Motion and Action Planning under LTL Specifications using Navigation Functions and Action Description Language

Meng Guo, Karl H. Johansson and Dimos V. Dimarogonas

Abstract— **We propose a novel framework to combine model-checking-based robot motion planning with action planning using action description languages, aiming to tackle task specifications given as Linear Temporal Logic (LTL) formulas. The specifications implicitly require both sequential regions to visit and the desired actions to perform at these regions. The robot's motion is abstracted based on sphere regions of interest in the workspace and the structure of navigation function(NF)-based controllers, while the robot's action map is constructed based on precondition and effect functions associated with the actions the robot is capable of. An optimal planner is designed that generates the discrete motion-and-action plan fulfilling the task specification, as well as the low-level hybrid controllers that implement this plan. The whole framework is demonstrated by a case study.**

I. Introduction

Navigation functions proposed by Rimon and Koditschek in [18] provide an easy-to-implement and provably correct point-to-point navigation algorithm, which has been successfully applied in both single [25] and multi-agent [9] navigation under different geometric constraints. Formal high-level languages such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) allow us to describe more complex planning objectives [4], [10], [19], [33]. Attempts to combine the strengths of both frameworks have recently appeared in [11], [24]. In particular, the robot's motion is abstracted by its dynamical transitions within a partition of the workspace, which consists of sphere regions around the points of interest. The task specifications are stated as LTL formulas over propositions satisfied at these regions. Then a high-level discrete plan, namely a sequence of regions to visit, is synthesized by off-the-shelf model-checking algorithms given the finite transition system and the LTL specification. This approach has been modified to take into account other navigating techniques like probabilistic roadmap method [8], [14] and rapidly-exploring random trees [1], [2] under certain complex motion objectives.

However, to solve planning problems of practical interest it is often necessary to perform various actions at different regions to achieve a goal. In other words, the purpose of "going somewhere" is to "do something". For example, a service robot can be assigned to bring a cup of coffee to the guest, which implicitly requires it to: move to the kitchen shelf, pick up a cup, move to the coffee machine, operate the coffee machine, take the coffee and bring it to the guest room. This plan is clearly a combination of transitions among different places of interest and performing various sequential actions. It would be inadequate to carry out the motion planning and action planning independently since the motion plan and action plan are closely related, i.e., "where to go" is motivated by "what to do there" and "what to do now" depends on "where it has been". Another observation is that some actions can only be performed when certain conditions are fulfilled and as a result certain state variables might be changed. Action description languages [28] like STRIPS [12], ADL [27] and PDDL [26] provide an intuitive and powerful way for describing the preconditions and effects of different operations/actions. However there has not been much work about how to incorporate the action description formalism with motion planning in general, and NF-based navigating techniques in particular.

Some relevant work integrates motion and action planning. In [33], since the underlying behaviors can only be performed at fixed regions, the specification is reinterpreted in terms of regions to visit. In [20], since the behaviors can be turned on and off at any time, independent atomic propositions are created for each behavior. The above approaches will not be applicable if some behaviors can only be performed when the workspace and the robot itself satisfy certain conditions, or choices have to be made among all the behaviors.

We propose to separate the domain-specific knowledge [22] such as the workspace model and the robot's mobility in the workspace, from the domain-independent knowledge such as the action map based on the actions the robot is capable of (given its on-board hardware and preprogrammed functionalities). One advantage is the increased modularity that our framework is adaptable whenever the workspace is modified or the task specification is changed. Another benefit is the considerably reduced size of the planning problem (compared with the exponential state space [6] of classic planning problems using STRIPS), because our framework avoids unfolding those state variables associated with the workspace properties and robot's positions.

The main contribution of this work is a generic framework to derive the complete description of the robot's functionalities within a certain workspace, such that any LTL task specification in terms of desired motions and actions can be treated. Furthermore, an optimal sequence of robot's motion and action that satisfies the given task specification and a low-level, real-time controller that implements this sequence are designed in a fully automated manner.

The authors are with the ACCESS Linnaeus Center, School of Electrical Engineering, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden. `mengg, kallej, dimos@kth.se`. This work was supported by the Swedish Research Council (VR) and EU STREP RECONFIG: FP7-ICT-2011-9-600825. The first and third authors are also affiliated with the KTH Centre for Autonomous Systems. The authors would like to thank Prof. John S. Baras for helpful suggestions.

The rest of the paper is organized as follows: Section II provides the essential preliminaries. In Sections III and IV, we discuss about the discretized abstraction of robot's mobility and actions. The way to synthesize the discrete plan and the low-level hybrid controller are provided in Section VII-C. Numerical simulations are presented in Section VII and concluding remarks are summarized in Section VIII.

II. Preliminaries

A. Navigation Function

The navigation function proposed by Rimon and Koditschek in [18] is given by

$$\Phi(q) = \frac{\gamma}{(\gamma^k + \beta)^{\frac{1}{k}}}\,, \tag{1}$$

where $k > 0$ is a design parameter, $q \in \mathbb{R}^n$ and $\Phi \in [0,\,1]$. In more detail, $\gamma = \|q - q_d\|^2$ represents an attractive potential from the goal where $q_d \in \mathbb{R}^n$ is the desired goal position, $\beta = \prod_{j=0}^{M} \beta_j$ is a repulsive potential from the sphere obstacles, where $\beta_0 \triangleq r_0^2 - \|q - q_0\|^2$ and $\beta_j \triangleq \|q - q_j\|^2 - r_j^2$. q_0, r_0 are the center and radius of the allowed workspace $\mathcal{W}_0 = \{q \in \mathbb{R}^n \,|\, \beta_0 > 0\}$; q_j, r_j are the center and radius of the sphere obstacles $\mathcal{O}_j = \{q \in \mathbb{R}^n \,|\, \beta_j < 0\}$, $j = 1, \cdots, M$. For brevity, we denote $\mathcal{W}_{\text{obs}} = \cup_{j=1}^{M} \mathcal{O}_j$ as the set of obstacle-occupied areas. It is assumed that $\mathcal{W}_0$ and $\mathcal{W}_{\text{obs}}$ satisfy the definition of a valid workspace in [18]. In particular, $\mathcal{O}_j \subset \mathcal{W}_0$ and $\mathcal{O}_i \cap \mathcal{O}_j = \emptyset$, $\forall i, j = 1, \cdots, M$. Moreover, it is required that $q_d \notin \mathcal{W}_{\text{obs}}$.

Besides its provable mathematical correctness, another strength of (1) is that it provides a straightforward motion planning algorithm. By following the negated gradient $-\nabla_q \Phi$, it is guaranteed that $\gamma \to 0$ when $t \to \infty$ and $\beta > 0$ holds for all $t \geq 0$, for sufficiently large k. That is to say, a collision free path is guaranteed from almost any initial position in the free space (except a set of measure zero) to any goal position in the free space given that the workspace is valid. Given a navigation function Φ, the way to compute its gradient is introduced in [18].

B. LTL and Büchi Automaton

Atomic propositions are Boolean variables that can be either true or false. In our case, they can be defined as any properties of the system the user is interested in, like "the robot is in region 1", "this region is occupied by obstacles" and "the robot has product A". We focus on the task specification φ given as an Linear Temporal Logic (LTL) formula. The basic ingredients of an LTL formula are a set of atomic propositions (APs) and several boolean and temporal operators. LTL formulas are formed according to the following grammar [3]: $\varphi ::= \mathsf{True} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \cup \varphi_2$, where $a \in AP$ and $\bigcirc$ (*next*), $\cup$ (*until*). For brevity, we omit the derivations of other useful operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*) and refer the readers to Chapter 5 of [3].

Given an LTL formula φ over a set of atomic propositions AP, there is a union of infinite words that satisfy φ: $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \,|\, \sigma \models \varphi\}$, where $\models\; \subseteq (2^{AP})^\omega \times \varphi$ is the satisfaction relation. There exists a Nondeterministic Büchi automaton (NBA) $\mathcal{A}_\varphi$ over 2^{AP} corresponding to φ, which is defined as:

$$\mathcal{A}_\varphi = (Q,\, 2^{AP},\, \delta,\, Q_0,\, \mathcal{F}), \tag{2}$$

where Q is a finite set of states; Q_0 is the initial state, 2^{AP} is an alphabets; $\delta \subseteq Q \times 2^{AP} \times Q$ is a transition relation and $\mathcal{F} \subseteq Q$ is a set of accepting states. An infinite run r of a NBA is an infinite sequence of states and is called accepting if $\texttt{Inf}(r) \cap \mathcal{F} \neq \emptyset$ where $\texttt{Inf}(r)$ is the set of states that appear in r infinitely often. Denote by $\mathcal{L}_\omega(\mathcal{A}_\varphi)$ the accepted language of $\mathcal{A}_\varphi$, which is the set of infinite words that have an accepting run in $\mathcal{A}_\varphi$, i.e., $\texttt{Words}(\varphi) = \mathcal{L}_w(\mathcal{A}_\varphi)$. There are fast translation algorithms [29] from an LTL formula to NBA. This translation process can be done in time and space $2^{\mathcal{O}(|\varphi|)}$ [3]. The state graph of a NBA has a vertex for each state and an edge between vertices q and q' whenever there exists $l \in 2^{AP}$ such that $(q,\, l,\, q') \in \delta$.

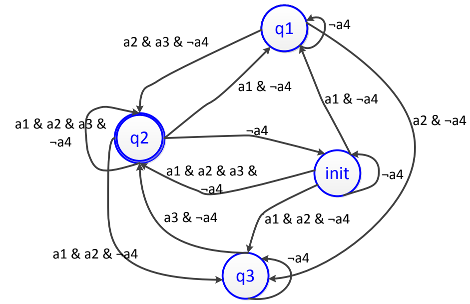

Fig. 1. The Büchi automaton corresponding to $\varphi = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\Diamond a_3) \wedge (\Box\neg a_4)$. In the case study of Section VII-B, $a_1 = \Psi_{r,2} \wedge \Psi_{b,2}$, $a_2 = \Psi_{r,4} \wedge \Psi_{b,4}$ $a_3 = \Psi_{r,3} \wedge \Psi_{b,5}$, $a_4 = \Psi_{p,3}$.

To give an example, the NBA corresponding to $\varphi = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\Diamond a_3) \wedge (\Box\neg a_4)$ is illustrated in Figure 1 by [29]. The transition from state q_1 to q_3 is given by $(q_1, l, q_3) \in \delta$ where the input alphabet $l = (a_2 \;\&\; \neg a_4)$, which is a short notation for four input alphabets $\{a_2\}$, $\{a_2,\, a_1\}$, $\{a_2,\, a_3\}$, $\{a_2,\, a_1,\, a_3\}$.

III. Abstraction of Robot Mobility

A. The Workspace Model

The workspace we consider is bounded by a large sphere $\pi_0 = \{q \in \mathbb{R}^n \,|\, \|q - q_0\| \leq r_0\}$, within which there exists N smaller spheres around the points of interest:

$$\pi_i = \mathcal{B}_{r_i}(q_i) = \{q \in \mathbb{R}^n \,|\, \|q - q_i\| \leq r_i\}, \tag{3}$$

where $q_i \in \mathbb{R}^n$ and $r_i > 0$ are the center and radius of the n-dimensional sphere areas $\mathcal{B}_{r_i}(q_i)$; r_i represent the margins we want to attain with respect to the points of interest, rather than necessarily some physical quantities, $\forall i = 1, \cdots, N$. Denote by $\Pi = \{\pi_1, \cdots, \pi_N\}$ the set of smaller sphere areas. It is assumed that π_0 and Π satisfy the conditions of a valid workspace introduced in Section II-A. Compared with other cell decomposition schemes like triangles [5], polygons [4] and hexagons [30], this sphere-area-based approach typically reduces the size of the resulting abstraction since they represent regions of interest, rather than a complete partition of the workspace.

Moreover, in order to indicate the robot's position, we define the set of atomic propositions $\Psi_r = \{\Psi_{r,i}\}$, $i = 1, 2 \cdots, N$, where

$$\Psi_{r,i} = \begin{cases} \texttt{True} & \text{if } q \in \pi_i \\ \texttt{False} & \text{otherwise.} \end{cases}$$

$\Psi_{r,i}$ can be evaluated by measurements from a real-time position system.

Beside the location of these regions, we also would like to know the properties satisfied by each region. Denote by $\Psi_p = \{\Psi_{p,1}, \cdots, \Psi_{p,I}\}$ the finite set of atomic propositions indicating those properties. Some elements of Ψ_p may reflect the designer's concern, like "this room is off limits", while the others are relevant to robot's actions discussed in Section IV. For instance, "this room has product A" is a property relevant to the action "pickup A".

B. Robot Dynamics

The robot is assumed to be holonomic, satisfying the single-integrator dynamics:

$$\dot{q} = u\,, \tag{4}$$

where $q,\, u \in \mathbb{R}^n$ are the position and control signal. As introduced in Section II-A, navigation functions provide an useful tool to navigate the robot within the sphere workspace introduced in the previous section.

Since there are no explicit representations of "obstacles" within the workspace π_0 and no fixed initial or goal regions, denote by $q_g \in \pi_g$ the center of the goal region $\pi_g \in \Pi$ and $\Pi_{\text{avoid}} \subset \Pi$ is the set of sphere regions to avoid. Moreover, $q_s \in \pi_s$ is the starting point within π_s and $\pi_s \in \Pi$. It is important to point out that the workspace remains valid no matter how the sphere regions in Π are classified since Π is assumed to be valid within π_0. It further implies that no matter which two regions in Π are chosen as initial and goal regions, there always exist a feasible path that leads the robot from the starting point in π_s to one point in π_g, without crossing any of the sphere areas in Π_{avoid} and always staying in π_0. The feasible path can be generated by following the negated gradient as introduced in Section II-A:

$$u \;=\; -\nabla_q \, \frac{\gamma_g}{(\gamma_g^k + \beta_{gs})^{\frac{1}{k}}} \tag{5}$$

where $\gamma_g = \|q - q_g\|^2$ and $\beta_{gs} = \prod_{j=0, j\neq s,g}^{M} \beta_j$. The way to construct β_j and compute the gradient $\nabla_q \Phi$ is introduced in Section II-A. Note that the asymptotic stability of the above controller guarantees the convergence to the neighborhood of q_g, namely π_g in finite time [25].

Remark 1: [23] also provides the NF-based control strategy for non-holonomic vehicles. The rest of the framework still applies by replacing (5) with the one proposed in [23].

TABLE I
ACTIONS DESCRIPTION FOR SECTION VII

Action	**Condition**	**Effect**
$act_{\mathcal{B},0}$	True	$\Psi_{b,0} = \mathsf{T}$, $\Psi_{b,\sim 0} = \mathsf{F}$
$act_{\mathcal{B},1}$	$\Psi_{p,1}\,\&\,\neg\Psi_{s,1}\,\&\,\neg\Psi_{s,2}$	$\Psi_{s,1} = \mathsf{T}$, $\Psi_{b,1} = \mathsf{T}$, $\Psi_{b,\sim 1} = \mathsf{F}$
$act_{\mathcal{B},2}$	$\Psi_{s,1}$	$\Psi_{s,1} = \mathsf{F}$, $\Psi_{b,2} = \mathsf{T}$, $\Psi_{b,\sim 2} = \mathsf{F}$
$act_{\mathcal{B},3}$	$\Psi_{p,2}\,\&\,\neg\Psi_{s,2}\,\&\,\neg\Psi_{s,1}$	$\Psi_{s,2} = \mathsf{T}$, $\Psi_{b,3} = \mathsf{T}$, $\Psi_{b,\sim 3} = \mathsf{F}$
$act_{\mathcal{B},4}$	$\Psi_{s,2}$	$\Psi_{s,2} = \mathsf{F}$, $\Psi_{b,4} = \mathsf{T}$, $\Psi_{b,\sim 4} = \mathsf{F}$
$act_{\mathcal{B},5}$	True	$\Psi_{b,5} = \mathsf{T}$, $\Psi_{b,\sim 5} = \mathsf{F}$

C. Weighted FTS

Given the workspace model and the robot dynamics, the abstraction of the robot's mobility is built as a weighted finite transition system (wFTS):

$$\mathcal{M} = (\Pi_{\mathcal{M}},\, act_{\mathcal{M}},\, \longrightarrow_{\mathcal{M}},\, \Pi_{\mathcal{M},0},\, \Psi_{\mathcal{M}},\, L_{\mathcal{M}},\, W_{\mathcal{M}}), \tag{6}$$

where (i) $\Pi_{\mathcal{M}} = \{\pi_1, \cdots, \pi_N\}$ is the finite set of states; (ii) $act_{\mathcal{M}}$ represents the controller (5); (iii) $\longrightarrow_{\mathcal{M}} \subseteq \Pi_{\mathcal{M}} \times act_{\mathcal{M}} \times \Pi_{\mathcal{M}}$ is the transition relation; (iv) $\Pi_{\mathcal{M},0} \subseteq \Pi_{\mathcal{M}}$ is the initial region the robot starts from; (v) $\Psi_{\mathcal{M}} = \Psi_r \cup \Psi_p$ is the set of atomic propositions; (vi) $L_{\mathcal{M}} : \Pi_{\mathcal{M}} \to 2^{\Psi_{\mathcal{M}}}$ is the labeling function standing for the properties that are satisfied at each region and $\Psi_{r,i} \in L_{\mathcal{M}}(\pi_i)$, $i = 1, \cdots, N$; (vii) $W_{\mathcal{M}} : \longrightarrow_{\mathcal{M}} \to \mathbb{R}^+$ represents the implementation (energy/time) cost [13], which is approximated by the straight-line distance between regions $\|q_i - q_j\| - r_i - r_j$, $\forall \pi_i,\, \pi_j \in \Pi$, $i \neq j$.

Remark 2: The workspace boundary π_0 is not included in $\Pi_{\mathcal{M}}$ as it is guaranteed by the controller (5) that the robot always stays within the workspace. Moreover, $\Psi_{r,i}$ is always true at state π_i.

Up to this point, given the wFTS $\mathcal{M}$ and an LTL task specification over $\Psi_{\mathcal{M}}$, various existing frameworks [4], [15], [16], [33] could be utilized to synthesize a discrete motion plan in terms of sequences of regions to visit, as also our approach discussed in Section VI-A. However when the task specification is stated as requirements on desired actions within different regions, the mobility abstraction $\mathcal{M}$ alone is not enough and a model of robot's actions is also needed.

IV. MODEL OF ROBOT ACTIONS

Classic planning formalisms, like STRIPS [12], ADL [27] and PDDL [26], provide an intuitive way to describe high-level actions the robot is capable of. Given a set of states and action names, each action is described by specifying its precondition and effect on the states. Here we utilize the same approach. Assume that the robot is capable of performing K different actions $\{act_{\mathcal{B},1}, \cdots, act_{\mathcal{B},K}\}$, implementable by the corresponding low-level controllers $\{\mathcal{K}_k\}$, $k = 1, 2 \cdots, K$. For brevity, denote by $Act_{\mathcal{B}} =$

$\{act_{\mathcal{B},0},\, act_{\mathcal{B},1}, \cdots,\, act_{\mathcal{B},K}\}$, where $act_{\mathcal{B},0} \triangleq$ *None* indicates that none of these K actions is performed. Moreover, we introduce another two sets of atomic propositions:

- $\Psi_s = \{\Psi_{s,j}\}$, represents the internal states of the robot, $j = 1, 2\cdots, J$, e.g., "the robot has product A".
- $\Psi_b = \{\Psi_{b,k}\}$ where $\Psi_{b,k} = \mathsf{True}$ if and only if action k is performed, $k = 0, 1, \cdots, K$. We assume that any two actions cannot be concurrent, i.e., at most one element of Ψ_b can be true.

The subscripts of Ψ_s and Ψ_b stand for the "state" and "behavior" of the robot. With Ψ_p, Ψ_s and Ψ_b, we can describe each action in $Act_{\mathcal{B}}$ by the precondition and effect functions.

A. Precondition and Effect

The precondition function

$$\texttt{Cond}: Act_{\mathcal{B}} \times 2^{\Psi_p} \times 2^{\Psi_s} \longrightarrow \mathsf{True/False}, \tag{7}$$

takes one action in $Act_{\mathcal{B}}$, subsets of Ψ_p and Ψ_s as inputs and returns a boolean value. Namely in order to perform that action, the conditions on the properties of the workspace Ψ_p and the robot's internal states Ψ_s have to be fulfilled. For instance, the action "pickup A" can only be performed when "the room has product A". While some actions like "take pictures" might be performed without such constraints and then the condition is simply a tautology, e.g., $\texttt{Cond} = \mathsf{True}$. Note the condition function for $act_{\mathcal{B},0}$ is defined as True.

The effect function

$$\texttt{Eff}: Act_{\mathcal{B}} \times (2^{\Psi_s} \times \Psi_b) \longrightarrow (2^{\Psi_s} \times \Psi_b), \tag{8}$$

represents the effect of the actions. As a result of performing action $act_{\mathcal{B},k}$, the robot's internal states Ψ_s might be changed and Ψ_b is changed to indicate which action is performed. More specifically,

- $\texttt{Eff}(act_{\mathcal{B},0}, w_s, \Psi_{b,k}) = (w_s,\, \Psi_{b,0})$, where $w_s \subseteq 2^{\Psi_s}$ and $\forall \Psi_{b,k} \in \Psi_b$. Performing $act_{\mathcal{B},0}$ does not change the robot's internal state and all elements in Ψ_b except $\Psi_{b,0}$ are set to false;
- $\texttt{Eff}(act_{\mathcal{B},k},\, w_s,\, \Psi_{b,l}) = (w'_s,\, \Psi_{b,k})$, where $w_s,\, w'_s \subseteq 2^{\Psi_s}$ and $\Psi_{b,l},\, \Psi_{b,k} \in \Psi_b$, is the effect function of $act_{\mathcal{B},k}$ for $k \neq 0$.

For example, once the action "pickup A" is performed, the propositions "the robot has A" and "'pickup A' is performed" become true. Note that the effect functions can not modify the properties of the workspace.

B. Action Map

Given Ψ_p, Ψ_s, Ψ_b and $Act_{\mathcal{B}}$, $\texttt{Cond}$, $\texttt{Eff}$, the *action map* is defined as a tuple

$$\mathcal{B} = (\Pi_{\mathcal{B}},\, Act_{\mathcal{B}},\, \Psi_p,\, \hookrightarrow_{\mathcal{B}},\, \Pi_{\mathcal{B},0},\, \Psi_{\mathcal{B}},\, L_{\mathcal{B}},\, W_{\mathcal{B}}), \tag{9}$$

where (i) $\Pi_{\mathcal{B}} \subseteq 2^{\Psi_s} \times \Psi_b$ is set of all assignments of Ψ_s and Ψ_b; (ii) Ψ_p serves as the input propositions, and 2^{Ψ_p} is the finite set of possible input assignments; (iii) the conditional transition relation $\hookrightarrow_{\mathcal{B}}$ is defined by $\pi_{\mathcal{B}} \times \alpha_{\mathcal{B}} \times 2^{\Psi_p} \times \pi'_{\mathcal{B}} \subseteq \hookrightarrow_{\mathcal{B}}$ if the following conditions hold:

(1) $\alpha_{\mathcal{B}} \in Act_{\mathcal{B}}$, $\pi_{\mathcal{B}}, \pi'_{\mathcal{B}} \in \Pi_{\mathcal{B}}$;
(2) $\texttt{Cond}(\alpha_{\mathcal{B}},\, 2^{\Psi_p},\, \pi_{\mathcal{B}}) = \mathsf{True}$;
(3) $\pi'_{\mathcal{B}} \in \texttt{Eff}\,(\alpha_{\mathcal{B}}, \pi_{\mathcal{B}})$.

(iv) $\Pi_{\mathcal{B},0} \subseteq 2^{\Psi_s} \times \Psi_{b,0}$ is the initial state; (v) $\Psi_{\mathcal{B}} = \Psi_s \cup \Psi_b$ is the set of atomic propositions; (vi) $L_{\mathcal{B}}(\pi_{\mathcal{B}}) = \{\pi_{\mathcal{B}}\}$, i.e., the labeling function is the state itself; (vii) $W_{\mathcal{B}} : \hookrightarrow_{\mathcal{B}} \to \mathbb{R}^+$ is the weight associated with each transition and $W_{\mathcal{B}}(\pi_{\mathcal{B}},\, \alpha_{\mathcal{B}},\, 2^{\Psi_p},\, \pi'_{\mathcal{B}})$ is estimated by the cost of action $\alpha_{\mathcal{B}}$.

Remark 3: The set of states $\Pi_{\mathcal{B}}$ is defined as $2^{\Psi_s} \times \Psi_b$ instead of $2^{\Psi_s} \times 2^{\Psi_b}$ because only one element in Ψ_b can be true. This reduces the size of the action map significantly.

Ψ_p can be viewed as external inputs [3] to the action map, i.e., within different regions the transition relations might be different due to their different properties. Moreover, $\mathcal{B}$ is nondeterministic in the sense that at each state $\pi_{\mathcal{B}}$ any action whose associated condition function is evaluated to be true, can be performed.

It is worth mentioning that the action map is constructed independently of the structure of the workspace where the robot will be deployed. Furthermore, given an instance of the workspace property Ψ_p, the action map $\mathcal{B}$ is equivalent to a wFTS as all conditional transition relations can be verified or falsified based on the definition of $\hookrightarrow_{\mathcal{B}}$.

V. Model of Complete Functionalities

As mentioned earlier, the abstraction of robot's mobility $\mathcal{M}$ from (6) and the robot's action map $\mathcal{B}$ from (9) are adequate for the controller synthesis within certain problem domain. However, in order to consider richer and more complex tasks involving both regions to visit and actions to perform within these regions, we need a complete model of robot's functionalities that combines these two parts. We propose the following way to compose $\mathcal{M}$ and $\mathcal{B}$:

$$\mathcal{R} = (\Pi_{\mathcal{R}},\, Act_{\mathcal{R}},\, \longrightarrow_{\mathcal{R}},\, \Pi_{\mathcal{R},0},\, \Psi_{\mathcal{R}},\, L_{\mathcal{R}},\, W_{\mathcal{R}}), \tag{10}$$

where (i) $\Pi_{\mathcal{R}} = \Pi_{\mathcal{M}} \times \Pi_{\mathcal{B}}$ is the set of states; (ii) $Act_{\mathcal{R}} = act_{\mathcal{M}} \cup Act_{\mathcal{B}}$ is the set of actions; (iii) $\longrightarrow_{\mathcal{R}} \subseteq \Pi_{\mathcal{R}} \times Act_{\mathcal{R}} \times \Pi_{\mathcal{R}}$ is the transition relation, defined by the following rules:

(1) $\langle \pi_{\mathcal{M}},\, \pi_{\mathcal{B}} \rangle \xrightarrow{act_{\mathcal{M}}}_{\mathcal{R}} \langle \pi'_{\mathcal{M}},\, \pi'_{\mathcal{B}} \rangle$ if $\pi_{\mathcal{M}} \xrightarrow{act_{\mathcal{M}}}_{\mathcal{M}} \pi'_{\mathcal{M}}$ and $\pi_{\mathcal{B}} \xrightarrow{act_{\mathcal{B},0}}_{\mathcal{B}} \pi'_{\mathcal{B}}$;

(2) $\langle \pi_{\mathcal{M}},\, \pi_{\mathcal{B}} \rangle \xrightarrow{\alpha_{\mathcal{B}}}_{\mathcal{R}} \langle \pi_{\mathcal{M}},\, \pi'_{\mathcal{B}} \rangle$ if $\pi_{\mathcal{B}} \times \alpha_{\mathcal{B}} \times L_{\mathcal{M}}(\pi_{\mathcal{M}}) \times \pi'_{\mathcal{B}} \subset \hookrightarrow_{\mathcal{B}}$, where $\alpha_{\mathcal{B}} \in Act_{\mathcal{B}}$;

(iv) $\Pi_{\mathcal{R},0} = \Pi_{\mathcal{M},0} \times \Pi_{\mathcal{B},0}$ contains the robot's initial region and initial internal state; (v) $\Psi_{\mathcal{R}} = \Psi_{\mathcal{M}} \cup \Psi_{\mathcal{B}}$ is the complete set of atomic propositions including Ψ_r, Ψ_p, Ψ_s and Ψ_b; (vi) $L_{\mathcal{R}} : \Pi_{\mathcal{R}} \to 2^{\Psi_{\mathcal{R}}}$ is the labeling function, $L_{\mathcal{R}}(\langle \pi_{\mathcal{M}},\, \pi_{\mathcal{B}} \rangle) = L_{\mathcal{M}}(\pi_{\mathcal{M}}) \cup L_{\mathcal{B}}(\pi_{\mathcal{B}})$; (vii) $W_{\mathcal{R}} : \longrightarrow_{\mathcal{R}} \to \mathbb{R}^+$, is the weight function on each transition, defined as:

(1) $W_{\mathcal{R}}(\langle \pi_{\mathcal{M}},\, \pi_{\mathcal{B}} \rangle,\, act_{\mathcal{M}},\, \langle \pi'_{\mathcal{M}},\, \pi'_{\mathcal{B}} \rangle) = W_{\mathcal{M}}(\pi_{\mathcal{M}},\, act_{\mathcal{M}},\, \pi'_{\mathcal{M}})$;

(2) $W_{\mathcal{R}}(\langle \pi_{\mathcal{M}},\, \pi_{\mathcal{B}} \rangle,\, \alpha_{\mathcal{R}},\, \langle \pi_{\mathcal{M}},\, \pi'_{\mathcal{B}} \rangle) = W_{\mathcal{B}}(\pi_{\mathcal{B}},\, \alpha_{\mathcal{R}},\, \pi'_{\mathcal{B}})$, if $\alpha_{\mathcal{R}} \in Act_{\mathcal{B}}$.

Remark 4: In the definition of $\hookrightarrow_{\mathcal{B}}$, $act_{\mathcal{B},0}$ is released automatically whenever the controller (5) is activated, because whenever the robot moves to a new region, this automatically indicates that no actions within $act_{\mathcal{B},k}$ are performed as we assume non-concurrent actions. The labeling

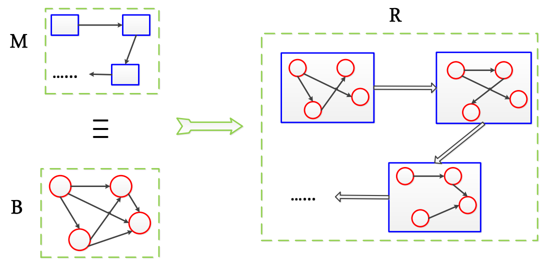

Fig. 2. The action map $\mathcal{B}$ is composed with each region of $\mathcal{M}$, giving a complete description $\mathcal{R}$ of robot's functionalities.

function $L_{\mathcal{M}}(\Pi_{\mathcal{M}})$ serves as inputs to $\mathcal{B}$ and the conditional transitions in $\mathcal{B}$ are verified or falsified.

Figure 2 illustrates the idea behind the process of parallel composition defined above. Blue squares represent the states of $\mathcal{M}$ and red cycles encode the states of $\mathcal{B}$. Loosely speaking, when composing them into $\mathcal{R}$, N copies of $\mathcal{B}$ are first made, corresponding to the N regions within the workspace. At the same time, the conditional transition relations in these copies are verified or falsified by verifying the conditions on the properties of each region. This is why the "verified" action map within different regions might have different structures as shown in Figure 2.

It is possible to construct the complete model $\mathcal{R}$ directly. But we argue that there are several advantages in constructing $\mathcal{M}$ and $\mathcal{B}$ first and then composing them into $\mathcal{R}$. Normally an abstraction for the robot's mobility is only valid for a specific workspace and it has to be modified whenever the structure of the workspace is changed. In contrast, the abstraction for robot's actions is relatively fixed, considering commercial robots with pre-programmed functionalities. By building $\mathcal{M}$ and $\mathcal{B}$ separately, each of them independently can serve as the input model to the controller synthesis machinery in Section VI-A. Moreover, $\mathcal{R}$ is more difficult to build manually than $\mathcal{M}$ and $\mathcal{B}$ separately and then taking their composition automatically. Another advantage is that the number of states in $\mathcal{R}$ is greatly reduced (compared with the exponential complexity $2^{|\Psi_{\mathcal{R}}|}$ [6] of classic planning problems using STRIPS). In particular, we avoid unfolding those propositional variables associated with the workspace properties as they are invariant and use the fact that only one element in Ψ_r or Ψ_b can be true.

The composed system $\mathcal{R}$ is a wFTS over the set of atomic propositions $\Psi_{\mathcal{R}}$. Recall that $\Psi_{\mathcal{R}} = \Psi_r \cup \Psi_p \cup \Psi_s \cup \Psi_b$. Among them, Ψ_r, Ψ_p are commonly seen in related work [1], [4], [5], [16] and [33], but Ψ_s, Ψ_b allow us to express richer requirements on the robot's internal states and actions directly, as for example where these actions are desired and the preferred sequence.

As introduced in Section II-B, LTL formulas φ can be used to specify various robot motion and action tasks, such as safety ($\square\neg\varphi_1$, globally avoiding φ_1), ordering ($\varphi_1 \cup (\varphi_2 \cup \varphi_3)$, φ_1, φ_2, φ_3 hold in sequence), response ($\varphi_1 \Rightarrow \varphi_2$, if φ_1 holds, φ_2 will hold in future). For instance, the task "eventually always drop A at region 1" can be expressed as

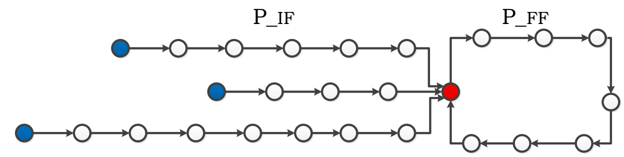

Fig. 3. For every accepting state $p_f \in \mathcal{F}_{\mathcal{P}}$ (in red), P_{IF} contains the paths from every initial state $p_0 \in Q_{\mathcal{P},0}$ (in blue) to p_f with the minimal costs; P_{FF} contains the path from p_f back to itself with the minimal cost.

$\varphi = \square\Diamond(\Psi_{r,1} \wedge \Psi_{b,2})$, where $\Psi_{r,1}$={the robot is in region 1} and $\Psi_{b,2}$={'drop A' is performed}. Notice that we do not even need to specify where to "pickup A" as it is modeled in the action map that to "drop A" the robot has to "pickup A" first at some regions that have A. We now state the problem we consider in this paper:

Problem 1: Given the weighted finite transition system $\mathcal{R}$ and an LTL formula φ over $\Psi_{\mathcal{R}}$, construct a discrete motion and action plan such that φ is satisfied and also the hybrid controller that implements this discrete plan.

VI. Motion and Action Planner

In this section, we provide the solution to Problem 1, which includes two major steps. First an optimal motion-and-action plan is derived by the searching for the optimal accepting run in the product automaton [3], [7], [31]. Then the hybrid controller that implements this discrete plan is synthesized in an automated manner.

A. Model Checking with Optimality

An infinite path fragment $\tau_{\mathcal{R}}$ of $\mathcal{R}$ is an infinite sequence of states

$$\pi_{\mathcal{R},0}\pi_{\mathcal{R},1}\pi_{\mathcal{R},2}\pi_{\mathcal{R},3}\cdots,$$

where $\pi_{\mathcal{R},0} \in \Pi_{\mathcal{R},0}$ and $(\pi_{\mathcal{R},i},\, \pi_{\mathcal{R},i+1}) \in \to_{\mathcal{R}}$, $\forall i > 0$. Its trace $\mathtt{trace}(\tau_{\mathcal{R}})$ defines a *word* of $\mathcal{R}$, given by the sequence of atomic propositions that are true in the states along this path. Namely, $\mathtt{trace}(\tau_{\mathcal{R}}) = L_{\mathcal{R}}(\pi_{\mathcal{R},0})\, L_{\mathcal{R}}(\pi_{\mathcal{R},1})\cdots$. Given an LTL formula φ, we want to find an infinite path $\tau_{\mathcal{R}}$ such that $\tau_{\mathcal{R}} \models \varphi$, i.e., $\mathtt{trace}(\tau_{\mathcal{R}}) \subseteq \mathtt{Words}(\varphi)$ [7]. The NBA $\mathcal{A}_{\varphi} = (Q,\, 2^{AP},\, \delta,\, Q_0,\, \mathcal{F})$ associated with φ from Section II-B allows us to check whether $\tau_{\mathcal{R}}$ satisfies φ by checking if $\mathtt{trace}(\tau_{\mathcal{R}})$ is accepted by $\mathcal{A}_{\varphi}$. In this paper, we use the automaton-based model-checking approach by checking the emptiness of the product Büchi automaton $\mathcal{R}\otimes\mathcal{A}_{\varphi}$, see [7] and Algorithm 11 in [3]. The product Büchi automaton is defined as a tuple

$$\mathcal{A}_{\mathcal{P}} = \mathcal{R} \otimes \mathcal{A}_{\varphi} = (Q_{\mathcal{P}},\, \delta_{\mathcal{P}},\, Q_{\mathcal{P},0},\, \mathcal{F}_{\mathcal{P}},\, W_{\mathcal{P}}), \qquad (11)$$

which consists of (i) $Q_{\mathcal{P}} = \Pi_{\mathcal{R}} \times Q$; (ii) the transition relation $(\langle\pi_{\mathcal{R}},\, q\rangle, \langle\pi'_{\mathcal{R}},\, q'\rangle) \in \delta_{\mathcal{P}}$ iff $(\pi_{\mathcal{R}},\, \pi'_{\mathcal{R}}) \in \longrightarrow_{\mathcal{R}}$ and $(q,\, L_{\mathcal{R}}(\pi_{\mathcal{R}}),\, q') \in \delta$; (iii) the set of initial states $Q_{\mathcal{P},0} = \Pi_{\mathcal{R},0} \times Q_0$; (iv) the set of accepting states $\mathcal{F}_{\mathcal{P}} = \Pi_{\mathcal{R}} \times \mathcal{F}$; (v) the weight function $W_{\mathcal{P}} : \delta_{\mathcal{P}} \to \mathbb{R}^+$, $W_{\mathcal{P}}(\langle\pi_{\mathcal{R}},\, q\rangle,\, \langle\pi'_{\mathcal{R}},\, q'\rangle) = W_{\mathcal{R}}(\pi_{\mathcal{R}},\, \pi'_{\mathcal{R}})$.

It is proven in [3] that there exists an infinite path of $\mathcal{R}$ satisfying φ if and only if $\mathcal{A}_{\mathcal{P}}$ has at least one accepting run. Then this accepting run could be projected to an infinite path

Algorithm 1: Function `optRun`(G, I, F)

Input: a weighted graph G, I, F.
Output: the optimal accepting run $r_{\mathcal{P},opt}$.
1. Compute the path with minimal cost from every initial vertex in I to every accepting vertex in F.

$$(D_{IF}, P_{IF}) = \texttt{MinPath}(\,G,\, I,\, F).$$

2. Compute the path with minimal cost from every accepting vertex in F and back to itself:

$$(D_{FF}, P_{FF}) = \texttt{MinCycl}(\,G,\, F).$$

3. For each column of D_{IF}, find the element with the minimal value and the corresponding cell in P_{IF} (with the same index). Save them sequentially in $1 \times M$ matrix D_{iF} and $1 \times M$ cell P_{iF}.
4. Find the element with the minimal value in $D_{iF} + \gamma\, D_{FF}$ and its index $f_{\min}$.
5. Optimal accepting run $r_{\mathcal{P},opt}$, prefix: the $f_{\min}$-th element of P_{iF}; suffix: the $f_{\min}$-th element P_{FF}.

in $\mathcal{R}$, the trace of which should satisfy φ automatically. In this paper, we consider the accepting runs with the following *prefix-suffix* structure:

$$r_{\mathcal{P}} = p_0\, p_1 \cdots (\, p_k \cdots\cdots p_n\, p_k\,)^{\omega}\,, \tag{12}$$

where $p_0 \in Q_{\mathcal{P},0}$ and $p_k \in \mathcal{F}_{\mathcal{P}}$. Namely, $r_{\mathcal{P}}$ consists of two parts: the prefix part that is executed only once from an initial state p_0 to one accepting state p_k and the suffix part which is repeated infinitely from this accepting state back to itself [3], [33]. An accepting run with the prefix-suffix structure has a finite representation as (12), and more importantly it allows us to define the *prefix-suffix cost* of an accepting run:

$$\begin{aligned}\texttt{Cost}(r_{\mathcal{P}}) &= \left(\sum_{i=0}^{k-1} W_{\mathcal{P}}(p_i,\, p_{i+1})\right)\\ &+ \gamma\left(\sum_{i=k}^{n-1} W_{\mathcal{P}}(p_i,\, p_{i+1}) \,+\, W_{\mathcal{P}}(p_n,\, p_k)\right)\,,\end{aligned} \tag{13}$$

of which the first summation represents the sum over the weights of transitions along the sequence of the prefix and the second summation for the suffix. Note that $\gamma \geq 0$ represents the relative weighting on the cost of transient response (the prefix) and steady response (the suffix) to the task specification [33].

Remark 5: The prefix-suffix structure is more of a way to formulate the total cost of an accepting run, rather than a conservative assumption. If an accepting run exists, by its definition at least one accepting state should appear in it infinitely often. Among all the finite number of cycles starting for this accepting state and back to itself there is one with the minimal cost. Thus an accepting run of the form (12) can be built using this minimal cycle as the periodic suffix.

Problem 2: Find an accepting run of the product automaton $\mathcal{A}_{\mathcal{P}}$ satisfying the prefix-suffix structure (12) and minimizing the cost by (13).

We call the solution to Problem 2 the optimal accepting run $r_{\mathcal{P},opt}$. Algorithm 1 takes as input arguments the weighted state graph [3] $G(\mathcal{A}_{\mathcal{P}}) = (Q_{\mathcal{P}},\, \delta_{\mathcal{P}},\, W_{\mathcal{P}})$ where $Q_{\mathcal{P}}, \delta_{\mathcal{P}}, W_{\mathcal{P}}$ are the set of vertices, edges and costs/distances associated with the edges, the set of initial vertices $I = Q_{\mathcal{P},0}$ and the set of accepting vertices $F = \mathcal{F}_{\mathcal{P}}$. It utilizes Dijkstra's algorithm [21] for computing the shortest path between pairs of vertices within a graph. In particular, denote the number of elements in I and F by $|I| = L$ and $|F| = M$. Function `MinPath` takes $(G,\, I,\, F)$ as inputs and outputs a $L \times M$ matrix D_{IF}, with the $(i_{\text{th}},\, j_{\text{th}})$ element containing the value of the minimal cost from I_i to F_j; and a $L \times M$ cell P_{IF}, with the $(i_{\text{th}},\, j_{\text{th}})$ cell containing the sequence of vertices appearing in the path with minimal cost from I_i to F_j. Function `MinCycl` is a variant of function `MinPath`, which outputs a $1 \times M$ matrix D_{FF}, with the j_{th} element containing the value of the minimal cost from F_j back to F_j; and a $1 \times M$ cell P_{FF} with the j_{th} cell containing the sequence of vertices appearing in the path with minimal cost from F_j back to F_j (as in Figure 3). Note that if a vertex is not reachable from another vertex, then the cost is $+\infty$.

B. Hybrid Controller Synthesis

The optimal accepting run $r_{\mathcal{P},opt}$ obtained from Algorithm 1 can be projected to an infinite path $\tau_{\mathcal{R}}$ of $\mathcal{R}$ by projecting $r_{\mathcal{P},opt}$ onto $\mathcal{R}$. The trace of $\tau_{\mathcal{R}}$ then automatically satisfies φ. $\tau_{\mathcal{R}}$ also fulfills the prefix-suffix structure [32], which gives the finite representation:

$$\tau_{\mathcal{R},opt} = \pi_{\mathcal{R},0}\, \pi_{\mathcal{R},1} \cdots (\, \pi_{\mathcal{R},k} \cdots\cdots \pi_{\mathcal{R},n}\, \pi_{\mathcal{R},k}\,)^{\omega}. \tag{14}$$

For each pair of sequential states $(\pi_{\mathcal{R},i},\, \pi_{\mathcal{R},i+1})$ in $\tau_{\mathcal{R},opt}$ there exists an action $\alpha_{\mathcal{R}} \in Act_{\mathcal{R}}$ such that $(\pi_{\mathcal{R},i},\, \alpha_{\mathcal{R}},\, \pi_{\mathcal{R},i+1}) \in \longrightarrow_{\mathcal{R}}$ from (10). Thus the underlying low-level control strategy can be synthesized by sequentially implementing the continuous controller associated with the actions along $\tau_{\mathcal{R}}$. In particular, if $\alpha_{\mathcal{R}} = act_{\mathcal{B},k}$, the controller $\{\mathcal{K}_k\}$ that implements $act_{\mathcal{B},k}$ is activated. If $\alpha_{\mathcal{R}} = act_{\mathcal{M}}$, the NF-based controller (5) is applied to drive the robot from one point in the starting region to one point in the goal region. The above arguments are summarized in Algorithm 2.

C. Complexity and Overall Framework

The correctness of the proposed solutions in Section VI follows from the problem formulation and the correctness of the Dijkstra's shortest path algorithm. Let $|\mathcal{M}|$ and $|\mathcal{B}|$ denote the size of the robot's mobility model and action map. The size of $\mathcal{A}_{\mathcal{P}}$ by (11) is $|\mathcal{A}_{\mathcal{P}}| = |\mathcal{M}| \cdot |\mathcal{B}| \cdot |2^{|\varphi|}|$. Algorithm 1 runs in $\mathcal{O}(|\mathcal{A}_{\mathcal{P}}| \cdot \log |\mathcal{A}_{\mathcal{P}}| \cdot |Q_{\mathcal{P},0}| \cdot |\mathcal{F}_{\mathcal{P}}|)$.

To summarize, the overall framework is shown in Algorithm 3. It is worth mentioning that $\mathcal{M}$, $\mathcal{B}$ are constructed only once for the robot within a certain workspace and φ can express any task specification in terms of required motions and actions. Steps 3, 5, 6 and 7 are performed automatically [3], [5]. Whenever a new task specification

Algorithm 2: Hybrid Controller Synthesis

Input: the optimal plan $\tau_{\mathcal{R}}$, associated with $r_{\mathcal{P},opt}$.
Output: the hybrid control strategy.
1. Follow the sequence of states along $\tau_{\mathcal{R}}$ by (14), namely for each transition $(\pi_{\mathcal{R},i},\, \pi_{\mathcal{R},i+1}) \in \tau_{\mathcal{R}}$, $\forall i = 1, \cdots, n-1$, repeat the following steps 2-4.
2. Let $(\pi_{\mathcal{R},i},\, \alpha_{\mathcal{R}},\, \pi_{\mathcal{R},i+1}) \in \longrightarrow_{\mathcal{R}}$, $\pi_{\mathcal{R},i} = \langle \pi_{\mathcal{M},s},\, \pi_{\mathcal{B}} \rangle$ and $\pi_{\mathcal{R},i+1} = \langle \pi_{\mathcal{M},g},\, \pi'_{\mathcal{B}} \rangle$.
3. If $\alpha_{\mathcal{R}} \in Act_{\mathcal{B}}$ and $\alpha_{\mathcal{R}} = act_{\mathcal{B},k}$, the controller $\mathcal{K}_k$ is activated until the predicate $\Psi_{b,k}$ is true.
4. If $\alpha_{\mathcal{R}} = act_{\mathcal{M}}$, the controller (5) is applied for the starting region $\pi_{\mathcal{M},s}$ and the goal region $\pi_{\mathcal{M},g}$, until the predicate $\Psi_{r,g}$ is true.

Algorithm 3: Robot motion and action planning under LTL specifications

1. Construct the abstraction of robot mobility $\mathcal{M}$.
2. Construct the action map $\mathcal{B}$.
3. Compose $\mathcal{M}$ and $\mathcal{B}$: $\mathcal{R} = \mathcal{M} \,|||\, \mathcal{B}$.
4. Given a LTL task specification φ over $\Psi_{\mathcal{R}}$, construct its associated NBA $\mathcal{A}_{\varphi}$.
5. Compute the product automaton $\mathcal{A}_{\mathcal{P}} = \mathcal{R} \otimes \mathcal{A}_{\varphi}$ and the associated directed graph $G(\mathcal{A}_{\mathcal{P}}) = (Q_{\mathcal{P}},\, \delta_{\mathcal{P}},\, W_{\mathcal{P}})$
6. Call Algorithm 1 to derive the optimal accepting run $r_{\mathcal{P},opt}$ and project it to $\mathcal{R}$ yielding $\tau_{\mathcal{R},opt}$.
7. Synthesize the hybrid controller as in Algorithm 2.

is given, the complete functionalities model $\mathcal{R}$ remains unchanged and steps 5, 6, 7 are repeated to synthesize the corresponding plan. Whenever the workspace is modified, only $\mathcal{M}$ needs to be re-constructed but the action map $\mathcal{B}$ remains the same and is reused in the following procedures.

VII. Case Study

In the following case study, we consider an autonomous robot that repetitively delivers various products from a source region to destination regions while at the same time avoids the prohibited regions and surveils over certain regions. All simulations are carried out in MATLAB on a desktop computer (3.06 GHz Duo CPU and 8GB of RAM). All computations were accomplished within one second.

A. System Model

We take into account a 2-D workspace for better visualization of the results. The workspace is bounded by region 0: $\pi_0 = \mathcal{B}_1([0.5\ \ 0.5]^T)$, where $[0.5\ \ 0.5]^T \in \mathbb{R}^2$ is the center point and 1 is the radius. Within π_0 there exist five sphere regions of interest: region 1: $\pi_1 = \mathcal{B}_{0.1}([0\ \ 0]^T)$, region 2: $\pi_2 = \mathcal{B}_{0.1}([1\ \ 0]^T)$, region 3: $\pi_3 = \mathcal{B}_{0.1}([1\ \ 1]^T)$, region 4: $\pi_4 = \mathcal{B}_{0.1}([0\ \ 1]^T)$, region 5: $\pi_5 = \mathcal{B}_{0.15}([0.5\ \ 0.5]^T)$. $\Psi_r = \{\Psi_{r,i}\}$ reflects the robot's position, $i = 1, \cdots, 5$. The robot is assumed to satisfy the single-integrator dynamics (4). Each region is potentially connected to any other region and the transition cost is estimated by the straight-line distance between them. There are three properties of concern: $\Psi_{p,1}$={this region has product A}, $\Psi_{p,2}$={this region has product B}, $\Psi_{p,3}$={this region is a office area}. It is assumed that region 1 has product A and B and region 5 is a office area. The robot starts from region 1. Then $\mathcal{M}$ can be easily constructed by (6).

The robot is assumed to be capable of five actions: $act_{\mathcal{B},1}$={pickup A}, $act_{\mathcal{B},2}$={drop A}, $act_{\mathcal{B},3}$={pickup B}, $act_{\mathcal{B},4}$={drop B}, $act_{\mathcal{B},5}$={take pictures}, and $act_{\mathcal{B},0}$={*None*} by definition. The associated costs are 20, 20, 20, 20, 15, 5, respectively. Ψ_b indicates which action is performed. Two propositions reflecting the robot's internal states are given by $\Psi_{s,1}$={the robot has A}, $\Psi_{s,2}$={the robot has B}. The effect and condition functions paired with each action in $Act_{\mathcal{B}}$ are listed in Table I after Section IV-B. $act_{\mathcal{B},0}$ and $act_{\mathcal{B},5}$ can be performed anytime while the others have conditions on Ψ_p and/or Ψ_s. Note that the conditions on $act_{\mathcal{B},1}$ and $act_{\mathcal{B},3}$ indicated that the robot can not hold product A and B at the same time. Assume that the robot initially has no product A or B. The resulting action map $\mathcal{B}$ is then constructed by (9), which has $6 \times 2^2 = 24$ states. We omit the digrams of $\mathcal{M}$ and $\mathcal{B}$ here due to limited space.

B. Task Specification

In plain English, the given task is to repeatedly transport product A to from region 1 to region 2 and product B from region 1 to region 4, while at the same time region 3 need to be under surveillance. Moreover, all office areas should be avoided during the whole mission. Related the propositions we have defined and the LTL, the task is reinterpreted as

> **Infinitely often**, drop A in region 2, drop B in region 4, take pictures within region 3. **Always**, avoid office areas.

The above task can be expressed in LTL format as

$$\varphi = \Box\Diamond(\Psi_{r,2} \wedge \Psi_{b,2}) \wedge \Box\Diamond(\Psi_{r,4} \wedge \Psi_{b,4}) \\ \wedge \Box\Diamond(\Psi_{r,3} \wedge \Psi_{b,5}) \wedge \Box(\neg\Psi_{p,3}).$$

The NBA $\mathcal{A}_{\varphi}$ corresponding to φ above is obtained from [29], with 4 states and 13 transitions. As can be seen here, the size of the resulting $\mathcal{A}_{\varphi}$ is relatively small even though the desired action is quite complex. The main reason is that we do not need to specify in φ that where the robot should go to pickup A and B.

C. Controller Synthesis

By following Algorithm 3, the composition $\mathcal{R} = \mathcal{M} \,|||\, \mathcal{B}$ is constructed, which has 90 states and 606 transitions (compared with 2^{15} states when unfolding $\Psi_{\mathcal{R}}$ blindly). The product automaton $\mathcal{A}_{\mathcal{P}} = \mathcal{R} \otimes \mathcal{A}_{\varphi}$ is given by (11), which has 480 states, out of which 120 are accepting states. Then Algorithm 1 is applied to the state graph $G(\mathcal{A}_{\mathcal{P}})$ to find the optimal accepting path of $\mathcal{A}_{\mathcal{P}}$. The final discrete plan is obtained by projecting this accepting run onto $\mathcal{R}$, which is interpreted in terms of the following sequence of actions in $\mathcal{R}$: pickup A in region 1 $\longrightarrow$ move to region 2 $\longrightarrow$ drop A $\longrightarrow$ move to region 3 $\longrightarrow$ take pictures $\longrightarrow$ move to region 1 $\longrightarrow$ pickup B $\longrightarrow$ move to region 4 $\longrightarrow$ drop B

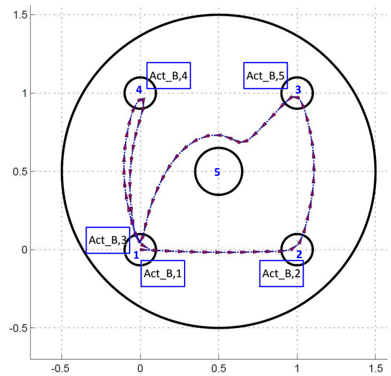

Fig. 4. The final action and motion trajectories fulfills the task specification. The robot stays within the workspace π_0 during the whole mission. Inside the blue boxes are the actions to perform at different regions.

$\longrightarrow$ move to region 1. Note that this sequence is cyclic and can be repeated as many times as needed. By summing up the costs of these actions, the total cost of this discrete plan is computed as 230. The corresponding hybrid control strategy is synthesized based on Algorithm 2. In Figure 4, the final trajectories are shown by the red arrowed lines and the actions performed during the motion are indicated by action names in the blue boxes.

VIII. Conclusion

In this paper we presented a systematic way to synthesize a hybrid control strategy for motion and action planning of an autonomous robot under LTL task specifications. The specifications take into account not only a sequence of regions to visit as in traditional motion planning algorithms, but also as the desired actions at these regions. The proposed framework is adaptable when the workspace model changes. Further research could involve the cases of reactive environments and multi-robot systems.

References

[1] A. Bhatia, L. E. Kavraki, M. Y. Vardi. Sampling-based motion planning with temporal goals. *IEEE International Conference on Robotics and Automation*, 2010.

[2] A. Bhatia, M. R. Maly, L. E. Kavraki, M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3): 55-64, 2011.

[3] C. Baier, J.-P Katoen. Principles of model checking. *The MIT Press*, 2008.

[4] C. Belta, V. Isler, G. J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5): 864-874, 2005.

[5] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, 14: 61-71, 2007.

[6] T. Bylander. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(2): 165204, 1994.

[7] E. M. Clarke, O. Grumberg, D. A. Peled. Model checking. *The MIT Press*, 1999.

[8] X. Ding, M. Kloetzer, Y. Chen, C. Belta. Automatic deployment of robotic teams. *IEEE Robotics Automation Magazine*, 18: 75-86, 2011.

[9] D. V. Dimarogonas and K. J. Kyriakopoulos. Decentralized Navigation Functions for Multiple Robotic Agents with Limited Sensing Capabilities. *Journal of Intelligent and Robotic Systems*. 48(3): 411-433, 2007.

[10] G. E. Fainekos, A. Girard, H. Kress-Gazit, G. J. Pappas. Temporal Logic Motion Planning for Dynamic Mobile Robots. *Automatica*, 45(2): 343-352, 2009.

[11] I. F. Filippidis, D. V. Dimarogonas, K. J. Kyriakopoulos. Decentralized Multi-Agent Control from Local LTL Specifications. *IEEE Conference on Decision and Control*, 2012.

[12] R. E. Fikes, Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3): 189-208, 1972.

[13] M. Guo, K. H. Johansson, D. V. Dimarogonas. Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces. *IEEE International Conference on Robotics and Automation*, 2013.

[14] S. Karaman, E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7): 846-894, 2011.

[15] S. Karaman, E. Frazzoli. Vehicle routing with linear temporal logic specifications: Applications to Multi-UAV Mission Planning. *Navigation, and Control Conference in AIAA Guidance*, 2008.

[16] M. Kloetzer, C. Belta. Automatic deployment of distributed teams of robots from temporal logic specifications. *IEEE Transactions on Robotics*, 26(1): 48-61, 2010.

[17] M. Kloetzer, X. C. Ding, C. Belta. Multi-robot deployment from LTL specifications with reduced communication, *IEEE Conference on Decision and Control*, 2011

[18] D. E. Koditschek, E. Rimon. Robot navigation functions on manifolds with boundary. *Advances Appl. Math.*, 11:412-442, 1990.

[19] H. Kress-Gazit, T. Wongpiromsarn, U. Topcu. Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics and Automation Magazine*, 2011.

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

[21] S. M. LaValle. Planning Algorithms. *Cambridge University Press*, 2006.

[22] H. Levesque, G. Lakemeyer. *Cognitive robotics. Handbook of Knowledge Representation*. Elsevier, 2007.

[23] S. G. Loizou, K. J. Kyriakopoulos. Closed loop navigation for multiple non-holonomic vehicles. *IEEE International Conference on Robotics and Automation*, 3: 4240-4245, 2003.

[24] S. G. Loizou, K. J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. *IEEE Conference on Decision and Control*, 2004.

[25] S. G. Loizou, A. Jadbabaie. Density Functions for Navigation Function Based Systems. *IEEE Conference on Decision and Control*, 2006.

[26] M. Ghallab, C. Aeronautiques, C. K. Isi, D. Wilkins. PDDL: The Planning Domain Definition Language. *Tech. report CVC TR98003/DCS TR1165*, Yale Center for Computational Vision and Control, 1998.

[27] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. *Proceedings of the first international conference on Principles of knowledge representation and reasoning*. Morgan Kaufmann Publishers Inc., 1989.

[28] M. Gelfond, V. Lifschitz. Action languages. *Electronic Transactions on AI*, 1998.

[29] D. Oddoux, P. Gastin. LTL2BA software: fast translation from LTL formulae to Büchi automaton. `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php`.

[30] A. S. Oikonomopoulos, S. G. Loizou, K. J. Kyriakopoulos. Coordination of multiple non-holonomic agents with input constraints. *IEEE International Conference on Robotics and Automation*, 869-874, 2009.

[31] T. Wongpiromsarn, U. Topcu, R. Murray. Receding horizon temporal logic planning, *IEEE Transactions on Automatic Control*, 2012.

[32] A. Ulusoy, S. L. Smith, C. Belta. Optimal Multi-robot path planning with LTL constraints: guaranteeing correctness through synchronization. *International Symposium on Distributed Autonomous Robotic Systems*, 2012.

[33] S. L. Smith, J. Tumova, C. Belta, D. Rus. Optimal path planning for surveillance with temporal logic constraints. *International Journal of Robotics Research*, 30(14): 1695-1708, 2011.