Timing Semantics for Abstraction and Execution of Synthesized High-Level Robot Control

Vasumathi Raman, Nir Piterman, Cameron Finucane, Hadas Kress-Gazit

***Abstract*—The use of formal methods for synthesis has recently enabled the automated construction of verifiable high-level robot control. Most approaches use a discrete abstraction of the underlying continuous domain, and make assumptions about the physical execution of actions given a discrete implementation; examples include when actions will complete relative to each other, and possible changes in the robot's environment while it is performing various actions. Relaxing these assumptions gives rise to a number of challenges during the continuous implementation of automatically-synthesized hybrid controllers. This paper presents several distinct timing semantics for controller synthesis, and compares them with respect to the assumptions they make on the execution of actions. It includes a discussion of when each set of assumptions is reasonable, and the computational trade-offs inherent in relaxing them at synthesis time.**

***Index Terms*—high-level behaviors, formal methods, temporal logic synthesis.**

I. Introduction

ROBOTICS has recently seen the successful application of formal methods to the construction of controllers for high-level autonomous behaviors, including reactive conditions and repeated goals [1]–[5]. Applications that involve such behaviors include search and rescue missions and autonomous vehicle control. Controllers for these tasks are traditionally constructed by hard-coding the high-level behaviors and combining low-level techniques like motion planning ad hoc. Such an implementation is not guaranteed to fulfill the desired requirements, motivating the use of formal methods to construct controllers that do provide guarantees.

One technique that has been successfully applied to high-level robot planning is Linear Temporal Logic (LTL) synthesis, in which a correct-by-construction controller is automatically synthesized from a formal task specification [4], [5]. Synthesis-based approaches operate on a discrete abstraction of the robot workspace and a formal specification of the environment assumptions and desired robot behavior in LTL. Synthesis algorithms automatically construct an automaton guaranteed to fulfill the specification on the discrete abstraction (if such an automaton exists). The automaton is then used to create a hybrid controller that calls low-level continuous controllers corresponding to each discrete transition. During the execution of this hybrid controller, a single transition between discrete states in the automaton may correspond to the simultaneous execution of several low-level controllers.

Consider an Aldebaran Nao [6] humanoid robot, whose available actions include motion of the arm (waving), a text-to-speech interface, and walking; walking between regions of interest takes significantly longer than the other actions. In the discrete abstraction of the above problem, the robot's state encodes its current location and whether it is waving or speaking. In general, a robot with multiple action capabilities will use low-level controllers that take varying amounts of time to complete. When reasoning about correctness, most approaches make assumptions about the physical execution of actions given a discrete implementation, such as when actions will complete relative to one another, and possible changes in the robot's environment while it is performing various actions. Relaxing these assumptions begets a host of challenges in the continuous implementation of automatically-synthesized hybrid controllers.

Extending the semantics of discrete-time temporal logic formulas to continuous or hybrid dynamics requires either careful definition of new semantics (see e.g. [7]) or a finite abstraction of the infinite time and state space, such as with the use of bisimulation relations [8] to define equivalence classes over states. Bisimulation relations have been proposed for various timed automata models of hybrid systems, including multi-rate and rectangular automata (see [8] for an overview of these methods). These bisimulations are time-abstract, in the sense that they do not explicitly encode time as a continuous variable. The authors of [9] show that time-abstract bisimulation is insufficient for controlling systems with general dynamics, and propose an alternative based on suffix-equivalence for trajectories, which produces finer but still finite abstractions.

We examine the correctness of continuous executions in the physical problem domain during discrete-time temporal logic control synthesis, using a discrete abstraction based on time-abstract bisimulations. Our framework addresses specifications in the Generalized Reactivity (1) (GR(1)) [10] fragment of LTL, which captures a large number of high-level tasks specified in practice. Related works such as [2], [11] restrict LTL specifications to the fragment which eliminates the "next" operator in order to define satisfaction of a formula over a continuous trajectory. In contrast, we permit the "next" operator, which allows us to specify immediate reactivity,

Manuscript received March 10, 2015. Vasumathi Raman is supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Cameron Finucane 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.

V. Raman is with the Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena CA 91125 (e-mail: vasu@caltech.edu).

N. Piterman is with the Department of Computer Science, Leicester University, Leicester, UK (e-mail: nir.piterman@leicester.ac.uk).

C. Finucane and H. Kress-Gazit are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853 (e-mail: cameron.finucane@gmail.com, hadaskg@cornell.edu).

as described in Section IV. We argue that the integration of computation with control implies that hybrid controllers operate with an inherent discrete-time semantics in addition to continuous-time semantics, and so it is meaningful to place restrictions on the "next" execution of the control loop when specifying desirable behavior.

Also relevant are previous works that incorporate other aspects of the physical execution into the discrete synthesis process. For example, the authors of [12], [13] optimize the synthesized discrete controllers with respect to a metric based on the physical workspace. Complementary to these approaches, we incorporate the continuous-time nature of the physical execution into the synthesis process.

Main Contributions: Ours is the first work to address the problem of synthesizing provably correct high-level control for atomic actions of varying relative execution times. We present several approaches to discrete synthesis and continuous execution, each suitable for different assumptions on the robot's physical capabilities and the environment in which it operates. Assumptions on robot actions range in strength from instantaneous actuation to arbitrary but finite relative execution times; the approaches also differ in responsiveness to events in the environment. This paper summarizes and extends results we previously presented in [4], [14], [15], including details of an enhanced synthesis algorithm that enables efficient synthesis for the approach in [15]. As a further contribution, in Section VI we compare these synthesis frameworks based on assumptions, complexity and resulting behavior, and illustrate the differences between them in detail with an example.

II. Background

Applying formal methods to the construction of provably correct robot controllers involves (a) a discrete abstraction in which the continuous reactive behavior of a robot is described in terms of a finite set of states, and (b) a temporal logic formalism for the specification, which in this work is Linear Temporal Logic (LTL) [16]. This section provides details on these components.

A. Linear Temporal Logic (LTL)

Syntax: Let AP be a set of atomic propositions. LTL formulas are defined by the recursive grammar:

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

where $\pi \in AP$, $\neg$ 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 from these operators.

Semantics: The truth of an LTL formula is evaluated over infinite sequences of states, usually executions of a finite state machine representing the system. Each state corresponds to an assignment of truth values to propositions, mapping each $\pi \in AP$ to `True` or `False`. Given an infinite sequence of truth assignments σ, the statement $\sigma \models \varphi$ denotes that σ satisfies formula φ at the first position. 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 with states corresponding to truth assignments on AP is said to satisfy φ if, for every execution σ of A, $\sigma \models \varphi$. The reader is referred to [16] for the formal semantics.

B. Synthesis

In reactive synthesis for robot control, the set of propositions AP is partitioned into two sets: a set of sensor propositions ($\mathcal{X}$) and a set of robot action and location propositions ($\mathcal{Y}$).

An LTL formula φ is *realizable* if there exists a strategy that, for every finite sequence of truth assignments to the sensor propositions, provides an assignment to the robot propositions such that every resulting infinite sequence of truth assignments to both sets of propositions satisfies φ. It turns out that such a strategy exists if and only if there is a finite state automaton that encodes it [17]. The synthesis problem is to find such a finite state automaton, i.e. one whose executions correspond to sequences of truth assignments that satisfy φ.

Definition 1 *A finite state automaton is a tuple* $A = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$ *where*

- Q *is a finite set of states.*
- $Q_0 \subseteq Q$ *is a set of initial states.*
- $\mathcal{X}$ *is a set of inputs (sensor propositions).*
- $\mathcal{Y}$ *is a set of outputs (location and action propositions).*
- $\delta : Q \times 2^\mathcal{X} \to 2^Q$ *is the transition relation. In this work, automata are restricted to be non-blocking, i.e.* $\delta(q,x) \neq \emptyset$ *for every* $q \in Q, x \in 2^\mathcal{X}$[1].
- $\gamma_\mathcal{X} : Q \to 2^\mathcal{X}$ *is a "transition" labeling, which associates with each non-initial state the set of environment propositions that are true over incoming transitions for that state. All transitions into a given state have the same truth assignment to inputs: this lets us associate with each state a unique truth assignment to* $\mathcal{X}$*. Note that if* $q' \in \delta(q,x)$ *then* $\gamma_\mathcal{X}(q') = x$*. Additionally, for* $q_0 \in Q_0$*, we define* $\gamma_\mathcal{X}(q_0) \subseteq 2^\mathcal{X}$ *as the set of environment propositions that are true in* q_0*.*
- $\gamma_\mathcal{Y} : Q \to 2^\mathcal{Y}$ *is a "state" labeling that maps each state to the set of robot propositions that are true in that state.*

Define $\gamma(q) = \gamma_\mathcal{X}(q) \cup \gamma_\mathcal{Y}(q)$ for $q \in Q$; intuitively, this labels each state with the input and output propositions that are true when the robot is in that state. Given a sequence of states $\sigma = q_0 q_1 q_2 ...$, define the sequence labeling $\Gamma(\sigma) = \gamma(q_0)\gamma(q_1)\gamma(q_2)....$ An automaton is *deterministic* if, for every $q \in Q$ and every $x \in 2^\mathcal{X}$, $|\delta(q,x)| = 1$. Unless mentioned explicitly, all automata considered in this work are deterministic. A deterministic automaton corresponds to a robot strategy, as described below.

Let $\delta(q) = \{\delta(q,x) \mid x \in 2^\mathcal{X}\}$ denote the set of possible successor states of state q. Finally, let $\delta_\mathcal{Y}(q,x) = \gamma_\mathcal{Y}(\delta(q,x))$; intuitively, this denotes the set of actions the robot takes when it senses input x while in state q.

[1]This assumption is without loss of generality, since a sink state with a self-transition can be added to replace any blocking transition.

Definition 2 *Given $\varphi = (\varphi_e \Rightarrow \varphi_s)$, deterministic automaton $A_\varphi = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$* realizes *$\varphi$ if $\forall \sigma = q_0 q_1 q_2 ... \in Q^\omega$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i)$, $\Gamma(\sigma) \models \varphi$.*

Given a task specification and a description of the workspace topology, this work considers formulas of the form $\varphi_e \Rightarrow \varphi_s$, where φ_e encodes assumptions about the environment's behavior and φ_s represents the desired robot (system) behavior. In turn, φ_e and φ_s have the structure $\varphi_e = \varphi_e^i \wedge \varphi_e^t \wedge \varphi_e^g$, $\varphi_s = \varphi_s^i \wedge \varphi_s^t \wedge \varphi_s^g$, where:

- φ_e^i and φ_s^i are non-temporal Boolean formulas over $\mathcal{X}$ and $\mathcal{Y}$ respectively, and constrain the initial sensor and robot proposition values, respectively.
- φ_e^t represents safety assumptions about behaviors of the environment, and consists of a conjunction of formulas of the form $\square A_i$ where each A_i is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X}$; here $\bigcirc\mathcal{X} = \{\bigcirc x \mid x \in \mathcal{X}\}$. Similarly, φ_s^t represents the robot safety constraints: it consists of a conjunction of $\square A_i$ formulas where each A_i is a Boolean formula over $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc\mathcal{X} \cup \bigcirc\mathcal{Y}$. Intuitively, φ_e^t and φ_s^t define the allowed environment and system transition relations, respectively. In each discrete time step, the environment's transition relation can depend only on the previous state, while the system transition relation can depend on both the previous state and the environment's current transition.
- $\varphi_e^g = \bigwedge_{i=1}^m \square\diamondsuit J_e^i$ and $\varphi_s^g = \bigwedge_{j=1}^n \square\diamondsuit J_s^j$ represent fairness assumptions on the environment and desired liveness guarantees for the system, respectively. Each J_e^i, J_s^j is a Boolean formula over $\mathcal{X} \cup \mathcal{Y}$, and represents an event that should occur infinitely often during controller execution.

Given a specification φ, consider the most general nondeterministic automaton over $\mathcal{X}$ and $\mathcal{Y}$, namely $N_\varphi = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_\mathcal{X}, \gamma_\mathcal{Y})$ such that for every $q \in Q$, $\delta(q,x) = \{q' \mid \exists \sigma \in (2^{\mathcal{X}\cup\mathcal{Y}})^\omega, \Gamma(qq')\sigma \models \varphi_e^t \Rightarrow \varphi_s^t\}$. Here σ is an infinite completion of the finite truth assignment sequence $\Gamma(qq')$ such that the resulting sequence satisfies $\varphi_e^t \Rightarrow \varphi_s^t$.

Finding a strategy for the robot that fulfills the specification φ can be thought of as exploring the above nondeterministic automaton N_φ in the aim of finding a deterministic automaton A_φ "contained" in it, which realizes the specification. Synthesizing a deterministic automaton that realizes an arbitrary LTL formula is doubly exponential in the size of the formula [17]. However, formulas of the form described above correspond to a fragment of LTL termed Generalized Reactivity or GR(1), for which the algorithm introduced in [10] permits synthesis in time polynomial in the size of the state space[2].

III. Approach 1: Instantaneous Actions with Synchronous Action Completion [4]

This section reviews the synthesis algorithm and continuous execution paradigm introduced in [4], wherein the discrete synthesis assumes instantaneous actions. This approach is implemented in the Linear Temporal Logic Mission Planning (LTLMoP) toolkit [19], which allows a specification written in a structured English grammar [20] to be transformed into a hybrid controller for use with physical robots and in simulation. Example 1, adapted from [14], will serve to demonstrate the stages of synthesis for a simple high-level task.

[2]The correctness of this synthesis algorithm relies on a technical assumption on the specifications, namely that they are well-separated [18].

A. Discrete Abstraction and Formal Specification

In this work, the relevant features of the continuous robot control problem are abstracted using a finite set of propositions. These propositions could result, for instance, from a discrete abstraction of the workspace and continuous robot dynamics [8], [21]. The approach presented in this paper is agnostic to the level of abstraction - it can be very fine-grained at the expense of the size of the state space, or very coarse at the expense of greater complexity of the abstraction grounding. Since we do not make any assumptions on the underlying robot dynamics, we do not present in detail how such abstractions can be obtained, but note that defining discrete abstractions for various settings is a topic of current research in the hybrid systems community.

For the purpose of presentation, we consider the following set of propositions:

- 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's camera is on)
- π_r for every location or region r (e.g., $\pi_{bedroom}$ is true if and only if the robot is in the bedroom).

Let $\mathcal{L} \subseteq \mathcal{Y}$ denote the set of location propositions. Thus, for every sensor input s, robot action a and region r, $\pi_s \in \mathcal{X}, \pi_r \in \mathcal{L}, \pi_a \in \mathcal{Y}\backslash\mathcal{L}$. 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 (represented by propositions π_{r_1} and π_{r_2}). The robot has one sensor, which senses a person (represented by 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 $\mathcal{X} = \{\pi_{person}\}$, $\mathcal{Y} = \{\pi_{r_1}, \pi_{r_2}, \pi_{camera}\}$, and $\mathcal{L} = \{\pi_{r_1}, \pi_{r_2}\}$.*

A high-level task is specified on this discrete abstraction using an LTL formula over $\mathcal{X} \cup \mathcal{Y}$. There are two types of properties allowed in a specification: *safety* properties, which state that "something bad never happens," and *liveness* conditions, which say "something good (eventually) happens."

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 the 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:

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

Note that the camera action is a simpler case, since it is modeled as having two binary states (on/off), and thus captured by a single Boolean proposition, assuming the availability of controllers for toggling its state.

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

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

B. Synthesis

The details of the GR(1) synthesis algorithm in [10] are provided below, and will be useful when comparing the alternatives in future sections. Given a set of propositions $P \subseteq AP$, $P \models \varphi$ denotes that all truth assignments setting $\pi \in P$ to `True` and $\pi \notin P$ to `False` satisfy φ. The semantics of μ-calculus formulae over N_φ is defined recursively as follows:

- The set of states $[\![\varphi]\!]$ is defined inductively on the structure of the μ-calculus formula. A Boolean formula φ is interpreted as the set of states $[\![\varphi]\!]$ in which φ is true, i.e. $[\![\varphi]\!] = \{q \in Q \mid \gamma(q) \models \varphi\}$.
- The logical operator $\circledcirc$ is defined as in [10]: $[\![\circledcirc \varphi]\!] = \{q \in Q \mid \forall x \in 2^{\mathcal{X}},\ \delta(q,x) \cap [\![\varphi]\!] \neq \emptyset\}$. Informally, this is the set of states q from which the robot can enforce that the next state will be in $[\![\varphi]\!]$, regardless of what the environment does next (i.e. for every $x \in 2^{\mathcal{X}}$). In Example 1, $[\![\circledcirc \pi_{r_2}]\!]$ is the set of all states in which the robot can move to region r_2, regardless of what the environment does, so $[\![\circledcirc \pi_{r_2}]\!] = \{q_0, q_1, q_2, q_3\}$ in Fig. 1.
- Let $\psi(X)$ denote a μ-calculus formula ψ with free variable X. $[\![\mu X.\psi(X)]\!] = \cup_i X_i$ where $X_0 = \emptyset$ and $X_{i+1} = [\![\psi(X_i)]\!]$. This is a least fixpoint operation, computing the smallest set of states X satisfying $X = \psi(X)$.
- $[\![\nu X.\psi(X)]\!] = \cap_i X_i$ where $X_0 = Q$ and $X_{i+1} = [\![\psi(X_i)]\!]$. This is a greatest fixpoint operation, computing the largest set of states X satisfying $X = \psi(X)$.

Synthesis is formulated as a two-player game between the robot and its uncontrolled environment, as captured by its sensors. The set of winning states for the robot is characterized by the μ-calculus formula $\varphi_{win} =$

$$
\nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} . \begin{bmatrix} \mu Y.\left(\bigvee_{i=1}^{m} \nu X.(J_s^1 \wedge \circledcirc Z_2 \vee \circledcirc Y \vee \neg J_e^i \wedge \circledcirc X)\right) \\ \mu Y.\left(\bigvee_{i=1}^{m} \nu X.(J_s^2 \wedge \circledcirc Z_3 \vee \circledcirc Y \vee \neg J_e^i \wedge \circledcirc X)\right) \\ \vdots \\ \mu Y.\left(\bigvee_{i=1}^{m} \nu X.(J_s^n \wedge \circledcirc Z_1 \vee \circledcirc Y \vee \neg J_e^i \wedge \circledcirc X)\right) \end{bmatrix}
$$

where J_e^i is the i^{th} environment liveness ($i \in \{1,...,m\}$), and J_s^j is the j^{th} system liveness ($j \in \{1,...,n\}$). Let $j \oplus 1 = (j \bmod n) + 1$. For $i \in \{1,...,m\}$ and $j \in \{1,...,n\}$, the greatest fixpoint $\nu X.(J_s^j \wedge \circledcirc Z_{j\oplus 1} \vee \circledcirc Y \vee \neg J_e^i \wedge \circledcirc X)$ characterizes the set of states from which the robot can force play to stay infinitely in states satisfying $\neg J_e^i$, thus falsifying the left-hand side of the implication $\varphi_e \Rightarrow \varphi_s$, or in finitely many steps reach a state in the set $Q_{win} = [\![J_s^j \wedge \circledcirc Z_{j\oplus 1} \vee \circledcirc Y]\!]$. The two outer fixpoints ensure that the robot wins from the set Q_{win} — μY ensures that play reaches a $J_s^j \wedge \circledcirc Z_{j\oplus 1}$ state in a finite number of steps, and νZ ensures that the robot can loop through the liveness properties in cyclic order. If every initial state is winning, we can extract an automaton realizing the specification from the intermediate steps of the above computation; details are available in [10].

Algorithm 1 Generalized synthesis algorithm from [10]

Input: φ, $\mathcal{X}$, $\mathcal{Y}$
Output: A_φ that realizes φ
1: Construct N_φ
2: Compute φ_{win}
3: **if** $\varphi_e^i \wedge \varphi_s^i \implies \varphi_{win}$ **then**
4: Extract strategy A_φ from intermediate values of X, Y, Z
5: **else**
6: FAIL
7: **end if**

When a specification is realizable, the above synthesis algorithm yields an automaton that implements the task on the discrete abstraction. If no such automaton exists, LTLMoP provides the user with a cause of unsynthesizability, as in [22], [23]. Fig. 1 depicts the automaton synthesized for Example 1. Each state of the automaton is labeled with the robot location and the truth assignment to action propositions in that state, and each transition is labeled with the truth assignment to sensor propositions required for that transition to be enabled. Incoming transitions therefore also determine the truth value of the sensor propositions for each state. The labels `r1`, `camera` and `person` represent π_{r_i}, π_{camera} and π_{person} respectively.

C. Continuous Execution

If an automaton is obtained, a controller that implements the corresponding continuous behavior is constructed by viewing the automaton as a hybrid controller; a transition between two states is achieved by the activation of one or more low-level (possibly continuous) controllers corresponding to each robot proposition. The atomic controllers used satisfy the bisimulation property [8], which ensures that every change in the discrete robot model can be implemented in the continuous domain (e.g., the motion controllers are guaranteed to drive the robot from one region to another regardless of the initial state within the region). The feedback motion controllers presented in [24] and [25] are among several that satisfy this property.

Consider the automaton in Fig. 1. Suppose the robot starts in room r_1, with its camera turned off and no person 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 automaton generated by the synthesis algorithm in [10] contains a transition from q_0 to q_1. To implement the transition (q_0, q_1), a motion controller and a controller for turning on the camera must both be invoked.

The controller synthesis framework presented in this section assumes that all robot actions are instantaneous. This assumption is justified when the robot controllers do not take a lot

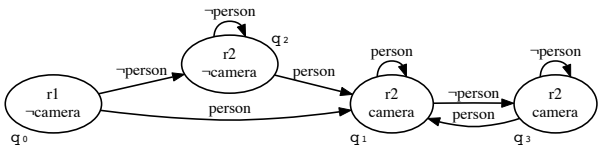


Fig. 1: Synthesized automaton for Example 1. Each state is labeled with the robot location and the truth assignment to action propositions in that state. Each transition is labeled with the truth assignment to sensor propositions that enables it.

of time to complete, such as in tasks that do not involve slow actions like motion. This assumption is also often violated in practice without consequence when all actions take similar amounts of time.

Under the continuous execution paradigm described in [4], all controllers except motion between adjacent regions 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; all controllers complete synchronously. Thus, to execute the transition (q_0, q_1) depicted in Fig. 1, the hybrid controller 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). Fig. 2(a) depicts the change in state for the transition (q_0, q_1), and how it corresponds to the progress of the continuous controllers.

D. Resulting Behavior

The approach to continuous execution in [4] has two undesirable qualities when actions are non-instantaneous (violating the assumption made at synthesis time):

Delayed Response: Since motion is executed first, this execution can result in a perceived lack of responsiveness to sensor inputs. For example, the camera should be turned on as soon as a person is sensed, regardless of the other actions to be performed. However for transition (q_0, q_1) in Fig. 1, using the approach to continuous execution in this section, the robot will not turn on its camera as soon as it senses a person, instead waiting until the transition to r_2 has been completed.

Unchecked Intermediate States: On the other hand, when non-motion controllers are non-instantaneous, continuous execution can admit intermediate states that are absent in the original automaton. For example, if the low-level controller for turning on the camera is non-instantaneous, the continuous execution of the controller in Example 1 will pass through the intermediate state $q_0^{\bar{fs}}$ (not present in the discrete automaton) with $\gamma_{\mathcal{Y}}(q_0^{\bar{fs}}) = \{\pi_{r_2}\}$, as depicted in Fig. 2(b). Although $q_0^{\bar{fs}}$ does not violate the specification in Example 1, this may not be true in general, as demonstrated by Example 2 below.

IV. Approach 2: Slow and Fast Action Classes with Synchronous Controller Activation [14]

While there are tasks for which the assumption of instantaneous actions is reasonable, a wider variety of tasks beg a more sophisticated model of the timing semantics of actuation. This includes all tasks that involve robot motion. In an attempt to relax the assumption of instantaneous actions, the robot's actions can be grouped into sets based on controller execution duration. This section summarizes the controller synthesis framework proposed in [14], where the robot actions were grouped into two such sets. To address the problem of delayed response that arises when using the approach in Section III, it is also desirable to be able to activate the camera simultaneously with the motion, to allow immediate reaction to the person sensor. This section therefore considers a continuous execution paradigm where all action controllers for a given transition are activated at the same time.

A. Problem Statement

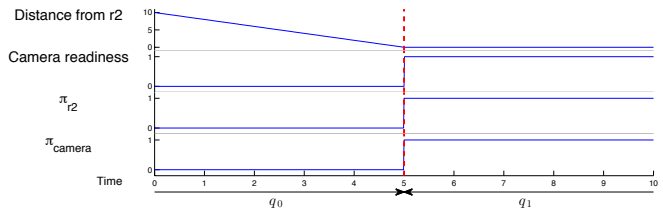
Assume that there are two kinds of low-level controllers — *fast* and *slow* — taking times t_F and t_S respectively, with $t_F \ll t_S$. More specifically, assume that motion is the only slow controller. The set of system propositions is partitioned based on the speed of the corresponding low-level controllers, into $\mathcal{Y}_S = \mathcal{L}$ and $\mathcal{Y}_F = \mathcal{Y} \backslash \mathcal{L}$ (i.e. location and non-location propositions). In Example 1, $\mathcal{Y}_F = \{\pi_{camera}\}$ and $\mathcal{Y}_S = \{\pi_{r_1}, \pi_{r_2}\}$.

Example 2 *Consider Example 1 with the added safety:*
$\square(\neg(\pi_{camera} \wedge \varphi_{r_1}))$ *(Do not activate the camera in r1)*

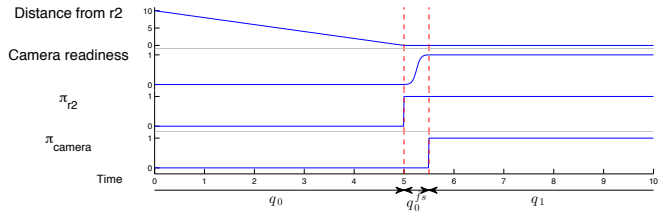
In this case, the execution resulting from turning on the camera and starting the motion to r_2 at the same time would pass through the intermediate state $q_0^{f\bar{s}}$ (Fig. 2(c)), not present in the discrete automaton. This execution would therefore be unsafe. Note that the camera turning on in r_1 is not captured by the discrete (and safe) model. For this example, the execution depicted in Fig. 2(b) is still safe, while that in Fig. 2(c) is not. It is desirable to obtain a controller that guarantees safety of intermediate states like $q_0^{f\bar{s}}$, which are not explicitly present in the synthesized automaton or checked during the existing synthesis process, but rather occur as artifacts of the continuous execution.

It may seem reasonable to circumvent these problems by allowing at most one robot action per transition. However, this could result not just in unnecessarily large automata, but also in newly unsynthesizable specifications. In fact, Example 1 (where the camera can turn on in r_1) would be unsynthesizable because the robot can never move from r_1 to r_2 if the environment infinitely alternates between person and no person (as it must toggle the camera on and off, and cannot move while doing so). So instead, we will explicitly model and certify intermediate states when synthesizing an automaton.

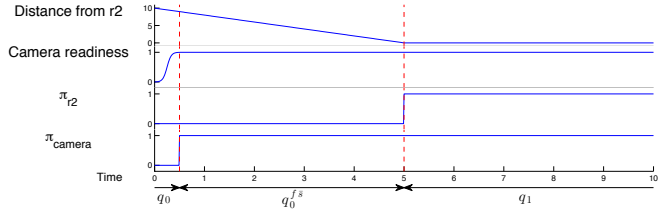
Given a specification φ, let (q, q') be a potential discrete transition in N_φ. Given $\mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$, define $\gamma_{\mathcal{Y}_F}(q) = \gamma_{\mathcal{Y}}(q) \cap \mathcal{Y}_F$ and $\gamma_{\mathcal{Y}_S}(q) = \gamma_{\mathcal{Y}}(q) \cap \mathcal{Y}_S$. Let $q^{f\bar{s}(q')}$ denote the discrete state with $\gamma_{\mathcal{Y}}(q^{f\bar{s}(q')}) = \gamma_{\mathcal{Y}_S}(q) \cup \gamma_{\mathcal{Y}_F}(q')$. This is the intermediate state in the transition between q and q', such that



(a) Motion completes first, instantaneous camera. This corresponds to the approach in [4] (assuming instantaneous fast actions).



(b) Actual execution corresponding to 2(a). Motion completes first, camera is non-instantaneous.



(c) Camera completes first when both controllers are activated together as in [14].

Fig. 2: Timing diagrams for executing (q_0, q_1) in Fig. 1

the fast actions have finished executing but the slow actions have not. Define

$$h(q,q') = \begin{cases} qq^{f\bar{s}(q')} & \text{if } \gamma_{\mathcal{Y}_S}(q) \neq \gamma_{\mathcal{Y}_S}(q') \wedge \gamma_{\mathcal{Y}_F}(q) \neq \gamma_{\mathcal{Y}_F}(q') \\ q & \text{otherwise} \end{cases}$$

The function $h(q,q')$ returns the intermediate state $q^{f\bar{s}(q')}$ only if both slow and fast actions change over the transition (q,q'). Note that if only the slow actions or only the fast actions change, there is no intermediate state in the continuous execution. In Example 1, $h(q_0,q_1) = q_0 q_0^{f\bar{s}(q_1)}$.

Definition 3 *Given $A = (Q, Q_0, \mathcal{X}, \mathcal{Y}, \delta, \gamma_{\mathcal{X}}, \gamma_{\mathcal{Y}})$, the* discrete lifting *of the set of its executions $\mathcal{E}_A = \{q_0 q_1 ... \in Q^\omega \mid q_{i+1} \in \delta(q_i)\}$ is $H_A^{FS} = \{h(q_0,q_1)...h(q_i,q_{i+1})... \mid q_0 q_1 ... \in \mathcal{E}_A\}$.*

H_A^{FS} defines a lifting onto the set of discrete states Q of all executions of automaton A when there are controllers of two completion times, and they are executed simultaneously to implement each discrete transition.

We assume known a set of safe states, Q_{safe}, which can be arbitrarily defined — in this paper, it is the set of all states not explicitly excluded by the safety properties φ_e^t and φ_s^t, and can be computed symbolically. An alternative is to explicitly enumerate the set of safe states.

Problem 1 *Given $\varphi, \mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$ and a set of safe states Q_{safe}, construct A_φ such that $\forall \sigma \in H_{A_\varphi}^{FS}, \sigma \in Q_{safe}^\omega$ (if such an automaton exists).*

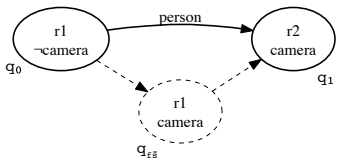


Fig. 3: Intermediate state with fast camera and slow motion for transition (q_1, q_1) in Fig. 1

Intuitively, the goal is to generate an implementing automaton such that every continuous execution contains only safe states.

B. Solution

In response Problem 1, we now present a synthesis algorithm and continuous execution paradigm that guarantees correctness when simultaneously executing low-level controllers of up to two different completion times for each discrete transition. The proposed framework explicitly introduces, at the discrete level, the intermediate states arising during continuous execution, and formally reasons about them. We previously presented a short version of this in [14].

To incorporate the relative execution times of the action controllers, Algorithm 1 is further constrained to generate only automata with safe intermediate states as follows. Given $\varphi, \mathcal{Y} = \mathcal{Y}_F \cup \mathcal{Y}_S$ and Q_{safe}, define the operator $\langle\!\langle \rangle\!\rangle_{FS}$ such that:

$$\begin{aligned} [\![\langle\!\langle \rangle\!\rangle_{FS}\varphi]\!] = \; & \{q \in Q \mid \forall x \in \delta_{\mathcal{X}}(q), \\ & \text{either} \\ & \quad \exists q' \in \delta(q,x) \cap [\![\varphi]\!] \text{ such that} \\ & \quad (\gamma_{\mathcal{Y}_F}(q) = \gamma_{\mathcal{Y}_F}(q') \text{ or } \gamma_{\mathcal{Y}_S}(q) = \gamma_{\mathcal{Y}_S}(q')), \\ & \text{or} \\ & \quad \exists q' \in \delta(q,x) \cap [\![\varphi]\!] \text{ such that} \\ & \quad \gamma_{\mathcal{Y}_F}(q) \neq \gamma_{\mathcal{Y}_F}(q') \text{ and } \gamma_{\mathcal{Y}_S}(q) \neq \gamma_{\mathcal{Y}_S}(q'), \\ & \quad \text{and } q^{f\bar{s}} \in Q_{safe}\} \end{aligned}$$

Thus, $[\![\langle\!\langle \rangle\!\rangle_{FS}\varphi]\!]$ is the set of states from which the system can in a single step force the play to reach a state in $[\![\varphi]\!]$, either:

- by executing actions of only one controller duration (fast or slow) so that there are no intermediate discrete states in the continuous execution, or
- by executing actions of both fast and slow controller durations, such that the intermediate state $q^{f\bar{s}}$ is safe.

Informally, $[\![\langle\!\langle \rangle\!\rangle_{FS}\varphi]\!]$ is a subset of $[\![\langle\!\langle \rangle\!\rangle\varphi]\!]$, with the additional constraint that if both fast and slow controllers are to be executed to implement a transition into $[\![\varphi]\!]$, the resulting intermediate state is safe.

Proposition 1 *Algorithm 1 with the operator $\langle\!\langle \rangle\!\rangle$ in formula φ_{win} replaced by $\langle\!\langle \rangle\!\rangle_{FS}$ yields a sound and complete synthesis algorithm that solves Problem 1.*

Proof: The only new property of the synthesis algorithm is to restrict the transitions in the resulting automaton to those for which the intermediate state is safe according to Q_{safe}. Therefore any A_φ synthesized by this modified algorithm realizes the specification as well as guaranteeing that $\forall \sigma \in H_{A_\varphi}^{FS}, \sigma \in Q_{safe}^\omega$; this shows soundness. On the other hand, by completeness of the original synthesis algorithm in [10],

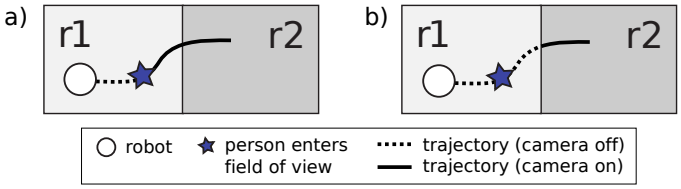


Fig. 4: Comparison of continuous trajectories and discrete events resulting from Approaches 1 and 2 for Example 1. a) The camera is turned on as soon as a person is sensed, according to Approach 2. b) When a person is sensed, motion is completed first, then the camera turns on. This corresponds to Approach 1. The observed trajectory under Approach 3 will depend on whether the camera is allowed in r_1.

and the fact that the only additional restriction is this property, Algorithm 1 will return such a strategy if one exists, and is thus complete for Problem 1. ■

C. Continuous Execution

The proposed synthesis algorithm is accompanied by a new execution paradigm that calls all low-level controllers corresponding to a discrete transition 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 then considered complete only when the motion completes. This approach produces controllers that are correct under the assumption that the environment does not change during execution of a discrete transition; any inputs that violate this assumption are ignored.

D. Resulting Behavior

Consider again the specification in Example 1, in which the robot has to move from its initial location r_1 to its destination r_2 and turn its camera on if it sees a person along the way. With the new execution paradigm, the hybrid controller turns the camera on immediately when a person is sensed. The trajectory that results from this controller is depicted in Fig. 4(a). Using the execution paradigm in Section III, even if the robot sensed a person while in the middle of r_1, it would only react to it once it completed its movement to region r_2. This is depicted in Fig. 4(b).

For Example 2, where the system safety condition includes "Always not camera in r_1" ($\square\neg(\pi_{camera} \wedge \varphi_{r_1})$), a state in which the system senses a person is only in $[\![\diamondsuit_{FS}\pi_{camera}]\!]$ if the robot can stay in the same region while turning on the camera. Recall that q_0 is the state in which the robot is in r_1 with the camera off. Observe that $q_0 \notin [\![\diamondsuit_{FS}\pi_{camera}]\!]$ (this means that in q_0, the robot cannot guarantee that the camera will be turned on in the next time step). This is because, if the environment sets π_{person} to true while the robot is still in r_1, the safety condition $\square(\neg(\pi_{camera} \wedge \varphi_{r_1}))$ prevents the robot from turning on the camera before first moving to r_2, and so the camera cannot be immediately activated since it might finish execution before the robot had left r_1. The corresponding specification is unrealizable under the new synthesis algorithm, whereas the original synthesis algorithm would return an automaton that included the transition (q_0, q_1) in Fig. 1. This difference is consistent with the observation that this transition is safe for the execution in Section III, under the assumption of instantaneous fast actions, but is unsafe if all action controllers are to be called simultaneously.

Since transitions are now explicitly non-instantaneous, we have to decide how to respond to changes in the environment once a transition has been started, until the destination state is reached. Consider again the transition (q_0, q_1) in Fig. 1, where the camera will be turned on immediately at the same time as 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 it has reached r_2. Then the transition (q_0, q_1) will be aborted, and the transition (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 once it turns on. To avoid such unsafe behaviors, the new execution paradigm 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 for the time taken to move between regions.

Note that, in the most general case, the relative execution times of the low-level controllers may be completely unknown, and therefore the safety of continuous executions corresponding to every possible ordering of controller execution times must be considered. The $\diamondsuit$ operator can be modified to check for the safety of every possible sequence of intermediate states during synthesis; however, this leads to a combinatorial explosion in the time taken for synthesis, since the number of such sequences is factorial in the size of the set of actions $\mathcal{Y}$.

V. Approach 3: Arbitrary Relative Action Completion [15]

This section proposes a synthesis framework that allows immediate reactivity as well as uninterrupted responsiveness to changes in the environment, and generalizes directly to arbitrary (but finite) action completion times. This approach relaxes all previous assumptions on the low-level controller execution durations, with a moderate computational overhead. In other words, it solves the following problem:

Problem 2 *Given a specification φ with $\mathcal{Y}$ such that the action represented by each $y \in \mathcal{Y}$ may have a different execution duration, construct a discrete automaton A_φ (if such an automaton exists) and continuous execution paradigm such that every resulting execution satisfies φ, even when the environment may change before an action has completed.*

To account for the non-instantaneous execution of continuous controllers, each robot action is viewed as the *activation* of the corresponding low-level controller rather than its complete execution, and a new sensor proposition is introduced in the discrete model to indicate whether the controller has finished executing. That is, the robot is able to sense when a low level controller has completed its action (e.g., the camera has turned on, or the robot has arrived in region r_1).

Original LTL (φ)	New LTL ($\hat{\varphi}$)
$\varphi_{r_1} \wedge \neg\pi_{camera}$	$\pi^c_{r_1} \wedge \neg\pi^c_{camera}$
$\square(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$	$\square(\bigcirc \pi_{person} \Rightarrow \bigcirc \pi_{camera})$
$\square((\pi_{camera} \Rightarrow \bigcirc \pi_{camera}))$	$\square((\pi^c_{camera} \Rightarrow \bigcirc \pi_{camera}))$
$\square\diamondsuit(\pi_{r_2})$	$\square\diamondsuit(\pi^c_{r_2})$

TABLE I: Proposition replacement for Example 1

A. Discrete Abstraction

The set of propositions is now modified to consist of:

- 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^c_a, π^c_r for the *completion* of the controller for turning action a on, or motion to region r (e.g., π^c_{camera} is true if and only if the camera has finished turning on, and $\pi^c_{bedroom}$ is true if and only if the robot has arrived in the bedroom). $\neg\pi^c_a$ represents the completion of the controller for turning action a off.[3]

Action/motion completion is modeled as an event sensed by the robot: for every sensor input s, robot action a and region r, $\pi_s, \pi^c_a, \pi^c_r \in \mathcal{X}$ and $\pi_a, \pi_r \in \mathcal{Y}$. For Example 1, $\mathcal{X} = \{\pi_{person}, \pi^c_{r_1}, \pi^c_{r_2}, \pi^c_{camera}\}$ and $\mathcal{Y} = \{\pi_{r_1}, \pi_{r_2}, \pi_{camera}\}$.

B. Formal Specification Transformation

Given this new discrete abstraction, the task specification now governs which robot actions can be activated, and how the action-completion sensors behave: it is revised as follows.

1) Proposition replacement in the original specification: Given a task specification $\varphi = (\varphi_e \Rightarrow \varphi_s)$ constructed for use with the synthesis framework in [4], it is modified as follows:

- Initial conditions specify the sensed state of the robot, so every occurrence of π_a and π_r in φ^i_s is replaced with an assumption on π^c_a and π^c_r in φ^i_e, respectively.
- Robot goals are predicated on the completion of actions (as sensed by the corresponding sensor). So every occurrence of π_a in φ^g_s is replaced with π^c_a. Similarly, robot goals refer to the sensed location π^c_r 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/motion completion as well as external events in the environment. The user input language, such as that presented in [20], must therefore allow distinguishing between a reference to π_a and π^c_a in safety conditions; this is discussed further in Section VI.

Table I presents the LTL formulas corresponding to the specification for Example 1 before and after proposition replacement. Notice that, as long as the action of turning on the camera has not completed, the robot is allowed to stop the controller that is turning on the camera.

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

$$\hat{\varphi}_{trans} = \bigwedge_r \square(\bigcirc \pi^c_r \Rightarrow \bigvee_{r' \in Adj(r)} \bigcirc \varphi_{r'})$$

For Example 1,

$$\begin{aligned}\hat{\varphi}_{trans} = \ & \square(\bigcirc \pi^c_{r_1} \Rightarrow (\bigcirc \varphi_{r_1} \vee \bigcirc \varphi_{r_2})) \\ & \wedge \square(\bigcirc \pi^c_{r_2} \Rightarrow (\bigcirc \varphi_{r_2} \vee \bigcirc \varphi_{r_1}))\end{aligned}$$

Here φ_{r_1} indicates that the robot is activating the controller to move towards r_1 and not activating the controller to move towards r_2 (i.e. $\varphi_{r_1} = \pi_{r_1} \wedge \neg\pi_{r_2}$); φ_{r_2} is defined symmetrically. The first conjunct in the above transition formula specifies that when the robot senses that it will be in r_1 in the next time step (i.e. $\bigcirc \pi^c_{r_1}$ is true), it can either stay in r_1 or activate the controller for moving towards r_2 (since the two regions are adjacent); the second conjunct is similarly defined for when the robot is in r_2..

The new transition formula $\hat{\varphi}_{trans}$ is included as a subformula of $\hat{\varphi}_s$. The specification resulting from the transformations in Sections V-B1 and V-B2 is denoted $\hat{\varphi} = \hat{\varphi}_e \Rightarrow \hat{\varphi}_s$.

3) Sensor Assumptions: Assumptions on the completion propositions model the effects of the robot activating its various controllers:

$$\begin{aligned}\varphi_c \ = \ & \bigwedge_r \square(\pi^c_r \Leftrightarrow \bigwedge_{r' \neq r} \neg\pi^c_{r'}) && (1) \\ & \wedge \bigwedge_r \bigwedge_{r' \in Adj(r)} \square(\pi^c_r \wedge \varphi_{r'} \Rightarrow (\bigcirc \pi^c_r \vee \bigcirc \pi^c_{r'})) && (2) \\ & \wedge \bigwedge_a \square(\pi^c_a \wedge \pi_a \Rightarrow \bigcirc \pi^c_a) && (3) \\ & \wedge \bigwedge_a \square(\neg\pi^c_a \wedge \neg\pi_a \Rightarrow \bigcirc \neg\pi^c_a) && (4)\end{aligned}$$

Conjunct (1) enforces mutual exclusion between the physical locations of the robot. Conjunct (2) governs how the location of the robot 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,

$$\begin{aligned}\varphi_c \ = \ & \square(\pi^c_{r_1} \Leftrightarrow \neg\pi^c_{r_2}) \wedge \square(\pi^c_{r_2} \Leftrightarrow \neg\pi^c_{r_1}) && (5) \\ & \wedge \square(\pi^c_{r_1} \wedge \varphi_{r_1} \Rightarrow \bigcirc \pi^c_{r_1}) && (6) \\ & \wedge \square(\pi^c_{r_1} \wedge \varphi_{r_2} \Rightarrow \bigcirc \pi^c_{r_1} \vee \bigcirc \pi^c_{r_2}) && (7) \\ & \wedge \square(\pi^c_{r_2} \wedge \varphi_{r_2} \Rightarrow \bigcirc \pi^c_{r_2}) && (8) \\ & \wedge \square(\pi^c_{r_2} \wedge \varphi_{r_1} \Rightarrow \bigcirc \pi^c_{r_2} \vee \bigcirc \pi^c_{r_1}) && (9) \\ & \wedge \square(\pi^c_{camera} \wedge \pi_{camera} \Rightarrow \bigcirc \pi^c_{camera}) && (10) \\ & \wedge \square(\neg\pi^c_{camera} \wedge \neg\pi_{camera} \Rightarrow \neg \bigcirc \pi^c_{camera}) && (11)\end{aligned}$$

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 (φ_{r_2}), then in the next time step it 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 being turned on, it will stay on.

[3] 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. For example, the intermediate stages of the camera turning on could be modeled separately, such as sensor cleaning, battery check, detecting external memory, etc.

4) Fairness conditions: In addition to the above safety conditions, the robot's environment must be constrained in order to ensure that every action/motion eventually completes, i.e. that the environment is in some sense "fair". A first approach is to add an environment assumption that every activation or deactivation of a controller eventually results in completion, i.e. the fairness conditions

$$\Box\Diamond(\pi_a \Rightarrow \bigcirc \pi_a^c) \quad (12) \qquad \Box\Diamond(\neg\pi_a \Rightarrow \bigcirc \neg\pi_a^c) \quad (13)$$

for every action a or region r.

However, this adds two fairness assumptions to the specification for every action. Since the synthesis algorithm scales linearly with the number of fairness assumptions [10], it is important to minimize the number of added assumptions. This can be 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. For this purpose, two new Boolean formulas $\varphi_a^{completion}$ and φ_a^{change} are defined for each action a as follows:

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

Formula $\varphi_a^{completion}$ holds when the activation (or deactivation) of the controller for a has completed execution. Formula φ_a^{change} holds when the robot changes its mind (such as by toggling the camera) due to events in the environment. 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:

$$\varphi_{loc}^{completion} = \bigvee_r (\varphi_r \wedge \bigcirc \pi_r^c) \qquad \varphi_{loc}^{change} = \bigvee_r (\varphi_r \wedge \neg \bigcirc \varphi_r)$$

The complete fairness assumption added is:

$$\varphi_{fair}^a = \Box\Diamond(\varphi_a^{completion} \vee \varphi_a^{change}) \quad (14)$$

Note that every execution satisfying both fairness conditions (12) and (13) described above for activation and deactivation also satisfies (14) and vice versa. Moreover, there is only one such assumption added for each action a (in the case of Example 1, there is one such assumption for the camera). Additionally, there is one assumption φ_{fair}^{loc} for motion. For Example 1,

$$\begin{aligned}
\varphi_{fair}^{camera} &= \Box\Diamond[(\pi_{camera} \wedge \bigcirc \pi_{camera}^c) \vee (\neg\pi_{camera} \wedge \neg \bigcirc \pi_{camera}^c) \\
&\quad \vee (\pi_{camera} \wedge \neg \bigcirc \pi_{camera}) \vee (\neg\pi_{camera} \wedge \bigcirc \pi_{camera})] \\
\varphi_{fair}^{loc} &= \Box\Diamond[(\varphi_{r_1} \wedge \bigcirc \pi_{r_1}^c) \vee (\varphi_{r_2} \wedge \bigcirc \pi_{r_2}^c) \\
&\quad \vee (\varphi_{r_1} \wedge \neg \bigcirc \varphi_{r_1}) \vee (\varphi_{r_2} \wedge \neg \bigcirc \varphi_{r_2})]
\end{aligned}$$

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

$$\varphi_{new} = \hat{\varphi}_e \wedge \varphi_c \wedge \textstyle\bigwedge_a \varphi_{fair}^a \wedge \varphi_{fair}^{loc} \Rightarrow \hat{\varphi}_s$$

C. Synthesis

Since the formulas $\varphi_a^{completion}$ and φ_a^{change} in the proposed liveness condition φ_{fair}^a govern both current and next time steps, the original synthesis algorithm in [10] cannot be applied as-is to synthesize an implementing automaton for the specification φ_{new}. 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 states as follows.

The μ-calculus is first extended with an operator $\circledcirc_C$ that maps a set of transitions to the set of states from which the robot can enforce those transitions. First define an operator $\ominus$ such that $[\![\ominus\varphi]\!] = \{(q,q') \mid q' \in [\![\varphi]\!]\}$. Informally, $\ominus$ returns all pairs of current and next states in the automaton whose second state is in $[\![\varphi]\!]$. Boolean operators can be used on these sets of current and next state pairs in the obvious way.

Now define $\circledcirc_C$ such that, given a formula φ, $[\![\circledcirc_C \varphi]\!] =$

- $\{q \in Q \mid \forall x \in \delta_{\mathcal{X}}(q), \exists q' \in \delta(q,x) \text{ s.t. } (q,q') \in [\![\ominus\varphi]\!]\}$ if φ describes a set of states
- $\{q \in Q \mid \forall x \in \delta_{\mathcal{X}}(q), \exists q' \in \delta(q,x) \text{ s.t. } (q,q') \in [\![\varphi]\!]\}$ if φ describes a set of current-next pairs of states

Then the new set of winning states is characterized by $\hat{\varphi}_{win} =$

$$\nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} \cdot \begin{bmatrix} \mu Y.\left(\bigvee_{i=1}^m \nu X. \circledcirc_C(J_s^1 \wedge \ominus Z_2 \vee \ominus Y \vee \neg J_e^i \wedge \ominus X)\right) \\ \mu Y.\left(\bigvee_{i=1}^m \nu X. \circledcirc_C(J_s^2 \wedge \ominus Z_3 \vee \ominus Y \vee \neg J_e^i \wedge \ominus X)\right) \\ \vdots \\ \mu Y.\left(\bigvee_{i=1}^m \nu X. \circledcirc_C(J_s^n \wedge \ominus Z_1 \vee \ominus Y \vee \neg J_e^i \wedge \ominus X)\right) \end{bmatrix}$$

Lemma V.1 *Algorithm 1 with the formula φ_{win} replaced by $\hat{\varphi}_{win}$ (denoted Algorithm 1$[\varphi_{win}/\hat{\varphi}_{win}]$) yields a sound and complete synthesis algorithm for formulas φ that admit the $\bigcirc$ operator in φ_e^g and φ_s^g.*

Proof: The only difference from φ_{win} in Section III is to replace $(J_s^j \wedge \circledcirc Z_{j\oplus 1} \vee \circledcirc Y \vee \neg J_e^i \wedge \circledcirc X)$ with $\circledcirc_C(J_s^j \wedge \ominus Z_{j\oplus 1} \vee \ominus Y \vee \neg J_e^i \wedge \ominus X)$. The intermediate stages of the computation of φ_{win} now compute states that can force goal *transitions* (rather than reach goal *states*). Therefore, every A_φ synthesized by Algorithm 1 guarantees fulfillment of these goal transitions, i.e. goals that admit $\bigcirc$. By the completeness of the original synthesis algorithm in [10], and the fact that the only additional restriction is this property, Algorithm 1 returns a strategy if one exists. Note that if none of the robot liveness conditions contain the $\bigcirc$ operator, the computations of φ_{win} and $\hat{\varphi}_{win}$ are identical. ■

Theorem V.2 *Algorithm 1$[\varphi_{win}/\hat{\varphi}_{win}]$, with input φ_{new} constructed from φ as in Section V-A, yields a sound and complete synthesis procedure to solve Problem 2.*

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:

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

This allows the additional liveness to instead be written as

$$\varphi'_{fair} = \Box\Diamond \pi_a^{completion,change}$$

However, this introduces one new proposition per robot action. The running time of the synthesis algorithm scales polyno-

mially with the size of the state space, which in turn scales exponentially with the number of propositions.

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[4]. However, if we do not use the enhanced synthesis algorithm presented above, 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 $x \in 2^{\mathcal{X}}$ and the corresponding next state q' in the automaton, the transition (q,q') is executed by simultaneously invoking the controllers associated with every action or location proposition that changes value from q to q'. Note that the current sensed state of the system, given by the completion propositions in x, determines which actions can be activated in q'. Transitions in the automaton execute instantaneously, as they correspond to activation or deactivation of controllers, but the controllers themselves may take several discrete transitions to finish executing.

E. Resulting Behavior

Fig. 5 depicts an excerpt of the automaton synthesized for Example 1 using the approach presented in this section. The full automaton has 11 states and is omitted for conciseness. Negated sensor labels are omitted from the transitions for a cleaner presentation. The label `c_r`$_i$ represents $\pi^c_{r_i}$.

In state q_0, the robot is trying to stay in r_1 (as indicated by the action π_{r_1} being true), and not activating its camera (π_{camera} is false); note that $q_0 \in Q_0$. Consider the transition (q_0,q_1), which is activated when the robot is in q_0 and does not sense a person. The robot is still in 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} being true in q_1. When in q_1, if the robot still does not sense a person, it either moves to q_3 or stays in q_1 depending on whether it has reached r_2 yet (i.e. depending on the truth value of $\pi^c_{r_2}$). On the other hand, suppose the robot senses a person in q_1 before it has reached r_2, the transition enabled is instead (q_1,q_2). In state q_2, the robot is still activating the motion controller to reach r_2, but is now additionally activating the camera; the transition to q_4 is taken once the robot is in r_2. Finally, the transition (q_1,q_5) is taken if the robot senses a person exactly as it reaches r_2 (as indicated by both sensor propositions π_{person} and $\pi^c_{r_2}$ 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, in the transition (q_0,q_2), which is activated when the robot is in q_0 and senses a person, the controllers for moving from r_1 to r_2 and for turning on the camera are being activated simultaneously, just as in Fig. 4(a). 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_0,q_2) is still taken (followed by a transition out of q_2 that corresponds to the person no longer being seen), and the camera is still being activated in q_2. This ensures that the person is not ignored, since the fact that the camera is being turned on is now explicitly represented, even though the person has disappeared.

[4]This can be reduced if some robot actions are mutually exclusive and can be encoded with a bit vector, as is true for motion between regions.

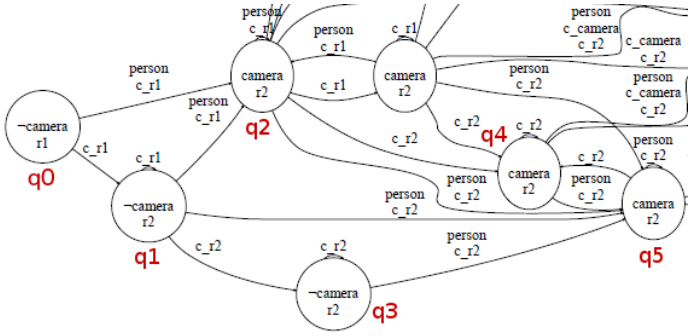


Fig. 5: Excerpt of automaton synthesized for Example 1 with Approach 3. Negated propositions are omitted from the transitions for clarity.

VI. Comparison of Approaches 1-3

Table II summarizes the differences between Approaches 1, 2 and 3 based on the assumptions they make about action execution times and the operating environment; the complexity of the synthesis algorithm; and the corresponding continuous execution.

The computational complexity of the synthesis algorithm presented in [10], which forms the basis for synthesis in all three frameworks, is $O(mn\Sigma^2)$, where m and n are the number of environment liveness conditions and system goals in the specification, respectively, and Σ denotes the state space corresponding to truth assignments to $\mathcal{X}$ and $\mathcal{Y}$. The new algorithms presented in Sections IV and V are also quadratic in the size of the state space and linear in the number of environment and system liveness conditions. However, for Approach 3, the size of the state space is larger by a factor of $O(2^{|\mathcal{Y}|})$, and the number of liveness conditions grows by $O(|\mathcal{Y}|)$. This results in a corresponding increase in synthesis time and size of the synthesized automata. For instance, the automaton synthesized for Example 1 has 3 states using Approach 1, 4 states using Approach 2, and 11 states using Approach 3; synthesis for this example takes roughly 4 times as long using Approach 3 as with the other two approaches. A thorough comparison of runtimes and solution sizes is conducted on the following example, and appears in Table III.

Example 3 *Consider the workspace depicted in Fig. 6. The robot starts in the region marked* `start` *and has to visit* `goal`*. However, if it sees a stop sign in region r_i (for $i \in R = \{1,2,4,5,7,8\}$), it cannot pass through that room. The robot also has to respond to seeing a person by turning on a camera, except in r_3, where the camera cannot be turned on. Additionally, the robot should wave when passing through r_6. There will never be stop signs in both $r_{\texttt{top}}$ and $r_{\texttt{bottom}}$ at the same time for the pairs $(\texttt{top},\texttt{bottom}) \in P = \{(1,2),(4,5),(7,8)\}$ (so the robot can always progress forward). There are initially no stop signs, and the robot starts with its camera turned off.*

Approach	Assumptions	$\lvert\Sigma\rvert$	Synthesis Complexity	Continuous Execution	Consequence of Violated Assumptions
1	Instantaneous actions	$2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{Y}\rvert}$	$O(mn\Sigma^2)$	Action activation staggered to ensure simultaneous completion	Delayed response, unsafe states
2	Two action speeds, environment constant on transitions	$2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{Y}\rvert}$	$O(mn\Sigma^2)$	Simultaneous action activation, environment ignored on transitions	Lack of response during transitions
3	No timing-related assumptions	$2^{\lvert\mathcal{X}\rvert+2\lvert\mathcal{Y}\rvert}$	$O((m+\lvert\mathcal{Y}\rvert)n\Sigma^2)$	Simultaneous action activation	NA

TABLE II: Comparison of the three synthesis frameworks discussed. Variables m and n are the number of environment liveness conditions and system goals in the specification φ, respectively, $\mathcal{X}$ and $\mathcal{Y}$ denote the environment and system variables, respectively, and Σ denotes the state space corresponding to truth assignments to these variables.

With the approach in [4], the corresponding specification is:

$$
\begin{aligned}
& \textstyle\bigwedge_{i\in R}\neg\pi_{stop\_r_i} && \text{\#Env Initial}\\
& \quad \text{(Env starts with no stop signs anywhere)}\\
\wedge\ & \textstyle\bigwedge_{(i,j)\in P}\Box(\neg(\pi_{stop\_r_i}\wedge\pi_{stop\_r_j})) && \text{\#Env Safety}\\
& \quad \text{(There will never be stop signs in both } \texttt{ri} \text{ and } \texttt{rj}\text{)}\\
\Rightarrow\ & \\
& (\varphi_{\texttt{start}}\wedge\neg\pi_{camera}\wedge\neg\pi_{wave}) && \text{\#Robot Initial}\\
& \quad \text{(Robot starts in } \texttt{start} \text{ with camera and waving off)}\\
\wedge\ & \textstyle\bigwedge_{i\in R}\Box(\bigcirc\pi_{stop\_r_i}\Rightarrow\neg\bigcirc\varphi_{r_i}) && \text{\#Robot Safety}\\
& \quad \text{(Do not go to } \texttt{ri} \text{ if you sense a stop sign in } \texttt{ri}\text{)}\\
\wedge\ & \Box(\varphi_{r_3}\Rightarrow\neg\pi_{camera}) && \text{\#Robot Safety}\\
& \quad \text{(Do not turn on the camera in } \texttt{r3}\text{)}\\
\wedge\ & \Box(\bigcirc\pi_{person}\wedge\neg\varphi_{r_3}\Rightarrow\bigcirc\pi_{camera}) && \text{\#Robot Safety}\\
& \quad \text{(Activate the camera if you see a person except in } \texttt{r3}\text{)}\\
\wedge\ & \Box(\varphi_{r_6}\Rightarrow\bigcirc\pi_{wave}) && \text{\#Robot Safety}\\
& \quad \text{(Wave when in } \texttt{r6}\text{)}\\
\wedge\ & \Box\Diamond(\varphi_{\texttt{goal}}) && \text{\#Robot Liveness}\\
& \quad \text{(Visit } \texttt{goal} \text{ infinitely often)}
\end{aligned}
$$

Approach 1: This specification is realizable under the assumption of instantaneous robot actions, via the synthesis approach in [4]. However, consider what happens under the continuous execution paradigm in Approach 1 when the robot is in `start`, and sees a stop sign in r_1. The robot will start to move towards r_2. Suppose that before the robot has entered r_2, the stop sign in r_1 disappears but one appears in r_2. The robot will abort the transition from `start` to r_2 and start heading to r_1, taking a different discrete transition instead. If the stop sign's location changes faster than the robot can change directions and move, the robot will be trapped in `start`, because it will keep changing its mind between the above two discrete transitions. The same problem manifests itself when the robot is in r_3 and r_6. This is therefore a high-level task that produces a controller under the synthesis approach in [4], but whose physical execution does not accomplish the specified behavior because of an inadequate modeling of the underlying physical system. This discrepancy is due to the fact that the synthesis approach in [4] assumes that when $\pi_{stop\_r_1}$ becomes true, the robot can move from `start` to r_2 instantaneously — this assumption is violated in practice.

Approach 2: With Approach 2, the specification is unrealizable. The reason for this is that the robot always needs to be able to stay in place if it is turning on the camera or waving, since these fast actions will complete first. This means that a stop sign cannot appear in the robot's current room. Adding the assumption $\Box(\bigcirc\varphi_{r_1}\Rightarrow\neg\bigcirc\pi_{stop\_r_i})$ to the environment allows synthesis to succeed. However, now if the stop sign changes location as the robot is moving, this change will simply be ignored, and the robot will continue on to r_2. This results in unsafe behavior, because the robot will end up in a room with a stop sign.

Approach 3: With the discrete abstraction, specification transformation and execution paradigm of Approach 3, the robot initial condition in the above specification changes to $\varphi^c_{\texttt{start}}\wedge\neg\pi^c_{camera}\wedge\neg\pi^c_{wave}$, and the robot goal becomes $\Box\Diamond(\varphi^c_{\texttt{goal}})$; similar changes apply to the rest of the specification to conform to the new timing semantics. Here again, the assumption that a stop sign will not be seen in the robot's current room is required; this is expressed as the environment assumption $\Box(\bigcirc\varphi^c_{r_i}\Rightarrow\neg\bigcirc\pi_{stop\_r_i})$. This specification (with the additional formulas introduced in Section V) is unrealizable, and no automaton is obtained. In this example, this is the safer, more desirable outcome, since there exists an environment strategy that toggles the stop signs between r_1 and r_2 and prevents the robot from fulfilling the specification. Using a timing-aware synthesis algorithm highlights this problem, and hints that we may need stronger assumptions on the environment in order to synthesize a controller.

One possible solution is to modify the specification to explicitly exclude the above pathological case, by adding safety assumptions to the environment that prevent the stop sign from appearing in room r_i while the robot is in motion towards r_i, i.e., $\Box\Diamond((\varphi_{r_i}\wedge\bigcirc\neg\varphi^c_{r_i})\Rightarrow\neg\bigcirc\pi_{stop\_r_i})$. This ensures that the robot can always move towards the goal region, making the specification realizable again. In this example, waving can be modeled as an instantaneous action, since it doesn't interact with other actions. As long as the waving controller is turned on when the robot enters r_6 (i.e. $\Box(\varphi^c_{r_6}\Rightarrow\bigcirc\pi_{wave})$), its completion is of no consequence. This reduces the number of added variables, and hence the running time of the synthesis algorithm. This shows that it is sometimes advantageous to reason about when the timing of continuous action execution is inconsequential. The set of actions that can be modeled as

Approach	#Conjuncts	Time (in secs)	#States
1	40	0.12	98
2	46	0.14	98
3	100	2.18	552

TABLE III: Specification length, synthesis time and automaton size for Example 3

instantaneous is currently a user-defined parameter; determining this automatically from the specification is a direction of future work.

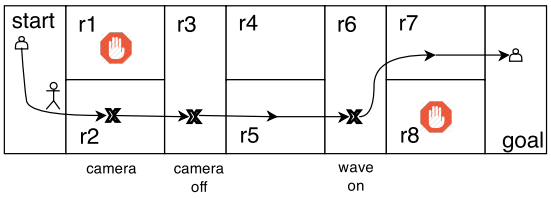
Simulations: Fig. 6 diagrams one simulated run of the automaton synthesized for Example 3 using each of the three approaches.

- The run in Fig. 6(a) is synthesized using Approach 1. The robot (which begins in `start`) senses a stop sign in r_1 and so moves to r_2. On its way to r_2, it senses a person, so once it arrives in r_2 it activates the camera. The robot then moves to r_3, turning the camera off once it enters r_3. It passes through r_5, and when it arrives in r_6, it waves. It then sees a stop sign in r_8 and so passes through r_7 to reach the `goal`.
- The run in Fig. 6(b) is synthesized using Approach 2. Now when the robot senses a person on its way to r_2, it immediately turns on the camera (so the camera turns on while the robot is still in r_1). The robot turns the camera off as it starts moving towards r_3. It passes through r_5 as before, and starts waving as soon as it starts moving towards r_6. As before, it sees a stop sign in r_8 and passes through r_7 to reach the `goal`.
- The run in Fig. 6(c) is synthesized using Approach 3. On its way to r_2, the robot senses a person and activates the camera controller. The camera turns on while the robot is in r_2. The robot deactivates the camera as it passes through r_3, but the camera only turns off when the robot is in r_5. When it arrives in r_6, it activates the controller for waving, which completes immediately.

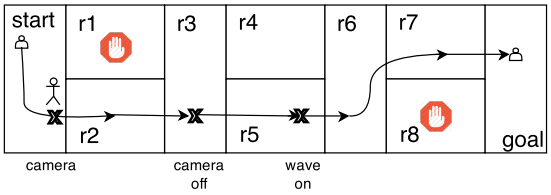
Table III compares the three approaches based on specification length, synthesis time and automaton size for Example 3. Synthesis was performed using implementations of the three versions of Algorithm 1 in the `slugs` synthesis tool[5] on a 1.3 GHz Intel Core i5 processor with 8GB of RAM.

Discussion: As illustrated in the above example, it may be the case that a specification is synthesizable in one synthesis framework but unsynthesizable in another. Recent work in [22], [23] has addressed the question of providing the user with feedback on a specification that has no implementing controller. In the situation described above, the user can now be alerted to the fact that the timing semantics of controller execution are responsible for the unsynthesizability of the specification, since unsafe intermediate states may occur. An unsynthesizable specification usually indicates that one or more assumptions are necessary in order to enable synthesis. For example, if a specification is not synthesizable under Approach 2 or 3, the assumption of instantaneous action

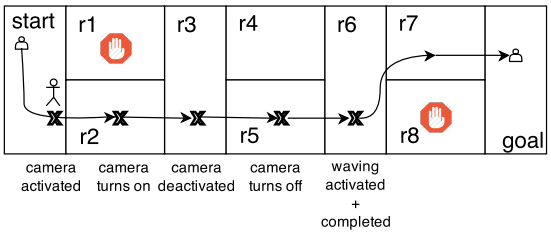
[5]Available at https://github.com/ltlmop/slugs



(a) Approach 1



(b) Approach 2



(c) Approach 3

Fig. 6: Simulated trajectories for Example 3

execution may be required so that Approach 1 can be used. Future research will analyze cases of unsynthesizability arising from the specific timing semantics chosen for controller synthesis, and present users with this information in a suitable manner. An additional direction to investigate is the automatic addition of environment assumptions to make the specification synthesizable. In Example 3, for instance, adding the environment fairness assumption $\square\diamondsuit(\pi^c_{r_4})$ results in a controller by explicitly requiring the environment to eventually let the robot through to r_4.

VII. Conclusions

This paper identifies and addresses a major challenge of applying formal methods in the physical domain of high-level robot control, namely that of achieving correct continuous behavior from high-level specifications when the low-level controllers have different execution durations. Three different approaches to timing semantics for controller synthesis are compared on the basis of the assumptions they make about the execution of low-level action controllers. Assumptions range in strength from instantaneous actions, to the case where robot actions are either *fast* or *slow*, to controllers whose relative completion times are unknown. The approaches are compared on factors including the complexity of the resulting synthesis algorithm, reactivity to sensor inputs, and the safety of intermediate states arising during execution. Future work includes analyzing specifications that have no implementation because of the timing semantics of the desired controllers, and presenting this information to users.

REFERENCES

[1] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *Special Issue of the IEEE Robotics and Automation Magazine on Formal Methods for Robotics and Automation*, vol. 18, no. 3, pp. 65–74, Sep. 2011.

[2] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Automat. Contr.*, vol. 53, no. 1, pp. 287–297, 2008.

[3] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, 2010, pp. 2689–2696.

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

[5] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*, K. H. Johansson and W. Yi, Eds. ACM ACM, 2010, pp. 101–110.

[6] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of NAO humanoid," in *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, 2009, pp. 769–774.

[7] J. M. Davoren, V. Coulthard, N. Markey, and T. Moor, "Non-deterministic temporal logics for general flow systems," in *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, 2004, pp. 280–295.

[8] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," in *Proceedings of the IEEE*, 2000, pp. 971–984.

[9] P. Bouyer, T. Brihaye, and F. Chevalier, "Control in o-minimal hybrid systems," in *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, 2006, pp. 367–378.

[10] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 911–938, 2012.

[11] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[12] G. Jing and H. Kress-Gazit, "Improving the continuous execution of reactive ltl-based controllers," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, 2013, pp. 5439–5445.

[13] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimal control with weighted average costs and temporal logic specifications," in *Robotics: Science and Systems VIII, University of Sydney, Sydney, NSW, Australia, July 9-13, 2012*, 2012.

[14] V. Raman, C. Finucane, and H. Kress-Gazit, "Temporal logic robot mission planning for slow and fast actions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 2012, pp. 251–256.

[15] V. Raman, N. Piterman, and H. Kress-Gazit, "Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, 2013, pp. 4075–4081.

[16] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 2001.

[17] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, 1989, pp. 179–190.

[18] U. Klein and A. Pnueli, "Revisiting synthesis of GR(1) specifications," in *Hardware and Software: Verification and Testing - 6th International Haifa Verification Conference, HVC 2010, Haifa, Israel, October 4-7, 2010. Revised Selected Papers*, 2010, pp. 161–181.

[19] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, 2010, pp. 1988–1993.

[20] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[21] P. Tabuada, *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.

[22] V. Raman and H. Kress-Gazit, "Explaining impossible high-level robot behaviors," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 94–104, 2013.

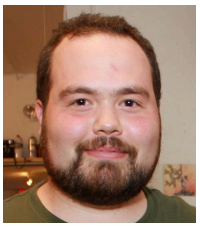
[23] ——, "Towards minimal explanations of unsynthesizability for high-level robot behaviors," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, 2013, pp. 757–762.

[24] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[25] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Nevada, USA, October 27 - November 1, 2003*, 2003, pp. 3546–3551.

Vasumathi Raman is a postdoctoral scholar in the Department of Computing and Mathematical Sciences at the California Institute of Technology. Her research explores algorithmic methods for designing and controlling autonomous systems, guaranteeing correctness with respect to user-defined specifications. She earned a PhD in Computer Science from Cornell University and a BA in Computer Science and Mathematics from Wellesley College.

Nir Piterman received his Ph.D. in Computer Science from the Weizmann Institute of Science in 2005. He is currently an Reader at the department of Computer Science in the University of Leicester. His research focuses on formal verification and automata theory. In particular he works on model checking, temporal logic, synthesis, game solving, and applications of formal methods to biological modeling.



Cameron Finucane worked on this project as a research assistant in the Autonomous Systems Lab at Cornell University. As a member of Professor Hadas Kress-Gazit's research group, his work focused on high-level robot control and the development of the LTLMoP toolkit. He previously studied electrical engineering and linguistics at the University of Pennsylvania and at Waseda University.

Hadas Kress-Gazit Hadas Kress-Gazit received her Ph.D. in Electrical and Systems Engineering from the University of Pennsylvania in 2008. She is currently an Assistant Professor at the Sibley School of Mechanical and Aerospace Engineering at Cornell University. Her research focuses on creating verifiable robot controllers for complex high-level tasks using logic, verification methods, synthesis, hybrid systems theory and computational linguistics. She is a recipient of the NSF CAREER award (2010) and the DARPA Young Faculty Award (2012)