executions of synthesized automata arising from the physical nature of the problem domain. There are a few previous approaches that tried to ensure that the provably correct discrete controller results in correct continuous executions. For example, the authors of [8] resynthesized portions of the high-level controller at execution time when changes in the physical domain make the controller execution infeasible. In contrast, this work incorporates the continuous nature of the physical execution in the synthesis process from the beginning, by modeling the

Vasumathi Raman and Hadas Kress-Gazit are supported by NSF CAREER CNS-0953365, ARO MURI (SUBTLE) W911NF-07-1-0216 and NSF ExCAPE. Nir Piterman is supported by FP7-PEOPLE-2011-IRSES MEALS.

[1]V. Raman is with the Department of Computer Science, Cornell University, Ithaca, NY, USA vraman@cs.cornell.edu

[2]N. Piterman is with the Department of Computer Science, University of Leicester, Leicester, UK nir.piterman@leicester.ac.uk

[3]H. Kress-Gazit is with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, USA hadaskg@cornell.edu

non-instantaneous nature of controller execution.

5

II. Background

This section provides an overview of the construction of provably correct robot controllers. The presented approach is implemented in the Linear Temporal Logic Mission Planning (LTLMoP) toolkit [3], which allows a specification written in a structured English grammar [6] to be transformed into a hybrid controller for use with real robots and in simulation. Example 1, adapted from [11], will serve to demonstrate the stages of synthesis for a simple high-level task.

A. Linear Temporal Logic (LTL)

Syntax: LTL formulas are defined recursively as:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \varphi,$$

where $\pi \in AP$ (a set of atomic propositions), $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is a strong "until". Conjunction ($\wedge$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$), "eventually" ($\Diamond$) and "always" ($\Box$) are derived operators.

Semantics: The truth of an LTL formula is evaluated over infinite sequences of states, corresponding to executions of a finite state machine representing the system. A state is an assignment of truth values to all propositions $\pi \in AP$. Given an infinite sequence of these states σ, the statement $\sigma \models \varphi$ denotes that σ satisfies formula φ. The statement $\sigma \models \Delta\varphi$ for $\Delta = (\bigcirc, \Box, \Diamond, \Box\Diamond)$ denotes that φ is true at the second position, at every position, at some position, and infinitely often in σ, respectively. A finite state machine A is said to satisfy φ if for every execution σ of A, $\sigma \models \varphi$. The reader is referred to [2] for a formal definition of the semantics.

B. Discrete Abstraction and Formal Specification

The relevant features of the continuous robot control problem are abstracted using a finite set of propositions consisting of:

- π_s for every sensor input s (e.g., π_{person} is true if and only if a person is sensed)
- π_a for every robot action a (e.g., π_{camera} is true if and only if the robot has turned on the camera)
- π_r for every location or region r (e.g., $\pi_{bedroom}$ is true if and only if the robot is in the bedroom).

The set of sensor propositions is denoted by $\mathcal{X}$, and the set of action and location (i.e., robot-controlled) propositions by $\mathcal{Y}$; $\mathcal{L} \subseteq \mathcal{Y}$ denotes the set of location propositions. Thus $\pi_s \in \mathcal{X}, \pi_r \in \mathcal{L}, \pi_a \in \mathcal{Y} \backslash \mathcal{L}$. The value of each $\pi \in \mathcal{X} \cup \mathcal{Y}$ is the abstracted binary state of a low-level black box component, such as a thresholded sensor value or the robot's location with respect to some partitioning of the workspace.

Example 1 *Consider a simple two-room workspace where the two adjacent locations are labeled r_1 and r_2 (corresponding to π_{r_1} and π_{r_2}). The robot has one sensor, which senses a person (represented by proposition π_{person}), and one action, which is to turn a camera on or off (π_{camera}). The robot starts in room r_1 with the camera off. When it senses a person, it must turn on the camera. Once the camera is on, it must stay on. Finally, the robot must visit room r_2 infinitely often.*

Here $\mathcal{X} = \{\pi_{person}\}$ and $\mathcal{Y} = \{\pi_{r_1}, \pi_{r_2}, \pi_{camera}\}$. A high-level task is specified on this discrete abstraction using an LTL formula over $\mathcal{X} \cup \mathcal{Y}$. Two types of properties are allowed: *safety* properties, which guarantee that "something bad never happens," and *liveness* conditions, which state that "something good (eventually) happens." This work considers tasks that can be specified using formulas of the form $\varphi_e \Rightarrow \varphi_s$, where φ_s represents the desired robot behavior, and φ_e encodes assumptions on the environment, including events external to the robot as well as the robot's internal state, as perceived by its sensors. φ_e and φ_s each contain initial conditions (φ_e^i, φ_s^i), safety requirements (φ_e^t, φ_s^t) and liveness conditions (φ_e^g, φ_s^g) for the environment and robot, respectively. Some approaches also provide a more user-friendly specification language and accompanying parser, to allow users unfamiliar with LTL to write specifications [6].

Since the robot can be in exactly one location at any given time, the formula $\varphi_r = \pi_r \wedge \bigwedge_{r' \neq r} \neg\pi_{r'}$ is used to represent the robot being in region r. The robot's motion in the workspace is governed by adjacency of regions, and the availability of controllers to drive it between adjacent regions. In LTLMoP, the adjacency relation is automatically encoded as a logic formula φ_{trans}. The adjacency relation for Example 1 is

$$\varphi_{trans} = \Box(\varphi_{r_1} \Rightarrow \bigcirc \varphi_{r_1} \vee \bigcirc \varphi_{r_2}) \wedge \Box(\varphi_{r_2} \Rightarrow \bigcirc \varphi_{r_2} \vee \bigcirc \varphi_{r_1}).$$

The task in Example 1 corresponds to the following LTL specification:

$(\varphi_{r_1} \wedge \neg\pi_{camera})$ #Initial
(Robot starts in region r1 with the camera off)
$\wedge \quad \Box(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$ #Safety
(Activate the camera if you see a person)
$\wedge \quad \Box((\pi_{camera} \Rightarrow \bigcirc \pi_{camera}))$ #Safety
(Camera stays on once turned on)
$\wedge \quad \Box\Diamond(\pi_{r_2})$ #Liveness
(Go to r2 infinitely often)

C. Synthesis

An LTL formula φ is *realizable* if there exists a finite state strategy that, for every finite sequence of truth assignments to sensor propositions, gives an assignment to the robot propositions such that every infinite sequence of truth assignments to both sets of propositions generated in this manner satisfies φ. The synthesis problem is to find a finite state machine encoding this strategy, i.e. whose executions satisfy φ.

Synthesizing an automaton that realizes an arbitrary LTL formula is doubly exponential in the size of the formula [10]. When restricted to LTL formulas of the form $\varphi_e \Rightarrow \varphi_s$ described above, the algorithm introduced in [9] permits synthesis in time polynomial in the size of the state space, while still capturing a large number of tasks specified in practice. Fig. 1 depicts the automaton synthesized for the above specification using this synthesis algorithm. Each state of the automaton is labeled with the subset of location and action propositions that are true in that state, and each tran-

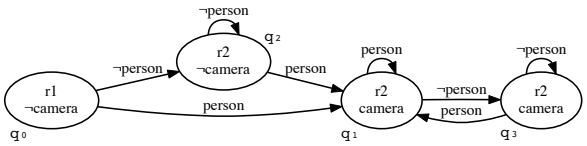


Fig. 1: Synthesized automaton for Example 1. States are labeled with the location and action propositions that are true in that state. Transitions are labeled with the sensor propositions that are true for that transition to be enabled.

sition is labeled with the sensor propositions that are true for that transition to be enabled. Incoming transitions therefore also determine the truth value of the sensor propositions for each state. The labels $\mathtt{r_i}, \mathtt{camera}$ and $\mathtt{person}$ represent π_{r_i}, π_{camera} and π_{person} respectively.

III. Continuous Execution

When a specification is realizable, synthesis yields an automaton that implements the specification in a discrete abstraction of the problem. If no such automaton exists, the user is presented with information about the cause of the unrealizability [12]. If an automaton is obtained, a controller that implements the corresponding continuous behavior is constructed by viewing the automaton as a hybrid controller, with a transition between two states achieved by the activation of one or more low-level continuous controllers corresponding to each robot proposition.

This section describes two approaches to physical execution of the synthesized automaton, and the challenges and shortcomings of each. Consider the workspace and task described in Example 1. Suppose the robot starts in room r_1, with its camera turned off and no persons sensed (so it is in the initial state q_0 in Fig. 1). Suppose it then senses a person. The safety condition $\square(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$ requires it to turn on the camera. In order to fulfill its patrol goal, it will also try to go to room r_2. So the discrete transition in the automaton generated by the synthesis algorithm in [9] will be to state q_1. To execute the transition (q_0, q_1) at the continuous level, a motion controller and a controller for turning on the camera must both be invoked.

A. Assuming Instantaneous Actions Except Motion [7]

Under the continuous execution paradigm in [7], all controllers except motion between adjacent regions (i.e. all fast actions) are assumed to have instantaneous execution. Given a discrete transition between two states with different locations, the motion controller for driving the robot between regions is activated first, and the remaining controllers only activated (or deactivated) once the robot has crossed into the new region. Thus, to execute the transition (q_0, q_1), the hybrid controller constructed for Example 1 first activates the controller for moving from r_1 to r_2, and only once that boundary has been crossed will it activate the (instantaneous) controller for turning on the camera, completing the discrete transition (q_0, q_1). This leads to a perceived delay in the robot turning on the camera in response to seeing a person.

B. Assuming Slow and Fast Actions [11]

Under the approach in [11], on the other hand, the robot's actions are grouped into two sets based on execution duration, and all actuator controllers are executed simultaneously. Thus, to execute the transition (q_0, q_1), the hybrid controller constructed for Example 1 activates the controller for turning on the camera (fast) simultaneously with that for moving from r_1 to r_2 (slow). The transition (q_0, q_1) is completed only when the motion completes. However, the execution paradigm ignores changes in the environment once a transition has begun (i.e. following activation of a fast controller), until the destination is reached. This approach therefore produces controllers that are correct under the assumption that the environment does not change during the execution of a discrete transition, and will ignore any such changes.

Note that in this execution, the camera will turn on before the motion completes, and the robot is still in r_1, effectively resulting in an intermediate discrete state. To ensure safety of the continuous execution, the synthesis algorithm was changed to ensure that all such intermediate states are safe [11]. The automaton produced by the modified synthesis algorithm contains only those transitions for which the intermediate states are safe.

IV. Problem Formulation

The chief drawback of the continuous execution paradigm in III-A is delayed reactivity. It is usually desirable to respond to sensor inputs in a "greedy" manner, i.e. as soon as possible; for example, the camera should be turned on as soon as a person is sensed, regardless of the other actions to be performed. However, with the approach to continuous execution in III-A, the robot will not turn on its camera until the transition to r_2 has been completed. As described in III-B, this approach may also result in unsafe continuous executions even though the discrete automaton is provably correct.

Furthermore, during the transition (q_0, q_1), if the person disappears before the motion completes, the transition is aborted, and the execution returns to q_0 (and then to an alternate successor state according to the automaton). Note that this causes the person to be ignored completely. On the other hand, if the camera is activated simultaneously with the motion, execution will pass through an intermediate state in which the fast action (camera) has completed but the slow action (motion) has not. If the robot safety formula required that the camera never be turned on in r_1, the execution resulting from simultaneous activation of the controllers would be unsafe even though the discrete solution is safe.

The correctness of the controllers generated in III-B, in comparison, relies on the assumption that the environment does not change during the execution of a discrete transition. Consider again the transition (q_0, q_1) in Fig. 1 where the camera will be turned on at the same time as the motion but will complete before the robot has reached r_2. Suppose the robot stops sensing a person after the camera has been turned on, but before reaching r_2. Then the transition (q_0, q_1) will be aborted, and (q_0, q_2) will be taken instead. This results in the

camera going from on to off, violating the safety condition that enforces persistence of the camera action.

In the above example, the execution paradigm proposed in III-B will ignore the disappearance of a person after the camera has turned on. Correctness is therefore at the expense of being fully responsive to the environment during the time taken to move between regions. Additionally, the approach in III-B assumes a known ordering on action completion times, reducing the number of intermediate states to be checked; it extends to an unknown ordering with an exponential blow-up in the number of intermediate states considered.

The shortcomings of the two existing approaches lead to the following problem, which this paper solves.

Problem 1 *Given a specification φ and a set of actions with unknown relative completion times, construct an automaton such that every continuous execution satisfies φ while allowing immediate reactivity as well as continual responsiveness to changes in the environment.*

V. Solution

This section proposes a solution to Problem 1 for arbitrary (but finite) action completion times. To account for the non-instantaneous execution of continuous controllers, each robot action is viewed as the *activation* of the corresponding low level controller, and a new sensor proposition is introduced in the discrete model to indicate whether the controller has completed execution.

A. Discrete Abstraction

The set of propositions is now modified to include:

- π_s for each sensor input s (e.g., π_{person} is true if and only if a person is sensed)
- π_a for the *activation* of each robot action a (e.g., π_{camera} is true if and only if the robot has activated the controller to turn on the camera). Similarly, $\neg\pi_a$ represents the activation of the controller for turning a off.
- π_r for the *initiation* of motion towards each region r (e.g., $\pi_{bedroom}$ is true if and only if the robot is trying to move to the bedroom). φ_r is defined as in Section II.
- π_a^c, π_r^c for the *completion* of the controller for turning action a on, or motion to region r (e.g., $\pi_{bedroom}^c$ is true if and only if the robot has arrived in the bedroom, and π_{camera}^c is true if and only if the camera has finished turning on). $\neg\pi_a^c$ represents the completion of the controller for turning action a off.[1]

Action/motion completion is modeled as an event sensed by the robot, and therefore $\mathcal{X} = \{\pi_a^c\} \cup \{\pi_r^c\} \cup \{\pi_s\}, \mathcal{Y} = \{\pi_a\} \cup \{\pi_r\}$. For Example 1, $\mathcal{X} = \{\pi_{person}, \pi_{r_1}^c, \pi_{r_2}^c, \pi_{camera}^c\}$ and $\mathcal{Y} = \{\pi_{r_1}, \pi_{r_2}, \pi_{camera}\}$.

B. Formal Specification Transformation

Given this discrete abstraction, the task specification now governs which actions can be activated by the robot, and how the action-completion sensors behave.

[1]Note that this paper considers actions other than motion to have on and off modes only, but the approach extends to other types of actions.

1) Proposition replacement in original specification: A specification $\varphi = (\varphi_e \Rightarrow \varphi_s)$ in [7] is modified as follows:

- Initial conditions specify the sensed state of the robot, so every occurrence of π_a in φ_s^i is replaced with π_a^c.
- Robot goals are predicated on the completion of actions (as sensed by the corresponding sensor). So every occurrence of π_a in φ_s^g is replaced with π_a^c. Similarly, robot goals refer to the sensed location π_r^c rather than just the activation of the motion controller π_r.
- Robot safety conditions govern which controllers are to be activated in response to events in the environment, and may refer to the sensing of action completion as well as events in the environment. The user input language, such as that presented in [6], must therefore allow distinguishing between a reference to π_a and π_a^c in safety conditions, as discussed in VI-B.

The resulting LTL specification is denoted $\varphi' = \varphi_e' \Rightarrow \varphi_s'$. Table I presents the LTL formulas corresponding to the specification for Example 1, provided in Section II, before and after proposition replacement.

2) Robot Transition Relation: The allowed robot motion now depends on the sensed location. Given a region r, let $Adj(r)$ denote the regions adjacent to r (including r itself). φ_{trans} II-B then changes as follows:

$$\varphi_{trans}' = \bigwedge_r \Box(\pi_r^c \Rightarrow \bigvee_{r' \in Adj(r)} \bigcirc \varphi_{r'})$$

For Example 1,

$$\varphi_{trans}' = \Box(\pi_{r_1}^c \Rightarrow (\bigcirc \varphi_{r_1} \vee \bigcirc \varphi_{r_2})) \wedge \Box(\pi_{r_2}^c \Rightarrow (\bigcirc \varphi_{r_2} \vee \bigcirc \varphi_{r_1}))$$

3) Sensor Assumptions: Additional assumptions define the effects of the robot activating its various controllers.

$$\begin{aligned}
\varphi_{eNew} \quad = \quad & \Box(\pi_r^c \Leftrightarrow \bigwedge_{r' \neq r} \neg\pi_{r'}^c) && (1)\\
& \wedge \bigwedge_r \bigwedge_{r' \in Adj(r)} \Box(\pi_r^c \wedge \varphi_{r'} \Rightarrow (\bigcirc \pi_r^c \vee \bigcirc \pi_{r'}^c)) && (2)\\
& \wedge \bigwedge_a \Box(\pi_a^c \wedge \pi_a \Rightarrow \bigcirc \pi_a^c) && (3)\\
& \wedge \bigwedge_a \Box(\neg\pi_a^c \wedge \neg\pi_a \Rightarrow \bigcirc \neg\pi_a^c) && (4)
\end{aligned}$$

Conjunct (1) enforces mutual exclusion between the physical locations of the robot. Conjunct (2) governs how the robot's location can change in a single time step in response to the activation of the motion controllers. Conjuncts (3-4) govern the completion of other actions in response to the activation of the corresponding controllers. In Example 1,

$$\begin{aligned}
\varphi_{eNew} \quad = \quad & \Box(\pi_{r_1}^c \Leftrightarrow \neg\pi_{r_2}^c) && (5)\\
& \wedge \Box(\pi_{r_1}^c \wedge \varphi_{r_1} \Rightarrow \bigcirc \pi_{r_1}^c) && (6)\\
& \wedge \Box(\pi_{r_1}^c \wedge \varphi_{r_2} \Rightarrow \bigcirc \pi_{r_1}^c \vee \bigcirc \pi_{r_2}^c) && (7)\\
& \wedge \Box(\pi_{r_2}^c \wedge \varphi_{r_2} \Rightarrow \bigcirc \pi_{r_2}^c) && (8)\\
& \wedge \Box(\pi_{r_2}^c \wedge \varphi_{r_1} \Rightarrow \bigcirc \pi_{r_2}^c \vee \bigcirc \pi_{r_1}^c) && (9)\\
& \wedge \Box(\pi_{camera}^c \wedge \pi_{camera} \Rightarrow \bigcirc \pi_{camera}^c) && (10)\\
& \wedge \Box(\neg\pi_{camera}^c \wedge \neg\pi_{camera} \Rightarrow \neg \bigcirc \pi_{camera}^c) && (11)
\end{aligned}$$

English specification	Original LTL (φ)	New LTL (φ')
Robot starts in region r_1 with the camera off	$\varphi_{r_1} \wedge \neg \pi_{camera}$	$\pi^c_{r_1} \wedge \neg \pi^c_{camera}$
Activate the camera if you see a person	$\square(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$	$\square(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$
Camera stays on once turned on	$\square((\pi_{camera} \Rightarrow \bigcirc \pi_{camera}))$	$\square((\pi^c_{camera} \Rightarrow \bigcirc \pi_{camera}))$
Go to r2 infinitely often	$\square\diamondsuit(\pi_{r_2})$	$\square\diamondsuit(\pi^c_{r_2})$

TABLE I: Replacing propositions in the task specification for Example 1

For example, conjunct (7) states that if the robot is in r_1 (i.e. $\pi^c_{r_1}$ is true) and is activating the controller to move to r_2 (i.e. φ_{r_2}), then in the next time step, the robot is either still in r_1 ($\pi^c_{r_1}$ is true) or has reached r_2 ($\pi^c_{r_2}$ is true). Conjunct (10) states that if the camera is already on and is supposed to be on, it will stay on.

4) Fairness conditions: In addition to the above safety conditions, additional constraints on the environment are required to ensure that every action/motion eventually completes, i.e. that the robot's environment is in some sense "fair". A naive approach is to add an environment assumption that every controller activation or deactivation eventually results in completion, i.e. the fairness conditions $\square\diamondsuit(\pi_a \Rightarrow \bigcirc \pi^c_a)$ and $\square\diamondsuit(\neg\pi_a \Rightarrow \bigcirc \neg\pi^c_a)$ for every action a.

However, this adds two fairness assumptions to the specification for every action. Since the synthesis algorithm scales linearly with the number of assumptions, it is important to minimize added assumptions. This is achieved by introducing a single fairness condition that incorporates the possibility that the robot is forced to "change its mind" by events in the environment. To this end, two new Boolean formulas $\varphi_a^{completion}$ and φ_a^{change} are defined for each action a:

$$\varphi_a^{completion} = (\pi_a \wedge \bigcirc \pi^c_a) \vee (\neg\pi_a \wedge \neg \bigcirc \pi^c_a))$$
$$\varphi_a^{change} = (\pi_a \wedge \neg \bigcirc \pi_a) \vee (\neg\pi_a \wedge \bigcirc \pi_a))$$

$\varphi_a^{completion}$ holds when the activation (or de-activation) of the controller for a has completed execution. φ_a^{change} captures the robot changing its mind (such as by toggling the camera). A single pair of formulas $\varphi_{loc}^{completion}, \varphi_{loc}^{change}$ suffices for motion since locations are mutually exclusive and the robot cannot try to move to two locations at once:

$$\varphi_{loc}^{completion} = \bigvee_r(\varphi_r \wedge \bigcirc \pi^c_r)), \qquad \varphi_{loc}^{change} = \bigvee_r(\varphi_r \wedge \neg \bigcirc \varphi_r))$$

The complete fairness assumption added is:

$$\varphi^a_{fair} = \square\diamondsuit(\varphi_a^{completion} \vee \varphi_a^{change})$$

Note that every execution satisfying both fairness conditions described above for activation and deactivation also satisfies this fairness condition. Moreover, there is only one such assumption added for each action a (in Example 1, there is one such assumption for the camera). Additionally, there is one assumption φ^{loc}_{fair} for motion. For Example 1,

$$\begin{aligned}\varphi^{camera}_{fair} &= \square\diamondsuit[(\pi_{camera} \wedge \bigcirc \pi^c_{camera}) \vee (\neg\pi_{camera} \wedge \neg \bigcirc \pi^c_{camera}) \\ &\quad \vee (\pi_{camera} \wedge \neg \bigcirc \pi_{camera}) \vee (\neg\pi_{camera} \wedge \bigcirc \pi_{camera})] \\ \varphi^{loc}_{fair} &= \square\diamondsuit[(\pi_{r_1} \wedge \bigcirc \pi^c_{r_1}) \vee (\pi_{r_2} \wedge \bigcirc \pi^c_{r_2}) \\ &\quad \vee (\pi_{r_1} \wedge \neg \bigcirc \pi_{r_1}) \vee (\pi_{r_2} \wedge \neg \bigcirc \pi_{r_2})]\end{aligned}$$

Given a task specification $\varphi = (\varphi_e \Rightarrow \varphi_s)$, the LTL specification used to synthesize a controller (after proposition replacement, changing the robot transition relation, and adding sensor assumptions and fairness conditions) is now

$$\varphi_{new} = \varphi'_e \wedge \varphi_{eNew} \wedge \textstyle\bigwedge_a \varphi^a_{fair} \wedge \varphi^{loc}_{fair} \Rightarrow \varphi'_s \wedge \varphi'_{trans}$$

C. Synthesis

Since the formulas $\varphi_a^{completion}$ and φ_a^{change} in the proposed liveness condition φ^a_{fair} govern both current and next time steps, the original synthesis algorithm in [9] cannot be applied as-is to synthesize an implementing automaton for the specification. Liveness conditions that incorporate temporal formulas with both current and next time steps are handled by changing the computation of the set of robot-winning strategies in the synthesis algorithm [9].

Note that it is possible to use the original synthesis algorithm (which only allows simple Boolean formulas in liveness conditions) to synthesize a controller by introducing a new proposition, $\pi_a^{completion,change}$, and the safety condition

$$\square(\bigcirc \pi_a^{completion,change} \iff (\varphi_a^{completion} \vee \varphi_a^{change}))$$

This allows the additional liveness to instead be written as

$$\varphi'_{fair} = \square\diamondsuit \pi_a^{completion,change}$$

However, this introduces one new proposition per robot action. The complexity of the synthesis algorithm scales with the size of the state space, which in turn scales exponentially with the number of propositions. In addition, extra propositions result in larger automata that are harder to read.

Even with the above change to the synthesis algorithm, one environment proposition must be added per robot action (corresponding to the sensor for action completion). In the worst case, the time taken for synthesis is therefore still increased by a factor of $2^{|\mathcal{Y}|}$ over the original approach. With the original synthesis algorithm, the increase is by a factor of $4^{|\mathcal{Y}|}$ (since two new propositions are required per action).

D. Continuous Execution

Given a state q, observed sensor values $X \subset \mathcal{X}$ and the corresponding next state q' in the automaton, the transition (q, q') is executed by simultaneously invoking the controllers corresponding to every action or location proposition π_a that changes value from q to q'. Note that the current sensed state of the system, as represented by X, determines which actions a can be activated in q'. Transitions in the automaton are instantaneous, as they correspond to activation or deactivation of controllers, but the controllers themselves may take several discrete transitions to complete execution.

The resulting controller exhibits the desired properties:

- actions are executed immediately in response to sensor events, eliminating the delayed reactivity of [7]

- safety of continuous executions is guaranteed even when the robot changes its mind
- the approach extends to any number of robot actions, with arbitrary relative timings, although the computational burden increases for a large number of actions.

VI. Test Cases

This section provides test cases illustrating the effectiveness of the proposed solution. The robot controllers for the examples presented were synthesized using LTLMoP.

A. Safety of physical execution

Fig. 3 depicts an excerpt of the automaton synthesized for Example 1. The full automaton has 7 states and is omitted for conciseness. Negated sensor labels are omitted from the transitions for clarity. The label `c_ri` represents $\pi^c_{r_i}$.

In state q_0, the robot is in r_2 (as indicated by $\pi^c_{r_2}$ being true on the incoming transitions) with the camera off (π^c_{camera} is false), and is trying to go to r_1 (as indicated by the action π_{r_1} being true). Consider the transition (q_0, q_1). The robot reaches r_1 (indicated by $\pi^c_{r_1}$ being true on the transition into q_1. It is now trying to go to r_2, indicated by π_{r_2}. In q_1, if the robot does not sense a person, it either moves back to q_0 or stays in q_1 depending on whether it has reached r_2 yet. On the other hand, suppose the robot senses a person in q_1 before it has reached r_2, the execution moves to state q_3, where the robot is still trying to get to r_2 but is now additionally activating the camera. On the other hand, the transition (q_1, q_2) is taken if the robot senses a person at the same time as it reaches r_2 (as indicated by sensor proposition $\pi^c_{r_2}$ being true on that transition).

Note that in the continuous execution of the above automaton, all the controllers are invoked at the same time. For example, when transitioning from q_1 (where the robot is in r_1) to q_3, the controller for moving to r_2 and for turning on the camera are being activated simultaneously. Any difference in their completion times is captured by the corresponding sensor propositions. Even if the person disappears before the motion from r_1 to r_2 is completed, the transition (q_1, q_3) is still taken (followed by a transition out of q_3 that corresponds to the person not being seen), and the camera is still being activated in q_3. This ensures that the person is not ignored, since there is an explicit state representing the fact that the camera is being turned on even though the person has disappeared.

B. Activation and Completion Dependent Safety Properties

Desired safety properties can now also be defined in terms of which actions have completed, rather than which actions are activated. For example, when requiring that the camera not be turned on in r_1, the added system safety can be either $\square(\pi^c_{r_1} \Rightarrow \neg \pi^c_{camera})$ or $\square(\pi^c_{r_1} \Rightarrow \neg \bigcirc \pi_{camera})$. The first of these enforces the requirement that the camera not physically be on while the robot is still in r_1, whereas the second states that the controller for turning on the camera will not be activated while the robot is in r_1. Note that both specifications will produce the same observed behavior, but the first one allows the robot to try to turn on the camera in r_1, whereas the second does not.

Recall from Section IV the challenge of ensuring safe execution when transitions are aborted due to changes in the environment. The safety condition enforcing that the camera stays on once turned on (in Example 1) can be expressed in terms of controller activation and completion, as either $\square(\pi_{camera} \Rightarrow \bigcirc \pi_{camera})$ or $\square(\pi^c_{camera} \Rightarrow \bigcirc \pi_{camera})$. The first of these prevents the robot from toggling π_{camera} (e.g., by flicking the camera switch on and off). The second prevents the robot from trying to turn the camera off once it has completely turned on; toggling π_{camera} is allowed until the camera has actually turned on. Similarly, the specification can be fine-tuned to distinguish between continuous controllers that can be aborted and those that cannot.

C. Unrealizability due to physical execution

Example 2 *Consider the workspace depicted in Fig. 2(a). The robot starts in r_1 and has to visit r_4. However, if it sees a stop sign in either r_2 or r_3, it cannot pass through that room. A safety assumption on the environment guarantees that there will never be stop signs in both r_2 and r_3 at the same time. There are initially no stop signs.*

Given this specification, one might expect an implementing controller that drives the robot from r_1 to r_4 via whichever room (r_2 or r_3) does not have a stop sign. However, if a stop sign appears while the robot is driving to this room, the robot has to turn around and try the other room. If this happens every time the robot starts moving towards r_4, it will never be able to reach r_4. With the approach in [7], the specification for Example 2 is:

$\wedge$ $\neg \pi_{stop\_sign\_in\_r_2} \wedge \neg \pi_{stop\_sign\_in\_r_3}$ #Env Initial
(Env starts with no stop sign in either r2 or r3)

$\wedge$ $\square(\neg(\pi_{stop\_sign\_in\_r_3} \wedge \pi_{stop\_sign\_in\_r_2}))$ #Env Safety
(There will never be stop signs in both r2 and r3)

$\Rightarrow$

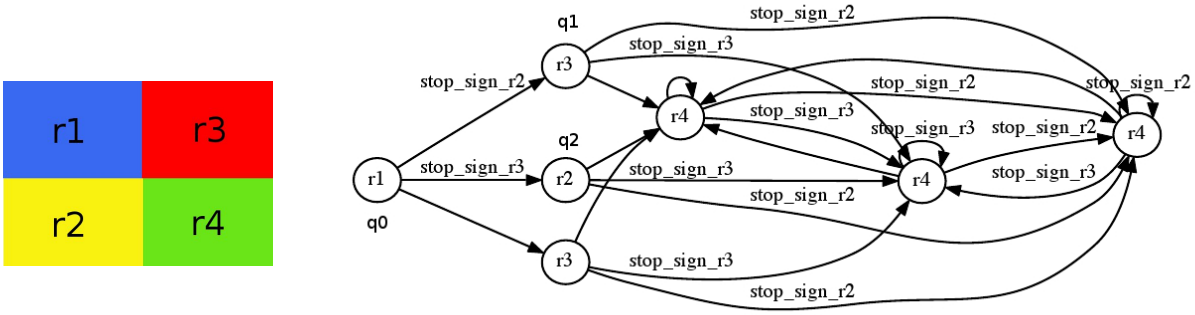
φ_{r_1} #Robot Initial
(Robot starts in r1)

$\wedge$ $\square(\bigcirc \pi_{stop\_sign\_in\_r_2} \Rightarrow \bigcirc \neg \varphi_{r_2})$ #Robot Safety
(Do not go to r2 if you sense a stop sign in r2)

$\wedge$ $\square(\bigcirc \pi_{stop\_sign\_in\_r_3} \Rightarrow \bigcirc \neg \varphi_{r_3})$ #Robot Safety
(Do not go to r3 if you sense a stop sign in r3)

$\wedge$ $\square \diamondsuit (\pi_{r_4})$ #Robot Liveness
(Visit r4 infinitely often)

This specification is realizable via the approach in [7], and the synthesized automaton is depicted in Fig. 2(b). However, consider what happens when the robot is in state q_0 (where it is in r_1), and sees a stop sign in r_2. The robot will start to move towards r_3 (and state q_1). Suppose that before the robot has entered r_3, the stop sign in r_2 disappears but one appears in r_3. The robot will abort the discrete transition (q_0, q_1) and start heading to r_2 to take the transition (q_0, q_2) instead. If the stop sign's location changes faster than the robot can move, the robot will be trapped in r_1, because it will keep changing its mind between the above two discrete



(a) Workspace for Example 2

(b) Synthesized automaton using approach in [7]

Fig. 2: Workspace and original automaton for Example 2. Negated sensor labels are omitted from the transitions for clarity.

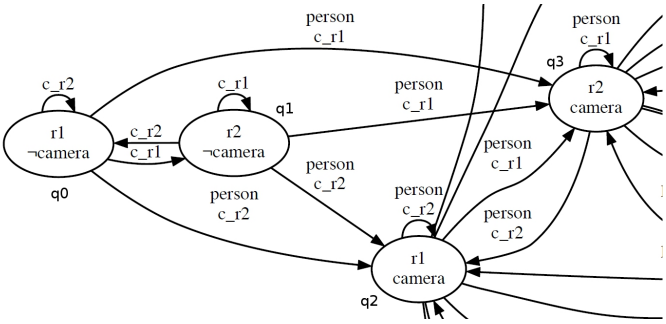


Fig. 3: Excerpt of automaton synthesized for Example 1 with new approach. Negated propositions on transitions are omitted for clarity.

transitions. This is therefore an example of a high-level task that produces a controller under the synthesis approach of [7], but whose physical execution does not accomplish the specified behaviour because of an inadequate modeling of the underlying physical system.

With the new discrete abstraction, task specification transformation and execution paradigm presented in this paper, the robot initial condition in the above specification changes to $\pi^c_{r_1}$, and the robot goal becomes $\square\diamondsuit(\pi^c_{r_4})$. This specification (with the additional formulas introduced in Section V) is unrealizable, and no automaton is obtained. As noted above, this the safer, more desirable outcome, since there exists an environment strategy that toggles the stop signs between r_2 and r_3 and prevents the robot from fulfilling the specification.

Recent work has tackled the question of analyzing specifications that are unrealizable [12]. Future research will explain cases of unrealizability arising from the relative execution times of the low-level controllers, and present users with this information in a useful manner. An additional direction to investigate is the automatic addition of environment assumptions to make the specification synthesizable. For example, in Example 2, adding the environment liveness $\square\diamondsuit(\pi^c_{r_4})$ results in a controller, by explicitly requiring the environment to eventually let the robot through to r_4.

VII. Conclusions

This paper describes a discrete abstraction, specification transformation, synthesis algorithm and execution paradigm that ensures correct continuous behaviors with immediate reactivity and continued responsiveness. By explicitly modeling the activation and completion of the continuous low-level controllers, the approach is able to handle actions of arbitrary relative execution times, solving problems that escape previous approaches. The synthesis algorithm is changed to lower the computational overhead of the controller construction. Future work includes analyzing specifications that are unrealizable under the new paradigm due to the relative execution times of the low-level controllers, and presenting users with this information in a useful manner.

References

[1] Amit Bhatia, Lydia E. Kavraki, and Moshe Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.

[2] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[3] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. LTLMoP: Experimenting with language, temporal logic and robot control. In *IROS*, pages 1988 – 1993, 2010.

[4] Karaman and Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *CDC*, Shanghai, China, December 2009.

[5] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transaction on Automatic Control*, 53(1):287–297, 2008.

[6] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

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

[8] Scott C. Livingston, Richard M. Murray, and Joel W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *ICRA*, pages 5163–5170, 2012.

[9] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380. Springer, 2006.

[10] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, New York, NY, USA, 1989.

[11] Vasumathi Raman, Cameron Finucane, and Hadas Kress-Gazit. Temporal logic robot mission planning for slow and fast actions. In *IROS*, pages 251–256, 2012.

[12] Vasumathi Raman and Hadas Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*, PP(99):1 –11, 2012.

[13] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Hybrid Systems: Computation and Control*, pages 101–110, 2010.