2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

LTL Robot Motion Control based on Automata Learning of Environmental Dynamics

Yushan Chen, Jana Tůmová and Calin Belta

***Abstract*— We develop a technique to automatically generate a control policy for a robot moving in an environment that includes elements with partially unknown, changing behavior. The robot is required to achieve an optimal surveillance mission, in which a certain request needs to be serviced repeatedly, while the expected time in between consecutive services is minimized. We define a fragment of Linear Temporal Logic (LTL) to describe such a mission and formulate the problem as a temporal logic game. Our approach is based on two main ideas. First, we extend results in automata learning to detect patterns of the partially unknown behavior of the elements in the environment. Second, we employ an automata-theoretic method to generate the control policy. We show that the obtained control policy converges to an optimal one when the unknown behavior patterns are fully learned. We implemented the proposed computational framework in MATLAB. Illustrative case studies are included.**

I. Introduction

In recent years, there has been increasing interest in using temporal logics, such as Linear Temporal Logic (LTL), Computation Tree Logic (CTL), μ-calculus, and regular expressions in mobile robotics [1]–[3]. As opposed to classical motion planning techniques [4], such approaches allow for high-level, expressive motion specifications *e.g.,* "Visit region R1, then R2, and then R3, infinitely often. Never enter R5 unless coming directly from R7." The existing works using LTL assume that a finite model of the robot motion in the environment is available. If this is deterministic, control strategies from specifications given as LTL formulas can be found through adaption of off-the-shelf model checking algorithms [5]. If the model is nondeterministic, the control problem can be mapped to a Rabin game [6], and to a Büchi [7] or GR(1) game if the specifications are restricted to fragments of LTL [1], [2]. If the model is probabilistic, the control problem reduces to generating a policy for a Markov Decision Process (MDP) such that the produced language satisfies a formula of a probabilistic temporal logic [8].

Note that all the above works assume that the environment is known, and it is either static or it can change according to known temporal logic rules [1], [2]. Our goal is to relax this assumption and learn these rules based on environmental observations. We show that under some reasonable assumptions, a model of the environment can be incrementally learned and used by the robot in a reactive control strategy.

This work was partially supported by ONR MURI N00014-09-1051, ARO W911NF-09-1-0088, AFOSR YIP FA9550-09-1-020, NSF CNS-0834260, and NSF CNS-1035588 at Boston University, and by LH11065 and GD102/09/H042 at Masaryk University. Y. Chen and C. Belta is with Boston University, {yushanc, cbelta}@bu.edu, and J. Tůmová is with Boston university and Masaryk University, xtumova@fi.muni.cz. Y. Chen is the corresponding author.

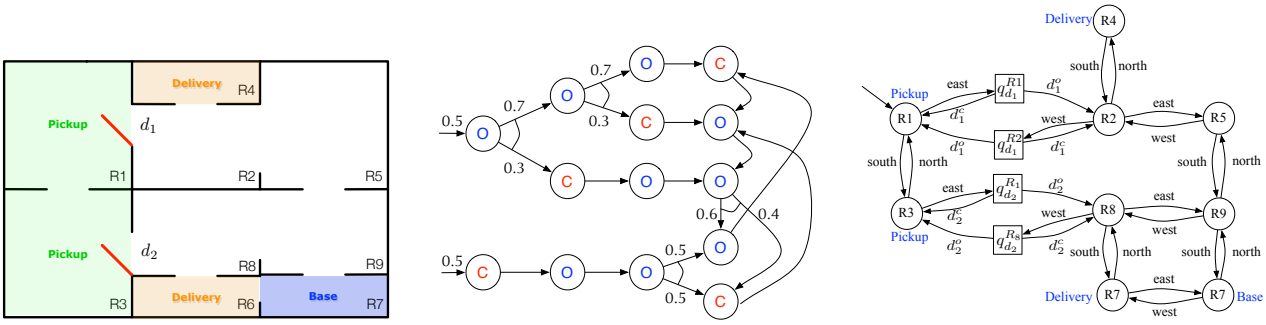
Research in theoretical linguistics [9], [10] show that while languages recognizable by Turing machines are learnable in principle, the corresponding algorithms require unreasonable amounts of time and resources. In [11], [12], it is shown that some subclasses of regular languages are feasibly learnable based on observations of sequences of admissible behaviors. The learned model asymptotically converges to the one that not only can generate all the behaviors already observed, but also predicts what other logically possible behaviors ought to be admissible based on the current observations.

In this paper, we bring together automata learning methods from the fields of theoretical linguistics and techniques from temporal logic games to develop a provably correct control strategy for a robot moving in an environment with unknown dynamics. Specifically, we model the unknown dynamics of the environment elements with a special subclass of ω-regular languages. Moreover, we consider surveillance missions, in which an "optimizing task" needs to be repeatedly serviced. Such missions are executed in infinite time, and therefore allow the robot to learn the environment model gradually. We provide a complete algorithm to learn the environment model, and combine the learning process with the MDP optimal control method proposed in [8] to find the optimal policies for the robot. As a result, the long term performance of the robot that learns while executing optimal policies based on partial information improves in time.

The contribution of this work is twofold. First, we define a specific fragment of LTL to describe the persistent surveillance tasks, which is carefully tailored to guarantee the optimality of the solution (the approach from [8] leads to a sub-optimal solution in general). Second, we show that, under some reasonable assumptions, the environment dynamics can be captured by a subclass of ω-regular languages, called stochastic strictly k-local languages. We extend results in automata learning to obtain a complete algorithm for learning this subclass. Due to space limitations, the algorithms, proofs, and complexity analysis are omitted. They are available in our technical report [13].

II. Models, Problem Formulation, and Approach

Notation: For a finite set $\mathcal{S}$, we use $|\mathcal{S}|$ and $2^{\mathcal{S}}$ to denote its cardinality and power set (set of all subsets), respectively. An infinite (finite) word $w = a_0a_1\ldots$ ($w = a_0a_1\ldots a_n$) over a set $\mathcal{S}$ is an infinite (finite) sequence of elements from $\mathcal{S}$. The length of a finite word w is denoted by $|w|$. The prefix of length m of an infinite word $w = a_0a_1\ldots$ is a finite sequence $a_0a_1\ldots a_m$. Given a word $w = a_0a_1a_2\ldots$, we use $w(i)$ to denote a_i. Given an element $a \in \mathcal{S}$, we use



(a) (b) (c)

Fig. 1: (a) A partitioned environment with door and region labels, where $\mathcal{V} = \{\mathsf{R1}, \ldots, \mathsf{R7}\}$. Regions R1 and R2, and R3 and R8 are separated by doors d_1 and d_2, respectively. The colored regions are labeled with different propositions, *e.g.,* "delivery" is possible in both R4 and R6. (b) An example of a door model $\mathcal{M}_{d_i}$, where c and o stand for d_i^c and d_i^o, respectively. The probabilities of the transitions are shown on top of the corresponding arrows, and they are omitted when the probabilities are 1. The incoming arrows and the corresponding probabilities show the initial distribution. (c) The game transition system $\mathcal{T}_G$ modeling the robot motion in the environment on the left. The robot and door states are represented by circles and squares, respectively. In this example, we have $U = \{east, west, south, north, wait\}$. The control $wait$ is omitted. $q_{d_1}^{\mathsf{R1}}, q_{d_1}^{\mathsf{R2}}$ and d_1^o, d_1^c are the states and inputs for door d_1, respectively.

a^k to denote word $w = aa\ldots a$, where $|w| = k$. The empty word is denoted by ϵ. $\mathcal{S}^*$ and $\mathcal{S}^+$ denote the set of all finite and finite nonempty words over $\mathcal{S}$, respectively. $\mathcal{S}^k$ denotes the set of all words over $\mathcal{S}$ with length k. $\mathcal{S}^\omega$ is the set of all infinite words over $\mathcal{S}$.

A. Environment, door, and robot model

We consider a robot moving in a partitioned environment such as the one illustrated in Fig. 1a. Assume that some adjacent regions of the environment are separated from each other by doors, which can be open or closed. The robot can move between adjacent regions when there is no door or an open door between them. The environment is modeled as a tuple:

$$\mathcal{E} = (\mathcal{V}, \rightarrow_{\mathcal{E}}, \Pi, L_{\mathcal{E}}, \mathcal{D}, F_{\mathcal{D}}), \tag{1}$$

where (i) $\mathcal{V}$ is a set of labels for the regions in the partitioned environment; (ii) $\rightarrow_{\mathcal{E}} \subseteq \mathcal{V} \times \mathcal{V}$ is the adjacency relation of the regions; (iii) Π is a set of atomic propositions; (iv) $L_{\mathcal{E}} : \mathcal{V} \rightarrow 2^{\Pi}$ is a labeling function over the set of regions, where $L_{\mathcal{E}}(v)$ represents the set of atomic propositions that hold true in region v; (v) $\mathcal{D} = \{d_i \mid i \in I\}$ is a set of door labels, where I is an index set; (vi) $F_{\mathcal{D}} : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{D} \cup \emptyset$ is a partial function, where $F_{\mathcal{D}}$ is defined for all $(v, v') \in \rightarrow_{\mathcal{E}}$; $F_{\mathcal{D}}(v, v') = d_i$ means that the adjacent regions v and v' are separated by door d_i, whereas $F_{\mathcal{D}}(v, v') = \emptyset$ indicates that the adjacent regions v and v' are not separated by any door.

The *status* of each door $d_i, i \in I$, belongs to a set $\Sigma_{d_i} = \{d_i^o, d_i^c\}$, where d_i^o and d_i^c stand for door d_i open and closed, respectively. We assume that the status of each door d_i evolves according to a finite discrete time Markov chain, defined as follows:

$$\mathcal{M}_{d_i} = (Q_{d_i}, \iota_{d_i}, P_{d_i}, \Sigma_{d_i}, L_{d_i}), \tag{2}$$

where (i) Q_{d_i} is the set of states; (ii) $\iota_{d_i} : Q_{d_i} \rightarrow [0, 1]$ is the initial distribution, such that $\sum_{q \in Q_{d_i}} \iota(q) = 1$; (iii) $P_{d_i} : Q_{d_i} \times Q_{d_i} \rightarrow [0, 1]$ is the transition probability function such that for $\forall q \in Q_{d_i}$, $\sum_{q' \in Q_{d_i}} P_{d_i}(q, q') = 1$; (iv) Σ_{d_i} is the output alphabet defined above; (v) $L_{d_i} : Q_{d_i} \rightarrow \Sigma_{d_i}$ is the labeling function. The paths of $\mathcal{M}_{d_i}$ are defined as infinite state sequences $r_{d_i} = q_0 q_1 \ldots$, such that $\iota_{d_i}(q_0) > 0$, $q_k \in Q_{d_i}$ and $P_{d_i}(q_k, q_{k+1}) > 0$, $\forall k \geq 0$. A path $r_{d_i} = q_0 q_1 \ldots$ generates an output word $o_{d_i} = L_{d_i}(q_0) L_{d_i}(q_1) \ldots$. The language of the door model $\mathcal{M}_{d_i}$, denoted by $\mathcal{L}(\mathcal{M}_{d_i})$, is defined as the set of all infinite words generated by all paths of $\mathcal{M}_{d_i}$. An example is given in Fig. 1b.

In this paper, we assume that the doors behave independently from the motion of the robot, and the door models $\mathcal{M}_{d_i}, i \in I$, are unknown to the robot. The time is uniformly discretized. For the sake of simplicity, we assume that the time domain is $\mathbb{N}$ and initially, $t = 0$. We assume that the robot can observe the status of each door at each time $t \in \mathbb{N}$ and memorize the status history of each door. The observed time behavior of door d_i can be represented as a function $Obs_i : \mathbb{N} \rightarrow \Sigma_{d_i}$, where $Obs_i(t)$ represents the status of door d_i (*i.e.,* open or closed) at time $t \in \mathbb{N}$. At time t, the status history of door d_i is $Obs_i(0) Obs_i(1) \ldots Obs_i(t)$, which is a prefix of an infinite word in $\mathcal{L}(\mathcal{M}_{d_i})$.

The motion capability of the robot in the environment is represented by a finite set of motion primitives U. With slight abuse of notation, we define a function $U(v)$, where $v \in \mathcal{V}$ and $U(v) \in U$, to represent the available motion primitives at region v. We define a function $M : \mathcal{V} \times U \rightarrow \mathcal{V}$ such that $M(v, u) = v'$, where $(v, v') \in \rightarrow_{\mathcal{E}}$, represents that the robot can move to region v' by applying the motion primitive $u \in U(v)$ at region v. Assume that the robot is in region v. During each interval $[t, t+1)$ the following events happen in the following order: 1) at time t, the robot observes the status of each door (*i.e.,* obtains $Obs_i(t)$ for all $i \in I$) and chooses a motion primitive $u \in U$ such that $M(v, u) = v'$; 2) after choosing u, the robot behaves as follows: (a) if $v = v'$ then the robot stays in v for $[t, t+1)$ (b) if $v \neq v'$, $F_{\mathcal{D}}(v, v') = d_i$ (*i.e.,* v and v' are separated by door d_i), and door d_i is closed then the robot is forced to stay in v for $[t, t+1)$ (c) if $v \neq v'$, and $F_{\mathcal{D}}(v, v') = \emptyset$ or the door $d_i = F_{\mathcal{D}}(v, v')$ is open,

the robot applies u and reaches v' 3) each door chooses its next status (*i.e.,* to open or close) and finishes executing its transition before $t+1$.

To capture how the unknown models of the doors affect the robot motion in the environment, we model the robot as a game transition system, denoted by $\mathcal{T}_G$. There are two players in the game: the robot (player) and the doors (adversary). The set of states of $\mathcal{T}_G$ is partitioned in two sets: robot set $\mathcal{V}$, which is the set of states (*i.e.,* regions in the environment $\mathcal{E}$) from which the robot can choose a transition, and door set $Q_{\mathcal{D}}$, which is the set of states from which the doors decide the transition. Each door d_i, which separates regions v and v', is represented by two states (one for each region), denoted by $q_{d_i}^{v}$ and $q_{d_i}^{v'}$. Thus, $Q_{\mathcal{D}} = \cup_{i\in I}\{q_{d_i}^{v}, q_{d_i}^{v'} \mid d_i = F_{\mathcal{D}}(v,v'),\ v,v' \in \mathcal{V}\}$. An example of a game transition system $\mathcal{T}_G$ is shown in Fig. 1c. Formally, we have

$$\mathcal{T}_G = (Q_G, q_G^{in}, \Sigma_G, \delta_G, \Pi, L_G), \tag{3}$$

where 1) $Q_G = \mathcal{V} \cup Q_{\mathcal{D}}$ is the finite set of states; 2) $q_G^{in} \in \mathcal{V}$ is the initial state; 3) $\Sigma_G = U \cup \Sigma_{\mathcal{D}}$ is the finite set of inputs, where $\Sigma_{\mathcal{D}} = \cup_{i\in I}\Sigma_{d_i}$; 4) $\delta_G : Q_G \times \Sigma_G \to Q_G$ is the transition function, such that for $v, v' \in \mathcal{V}$ and $u \in U(v)$, we have (a) $v' = \delta_G(v,u)$, if and only if $M(v,u) = v'$ and $F_{\mathcal{D}}(v,v') = \emptyset$, and (b) $q_{d_i}^{v} = \delta_G(v,u)$, $v = \delta_G(q_{d_i}^{v}, d_i^{c})$ and $v' = \delta_G(q_{d_i}^{v}, d_i^{o})$, if and only if $M(v,u) = v'$ and $d_i \in F_{\mathcal{D}}(v,v')$; 5) Π is the set of atomic propositions; 6) $L_G = L_{\mathcal{E}}$ is the labeling function over the states in $\mathcal{V}$.

Note that the game transition system always initiates from a robot state. The transition function δ_G is defined in such a way that for all states in $\mathcal{V}$, only motion primitives $u \in U$ are the available inputs, while for all states in $Q_{\mathcal{D}}$, only the status of the doors (*i.e.,* d_i^{c}, d_i^{o}) are the available inputs. A transition $\delta_G(q,\sigma) = q'$ indicates that, while the system is in state q, it can take a transition to state q' under input σ. When a motion primitive is applied, the game transition system transits from a robot state to another robot state (during one time interval), *i.e.,* makes a transition from a robot state to another or makes two transitions - one to a door state and the other from the door state to a robot state. An *input word* of $\mathcal{T}_G$ is defined as an infinite sequence $w_G = \sigma_0\sigma_1\ldots \in \Sigma_G^{\omega}$. A *run* of $\mathcal{T}_G$ produced by the input word w_G is an infinite sequence $r_G = q_0q_1\ldots$ with the property that q_0 is the initial state and $q_{k+1} = \delta_G(q_k,\sigma_k)$, for all $k \geq 0$. A robot run, denoted by r_{rob}, can be obtained by deleting all the door states from r_G. A run of the robot $r_{rob} = v_0v_1\ldots$ produced by r_G generates an *output word* $o_G = L_G(v_0)L_G(v_1)\ldots$.

Remark 1: To keep the exposition simple, we assume that all transitions of the robot take one time interval. We can accommodate different times by adding weights on the transitions of the game transition system.

B. Task Specification

We employ a fragment of Linear Temporal Logic (LTL) to describe robot motion specifications. A detailed description of the syntax and semantics of LTL is beyond the scope of this paper and can be found in [14]. Roughly, an LTL formula is built up from a set of atomic propositions Π, standard Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and temporal operators X (next), U (until), F (eventually), G (always). The semantics of LTL formulas are given over infinite words over 2^{Π}, such as the output words generated by the game transition system from Eqn. (3). Assume that ϕ, ϕ_1, and ϕ_2 are LTL formulas over Π and $o_G = L_G(v_0)L_G(v_1)\ldots$ is an output word of $\mathcal{T}_G$. Word o_G satisfies atomic proposition α if α is satisfied in the first position of the word o_G (*i.e.*, $\alpha \in L_G(v_0)$). Formula $\phi_1\,\mathsf{U}\,\phi_2$ intuitively means that ϕ_2 is true eventually, and ϕ_1 is true at least until ϕ_2 is true. Formula $\mathsf{G}\,\phi$ states that ϕ holds at all positions of the word, and $\mathsf{F}\,\phi$ states that ϕ becomes eventually true. More expressiveness can be achieved by combining the above operators.

In this paper, we consider robot missions requiring infinite executions of the robot, such as surveillance, persistent monitoring, and pickup-delivery tasks. Specifically, the robot needs to repeatedly accomplish a set of *tasks*. Each task is represented as a Boolean combination of atomic propositions in Π. A task φ is accomplished if the robot visits a region, where φ is satisfied. We denote the set of all tasks by Φ_{GF}. For instance, assume that regions R1, and R3 in Fig. 1a are pickup regions. A task $\mathsf{Pickup} \in \Phi_{\mathsf{GF}}$ means that the robot needs to repeatedly visit a pickup region, *i.e.,* one of the regions R1 and R3. Furthermore, the robot might be required to service the tasks from Φ_{GF} in a certain order. For instance, the robot might need to first visit a pickup region and then a delivery region without revisiting a pickup region in between. Besides that, we allow for specifying safety properties, *i.e.,* properties of the type "nothing bad ever happens" (*e.g.,* the robot keeps avoiding certain regions), and request-response properties of the form "each request requires a response" (*e.g.,* whenever a certain region is visited, then another region needs to be visited after that).

We assume that the robot always needs to keep servicing a special, so-called optimizing task, denoted by $\pi \in 2^{\Pi}$, for instance, visiting a base. We say that the robot *completes a task cycle* each time it accomplishes all the tasks in Φ_{GF}, while following the demanded order, satisfying the safety properties, responding to each request, and returns to the region where the optimizing task is satisfied. Formally, we use a subclass of LTL formulas over Π to specify the robotic mission. The specification is in the form of

$$\begin{aligned}\phi \;=\; & \mathsf{GF}\pi \;\wedge \bigwedge_{\varphi\in\Phi_{\mathsf{GF}}} \mathsf{GF}\varphi \;\wedge \bigwedge_{\varphi\in\Phi_{\mathsf{GF}}} \mathsf{G}(\pi \Rightarrow \pi\mathsf{U}(\neg\pi\mathsf{U}\varphi)) \;\wedge \\ & \psi_{\text{ord}} \wedge \psi_{\text{safe}} \wedge \psi_{\text{react}}, \text{where } \psi_{\text{ord}}, \psi_{\text{safe}}, \text{and } \psi_{\text{react}} \text{ are} \\ & \text{recursively defined as:}\end{aligned}$$

$$\begin{aligned}\psi_{\text{ord}} &= \psi_{\text{ord}} \wedge \psi_{\text{ord}} \mid \psi_{\text{ord}} \vee \psi_{\text{ord}} \mid \mathsf{G}(\varphi_1 \Rightarrow \varphi_1\mathsf{U}(\neg\varphi_1\mathsf{U}\varphi_2)) \mid \epsilon,\\ \psi_{\text{safe}} &= \psi_{\text{safe}} \wedge \psi_{\text{safe}} \mid \psi_{\text{safe}} \vee \psi_{\text{safe}} \mid \mathsf{G}\neg\xi \mid \epsilon,\\ \psi_{\text{react}} &= \psi_{\text{react}} \wedge \psi_{\text{react}} \mid \psi_{\text{react}} \vee \psi_{\text{react}} \mid \mathsf{G}(\xi_1 \Rightarrow (\neg\pi\mathsf{U}\xi_2)) \mid \epsilon,\end{aligned} \tag{4}$$

where $\varphi, \varphi_1, \varphi_2 \in \Phi_{\mathsf{GF}}$, and ξ, ξ_1, ξ_2 are boolean combinations of the atomic propositions from Π. The terms $\mathsf{GF}\pi$ and $\bigwedge_{\varphi\in\Phi_{\mathsf{GF}}} \mathsf{GF}\varphi$ enforce that the optimizing task π and each task

$\varphi \in \Phi_{\mathsf{GF}}$ be accomplished infinitely many times. The term $\bigwedge_{\varphi\in\Phi_{\mathsf{GF}}} \mathsf{G}(\pi \Rightarrow \pi \mathsf{U}(\neg\pi \mathsf{U}\varphi))$ ensures that once the robot accomplishes the optimizing task π, it can not redo it before completing all the tasks in Φ_{GF}. The formula ψ_{ord} specifies the relative order between the tasks in Φ_{GF} during one task cycle. The formula ψ_{safe} ensures that all the dangerous areas are not visited. The formula ψ_{react} specifies reactive tasks, *i.e.,* ξ_1 triggers task ξ_2. Moreover, we use $\neg\pi \mathsf{U}\xi_2$ to ensure that once ξ_1 is requested, the response ξ_2 happenes within the same task cycle, *i.e.,* before revisit to π. $\psi_{\text{ord}} = \epsilon$ expresses that there are no requirements on the order of tasks in Φ_{GF}, and similarly $\psi_{\text{safe}} = \epsilon$ and $\psi_{\text{react}} = \epsilon$ mean that there are no safety and request-response requirements, respectively.

C. Problem Formulation

Our history dependent robot control policy is in the form of an infinite sequence $\mathcal{C} = \{\mu_0, \mu_1, \ldots\}$ where $\mu_t : \mathcal{V}^t \times \prod_{i\in I}(\Sigma_{d_i})^t \to U$. Given the sequence of the regions visited by the robot $v_0 v_1 \ldots v_t$ and the status history $Obs_i(0)Obs_i(1)\ldots Obs_i(t)$ of door d_i for all $i \in I$, the policy μ_t returns the motion primitive $u_t \in U(v_t)$ to be applied at time t. Given the control policy $\mathcal{C}$, the status of the doors $\{Obs_i(0)Obs_i(1)\ldots, i \in I\}$, and the initial position of the robot, the run of the robot is an infinite sequence $r_{\mathcal{C}} = v_0 v_1 \ldots$, which satisfies the following conditions: for all times t, (a) $\delta_G(v_t, u_t) = v_{t+1}$, or (b) $\exists d_i$ such that $\delta_G(v_t, u_t) = q_{d_i}^{v_t}$ and $\delta_G(q_{d_i}^{v_t}, Obs_i(t)) = v_{t+1}$. Note that $r_{\mathcal{C}}$ is a robot run produced by the game transition system $\mathcal{T}_G$. Given $\mathcal{C}$ and the initial position of the robot, $r_{\mathcal{C}}$ is not unique due to the probabilistic transitions in the door models.

We would like to obtain an optimal policy such that the robot mission ϕ is accomplished, and the expected time in between consecutive satisfactions of the optimizing task π is minimized. Given a run $r_{\mathcal{C}} = v_0 v_1 v_2 \ldots$, we use $C(r_{\mathcal{C}}, T)$ to denote the number of task cycles completed until time T plus 1 (for technical reasons to become clear later, we set $C(r_{\mathcal{C}}, 0) = 1$). Given $\mathcal{C}$, we define the average time per cycle (ATPC) starting from the initial position q_G^{in} as follows:

$$J(q_G^{in}) = \limsup_{T\to\infty} E\left(\frac{T}{C(rc, T)}\right), \tag{5}$$

where $E(\cdot)$ denotes the expected value. Now we are ready to formulate the problem:

Problem 1: GIVEN a game transition system $\mathcal{T}_G$ (Eqn. (3)), a formula ϕ in the form of Eqn. (4), and the set of observed status of all doors $\{Obs_i, i \in I\}$, FIND a robot control policy $\mathcal{C}$, such that (i) all runs of the robot generated by $\mathcal{C}$ satisfy ϕ, and (ii) $J(q_G^{in})$ is minimized.

D. Summary of technical approach

We propose a two-step approach to Prob. 1. In the first part (Sec. III), we study how to learn the door models, based on the observed statuses of the doors. We show that, under some reasonable assumptions, the learned door models eventually converge to the real models. In the second part (Sec. IV), we synthesize a control policy for the robot to achieve a persistent surveillance mission given as a fragment of LTL, by incorporating the learned door models. We show that for this type of mission, learning can be adapted to improve the long-term performance of the robot (*i.e.,* minimizing the time in between servicing the optimizing task π).

III. Learning the door models

In this section, we focus on learning a door model $\mathcal{M}_{d_i}$. At each time t, the status history up to time t, *i.e.,* a *positive sample* of the door language, is known. However, no examples of forbidden door behavior (*negative samples*) are available. Although it is well-known that in general, a language cannot be learned only from its positive samples, there are subclasses of languages that can [12]. We characterize an extension to the strictly k-local subclass of regular languages [12] called *stochastic strictly k-local ω-regular languages*, that 1) are rich enough to represent interesting door behaviors, and 2) can be learned from positive samples. We derive an algorithm for learning a door model (*i.e.,* Markov chain defined in Eqn. (2)) from a prefix of a word generated by $\mathcal{M}_{d_i}$ assuming that the generated language is within this subclass. By running the learning algorithm iteratively at each time $t \in \mathbb{N}$, the learned door model eventually converges to the real door model.

A. Door Language

Definition 1: A finite string $y \in \Sigma^*$ is a *factor* of an infinite string $w \in \Sigma^\omega$ if and only if $\exists x \in \Sigma^*, w' \in \Sigma^\omega$ such that $w = xyw'$. If $|y| = k$, then y is a k-factor of w. For example, $F_3((ooc)^\omega) = \{ooc, oco, coo\}$. The set of all k-factors is denoted by $F_k(\Sigma^\omega)$.

Definition 2: An ω-regular language $\mathcal{L} \subseteq \Sigma^\omega$ is *strictly k-local* if and only if there exists a finite set of *allowed* k-factors $\mathsf{S} \subseteq F_k(\Sigma^\omega)$ such that $\mathcal{L} = \{w \in \Sigma^\omega \mid F_k(w) \subseteq \mathsf{S}\}$.

We denote by $\mathcal{L}(\mathsf{S})$ the strictly k-local language defined by the set of allowed k-factors S. Given k and a set of allowed k-factors S, the strictly k-local language $\mathcal{L}(\mathsf{S})$ can be learned using a slight modification of the algorithm for learning strictly k-local languages of finite words presented in [15]. The details will be discussed later in this section.

Assumption 1: We assume that for all $i \in I$, the door language $\mathcal{L}(\mathcal{M}_{d_i})$ is strictly k-local, where k is known.

Although this assumption does not allow the door to behave arbitrarily, it still allows it to capture many rich patterns and rules in the door behavior. For instance, the Markov chain depicted in Fig. 1b represents that the door stays open exactly twice or three times after each close. The language is strictly 4-local defined by a set of allowed sequences $\mathsf{S} = \{oooc, ooco, ocoo, cooo, cooc\}$.

Given Assump. 1, the language $\mathcal{L}(\mathcal{M}_{d_i})$ can be learned using a modification of the algorithm introduced in [15]. However, to fully learn the door model, we also need to learn the correct probabilities in the transition probability function of $\mathcal{M}_{d_i}$. This can be done for the *stochastic* extension of strictly k-local languages defined in the following.

Given a Markov chain $\mathcal{M} = (Q, \iota, P, \Sigma, L)$, we define a probability function $\mathsf{P}_{\mathcal{M}}^{\text{out}} : \Sigma^+ \times \Sigma \to [0, 1]$ as follows: $\mathsf{P}_{\mathcal{M}}^{\text{out}}(x, \sigma) = \sum_{q_0\ldots q_n \in g(x\sigma)} P(q_{n-1}, q_n)$, where

$g(x\sigma) = \{q_0 \dots q_n \mid L(q_0)L(q_1)\dots L(q_n) = x\sigma\}$. Intuitively, $\mathsf{P}^{\text{out}}_{\mathcal{M}}(x,\sigma)$ is the probability that a prefix x of an output word of $\mathcal{M}$ is followed by the symbol σ. For instance, for Markov chain $\mathcal{M}$ in Fig 1b, $\mathsf{P}^{\text{out}}_{\mathcal{M}}(ooocoo, o) = 0.6$.

Definition 3: Given a set of allowed factors $\mathsf{S} \subseteq F_k(\Sigma^\omega)$ and a probability distribution function $\mathsf{P} : \mathsf{S} \times \Sigma \to [0,1]$, a *stochastic strictly k-local language* $\mathcal{L}(\mathsf{S},\mathsf{P})$ is a language $\{w \in \Sigma^\omega \mid F_k(w) \subseteq \mathsf{S}\}$ generated by a Markov chain $\mathcal{M} = (Q, \iota, P, \Sigma, L)$, such that $\mathsf{P}^{\text{out}}_{\mathcal{M}}(y,\sigma) = \mathsf{P}(z,\sigma)$, $\forall y = xz$, where $z \in \mathsf{S}$ and $x \in \Sigma^*$.

In other words, for a stochastic strictly k-local language, the probability that a prefix of a word from $\mathcal{L}$ is followed by a symbol $\sigma \in \Sigma$ depends only on the last k-factor of this prefix. For example, the language generated by $\mathcal{M}$ in Fig. 1b is a stochastic 4-local language, where $\mathsf{P}^{\text{out}}_{\mathcal{M}}(cooocoo, o) = \mathsf{P}(ocoo, o) = 0.6$, $\mathsf{P}^{\text{out}}_{\mathcal{M}}(ooocoo, o) = \mathsf{P}(ocoo, o) = 0.6$, etc.

Assumption 2: We assume that for all $i \in I$, the door language $\mathcal{L}(\mathcal{M}_{d_i})$ is stochastic strictly k-local, where k is known. Furthermore, let all words initiate with a given k-factor $y_{\text{init}} \in F_k(\Sigma^\omega)$.

B. Markov Chain Learning

Given k, alphabet Σ, set of allowed factors $\mathsf{S} \subseteq F_k(\Sigma^\omega)$, probability distribution function $\mathsf{P} : \mathsf{S} \times \Sigma \to [0,1]$, and initial k-factor $y_{\text{init}} \in \Sigma^\omega$, the stochastic strictly k-local ω-regular language $\mathcal{L}(\mathsf{S},\mathsf{P})$ initiating with y_{init} can be learned using a modification of the algorithm for learning strictly k-local regular languages introduced in [15]. The original algorithm is adapted to deal with stochastic ω-regular languages instead of regular languages and can be found in the technical report [13]. Specifically, the algorithm returns a Markov chain $\mathcal{M} = (Q, \iota, P, \Sigma, L)$, such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathsf{S},\mathsf{P})$.

Assume, that at any time $t \in \mathbb{N}, t \geq k$ the status history $Obs_i(0)\dots Obs_i(t)$ of each door d_i is given. Based on this information, we can compute an approximation of the set of allowed factors $\mathsf{S}^t_{d_i}$ as a set of all k-factors that appear in $Obs_i(0)\dots Obs_i(t)$, an approximation of the probability function $\mathsf{P}^t_{d_i}$ via frequency analysis, and an initial k-factor y_{init}. Hence, we can learn an approximation of the door model $\mathcal{M}^t_{d_i}$ at time t. Formally, we have

$$
\begin{aligned}
y_{\text{init},d_i} &= Obs_i(0)\dots Obs_i(k-1),\\
\mathsf{S}^t_{d_i} &= \{y \mid y \text{ is a } k\text{-factor of } Obs_i(0)\dots Obs_i(t)\},\\
\mathsf{P}^t_{d_i}(y, d_i^c) &= \frac{N^t_{yd_i^c}}{N^t_{yd_i^c} + N^t_{yd_i^o}}, \mathsf{P}^t_{d_i}(y, d_i^o) = \frac{N^t_{yd_i^o}}{N^t_{yd_i^c} + N^t_{yd_i^o}},
\end{aligned}
$$

where $N^t_{yd_i^c}$ and $N^t_{yd_i^o}$ are the number of occurrences of $k+1$ factors yd_i^c, and yd_i^o in $Obs_i(0)\dots Obs_i(t)$, respectively.

Theorem 1: Given Assump. 2, when time $t \to \infty$, the door model approximation $\mathcal{M}^t_{d_i}$ almost surely generates a language $\mathcal{L}(\mathcal{M}^t_{d_i}) = \mathcal{L}(\mathcal{M}_{d_i})$ (*i.e.,* the probability that $\mathcal{M}^t_{d_i}$ generates $\mathcal{L}(\mathcal{M}_{d_i})$ is 1).

IV. Optimal LTL Control Synthesis

Given the door model approximation $\mathcal{M}^t_{d_i} = (Q^t_{d_i}, \iota^t_{d_i}, P^t_{d_i}, \Sigma^t_{d_i}, L^t_{d_i})$, we aim to find a control policy $\mathcal{C}$ such that 1) all runs of the robot under $\mathcal{C}$ satisfy the LTL formula ϕ, and 2) $J(q^{in}_G)$ is minimized. We first construct a Markov Decision Process (MDP) to capture the motion of the robot when it interacts with the doors in the environment. The MDP can be seen as a product of the game transition system $\mathcal{T}_G$ and $\mathcal{M}^t_{d_i}$, $i \in I$ and it is defined as follows:

$$\mathcal{M}_D = (Q_D, q^{in}_D, U_D, P_D, \Pi, L_D), \tag{6}$$

where 1) $Q_D \subseteq \mathcal{V} \times Q^t_{d_1} \times \dots \times Q^t_{d_{|I|}}$ is the finite set of states; 2) $q^{in} = (q^{in}_G, q^{in}_{d_1}, \dots, q^{in}_{d_{|I|}}) \in Q_D$ is the initial state, where $\iota^t_{d_i}(q^{in}_{d_i}) = 1$, $\forall i \in I$; 3) U_D is the set of controls (*i.e.,* motion primitives) and we define the function $U_D(v, q_{d_1}, \dots, q_{d_{|I|}}) = U(v)$ to represent the available actions at state $(v, q_{d_1}, \dots, q_{d_{|I|}})$; 4) $P_D : Q_D \times U_D \times Q_D \to [0,1]$ is the transition probability function such that $P_D((v, q_{d_1}, \dots, q_{d_{|I|}}), u, (v', q'_{d_1}, \dots, q'_{d_{|I|}})) = \prod_{i\in I} P^t_{d_i}(q_{d_i}, q'_{d_i})$, if and only if (a) $\delta_G(v,u) = v'$, or (b) $\delta_G(v,u) = q^v_{d_i}$, $\delta_G(q^v_{d_i}, d^o_i) = v'$, $L^t_{d_i}(q_{d_i}) = d^o_i$, or (c) $\delta_G(v,u) = q^v_{d_i}$, $\delta_G(q^v_{d_i}, d^c_i) = v'$, $L^t_{d_i}(q_{d_i}) = d^c_i$, and $P_D((v, q_{d_1}, \dots, q_{d_{|I|}}), u, (v', q'_{d_1}, \dots, q'_{d_{|I|}})) = 0$, otherwise; 5) Π is the set of atomic propositions; 6) $L_D : Q_D \to 2^\Pi$ is the labeling function inherited from the game transition system $\mathcal{T}_G$, such that $L_D(v, q_{d_1}, \dots, q_{d_{|I|}}) = L_G(v)$.

We define a control function $\eta : Q_D \to U_D$ such that $\eta(q) \in U_D(q)$, $\forall q \in Q_D$. An infinite sequence of control functions $\{\eta_0, \eta_1, \dots\}$ is called an MDP policy. Given the initial state, an infinite sequence $r_D = q_0 q_1 \dots$ on $\mathcal{M}_D$ generated under $\{\eta_0, \eta_1, \dots\}$ is called a path on $\mathcal{M}_D$ if $P_D(q_t, \eta_t(q_t), q_{t+1}) > 0$, $\forall t$. A path $r_D = q_0 q_1 \dots$ on $\mathcal{M}_D$ generates an *output* word $o_D = L_D(q_0)L_D(q_1)\dots$ on $\mathcal{M}_D$. We say a path r_D on $\mathcal{M}_D$ satisfies an LTL formula if and only if its corresponding output word satisfies the LTL formula. We say an MDP policy satisfies an LTL formula almost surely if and only if the probability that a path generated by the policy satisfy the LTL formula is 1 (see [14]). The path $r_D = q_0 q_1 \dots$ on $\mathcal{M}_D$ can be mapped to a run $r_G = v_0 v_1 \dots$ on the game transition system, where v_t is the 1st element in q_t, $\forall t$. Given r_D, we can see that each visit to a state Q_D satisfying π corresponds to completion of one task cycle in r_G. A policy $\{\eta_0, \eta_1, \dots\}$ on $\mathcal{M}_D$ can be mapped to $\mathcal{T}_G$ to obtain a control policy $\mathcal{C}$. When the door models are fully learned, the ATPC cost function on $\mathcal{T}_G$ defined in Eqn. (5) under $\mathcal{C}$ is equal to the expected cost in between visiting states in Q_D satisfying π under $\{\eta_0, \eta_1, \dots\}$. However, it is not the case when the door models are not equal to the real models, since $\mathcal{M}_D$ is only an approximation of the robot movement in the environment.

Therefore, when the door models are fully learned, our synthesis problem is converted to the problem of finding a control policy $\{\eta_0, \eta_1, \dots\}$ on MDP $\mathcal{M}_D$, such that the policy satisfies the LTL formula almost surely, and the expected cost in between visiting states in Q_D satisfying π is minimized. Specifically, we use an automata-theoretical approach similar to that in [8] to generate the desired policy. Prop. 1 shows that there exists at least one MDP policy on $\mathcal{M}_D$ that satisfies ϕ almost surely. This requirement is a necessary condition for employing the algorithm.

Proposition 1: Under the assumption that each door is not allowed to be closed for infinite time, there exists an MDP

policy on $\mathcal{M}_D$ satisfying ϕ almost surely if there exists a robot run of $\mathcal{T}_G$ satisfying ϕ.

Note that in general, the algorithm proposed in [8] only returns a sub-optimal policy for the given MDP. We show that the algorithm can be used to find an optimal policy while guaranteeing the satisfaction of the task specification, if we use the carefully tailored fragment of LTL (Eqn. (4)) to capture the robot missions (see our technical report [13]).

Proposition 2: Given ϕ in the form of Eqn. (4), we can use the algorithm proposed in [8] to find an optimal policy for $\mathcal{M}_D$, which satisfies ϕ almost surely and minimizes the expected cost in between visiting states in Q_D satisfying π.

Since the learned door model might not be equal to the actual door model, we need to update the door model approximation. We can use the approximation to construct M_D and then obtain a control policy. This policy might not be optimal. We suggest an iterative procedure which, whenever a more precise approximation of $\mathcal{M}^t_{d_i}$ is learned, recomputes the MDP and the control policy. Based on Thm. 1 and Prop. 2, $\mathcal{M}^t_{d_i}$ converges to $\mathcal{M}_{d_i}$ eventually, and the computed control policy will ensure that the robot satisfies ϕ and the long-term performance of the robot is optimized.

V. Case Study

The algorithmic framework developed in this paper was implemented in MATLAB, and used in conjunction with a simulator to demonstrate simulations of a robot performing missions in the environment shown in Fig. 1a. Here we provide an example as case study. The model for door d_1 is shown in Fig. 1b. The language of door d_2 is a strictly-local language with set of allowed factors $\{oc, co\}$. The goal of the robot is to continuously perform a pickup-delivery task, and return to the base once in a while. The robot is required to pick up items at one of the states marked by Pickup (see Fig. 1a), and drop them off at one of the states marked by Delivery. The robot is required to finish the pickup-delivery task before going back to the base. This task can be written as the following formula, which is in the form of Eqn. (4):

$$\begin{aligned}\phi \;=\; & \mathsf{GF}(\mathsf{Base}) \wedge \mathsf{GF}(\mathsf{Pickup}) \wedge \\ & \mathsf{G}(\mathsf{Base} \rightarrow \mathsf{Base}\ \mathsf{U}\ ((\neg\mathsf{Base})\ \mathsf{U}\ \mathsf{Pickup}))) \wedge \\ & \mathsf{G}(\mathsf{Pickup} \rightarrow ((\neg\mathsf{Base})\ \mathsf{U}\ \mathsf{Delivery})),\end{aligned}$$

where $\mathsf{GF}(\mathsf{Base})$ and $\mathsf{GF}(\mathsf{Pickup})$ enforce the robot to repeatedly pick up items and return to base. $\mathsf{G}(\mathsf{Base} \rightarrow \mathsf{Base}\ \mathsf{U}\ ((\neg\mathsf{Base})\ \mathsf{U}\ \mathsf{Pickup})))$ and $\mathsf{G}(\mathsf{Pickup} \rightarrow ((\neg\mathsf{Base})\ \mathsf{U}\ \mathsf{Delivery}))$ ensure that the items are picked up and delivered in each task cycle. The optimizing task is going back to Base, and we aim to minimize the expected time in between consecutive visits to the base. In this example, given ϕ, the door and robot models, we computed the real optimal control policy $\mathcal{C}^*$ and its corresponding ATPC cost $J^*(q^{in}_G)$. We have $J^*(q^{in}_G) = 8$, *i.e.,* in average, the time in between consecutive visits to the base is equal to 8 time intervals.

Then we assumed that the door models were unknown and ran our algorithm. The robot learned the door models gradually using the method introduced in Sec. III. Every time when a new door model is learned, the control policy is recomputed. Specifically, the MDP $\mathcal{M}_D$ contains approximately 90 states. The generated product MDP contains around 810 states. The re-computation of the optimal control policy takes about 9s. Also, we use simulations to compute the ATPC cost $J(q^{in}_G)$ for policies generated at different time. In Fig. 2, we show that the ATPC cost is approximate 20 when the control policy is first generated and it decreases to a value that is very close to the optimal cost (*i.e.,* $J^*(q^{in}_G) = 8$) after time 20. In other words, the obtained control policy is very close to the optimal solution after time 20.

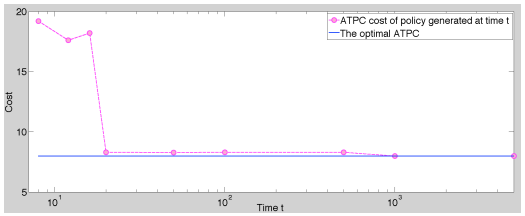


Fig. 2: The ATPC costs of the control policies generated at different time. As shown in the figure, starting from time 20, the obtained control policy is very close to the optimal one.

We use a simulator to demonstrate the motion of the robot. Our video submission displays this simulation (also available at `http://hyness.bu.edu/ICRA2012/`).

Acknowledgment: We thank Jeffrey Heinz and Herbert Tanner at Univ. of Delaware for inspiring discussions.

References

[1] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc CDC and CCC*, Shanghai, China, December 2009, pp. 5997–6004.

[2] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robotics and Automation Magazine*, vol. 15, no. 1, pp. 30–38, 2008.

[3] S. Karaman and E. Frazzoli, "Complex mission optimization for multiple-UAVs using linear temporal logic," in *Proc ACC*, Seattle, USA, 2008, pp. 2003–2009.

[4] S. M. LaValle, *Planning algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[5] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proc ICRA*, 1995.

[6] J. Tumova, B. Yordanov, C. Belta, I. Cerna, and J. Barnat, "A symbolic approach to controlling piecewise affine systems," in *Proc CDC*, Atlanta, GA, 2010.

[7] M. Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Proc HSCC*, ser. LNCS. Spinger Verlag, 2008, pp. 287–300.

[8] X. Ding, S. L. Smith, C. Belta, and D. Rus, "MDP optimal control under temporal logic constraints," in *Proc CDC*, Orlando, USA, 2011.

[9] J. J. Horning, "A study of grammatical inference," Ph.D. dissertation, Stanford University, 1969.

[10] L. Becerra-Bonache, A. H. Dediu, and C. Tîrnauca, "Learning DFA from correction and equivalence queries." in *ICGI*, ser. LNCS, vol. 4201. Springer, 2006, pp. 281–292.

[11] D. Angluin, "Inference of reversible languages," *J. ACM*, vol. 29, no. 3, pp. 741–765, 1982.

[12] J. Heinz, "String extension learning," in *Proc ACL*, Uppsala, Sweden, 2010, pp. 897–906.

[13] Y. Chen, J. Tumova, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," Boston University, CISE Technical Report, Tracking Number 2012-IR-0002, (available at http://www.bu.edu/systems/publications/search-publications-database/).

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

[15] P. Garcia, E. Vidal, and J. Oncina, "Learning locally testable languages in the strict sense." in *Proc ALT*, Tokyo, Japan, 1990, pp. 325–338.