Maximally Satisfying LTL Action Planning

Jana Tumova, Alejandro Marzinotto, Dimos V. Dimarogonas, Danica Kragic

***Abstract*— We focus on autonomous robot action planning problem from Linear Temporal Logic (LTL) specifications, where the *action* refers to a "simple" motion or manipulation task, such as "go from A to B" or "grasp a ball". At the high-level planning layer, we propose an algorithm to synthesize a maximally satisfying discrete control strategy while taking into account that the robot's action executions may fail. Furthermore, we interface the high-level plan with the robot's low-level controller through a reactive middle-layer formalism called Behavior Trees (BTs). We demonstrate the proposed framework using a NAO robot capable of walking, ball grasping and ball dropping actions.**

I. Introduction

Synthesis of correct-by-design robot controllers to achieve a complex, high-level, recurrent task has recently received an increasing amount of attention. In particular, temporal logics have been suggested to express robot motion tasks, such as combinations of surveillance ("periodically visit region A"), sequencing ("visit region A, then B, then C"), safety ("always avoid region D"), and many others. Leveraging ideas from formal verification [1], a number of techniques have been developed to find a controller to achieve Linear Temporal Logic (LTL) [2], [3], [4], [5], [6], [7] task specifications.

Many of the suggested approaches build on a three-step hierarchical procedure: First, the robot in its environment is abstracted into a finite, discrete state-transition structure using e.g., sampling-based methods or cell decompositions. Second, a discrete plan that satisfies the temporal logic task is synthesized. Third, the plan is projected onto a correct-by-design robot controller. The correlation between a discrete plan and a robot controller has been investigated quite widely in robot motion planning context [2], [5], [8], whereas the literature focused on low-level implementation of both *motion and manipulation* discrete plans has been, to our best knowledge, limited to several standalone studies [9]. In this work, we enhance the temporal logic specification of "where to go" with "what to do" there and we aim to propose a framework that 1) bridges the high-level plan with the low-level motion, grasping, and other controllers, 2) is modular and extensible to handle different robots and their capabilities, 3) is reactive and able to cope with the unreliability of the robot's sensors and actuators leading to failures of its actions, and 4) guarantees that the temporal logic task is met as closely as possible.

The authors are with the Centre for Autonomous Systems, Royal Institute of Technology (KTH), SE-100 44 Stockholm, Sweden. J. Tumova and D. Dimarogonas are also with the Automatic Control Lab and the ACCESS Linnaeus Center. A. Marzinotto and D. Kragic are also with Computer Vision and Active Perception Lab. This work was supported by the EU STREP RECONFIG. e-mail: {tumova|almc|dimos|dani}@kth.se.

A. Our Approach and Contribution

The problem addressed in this paper can be formulated as follows. Similarly as in some related literature (e.g., [3], [6], [10]) we assume that the robot's states and action capabilities are at the highest level of abstraction captured through a finite, discrete transition system. Due to imprecisions in robot's sensors and actuators, each attempt to execute an action might result into a success or a failure. As the specification language, we use a surveillance fragment of the State-Event variant of LTL (SE-LTL) [11], [3] allowing for requirements and restrictions on both the robot's locations and its actions.

Our approach builds on using *Behavior Trees* (BTs) as a middle-layer interface between the high-level planner and the low-level robot controller. The idea has been recently proposed in [12], resulting in an open source BT library for the Robot Operating System (ROS) [13]. The main feature indicating that BTs are suitable for middle-layer controller representation is their *modularity*; every sub-tree of a BT can be treated as a standalone entity that represents a certain action or a task. BTs are built using sub-tree blocks in a hierarchical fashion, subsuming thus multiple levels of abstraction and allowing to easily replace an abstract action/task node with a concrete action/task implementation [12]. Additionally, BTs are equipped with control-flow nodes to capture various conditional, or sequential sub-tree executions, hence making BTs an appropriate formalism for reactive controllers.

Our goal is to find a reactive controller that takes the form of a BT, takes into account the action execution successes and failures, and guarantees that the resulting robot's behavior is as close to the satisfaction of the SE-LTL task as possible in case the task accomplishment cannot be ensured.

Related work focused on techniques for reactive temporal logic planning include e.g., studying synthesis for nondeterministic systems [14], [6] or general-reactivity goals [2], a receding horizon approach to synthesis [5], or synergistic interface between a planner and a controller [4] . As opposed to them, we consider a specific kind of nondeterminism that we handle algorithmically while employing ideas from synthesis for deterministic systems. Other related works aim at planning under unsatisfiable temporal logic specifications, such as [15], [16], [17], where the authors focus on least-violating planning with respect to a proposed metric. Other authors [18], [10] propose to systematically revise the given temporal logic specification to be satisfiable by the given model and close to the original formula.

The rest of the paper is organized as follows. In Sec. II, we fix necessary preliminaries in transition systems, temporal logics, and automata. In Sec. III, we formalize the discrete action planning problem and in Sec. IV we introduce the details of the proposed solution. Sec. V discusses the implementation of a discrete plan by a robot. Sec. VI presents several illustrative examples and experimental results. We conclude and outline future research directions in Sec. VII.

II. Preliminaries

A. Notation

Given a set S, let $|\mathsf{S}|$, 2^{S}, S^* and S^ω denote the cardinality of S, the set of all subsets of S, and the set of all finite (possibly empty) and infinite sequences of elements of S, respectively. Given a finite sequence s_{fin} and a finite or an infinite sequence s, we denote by $\mathsf{s}_{fin}\mathsf{s}$ their concatenation and by s_{fin}^ω the infinite sequence $\mathsf{s}_{fin}\mathsf{s}_{fin}\mathsf{s}_{fin}\ldots$. The concatenation of a set of finite sequences S_{fin}, and a set of finite or infinite sequences S, is $\mathsf{S}_{fin}\cdot\mathsf{S} = \{\mathsf{s}_{fin}\mathsf{s} \mid \mathsf{s}_{fin}\in\mathsf{S}_{fin}, \mathsf{s}\in\mathsf{S}\}$.

B. Transition System, Linear Temporal Logic, and Automata

Definition 1 (Transition System) *A* labeled transition system *(TS) is a tuple $\mathcal{T} = (S, s_{init}, \Sigma, \to, \Pi, L)$, where S is a finite set of states; $s_{init}\in S$ is the initial state; Σ is a finite set of* events *(or* actions*); $\to$: $S\times\Sigma\times S$ is a transition relation; Π is a set of* atomic propositions*, such that $\Sigma\cap\Pi=\emptyset$; and $L: S\to 2^\Pi$ is a labeling function that assigns to each state $s\in S$ exactly the subset of atomic propositions $L(s)$ that hold true in there.*

For convenience, we use $s\xrightarrow{\sigma}s'$ to denote the transition $(s,\sigma,s')\in\to$ from state s to s' under action σ. The transition system $\mathcal{T}$ is *deterministic* if for all $s\in S, \sigma\in\Sigma$, there exist at most one $s'\in S$, such that $s\xrightarrow{\sigma}s'$. A *trace* of $\mathcal{T}$ is an infinite alternating sequence of states and events $\tau = s_1\sigma_1 s_2\sigma_2\ldots\in(S\cdot\Sigma)^\omega$, such that $s_1 = s_{init}$ and $s_i\xrightarrow{\sigma_i}s_{i+1}$, for all $i\geq 1$. A *trace fragment* is a finite subsequence $\tau_{fin} = s_i\sigma_i\ldots s_j\sigma_j\in(S\cdot\Sigma)^*$ of an infinite trace $\tau = s_1\sigma_1\ldots\sigma_{i-1}s_i\sigma_i\ldots s_j\sigma_j s_{j+1}\ldots$. A trace $\tau = \tau_{pre}(\tau_{suf})^\omega$ is called a trace with a prefix-suffix structure; τ_{pre} is the trace prefix and τ_{suf} is the periodically repeated trace suffix. The set of all traces of $\mathcal{T}$ and the set of all traces of all transition systems are denoted by $\mathfrak{T}(\mathcal{T})$ and $\mathfrak{T}$, respectively. Analogously, $\mathfrak{T}_{fin}(\mathcal{T})$ and $\mathfrak{T}_{fin}$ denote the set of all finite trace fragments of $\mathcal{T}$ and of all transition systems, respectively. A trace $\tau = s_1\sigma_1 s_2\sigma_2\ldots$ of $\mathcal{T}$ produces a *word* $w(\tau) = w(1)w(2)\ldots$, where $w(i) = L(s_i)\cup\{\sigma_i\}$.

Definition 2 (Trace Fragments Distance) *Given $\tau_{fin} = s_i\sigma_i\ldots s_j\sigma_j\in\mathfrak{T}_{fin}$, and a subset of indexes $I\subseteq\{i,\ldots,j\}$, the* restricted *trace fragment $\tau_{fin|I}$ is created by removing the states and actions labeled with indexes from I, i.e., $\forall\ell_k\in I$, $\tau_{fin|I} = s_i\sigma_i\ldots s_{\ell_k-1}\sigma_{\ell_k-1}s_{\ell_k+1}\sigma_{\ell_k+1}\ldots s_j\sigma_j$.*

The distance *between trace fragments $\tau_{fin},\tau'_{fin}\in\mathfrak{T}_{fin}$ is* $\mathsf{dist}(\tau_{fin},\tau'_{fin}) = \min(|I|+|I'|)$, *s.t.* $w(\tau_{fin|I}) = w(\tau'_{fin|I'})$.

Intuitively, the distance $\mathsf{dist}(\tau_{fin},\tau'_{fin})$ is the number of states and actions that have to be removed from and added to τ_{fin} for it to produce the same word as τ'_{fin}.

A (control) strategy $\Omega: (S\cdot\Sigma)^*\cdot S\to\Sigma$ for $\mathcal{T}$ is a function that assigns to each trace prefix followed with a state the next action to be executed. A trace $s_1\sigma_1 s_2\sigma_2\ldots$ is a trace *under* the controller Ω if for all $i\geq 1$, it holds that $\sigma_i = \Omega(s_1\sigma_1\ldots s_i)$. If $\mathcal{T}$ is deterministic, then there is only one trace under each Ω, and thus we talk about the trace as about the strategy itself. A *finite-memory* control strategy is intuitively one, whose outputs depend only on last m states and actions. Such a strategy can be represented an Input/Output (I/O) automaton.

Definition 3 (I/O Automaton) *An I/O automaton is a tuple $\mathcal{I} = (Q, q_{init}, \Sigma_{in}, \Sigma_{out}, \delta)$, where Q is a finite set of states; q_{init} is the initial state; Σ_{in} is an input alphabet; Σ_{out} is an output alphabet; and $\delta: Q\times\Sigma_{in}\to Q\times\Sigma_{out}$ is a partial transition function.*

A run of an I/O automaton $\mathcal{I}$ over an *input* word $\iota_1\iota_2\ldots\in\Sigma_{in}^\omega$ is a sequence of states $q_1q_2\ldots$, with the property that $q_1 = q_{init}$, and $\forall i\geq 1$, $\exists o_i\in\Sigma_{out}$, such that $\delta(q_i,\iota_i) = (q_{i+1},o_i)$. The *output* sequence of such a run is $o_1o_2\ldots$.

The I/O automaton $\mathcal{I}$ can serve as a finite-memory control strategy for a transition system $\mathcal{T} = (S, s_{init}, \Sigma, \to, \Pi, L)$ as follows. Assume that $\Sigma_{in} = S$, and $\Sigma_{out} = \Sigma$. Then, the input words for $\mathcal{I}$ are state sequences of $\mathcal{T}$ and the output sequences are action sequences for $\mathcal{T}$. The transition system $\mathcal{T}$ is *controlled* by $\mathcal{I}$ in the following way: Assume that s is the current state of $\mathcal{T}$, q is the current state of $\mathcal{I}$, and $\delta(q,s) = (q',a)$. Then q' becomes the new current state of $\mathcal{I}$, and a state s', such that $s\xrightarrow{a}s'$ becomes the new current state of $\mathcal{T}$. There might be multiple states s', such that $s\xrightarrow{a}s'$, in which case, one of them is chosen nondeterministically, corresponding to the fact that the application of the action a may have several different effects. If there is no s', such that $s\xrightarrow{a}s'$ in $\mathcal{T}$, then $\mathcal{I}$ is not a valid control strategy for $\mathcal{T}$.

State/Event Linear Temporal Logic: To specify the robot's goals, we will use state/event variant of LTL [11] that allows to capture requirements on both the robot's state or its location and the actions it executes.

Definition 4 (SE-LTL) *Let us consider transition system $\mathcal{T} = (S, s_{init}, \Sigma, \to, \Pi, L)$. A State/Event Linear Temporal Logic (SE-LTL) formula ϕ over the set of atomic propositions Π and the set of events Σ is defined inductively as follows:*

1) *every $\pi\in\Pi$ and every $\sigma\in\Sigma$ is a formula, and*
2) *if ϕ_1 and ϕ_2 are formulas, then $\phi_1\vee\phi_2$, $\neg\phi_1$, $\mathsf{X}\,\phi_1$, $\phi_1\,\mathsf{U}\,\phi_2$, $\mathsf{G}\,\phi_1$, and $\mathsf{F}\,\phi_1$ are each formulas,*

where $\neg$ (negation) and $\vee$ (disjunction) are standard Boolean connectives, and X, U, G, and F are temporal operators.

Informally speaking, the trace $\tau = s_1\sigma_1\ldots$ of $\mathcal{T}$ *satisfies* the atomic proposition π if π is satisfied in s_1 and the event σ if $\sigma_1 = \sigma$. The formula $\mathsf{X}\,\phi$ states that ϕ holds in the following state, whereas $\phi_1\,\mathsf{U}\,\phi_2$ states that ϕ_2 is true eventually, and ϕ_1 is true at least until ϕ_2 is true.

The formulas $\mathsf{F}\phi$ and $\mathsf{G}\phi$ state that ϕ holds eventually and always, respectively. For the full semantics of SE-LTL, see [11].

The trace-satisfaction relation can be easily extended to satisfaction relation for words over $2^{\Pi\cup\Sigma}$. Particularly, a word w satisfies an SE-LTL formula ϕ over Π and Σ if there exists a trace $\tau \in \mathfrak{T}$, such that $\tau \models \phi$ and $w = w(\tau)$. The *language* of all words that satisfy ϕ is denoted by $\mathcal{L}(\phi)$.

Büchi Automata: An alternative way to capture a property defined by an SE-LTL formula is a Büchi automaton.

Definition 5 (Büchi Automaton) *A Büchi automaton (BA) is a tuple $\mathcal{B} = (Q, q_{init}, \Upsilon, \delta, F)$, where Q is a finite set of states; $q_{init} \in Q$ is the initial state; Υ is an input alphabet; $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation; and F is the acceptance condition.*

A *run* of a Büchi automaton $\mathcal{B}$ *over* an input word $w = w(1)w(2)\ldots \in \Upsilon^\omega$ is a sequence $\rho = q_1 w(1) q_2 w(2)\ldots$, such that $q_1 = q_{init}$, and $(q_i, w(i), q_{i+1}) \in \delta$, for all $i \geq 1$. A run ρ is *accepting* if it intersects F infinitely many times. A word w is *accepted* by $\mathcal{B}$ if there exists an accepting run over w. The *language* of all words accepted by $\mathcal{B}$ is denoted by $\mathcal{L}(\mathcal{B})$. A Büchi automaton is *tight* if and only if for all $w \in \mathcal{L}(\mathcal{B})$ there exists an accepting run $q_1 w(1) q_2 w(2)\ldots$ with the property that $w(i)q_{i+1}w(i+1) = w(j)q_{j+1}w(j+1) \Rightarrow q_i = q_j$. Any SE-LTL formula over Π and Σ can be translated into a BA with $\Upsilon = 2^{\Pi\cup\Sigma}$, such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{B})$ using standard techniques [11]. Furthermore, some translation algorithms (e.g., [19]), guarantee that the resulting BA is tight [20].

Graph Theory: Any automaton or transition system can be viewed as a graph $\mathcal{G} = (V, E)$ with the set of vertices V equal to the set of states, and the set of edges E given by the transition function in the expected way. A *simple path* in $\mathcal{G}$ is a sequence of states $v_i \ldots v_l$ such that $(v_j, v_{j+1}) \in E$, for all $i \leq j < l$, and $v_j = v_{j'} \Rightarrow j = j'$, for all $i \leq j, j' \leq l$. A *cycle* is a sequence of states $v_i \ldots v_l v_{l+1}$, where $v_i \ldots v_l$ is a simple path and $v_{l+1} = v_i$. A weighted graph is a tuple $\mathcal{G} = (V, E, W)$, where $W : E \to \mathbb{N}$ is a a function assigning weights to the edges. The length of a path $v_i \ldots v_l$ is the sum of its respective weights $\mathsf{len}(v_i \ldots v_l) = \sum_{j\in\{i,\ldots,l-1\}} W(v_j, v_{j+1})$. A shortest path from v to v' is a path minimizing its length and can be found efficiently using e.g., Dijkstra's shortest path algorithm (see, e.g. [21]).

III. Problem Formulation and Approach

Our goal in this work is to algorithmically synthesize a controller for an autonomous robot that guarantees that the desired long-term goal is met as closely as possible, even in cases when the robot fails to execute some of the actions in its repertoire. Our approach to the problem builds on formal methods; we model the expected robot's action capabilities as a transition system and the goal as an SE-LTL formula. The proposed solution comprises two steps. First, we define a quantitative metric to measure the *level of noncompliance* of a robot's behavior with respect to the given SE-LTL formula and we synthesize a *maximally satisfying* discrete, reactive control strategy in the form of an I/O automaton. Second, we implement the control strategy through an automatic translation into a middle-layer controller that takes the form of a BT. This step interfaces the high-level planner with the low-level controller that executes action primitives in a continuous workspace. As a proof of concept, we have implemented the solution in a testbed with a NAO robot executing walking, ball grasping and ball dropping motion primitives.

In the remainder of this section, we focus on formalizing the first of the mentioned steps, i.e. the maximally satisfying control strategy synthesis problem. The second step, i.e. the implementation of the strategy in the continuous domain is discussed later on in Section V.

Maximally Satisfying Discrete Action Planning Problem: Similarly as in some related papers (e.g., [3], [6], [10]) we model the robot as a TS $\mathcal{T} = (S, s_{init}, \Sigma, \to, \Pi, L)$. The set of states S encode the possible states of the robot, such as its position in the environment, or the object it is carrying. The atomic propositions and the labeling function capture properties of interest of the robot's states. The actions in Σ abstract the *action primitives*, such as "move from position A to position B", or "grasp a ball" and the transition function expresses the evolution of the robot's states upon a successful execution of an action primitive. If the execution of an action fails, the robot stays at the same state. Formally, we define a *success-deterministic* transition system as the robot's model.

Definition 6 (Success-deterministic Transition System) *A success-deterministic transition system $\mathcal{T}$ is a syntactically deterministic transition system with specific nondeterministic semantics defined as follows: The traces of $\mathcal{T}$ are sequences $s_1\sigma_1 s_2\sigma_2\ldots$ and for each $i \geq 1$ it holds that*

- *either $s_i \xrightarrow{\sigma_i} s_{i+1}$ (σ_i succeeds in s_i), or*
- *$s_i = s_{i+1}$, and $\exists s'_{i+1}$, s.t. $s_i \xrightarrow{\sigma_i} s'_{i+1}$ (σ_i fails in s_i).*

Intuitively, the success-deterministic transition system has two possible outcomes for each action execution attempt. If the action succeeds, the transition is followed as expected; if it fails, the system's state does not change.

The specification of the robot's task takes a form of an SE-LTL formula ϕ. We specifically focus on recurrent tasks, that involve, e.g., periodic visits to selected regions of the environment or repeated execution of certain actions. To pose the problem formally, we consider a special surveillance event σ_{sur} to be periodically executed. Intuitively, the execution of σ_{sur} indicates that the robot has finished a single *surveillance cycle*. The action σ_{sur} is available in $S_{sur} \subseteq S$, and for all $s \in S_{sur}$, the transition enabled by σ_{sur} is $s \xrightarrow{\sigma_{sur}} s$ only. The attempt to execute the event σ_{sur} is always successful.

The SE-LTL formula ϕ is of form

$$\phi = \varphi \wedge \mathsf{GF}\sigma_{sur}, \tag{1}$$

where φ is any SE-LTL formula over Π and Σ, such that σ_{sur} does not appear in φ. In other words, there are no

other demands imposed on the execution of the surveillance event σ_{sur} apart from its infinite repetition, whereas arbitrary requirements and restrictions can be prescribed for the other actions and atomic propositions.

Example 1 *In Fig. 1 we give an example of a NAO robot's workspace partitioned into 9 regions; 6 rooms and 3 corridor regions. We assume the following set of action and motion primitives Σ: r, b, l, t (move to the neighboring region on the right, bottom, left, or top, respectively), grab (grab a ball), drop (drop the ball), and $\sigma_{sur} = light\_up$. The robot's state is compound of the region the robot is present at and whether or not it holds a ball. The robot is not allowed to cross a black line, and therefore the motion primitives r, b, l, t are not enabled in all the states. Similarly, the action $grab$ is enabled only in the set of regions $G \subseteq \{R_1, \ldots, R_6, C_1, \ldots, C_3\}$, where an object to be grabbed is expected to be present. Initially, the NAO is present in R_1, with no object in its hand. The transition system model is $\mathcal{T} = (S, s_{init}, \Sigma, \rightarrow, \Pi, L)$, where*

- $S = \{R_1, \ldots, R_6, C1, \ldots, C_3\} \times \{1, 0\}$; $s_{init} = (R_1, 0)$; $\Sigma = \{r, b, l, t, grab, drop, light\_up\}$; $\Pi = \{R_1, \ldots, R_6\}$;
- $(R_1, 0) \xrightarrow{light\_up} (R_1, 0)$; *for all* $i \in \{1, 2, 3\}, x \in \{1, 0\}$: $(C_i, x) \xrightarrow{r} (C_{i+1}, x)$, $(C_{i+1}, x) \xrightarrow{l} (C_i, x)$ $(R_i, x) \xrightarrow{b} (C_i, x)$, $(C_i, x) \xrightarrow{t} (R_i, x)$, $(R_{i+3}, x) \xrightarrow{t} (C_i, x), (C_i, x) \xrightarrow{b} (R_{i+3}, x)$, *for all* $R_i \in G$: $(R_i, 0) \xrightarrow{grab} (R_i, 1)$, *for all* $i \in \{1, \ldots, 6\}$: $(R_i, 1) \xrightarrow{drop} (R_i, 0)$;
- *for all* $i \in \{1, 2, 3\}, x \in \{1, 0\}$: $L((R_i, x)) = \{R_i\}$, $L((R_{i+3}, x)) = \{R_{i+3}\}$, $L((C_i, x)) = \emptyset$, .

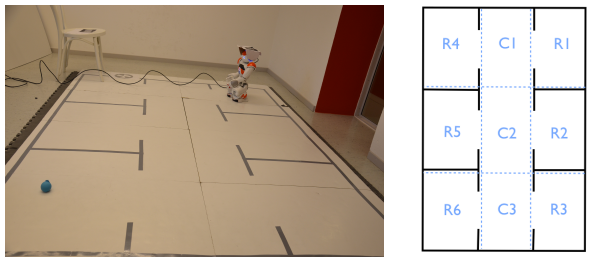

Fig. 1: A NAO robot's workspace (left). A scheme of the workspace partitioned in regions (right).

An example of an SE-LTL robot's task is to periodically 1) survey R_5, 2) grab a ball in R_4 and 3) bring it to R_2: $\phi = \mathsf{GF} R_5 \wedge \mathsf{GF}(R_4 \wedge grab \wedge \mathsf{F}(R_2 \wedge drop)) \wedge \mathsf{GF}\ light\_up$.

Before stating our problem formally, let us introduce several intermediate definitions.

Definition 7 (Surveillance Trace Fragment) *A finite trace fragment $\tau_{sur} = s_i \sigma_i \ldots s_j \sigma_j \in \mathfrak{T}_{fin}$ is called a* surveillance fragment *if $\sigma_j = \sigma_{sur}$, and $\sigma_\ell \neq \sigma_{sur}$ for all $i \leq \ell < j$.*

In other words, a surveillance trace fragment ends with a surveillance event, and it does not contain any other surveillance event besides this one. Let $\mathfrak{T}_{sur}(\mathcal{T})$ and $\mathfrak{T}_{sur}$ denote the set of all surveillance trace fragments of $\mathcal{T}$ and of all transition systems, respectively.

Definition 8 (Level of Noncompliance) *The* level of noncompliance $\lambda(\tau_{sur})$ *of a surveillance fragment $\tau_{sur} \in \mathfrak{T}_{sur}$ is the minimal distance (see Def. 2) to a trace fragment $\tau'_{sur} \in \mathfrak{T}_{sur}$, with the property that for some surveillance trace prefix $\tau_{surp} \in \mathfrak{T}_{sur}$, $\tau_{surp}(\tau'_{sur})^\omega \models \phi$.*

Intuitively, the level of noncompliance for a surveillance fragment τ_{sur} expresses how far the infinite repetitive execution of τ_{sur} is from satisfying the LTL formula ϕ. Note that if there exists a prefix $\tau_{surp} \in \mathfrak{T}_{sur}$, such that $\tau_{surp}(\tau_{sur})^\omega \models \phi$, then $\lambda(\tau_{sur}) = 0$. Dually, if there does not exist any $\tau_{surp}(\tau'_{sur})^\omega \models \phi$, then $\lambda(\tau_{sur}) = \infty$ for all trace fragments $\tau_{sur} \in \mathfrak{T}_{sur}$. Furthermore, we set $\lambda(\tau_{fin}) = \infty$, for all trace fragments $\tau_{fin} \in \mathfrak{T}_{fin} \setminus \mathfrak{T}_{sur}$.

Remark 1 *We limit our attention to traces with a prefix-suffix structure only, and hence, on finite memory controllers. This assumption is in fact not restrictive; well-known results from model-checking show, that the existence of a trace of $\mathcal{T}$ satisfying an LTL or an SE-LTL formula implies the existence of such a trace with a prefix-suffix structure [1].*

We aim to propose a control strategy synthesis algorithm that copes with possible failures of the action primitives executions. Technically, the action failures can be captured by introducing transitions $s \xrightarrow{\sigma} s$, for all $\sigma \in \Sigma$ and $s \in S$ in $\mathcal{T}$, thus making $\mathcal{T}$ nondeterministic and treating the problem using a standard control strategy synthesis algorithm for a nondeterministic system. Such an approach is very conservative as the control strategy Ω is required to guarantee the satisfaction of ϕ regardless the action failures, including the unlikely scenario when all of the action fail at all times. Due to such cases, the desired Ω often does not exist at all.

In contrast, our approach is to find a strategy that responds to the action failures locally to achieve maximal level of satisfaction of the desired specification. Our default assumption is that all actions can be successfully executed; if it is found out that an action fails upon the run of the system, we change our original assumption to the belief that this particular action would fail for the rest of the current surveillance fragment as well, however it would be executed successfully again later, after the current surveillance fragment is finished.

Formally, consider a trace prefix $\tau_{pre} = \tau_{sur_1} \ldots \tau_{sur_m} s_i \sigma_i \ldots s_j \sigma_j$ of $\mathcal{T}$, where

- $\tau_{sur_\ell}, 1 \leq \ell \leq m$ are surveillance fragments, and
- $\sigma_\ell \neq \sigma_{sur}$, for all $i \leq \ell \leq j$.

We denote by $\mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$ the set of all traces $\tau = \tau_{pre}\tau_{suf}$ of $\mathcal{T}$ with the prefix τ_{pre} that satisfy the following conditions:

- $\tau = \overbrace{\tau_{sur_1} \ldots \tau_{sur_m} s_i \sigma_i \ldots s_j \sigma_j}^{\tau_{pre}} \overbrace{s_{j+1} \sigma_{j+1} \ldots s_k \sigma_k (\tau_{sur})^\omega}^{\tau_{suf}}$, s.t. $s_i \sigma_i \ldots s_k \sigma_k$, and τ_{sur} are surveillance fragments;
- for all $i \leq \ell \leq j, j < \ell' \leq k$, s.t. $s_\ell = s'_\ell$ and $\sigma_\ell = \sigma'_\ell$ it holds that σ_ℓ fails in s_ℓ iff $\sigma_{l'}$ fails in s'_l; and
- for all $\ell > k$ it holds that σ_ℓ succeeds in s_ℓ.

The quality of the trace τ is measured in terms of the level of noncompliance of its surveillance fragments. Namely, we are interested in two measures:

- *Long-term level of noncompliance* $\lambda_\infty(\tau) = \lambda(\tau_{sur})$
- *Transient level of noncompliance* $\lambda_\downarrow(\tau) = \lambda(s_i \ldots \sigma_k)$.

Note that given a trace prefix τ_{pre}, and a controller Ω, there is only one trace under Ω that belongs to $\mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$, since the nondeterminism is resolved by assuming explicitly which actions fail and which succeed.

We are now ready to state our problem, Problem 1.

Problem 1 (Discrete Planning Problem) Given *a robot modeled as a success-deterministic transition system $\mathcal{T}$ and a SE-LTL formula $\phi = \varphi \wedge \mathsf{GF}\sigma_{sur}$,* synthesize *an I/O automaton representing a so-called* maximally satisfying *control strategy function Ω that satisfies the following conditions: For all prefixes τ_{pre} of $\mathcal{T}$, the trace $\tau \in \mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$ under Ω*

(i) *minimizes $\lambda_{\infty}(\tau)$ among all traces in $\mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$, and*
(ii) *minimizes $\lambda_{\downarrow}(\tau)$ among all traces in $\mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$ satisfying condition* (i)*.*

IV. Solution to the Discrete Planning Problem

Let us first focus on solving Problem 1 under the assumption that $\mathcal{T}$ is purely deterministic, i.e., we first assume that none of the actions can fail. We propose an algorithm to find a maximally satisfying control trace τ.

Following the automata-based approach to model checking [1] and similar ideas as in [15], [16], we first construct a specialized weighted product automaton $\mathcal{P}$ that captures not only the traces of $\mathcal{T}$ that satisfy $\mathcal{B}$, but also the level of noncompliance of individual trace fragments through its weights. Using standard graph algorithms, we find a lasso-shaped accepting run in $\mathcal{P}$ that projects onto the maximally satisfying trace τ of $\mathcal{T}$ with a prefix-suffix structure.

Definition 9 (Weighted Product Automaton) *A product of a deterministic TS $\mathcal{T} = (S, s_{init}, \Sigma, \rightarrow, \Pi, L)$ and a tight BA $\mathcal{B} = (Q, q_{init}, 2^{\Pi \cup \Sigma}, \delta, F)$ is a weighted BA $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \Sigma, \delta_{\mathcal{P}}, F_{\mathcal{P}}, W_{\mathcal{P}})$, where $Q_{\mathcal{P}} = S \times Q$; $q_{init,\mathcal{P}} = (s_{init}, q_{init})$; $F_{\mathcal{P}} = S \times F$; and*

- $((s,q), \sigma, (s',q')) \in \delta_{\mathcal{P}}$, $W_{\mathcal{P}}((s,q), \sigma, (s',q')) = 0$ *if*
  a) $s \xrightarrow{\sigma} s'$ *in* $\mathcal{T}$, *and* $(q, L(s) \cup \{\sigma\}, q') \in \delta$;
- $((s,q), \sigma, (s',q')) \in \delta_{\mathcal{P}}$, $W_{\mathcal{P}}((s,q), \sigma, (s',q')) = 1$ *if*
  b) $s \xrightarrow{\sigma} s'$ *in* $\mathcal{T}$, *and* $q = q'$, *or*
  c) $s = s'$, *and* $(q, L(s) \cup \{\sigma\}, q') \in \delta$;

A run $\rho = (s_1, q_1)\sigma_1(s_2, q_2)\sigma_2 \ldots$ of $\mathcal{P}$ projects onto the trace $\tau = \alpha(\rho) = s_1\sigma_1 s_2 \sigma_2 \ldots$ and onto the run $\rho_{\mathcal{B}} = \beta(\rho) = q_1\sigma_1 q_2 \sigma_2 \ldots$. Analogously as for $\mathcal{T}$, we define surveillance run fragments for $\mathcal{P}$; ρ_{fin} is a surveillance run fragment if $\alpha(\rho_{fin})$ is a surveillance trace fragment. We say that a run ρ is with a *surveillance prefix-suffix structure* if $\rho = \rho_{surp}(\rho_{sur})^{\omega}$ in $\mathcal{P}$, such that ρ_{surp}, ρ_{sur} are surveillance fragments. The product automaton can be viewed as a graph (see Sec. II-B) and therefore, with a slight abuse of notation, we use $\mathsf{len}(\rho_{fin})$ to denote the sum of weights on a finite run fragment ρ_{fin}. The accepting runs of the weighted product automaton project onto traces of $\mathcal{T}$ and the weights of the transitions in $\mathcal{P}$ capture the level of noncompliance of the traces as explained through the following two lemmas.

Lemma 1 *Let $\tau = \tau_{surp}(\tau_{sur})^{\omega}$ be a trace of $\mathcal{T}$, such that $\tau_{surp}, \tau_{sur} \in \mathfrak{T}_{sur}(\mathcal{T})$ are surveillance fragments. Then there exists an accepting run $\rho_{surp}(\rho_{sur})^{\omega}$ in $\mathcal{P}$, such that*

(i) $\alpha(\rho_{surp}) = \tau_{surp}, \alpha(\rho_{sur}) = \tau_{sur}$
(ii) $\lambda(\tau_{sur}) = \mathsf{len}(\rho_{sur})$.

Vice versa, any accepting run $\rho = \rho_{surp}(\rho_{sur})^{\omega}$ in $\mathcal{P}$ with a surveillance prefix-suffix structure projects onto a trace $\tau = \alpha(\rho)$ satisfying conditions (i)-(ii) *above.*

Proof: The proof is lead via induction w.r.t. $\lambda(\tau_{sur})$.

Let $\tau = \tau_{surp}(\tau_{sur})^{\omega}$ be a trace of $\mathcal{T}$, s.t. $\lambda(\tau_{sur}) = 0$. Then there exists τ'_{surp}, such that $w(\tau') = \tau'_{surp}(\tau_{sur})^{\omega}$ is accepted by $\mathcal{B}$. Following classical results from model-checking [1], we get that there exists an accepting run $\rho'_{surp}(\rho_{sur})^{\omega}$ of $\mathcal{P}$ that projects onto τ'. All transitions along this run are of type *a)*, and therefore $\mathsf{len}(\rho_{sur}) = 0$. By replacement of τ'_{surp} with τ_{sur}, we replace also $\rho'_{surp} = (s_1, q_1)\sigma_1 \ldots (s_i, q_i)\sigma_i$ with a run fragment ρ_{surp} that ends with $(s_i, q_i)\sigma_i$, is over τ_{sur}. Such a fragment can be obtained following a sequence of *c)*-type transitions to state (s_1, q_i) and then *b)*-type transitions to (s_i, q_i). For analogous reasons, any accepting run $\rho_{surp}(\rho_{sur})^{\omega}$ in $\mathcal{P}$ with $\mathsf{len}(\rho_{sur}) = 0$ projects onto a trace $\tau = \tau_{surp}(\tau_{sur})^{\omega}$ with $\lambda(\tau_{sur}) = 0$.

Let the lemma hold for all $1 \leq k < \lambda(\tau_{sur})$. Consider a trace $\tau = \tau_{surp}(\tau_{sur})^{\omega}$ of $\mathcal{T}$ with $\lambda(\tau_{sur}) = k + 1$. Then, necessarily either $\tau = uv$ and there exists $\tau'_{sur} = us\sigma v$ with $\lambda(\tau'_{sur}) = k$, or $\tau = us\sigma v$ and there exists $\tau'_{sur} = uv$ with $\lambda(\tau'_{sur}) = k$. For the former case, there exists $\rho'_{sur} = \rho_u \rho_v$ with $\mathsf{len}(\rho'_{sur}) = k$. By injecting a c)-type transition between ρ_u and ρ_v we obtain run ρ_{sur}. Thanks to the tightness of $\mathcal{B}$, $\mathsf{len}(\rho_{sur}) = k + 1$ and $\lambda(\rho_{sur}) = \tau_{sur}$. Analogous argument holds for the latter case with the difference of injecting a b)-type transition. Dually, an accepting run $\rho_{surp}(\rho_{sur})^{\omega}$ in $\mathcal{P}$ with $\mathsf{len}(\rho_{sur}) = k+1$ projects onto a trace $\tau = \tau_{surp}(\tau_{sur})^{\omega}$ with $\lambda(\tau_{sur}) = k + 1$. Thus, the proof is complete. ■

Lemma 2 *An accepting run $\rho = \rho_{surp}(\rho_{sur})^{\omega}$ of $\mathcal{P}$ with a surveillance prefix-suffix structure that minimizes $\mathsf{len}(\rho_{sur})$ among all such runs of $\mathcal{P}$ projects onto a maximally satisfying trace $\tau = \alpha(\rho)$ of $\mathcal{T}$.*

Proof: The proof follows directly from Lemma 1. ■

Lemmas 1 and 2 give us a guideline, how to find a maximally satisfying trace τ of a deterministic TS $\mathcal{T}$; when viewing $\mathcal{P}$ as a graph, a shortest cycle that is reachable from the initial state and contains both an accepting state and a surveillance action represents a run ρ of $\mathcal{P}$ that projects onto the sought τ. The algorithm is summarized in Alg. 1, where the functions `min_frag` and `min_sur_frag` compute the shortest run fragment and the shortest surveillance run fragment, respectively, using standard graph methods.

Finally, the maximally satisfying trace $\tau = \tau_{surp}(\tau_{sur})^{\omega}$, $\tau_{surp} = s_1\sigma_1 \ldots s_{i-1}\sigma_{i-1}$, $\tau_{sur} = s_i\sigma_i \ldots s_j\sigma_{sur}$ is translated into an I/O automaton $\mathcal{I} = (Q_{\mathcal{I}}, q_{init,\mathcal{I}}, \Sigma_{in}, \Sigma_{out}, \delta_{\mathcal{I}})$, where $Q_{\mathcal{I}} = \{q_1, \ldots, q_j\}$, $q_{init,\mathcal{I}} = q_1$, $\Sigma_{in} = \{succ, fail\}$, $\Sigma_{out} = \Sigma$, for all $1 \leq \ell < j$, $\delta_{\mathcal{I}}(q_\ell, succ) = (q_{\ell+1}, \sigma_\ell)$, and $\delta_{\mathcal{I}}(q_j, succ) = (q_i, \sigma_{sur})$. Note that state q_i can be reached

Algorithm 1: Maximally-satisfying Ω for a determ. $\mathcal{T}$

Input: Product $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \Sigma, \delta_\mathcal{P}, F_\mathcal{P}, W_\mathcal{P})$
Output: Control strategy given as a trace τ of $\mathcal{T}$
for all $p_s \in \{(s,q) \mid s \in S_{sur}\}$, $p_f \in F_\mathcal{P}$ **do**
  $\rho_{fin}(p_a, p_f)$ = `min_frag` $(\mathcal{P}, p_s, p_f)$
  $\rho_{fin}(p_f, p_s)$ = `min_sur_frag` $(\mathcal{P}, p_f, p_s)$
end
$\rho_{sur}(p_s)$ = `concatenate` $(\rho_{fin}(p_s, p_f), \rho_{fin}(p_f, p_s))$
$Opt = \{p_s$ minimizing $\mathsf{len}(\rho_{sur}(p_s))\}$
pick any $p_s \in Opt$ reachable from p_{init}
ρ_{surp} = `min_sur_frag` $(\mathcal{P}, p_{init}, p_s)$
$\tau = \alpha(\rho_{surp}(\rho_{sur})^\omega)$

Algorithm 2: Maximally-satisfying τ for a given τ_{pre}

Input: $\mathcal{T} \otimes \mathcal{B} = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \Sigma, \delta_\mathcal{P}, F_\mathcal{P}, W_\mathcal{P})$, τ_{pre}, *failed*
Output: Control strategy given as a trace $\tau \in \mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$
for all $p_j = (s_j, q)$, *for some* $q \in Q$, $p_s \in Opt$ *(Alg. 1)* **do**
  $\rho_{fin}(p_j, p_s)$ = `min_restrict_sur` $(\mathcal{P}, p_j, p_s, failed)$
end
$Opt\_transient = \{(p_j, p_s)$ minimizing $\mathsf{len}(\rho_{fin}(p_j, p_s))\}$
pick any $(p_j, p_s) \in Opt\_transient$
$\tau = \tau_{pre}\, \alpha(\rho_{fin}(p_j, p_s)(\rho_{sur}(p_s))^\omega)$

only right after execution of a surveillance event, hence we call it a *cycle start* and we remember the corresponding system state $start_i = s_i$.

Let us now consider a success-deterministic TS $\mathcal{T}$, its fixed trace prefix $\tau_{pre} = \tau_{sur_1} \dots \tau_{sur_m} s_i \sigma_i \dots s_j \sigma_j$ and a subset of indexes $failed \subseteq \{i, \dots, j\}$ meaning that σ_ℓ failed in s_ℓ, for all $\ell \in failed$. Let us concentrate on finding a trace $\tau = \tau_{pre} s_{j+1} \sigma_{j+1} \dots s_k \sigma_k (\tau_{sur})^\omega$ that minimizes the long-term and the transient level of noncompliance among the traces in $\mathfrak{T}_{\rightsquigarrow}(\mathcal{T}, \tau_{pre})$. The solution is given in Alg. 2 and involves two steps; finding all suitable, i.e., the shortest fragments τ_{sur}, and finding the shortest surveillance fragment leading to some of the found fragments τ_{sur} while taking into account that actions defined by *failed* are forbidden in the respective states (function `min_restrict_sur`). Because τ_{sur} corresponds to the long-term behavior and we assume all actions will be succeeding then, the suitable suffixes τ_{sur} are already determined by Opt in Alg. 1.

The remainder of the solution lies in enhancing the I/O automaton $\mathcal{I} = (Q_\mathcal{I}, q_{init,\mathcal{I}}, \Sigma_{in}, \Sigma_{out}, \delta_\mathcal{I})$ constructed for a deterministic variant of the system $\mathcal{T}$ with systematic action failures. Consider a prefix $\tau_{pre} = \tau_{sur_1} \dots \tau_{sur_m} s_i \sigma_i \dots s_j \sigma_j$, such that $failed = \{j\}$, and the current state q of the I/O automaton. The value $\delta_I(q, fail)$ is not defined. We apply Alg. 2 to find the desired trace suffix and we extend the automaton $\mathcal{I}$ accordingly: Given that the found suffix is $\tau_{suf} = s_{j+1} \sigma_{j+1} \dots s_k \sigma_k (s_{k+1} \sigma_{k+1} \dots s_l \sigma_l)^\omega$, we introduce new states $q'_{j+2} \dots q'_k$ in case that $s_{k+1} = start_i$, for some cycle start $q_i \in Q_\mathcal{I}$, or $q'_{j+2} \dots q_l$ otherwise. We set $\delta_I(q, fail) = (q'_{j+2}, \sigma_{j+1})$, and similarly as before, $\delta_I(q_\ell, succ) = (q_{\ell+1}, \sigma_\ell)$, for all $j+2 \le \ell < k$, or $j+2 \le \ell < m$, and $\delta_I(q_k, succ) = (q_{cyc}, \sigma_\ell)$ or $\delta_I(q_m, succ) = (q_{k+1}, \sigma_\ell)$, respectively. The rest of the solution to Problem 1 is obtained inductively in the straightforward way. The I/O automaton $\mathcal{I}$ is systematically searched through and whenever $\delta_I(q, fail)$ is not defined for some q, the action failure is simulated. The overall procedure is finite as Opt is finite as well as every surveillance fragment.

V. Action Plan Execution

The I/O automaton $\mathcal{I}$ obtained as a solution to Problem 1 represents a discrete action plan for an abstract model of the considered robotic system, however its application in such a dynamical system itself is not straightforward. Hence, in this section, we focus on the actual execution of the control strategy. Specifically, we propose an automatic translation of the I/O automaton $\mathcal{I}$ into a BT that maps abstract action primitives from Σ onto their respective continuous controllers and hence serves as a bridge between the high-level planning and the low-level control layers. We first present BTs following the notation in [12], where BTs were introduced in the context of robot control. Second, we extend the BTs with new features that enrich their expressive power to match the semantics of I/O automata and we focus on the translation from an I/O automaton to a BT itself.

Behavior Trees: Behavior Trees (BTs) are *rooted trees* that are used to represent and execute plans; *actions* and *conditions* are placed in its leaves, connected through internal *control-flow* nodes. The nodes are systematically evaluated, having a *Running*, *Success*, or *Failure* status. The node types are summarized in Table I.

Node	Symbol	Succeeds if	Fails if	Runs if
Root	∅	tree `S`	tree `F`	tree `R`
Selector(*)	?, ?*	1 `Ch S`	N `Ch F`	1 `Ch R`
Sequence(*)	→, →*	N `Ch S`	1 `Ch F`	1 `Ch R`
Parallel	⇉	$\geq K$ `Ch S`	$\geq L$ `Ch F`	otherwise
Decorator	◈	varies	varies	varies
Action	□	$X_\sigma(t) \in \mathsf{S}_\sigma$	$X_\sigma(t) \in \mathsf{F}_\sigma$	$X_\sigma(t) \in \mathsf{R}_\sigma$
Condition	○	$X_c(t) \in \mathsf{S}_c$	$X_c(t) \in \mathsf{F}_c$	never

TABLE I: The node types of a BT. `Ch` ≡ *children*, `S` ≡ *Success*, `F` ≡ *Failure*, `R` ≡ *Running*. $N \equiv$ *# children*, $K, L \in \mathbb{N}$ are node parameters, $X_\sigma(t)$, $X_c(t)$ are the continuous states, and $\mathsf{S}_\sigma, \mathsf{F}_\sigma$, and R_σ, and $\mathsf{S}_c, \mathsf{F}_c$ are the respective success, failure, and running subsets of the action σ, and the condition c, respectively.

The execution of a BT is governed through *ticks* representing discrete evaluation cycles. The tick is a periodic enabling signal, generated at the *Root* of the BT at every Δ_T. The tick is instantly propagated through the tree and triggers each node's pre-defined algorithm upon reaching it. For details on the relevant nodes see Alg. 3. Broadly speaking, control-flow nodes redirect the course of the program depending on the *Success* or *Failure* of their children, whereas action and condition leaves are evaluated over continuous variables and optionally perform a control algorithm. The tick frequency is assumed to be sufficiently high to respond to changes in the environment and control the robot in real-time.

Consider the *continuous trajectory* $(\sigma, \Delta_\sigma, X_\sigma(t), U_\sigma(t))$ associated with the action σ in a time interval $[0, \Delta_\sigma]$ during which σ is active, where $U_\sigma(t) : [0, \Delta_\sigma] \to \mathcal{U}$ is

Algorithm 3: BT Execution at each Δ_T

```
switch Node Type do
    case Root return Tick(child(0))
    endsw
    case Selector*
        for i ← run-index to N do
            status ← Tick(child(i))
            if status = Running then
                run-index ← i; return Running
            if status = Success then
                run-index ← 1; return Success
            end
        end
        run-index ← 1; return Failure
    endsw
    case Sequence*
        for i ← run-index to N do
            status ← Tick(child(i))
            if status = Running then
                run-index ← i; return Running
            if status = Failure then
                run-index ← 1; return Failure
            end
        end
        run-index ← 1; return Success
    endsw
    case Action σ
        if X_σ(t) ∈ S_σ then return Success
        if X_σ(t) ∈ F_σ then return Failure
        if X_σ(t) ∈ R_σ then
            U_σ(t) ← γ_σ(X_σ(t)); return Running
        end
    endsw
    case Condition
        if X_σ(t) ∈ S_σ then return Success
        if X_σ(t) ∈ F_σ then return Failure
    endsw
    ...
endsw
```

a piecewise continuous function, and $X_\sigma(t) : [0, \Delta_a] \to \mathcal{X}$ is a continuous piecewise differentiable function, such that $\forall t \in [0, \Delta_\sigma]$, $X_\sigma(t)$ stays in the invariant set and $f(X_\sigma(t), U_\sigma(t)) = \dot{X}_\sigma(t)$ except for the points of discontinuity. The *execution* of a BT is a sequence of *continuous trajectories*: $(\sigma_1, \Delta_{\sigma_1}, X_{\sigma_1}(t), U_{\sigma_1}(t)) \xrightarrow{status_1} (\sigma_2, \Delta_{\sigma_2}, X_{\sigma_2}(t), U_{\sigma_2}(t)) \xrightarrow{status_2} \cdots$. The times of the action node return status is *Success* or *Failure* are $t_{\sigma_1}, t_{\sigma_2}, \ldots$, where $t_{\sigma_n} = \sum_{i \in \{1,\ldots,n\}} \Delta_{\sigma_i}$. If $X_{\sigma_j}(t_{\sigma_j}) \in \mathtt{S}_{\sigma_j}$, i.e., if σ_j succeeded, then $status_j = \mathtt{S}_{\sigma_j}$. If $X_{\sigma_j}(t_{\sigma_j}) \in \mathtt{F}_{\sigma_j}$, i.e., if σ_j failed, then $status_j = \mathtt{F}_{\sigma_j}$.

To obtain the abstract *action-driven* BT, we, simply put, ignore the *Running* statuses of actions. At times $t_1, t_2 \ldots$, where $t_n = n \cdot \Delta_T, \forall n \geq 0$, the tick-driven BT produces the sequence of action evaluations that takes the following form: $\mathtt{R}_{\sigma_1}(1) \ldots \mathtt{R}_{\sigma_1}(j_1)\mathtt{S}/\mathtt{F}_{\sigma_1}(j_1{+}1) = \mathtt{R}_{\sigma_2}(1) \ldots \mathtt{S}/\mathtt{F}_{\sigma_i}(j_i{+}1) = \mathtt{R}_{\sigma_{i+1}}(1) \ldots$, where $\mathtt{R}_\sigma$, $\mathtt{S}_\sigma$, and $\mathtt{F}_\sigma$ denotes that action σ is *Running*, *Success* or *Failure*, respectively. The action-driven BT produces a sequence of successes and failures $\mathtt{S}/\mathtt{F}_{\sigma_1}\mathtt{S}/\mathtt{F}_{\sigma_2} \ldots$, i.e., a sequence $status_1, status_2, \ldots$, hence allowing us to view each action as an atomic primitive.

The action-driven BT represents a function $(\Sigma \times \{\mathtt{S}_\sigma, \mathtt{F}_\sigma \mid \sigma \in \Sigma\})^* \to \Sigma$, where Σ is the set of BT actions. In other words, given the history of the actions executed and their statuses, the next action to be executed is determined. As such, if all actions succeed, the BT can be viewed as a control strategy for a deterministic transition system $\mathcal{T} = (S, s_{init}, \Sigma, \to, \Pi, L)$, in which all actions are assumed to succeed, too.

I/O Automaton to BT Translation: To translate an I/O automaton $\mathcal{I} = (Q_\mathcal{I}, q_{init,\mathcal{I}}, \Sigma_{in}, \Sigma_{out}, \delta_\mathcal{I})$ into a BT. Since standard BTs lack mechanisms to explicitly store history of their execution (see [12] for a discussion), which is in an I/O automaton provided through its states, we introduce two global variables: q_{curr} to remember a current state of the I/O automaton and ls to store the success/failure status of the latest executed action. Furthermore, we introduce the *Update Decorator* to maintain the correct values of the global variables, which is presented in Alg. 4.

Algorithm 4: Update Decorator

```
status ← Tick(child)
if status = Success then
    ls ← Success; q_curr ← δ_I(q_curr, succ)
else if status = Failure then
    ls ← Failure; q_curr ← δ_I(q_curr, fail)
return status
```

The translation procedure is illustrated in Fig. 2. Note that the action nodes can be easily substituted with sub-trees that implement the respective actions.

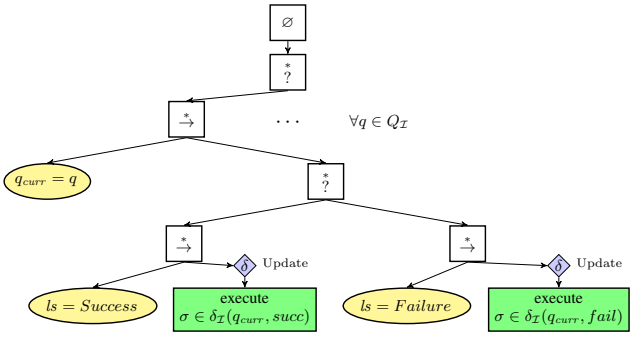

Fig. 2: Scheme of the I/O Automata BT translation using global variables and Update Decorator.

With the translation from I/O automata to BTs, we conclude the solution to Problem 1 and its execution by a robot.

VI. Experiments

We implemented the proposed solution in Robot Operating System (ROS) and tested it in a NAO robot testbed described in Example 1. The NAO humanoid robot has basic motion and grasping capabilities. Due to low resolution of its native vision, we used a custom positioning system with a camera overlooking its workspace. To recognize the robot's position, its head was equipped with an easily distinguishable pattern.

To illustrate the results of our approach, we consider three different SE-LTL tasks involving motion between regions,

grasping and dropping a ball. The grasping action has shown as the most critical, often failing one. Thus, for simplicity of presentation, in Fig. 3 we depict resulting trajectories while considering only grasping action failures.

Case A: Periodically grab a ball in R_6 and drop it in R_2:

$$\mathsf{GF}(R_4 \wedge grab \wedge \mathsf{F}(R_2 \wedge drop)) \wedge \mathsf{GF}\, light\_up.$$

The executed surveillance cycle is depicted in Fig. 3 on the left. After a grasp failure, room R_2 is not visited as $R_2 \wedge drop$ cannot be satisfied.

Case B: Periodically grab a ball in R_4 or in R_5 and drop it in R_2 (illustrated in Fig. 3 in the middle):

$$\mathsf{GF}((R_4 \wedge grab \vee R_5 \wedge grab) \wedge \mathsf{F}(R_2 \wedge drop)) \wedge \mathsf{GF}\, light\_up.$$

Case C: Periodically visit R_2, R_3, R_6, R_5 and R_4 in this order, while in R_6 grab a ball and drop it later on in R_4 (illustrated in Fig. 3 on the right). The formula says that if in R_1 then no other room should be visited until R_2 is visited, then no other room should be visited untill R_4 is visited, etc., until R_1 is reached again:

$$\mathsf{G}(R_1 \Rightarrow \bigwedge_{i\neq 1} \neg R_i \,\mathsf{U}\, R_2 \wedge (\bigwedge_{i\neq 2} \neg R_i \,\mathsf{U}\, R_3 \wedge (\bigwedge_{i\neq 3} R_i \,\mathsf{U}\, (R_6 \wedge grab) \wedge (\bigwedge_{i\neq 6} \neg R_i \,\mathsf{U}\, R_5 \wedge (\bigwedge_{i\neq 5} \neg R_i \,\mathsf{U}\, (R_4 \wedge drop) \wedge (\bigwedge_{i\neq 4} \neg R_i \,\mathsf{U}\, R_1)))))) \wedge \mathsf{GF}\, light\_up.$$

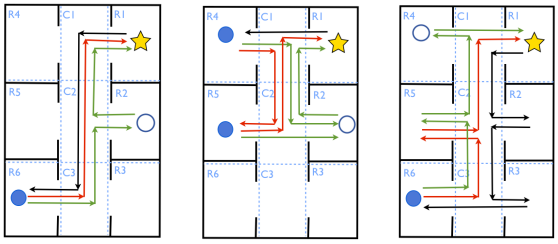

Fig. 3: NAO robot's surveillance cycles for test cases A,B,C. The yellow star represents *light_up* action. The filled and empty circles depict *grasp* and *drop*, respectively. The black arrows illustrate the beginning of a surveillance cycle, before *grasp* is attempted. The continuation of the trajectory after a successful and a failed *grasp* is in green and red, respectively.

Results from the experimental run of Case B in the testbed are demonstrated in the movie that accompanies this paper.

VII. Conclusions and Future Work

We have proposed an algorithm to find maximally satisfying action plans and interfaced the high-level planner with a low-level robot controller through automatic translation to BTs. As a future work, we will explore an option to update the BT upon a run of a system instead of its offline generation.

Acknowledgement: We would like to thank Meng Guo and Michele Colledanchise for their help on the NAO testbed.

References

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

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

[3] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal Path Planning for Surveillance with Temporal Logic Constraints," *International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[4] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 55 –64, 2011.

[5] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[6] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2013, pp. 3197–3204.

[7] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Motion and action planning under LTL specifications using navigation functions and action description language," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 240–245.

[8] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469 – 482, 2010.

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

[10] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5010–5017.

[11] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha, "State/event-based software model checking," in *Integrated Formal Methods*. Springer, 2004, pp. 128–147.

[12] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5420–5427.

[13] A. Marzinotto, M. Colledanchise, and I. Tjernberg. (2013) Behavior trees library for the robot operating system (ROS). [Online]. Available: https://github.com/almc/behavior_trees

[14] M. Kloetzer and C. Belta, "Managing non-determinism in symbolic robot motion planning and control," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[15] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2013, pp. 1–10.

[16] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2013, pp. 3217–3224.

[17] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2013, pp. 353–362.

[18] K. Kim and G. Fainekos, "Approximate solutions for the minimal revision problem of specification automata," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[19] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Proceedings of the Fifteenth IFIP WG6. 1 International Symposium on Protocol Specification, Testing and Verification*. IFIP, 1995.

[20] V. Schuppan and A. Biere, "Shortest counterexamples for symbolic model checking of LTL with past," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3440, pp. 493–509.

[21] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*. MIT press, 2001.