Control in Belief Space with Temporal Logic Specifications

Cristian-Ioan Vasile[1], Kevin Leahy[2], Eric Cristofalo[2], Austin Jones[3], Mac Schwager[4] and Calin Belta[2]

***Abstract*— In this paper, we present a sampling-based algorithm to synthesize control policies with temporal and uncertainty constraints. We introduce a specification language called *Gaussian Distribution Temporal Logic (GDTL)*, an extension of Boolean logic that allows us to incorporate temporal evolution and noise mitigation directly into the task specifications, e.g. "Go to region A and reduce the variance of your state estimate below 0.1 m^2." Our algorithm generates a transition system in the belief space and uses local feedback controllers to break the *curse of history* associated with belief space planning. Furthermore, conventional automata-based methods become tractable. Switching control policies are then computed using a product Markov Decision Process (MDP) between the transition system and the Rabin automaton encoding the task specification. We present algorithms to translate a GDTL formula to a Rabin automaton and to efficiently construct the product MDP by leveraging recent results from incremental computing. Our approach is evaluated in hardware experiments using a camera network and ground robot.**

I. Introduction

In this work, we use sampling-based techniques to synthesize switched closed-loop control policies that are guaranteed to drive a dynamical system with observation noise while achieving high-level tasks given as temporal logic formulae. Significant observation and actuation noises are inherent in many engineering applications, such as robotics or power networks, in which control actions must be made in real time in response to uncertain or incomplete state and model information. Temporal logic formulae interleave Boolean logic and temporal operators with system properties to specify rich global behaviors. In the domain of robotics, an example of a task that can be encoded in temporal logic is "Periodically clean the living room and then the bathroom. Put the trash in the bin in the kitchen or outside. Go to a charging station after cleaning is complete. Always avoid the bedroom." In the absence of observation noise, tools from formal synthesis can be used to synthesize control policies that ensure these rich specifications are met [1]. On the other hand, modern control techniques can be used to synthesize controllers automatically to enforce properties such as "drive the state of the system to a safe set while avoiding unsafe states" under observation and dynamics noise [2]–[7]. In this work, we present an automatic, hierarchical control synthesis algorithm that extends tools from formal synthesis and stochastic control to enforce temporal logic specifications. Though our approach is quite general, we use examples from robotic navigation throughout the paper to motivate our approach. We evaluate our algorithm with experiments using a wheeled robot with noisy actuators localized by a noisy, static camera network performing a persistent navigation task.

While synthesizing control policies to enforce temporal logic properties under dynamics noise has been extensively considered in the literature [8], observation noise has only recently been considered [1], [9]–[12]. One of the technical challenges of incorporating observation noise into formal synthesis is that satisfaction of temporal logic properties is in general defined with respect to the state trajectory of the system rather than the evolution of the belief (as measured by a posterior probability distribution) about this state. In this paper, we introduce the paradigm of Gaussian distribution temporal logic (GDTL) which allows us to specify properties involving the uncertainty in the state of the system, e.g. "Ensure that the uncertainty (measured by variance) of the robot's x position is always below 0.1 m^2". GDTL formulae can be translated to Rabin automata using off-the-shelf tools [9].

The problem of synthesizing controllers to enforce a GDTL specification is in general a discrete time, continuous space partially observable Markov decision process (POMDP). Our approach approximates the optimal solution with a computationally feasible hierarchical sampling-based control synthesis algorithm. Most existing sampling-based algorithms sample points directly in belief space [13]–[16], which requires synthesizing distribution-to-distribution controllers. Such synthesis problems are computationally difficult and may require significant modeling on the part of a control designer. To circumvent these challenges, we base the core of our algorithm on feedback information roadmaps (FIRMs). The FIRM motion planner extends probabilistic roadmaps (PRMs) [17], to handle observation noise. In FIRM, points are sampled directly in the state space (rather than in belief space) and feedback control policies, e.g. linear quadratic Gaussian (LQG) controllers, stabilize the system about nodes along paths in the roadmap. The behavior of the closed-loop system is then used to predict how the state estimate evolves. The associated trajectories of the estimate induce a roadmap in the belief space.

If the goal of the problem were only to reach a given region of the belief space, one could construct a switched

*This work was supported by NSF NRI-1426907, NSF CMMI-1400167.

[1]Cristian-Ioan Vasile is with the Division of Systems Engineering, Boston University, 15 Saint Mary's Street, Brookline, MA 02446, USA cvasile@bu.edu

[2]Kevin Leahy, Eric Cristofalo and Calin Belta are with the Department of Mechanical Engineering, Boston University, 110 Cummington Mall, Boston, MA 02215, USA {kjleahy, emc73, cbelta}@bu.edu

[3]Austin Jones is with Mechanical Engineering and Electrical Engineering at Georgia Institute of Technology, North Ave NW, Atlanta, GA 30332, USA austinjones@gatech.edu

[4]Mac Schwager is with the Department of Aeronautics and Astronautics, Stanford University, Durand Building, 496 Lomita Mall, Stanford, CA 94305, USA schwager@stanford.edu

controller by finding a path in the roadmap from the initial distribution to a node contained within the goal set and then applying the corresponding sequence of controllers. During the application of the controller, however, we do not have any guarantees about whether or not the evolution of the system will violate the given specification. Therefore, we can only estimate with what probability the given controller drives the distribution to the next collection of nodes without violating the specification. This allows us to construct a Markov decision process in which the states correspond to nodes, actions correspond to controller pairs, and transition probabilities correspond to the probability of the closed-loop system reaching the next node without violating the specification. Applying dynamic programming to this system yields a policy that maps the current region of belief states to the pair of controllers to be applied. Combining the policy with the synthesized LQG controllers yields a state-switched feedback controller that satisfies the system specifications with some minimum probability.

Given a Rabin automaton constructed from a GDTL formula and a FIRM, we construct a graph product between the two, called the GDTL-FIRM, to check if the state space has been sampled sufficiently to synthesize a switched controller satisfying the specification with positive probability. We use techniques similar to those in sampling-based formal synthesis work [18]–[22] to construct the GDTL-FIRM incrementally until we find a policy with sufficiently high satisfaction probability.

II. Gaussian Distribution Temporal Logic

In this section, we define Gaussian Distribution Temporal Logic (GDTL), a predicate temporal logic defined over the space of Gaussian distributions with fixed dimension.

Notation: Let Σ be a finite set. The cardinality, power set, Kleene- and ω-closures of Σ are denoted by $|\Sigma|$, 2^{Σ}, Σ^* and Σ^ω, respectively. $A \subseteq \mathbb{R}^n$ and $B \subseteq \mathbb{R}^m$, $n, m \geq 0$, we denote by $\mathcal{M}(A, B)$ the set of functions with domain A and co-domain B, where A has positive measure with respect to the Lebesgue measure of $\mathbb{R}^n$. The set of all positive semi-definite matrices of size $n \times n$, $n \geq 1$, is denoted by S^n. $\mathbb{E}[\cdot]$ is the expectation operator. The $m \times n$ zero matrix and the $n \times n$ identity matrix are denoted by $\mathbf{0}_{m,n}$ and $\mathbf{I}_n$, respectively. The supremum and Euclidean norms are denoted by $\|\cdot\|_\infty$ and $\|\cdot\|_2$, respectively.

Let $\mathcal{G}$ denote the Gaussian belief space of dimension n, i.e. the space of Gaussian probability measures over $\mathbb{R}^n$. For brevity, we identify the Gaussian measures with their finite parametrization, mean and covariance matrix. Thus, $\mathcal{G} = \mathbb{R}^n \times S^n$. If $\mathbf{b} = b^0 b^1 \ldots \in \mathcal{G}^\omega$, we denote the suffix sequence $b^i b^{i+1} \ldots$ by $\mathbf{b}^i$, $i \geq 0$.

Definition 1 (GDTL Syntax)**.** The *syntax* of Gaussian Distribution Temporal Logic is defined as

$$\phi := \top \mid f \leq 0 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U} \phi_2,$$

where $\top$ is the Boolean constant "True", $f \leq 0$ is a predicate over $\mathcal{G}$, where $f \in \mathcal{M}(\mathcal{G}, \mathbb{R})$, $\neg$ is negation ("Not"), $\wedge$ is conjunction ("And"), and $\mathcal{U}$ is "Until".

For convenience, we define the additional operators: $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\Diamond\, \phi \equiv \top \mathcal{U} \phi$, and $\Box\, \phi \equiv \neg \Diamond \neg\phi$, where $\equiv$ denotes semantic equivalence.

Definition 2 (GDTL Semantics)**.** Let $\mathbf{b} = b^0 b^1 \ldots \in \mathcal{G}^\omega$ be an infinite sequence of belief states. The *semantics* of GDTL is defined recursively as

$$\begin{aligned}
&\mathbf{b}^i \models \top \\
&\mathbf{b}^i \models f \leq 0 \quad &\Leftrightarrow \quad & f(b^i) \leq 0 \\
&\mathbf{b}^i \models \neg\phi \quad &\Leftrightarrow \quad & \neg(\mathbf{b}^i \models \phi) \\
&\mathbf{b}^i \models \phi_1 \wedge \phi_2 \quad &\Leftrightarrow \quad & (\mathbf{b}^i \models \phi_1) \wedge (\mathbf{b}^i \models \phi_2) \\
&\mathbf{b}^i \models \phi_1 \vee \phi_2 \quad &\Leftrightarrow \quad & (\mathbf{b}^i \models \phi_1) \vee (\mathbf{b}^i \models \phi_2) \\
&\mathbf{b}^i \models \phi_1 \mathcal{U} \phi_2 \quad &\Leftrightarrow \quad & \exists j \geq i \text{ s.t. } (\mathbf{b}^j \models \phi_2) \\
& & & \wedge\, (\mathbf{b}^k \models \phi_1, \forall k \in \{i, \ldots j-1\}) \\
&\mathbf{b}^i \models \Diamond\, \phi \quad &\Leftrightarrow \quad & \exists j \geq i \text{ s.t. } \mathbf{b}^j \models \phi \\
&\mathbf{b}^i \models \Box\, \phi \quad &\Leftrightarrow \quad & \forall j \geq i \text{ s.t. } \mathbf{b}^j \models \phi
\end{aligned}$$

The word $\mathbf{b}$ satisfies ϕ, denoted $\mathbf{b} \models \phi$, if and only if $\mathbf{b}^0 \models \phi$.

By allowing the definition of the atomic predicates used in GDTL to be quite general, we can potentially enforce interesting and relevant properties on the evolution of a system through belief space. Some of these properties include

- Bounds on determinant of covariance matrix $det(P)$. This is used when we want to bound the overall uncertainty about the system's state.
- Bounds on trace of covariance matrix $Tr(P)$. This is used when we want to bound the uncertainty about the system's state in any direction.
- Bounds on state mean $\hat{x}$. This is used when we want to specify where in state space the system should be.

Example 1. Let R be a system evolving along a straight line with state denoted by $x \in \mathbb{R}$. The belief space for this particular robot is thus $(\hat{x}, P) \in \mathbb{R} \times [0, \infty)$, where $\hat{x}$ and P are its state estimate and covariance obtained from its sensors. The system is tasked with going back and forth between two goal regions (denoted as $\pi_{g,1}$ and $\pi_{g,2}$ in the top of Fig. 1). It also must ensure that it never overshoots the goal regions or lands in obstacle regions $\pi_{o,1}$ and $\pi_{o,2}$. The system must also maintain a covariance P of less than 0.5 m^2 at all times and less than 0.3 m^2 when in one of the goal regions. These requirements can be described by the GDTL formula

$$\begin{aligned}
\phi_{1d} \quad &= \quad \phi_{avoid} \wedge \phi_{reach} \wedge \phi_{u,1} \wedge \phi_{u,2}\,, \text{ where} \\
\phi_{avoid} \quad &= \quad \Box\, \neg((box(\hat{x}, -4, 0.35) \leq 1) \\
& \qquad \vee (box(\hat{x}, 4, 0.35) \leq 1)) \\
\phi_{reach} \quad &= \quad \Box\ \Diamond\, (box(\hat{x}, -2, 0.35) \leq 1) \\
& \qquad \wedge \Box\ \Diamond\, (box(\hat{x}, 2, 0.35) \leq 1) \\
\phi_{u,1} \quad &= \quad \Box\, (P < 0.5) \\
\phi_{u,2} \quad &= \quad \Box\, ((box(\hat{x}, -2, 0.35) \leq 1) \\
& \qquad \wedge (box(\hat{x}, 2, 0.35) \leq 1)) \Rightarrow (P < 0.3)\,,
\end{aligned} \tag{1}$$

where $box\left(\hat{x}, x_c, a\right) = \left\|a^T\left(\hat{x} - x_c\right)\right\|_\infty$ is a function bounding $\hat{x}$ inside an interval of size $2\left|a\right|$ centered at x_c. Subformula ϕ_{avoid} encodes keeping the system away from the obstacle regions. Subformula ϕ_{reach} encodes periodically visiting the goal regions. Subformula $\phi_{u,1}$ encodes maintaining the uncertainty below 0.5 m^2 globally and subformula $\phi_{u,2}$ encodes maintaining the uncertainty below 0.3 m^2 in the goal regions.

The belief space associated with this problem is shown in the bottom of Fig. 1. The curves in the figure correspond to the borders between the satisfaction and violation of predicates in (1), e.g. the level sets that are induced by the predicates when inequalities are replaced with equality. In the figure, + denotes that the predicate is satisfied in that region and - indicates that it is not. An example trajectory that satisfies (1) is shown in black. Note that every point in this belief trajectory has covariance P less than 0.5, which satisfies $\phi_{u,1}$. Further, the forbidden regions in ϕ_{avoid} (marked with red stripes) are always avoided while each of the goal regions in ϕ_{reach} (marked with green stars) are each visited. Further, whenever the belief is in a goal region, it has covariance P less than 0.3, which means $\phi_{u,2}$ is satisfied.

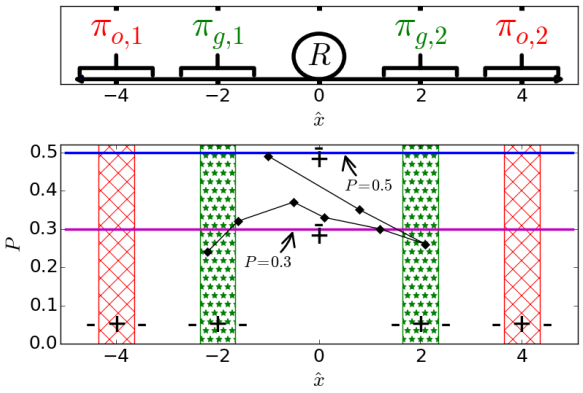


Fig. 1. (Top) The state space of a system evolving along one dimension and (Bottom) the predicates from (1) as functions of the belief of the system from Ex. 1.

III. Problem Formulation

In this section, we define the problem of controlling a system to satisfy a given GDTL formula with maximum probability.

A. Motion and sensing models

We assume the system has noisy linear time invariant (LTI) dynamics given by

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{2}$$

where $x_k \in \mathcal{X}$ is the state of the system, $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space, $A \in \mathbb{R}^{n\times n}$ is the dynamics matrix, $B \in \mathbb{R}^{n\times p}$ is the control matrix, $u_k \in \mathcal{U}$ is a control signal, $\mathcal{U} \subseteq \mathbb{R}^p$ is the control space, and w_k is a zero-mean Gaussian process with covariance $Q \in \mathbb{R}^{n\times n}$. The state is observed indirectly according to the linear observation model

$$y_k = Cx_k + v_k, \tag{3}$$

where $y_k \in \mathcal{Y}$ is a measurement, $\mathcal{Y} \subseteq \mathbb{R}^m$ is the observation space, $C \in \mathbb{R}^{m\times n}$ is the observation matrix and v_k is a zero-mean Gaussian process with covariance $R \in \mathbb{R}^{m\times m}$. We assume the LTI system (2), (3) is controllable and observable, i.e., (A, B) is a controllable pair and (A, C) is an observable pair. Moreover, we assume that C is full rank. These assumptions apply to many systems, including nonlinear systems that can be linearized to satisfy the assumptions.

The belief state at each time step is characterized by the *a posteriori* state and error covariance estimates, $\hat{x}_k$ and P_k, i.e., $b^k = (\hat{x}_k, P_k)$. The belief state is maintained via a Kalman filter [23], which we denote compactly as

$$b^{k+1} = \tau(b^k, u_k, y_{k+1}), \quad b^0 = (\hat{x}_0, P_0)\,, \tag{4}$$

where b^0 is the known initial belief about the system's state centered at $\hat{x}_0$ with covariance P_0. For a belief state $(x, P) \in \mathcal{G}$ we denote by $N_\delta(x, P) = \{b \in \mathcal{G} \mid \|b - (x, P)\|_{\mathcal{G}} \leq \delta\}$ the uncertainty ball of radius δ in the belief space centered at (x, P), where $\|\cdot\|_{\mathcal{G}}$ over $\mathcal{G}$ is a suitable norm in $\mathcal{G}$.

The robot model together with the Kalman filter may be interpreted as a POMDP [2], [24], [25].

B. Problem definition

Definition 3 (Policy)**.** A control policy for the system is a feedback function from the belief space $\mathcal{G}$ to the control space, e.g., $\mu : \mathcal{G} \to \mathcal{U}$. Denote the space of all policies by $\mathbb{M} = \mathcal{M}(\mathcal{G}, \mathcal{U})$.

We now introduce the main problem under consideration in this work:

Problem 1 (Maximum Probability Problem)**.** Let ϕ be a given GDTL formula and let the system evolve according to dynamics (2), with observation dynamics (3), and using a Kalman filter defined by (4). Find a policy μ^* such that

$$\begin{aligned} \mu^* &= \arg\max_{\mu\in\mathbb{M}} Pr[\mathbf{b} \models \phi] \\ &\text{subject to (2), (3), (4).} \end{aligned} \tag{5}$$

IV. Solution

In our approach, we use sampling-based techniques to generate paths throughout the state space. Local controllers drive the systems along these paths and stabilize at key points. The closed-loop behavior of the system induces paths in the belief space. The FIRM describes the stochastic process that generates these paths. We build an MDP by combing the FIRM with a Rabin automaton which then allows us to check if sample paths satisfy a GDTL formula. We compute transition probabilities and intersection probabilities (probability of intersecting a good or bad set from the Rabin automaton's acceptance condition) for each edge in this structure. We use dynamic programming to find the policy in this structure that maximizes the probability of satisfying the formula. The resulting policy can then be translated to

a non-stationary switched local controller that approximates the solution to Pb. 1. An important property of the proposed solution is that all operations are incremental with respect to the size of the FIRM. Note that the proposed solution may be applied to nonlinear systems whose linearizations around random samples in the state space satisfy the assumptions in Sec. III-A. The details of our solution Alg. 1 are presented below.

A. Sampling-based algorithm

We propose a sampling-based algorithm to solve Pb. 1 that overcomes the curse of dimension and history generally associated with POMDPs. In short, a sampling-based algorithm iteratively grows a graph $\mathcal{T}$ in the state space, where nodes are individual states, and edges correspond to motion primitives that drive the system from state to state [26]. The extension procedure is biased towards exploration of uncovered regions of the state space. Similar to [18], we adapt sampling-based methods to produce finite abstractions (e.g., graphs) of the belief space. Alg. 1 incrementally constructs a transition system $\mathcal{T} = (\mathfrak{B}_\mathcal{T}, B_0, \Delta_\mathcal{T}, \mathcal{C}_\mathcal{T})$, where the state space $\mathfrak{B}_\mathcal{T}$ is composed of belief nodes, i.e., bounded hyperballs in $\mathcal{G}$, $\Delta_\mathcal{T}$ is the set of transitions, and $\mathcal{C}_\mathcal{T}$ is a set of controllers associated with edges. The center of a belief node is a belief state $b = (x, P^\infty)$, where the mean x is obtained through random sampling of the system's state space, and P^∞ is the stationary covariance. The initial belief node is denoted by B_0.

Sampling-based algorithms are built using a set of primitive functions that are assumed to be available:

- $sample(\mathcal{X})$ generates random states from a distribution over the state space $\mathcal{X}$,
- $nearest(x^r, \mathcal{T}) = \arg\min_{x^u}\{\|x^r - x^u\|_2 \mid \exists P^u \wedge N_\delta(x^u, P^u) \in \mathfrak{B}_\mathcal{T}\}$ returns the mean x^u of a belief node's center in $\mathcal{T}$ such that x^u is closest to the state x^r using the metric defined on $\mathcal{X}$,
- $near(B_n, \mathfrak{B}_\mathcal{T}, \gamma)$ returns the closest γ belief nodes in $\mathfrak{B}_\mathcal{T}$ to B_n with respect to the distance between their centers induced by $\|\cdot\|_\mathcal{G}$, and
- $steer(x^i, x^t)$ returns a state obtained by attempting to drive the system from x^i towards x^t.

Using these primitive functions, an extension procedure $extend(\mathcal{X}, \mathcal{T})$ of the transition system $\mathcal{T}$ can be defined as:

1) generate a new sample $x^r \leftarrow sample(\mathcal{X})$,
2) find nearest state $x^u \leftarrow nearest(x^r, \mathcal{T})$, and
3) drive the system towards the random sample $x^n \leftarrow steer(x^u, x^r)$.

For more details about sampling-based algorithms, primitive functions and their implementations see [21], [26], [27].

Transitions are enforced using local controllers which are stored in $\mathcal{C}_\mathcal{T}$. i.e., we assign to each edge $e \in \Delta_\mathcal{T}$ a local controller $ec_e \in \mathcal{C}_\mathcal{T}$. Under the assumptions of our model [18], the local controllers are guaranteed to stabilize the system to belief nodes along a path in finite time. Thus we abstract the roadmap to a deterministic system. In Alg. 1, local controllers are generated using the method $localController()$. The design of the node controllers is presented Sec. V.

The algorithm checks for the presence of a satisfying path using a deterministic Rabin automaton (DRA) $\mathcal{R}$ that is computed from the GDTL specification using an intermediate linear temporal logic (LTL) construction [9]. There exist efficient algorithms that translate LTL formulae into Rabin automata [28]. We denote the set of predicates in GDTL formula ϕ as F_ϕ.

Definition 4 (Rabin Automaton)**.** A (deterministic) Rabin automaton is a tuple $\mathcal{R} = (S_\mathcal{R}, s_0^\mathcal{R}, \Sigma, \delta, \Omega_\mathcal{R})$, where $S_\mathcal{R}$ is a finite set of states, $s_0^\mathcal{R} \in S_\mathcal{R}$ is the initial state, $\Sigma \subseteq 2^{F_\phi}$ is the input alphabet, $\delta : S_\mathcal{R} \times \Sigma \to S_\mathcal{R}$ is the transition function, and $\Omega_\mathcal{R}$ is a set of tuples $(\mathcal{F}_i, \mathcal{B}_i)$ of disjoint subsets of $S_\mathcal{R}$ which correspond to good ($\mathcal{F}_i$) and bad ($\mathcal{B}_i$) states.

A transition $s' = \delta(s, \sigma)$ is also denoted by $s \xrightarrow{\sigma}_\mathcal{R} s'$. A trajectory of the Rabin automaton $\mathbf{s} = s_0 s_1 \ldots$ is generated by an infinite sequence of symbols $\boldsymbol{\sigma} = \sigma_0 \sigma_1 \ldots$ if $s_0 = s_0^\mathcal{R}$ is the initial state of $\mathcal{R}$ and $s_k \xrightarrow{\sigma_k}_\mathcal{R} s_{k+1}$ for all $k \geq 0$. Given a state trajectory $\mathbf{s}$ we define $\vartheta_\infty(\mathbf{s}) \subseteq S_\mathcal{R}$ as the set of states which appear infinitely many times in $\mathbf{s}$. An infinite input sequence over Σ is said to be accepted by a Rabin automaton $\mathcal{R}$ if there exists a tuple $(\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R}$ of good and bad states such that the state trajectory $\mathbf{s}$ of $\mathcal{R}$ generated by $\boldsymbol{\sigma}$ intersects the set $\mathcal{F}_i$ infinitely many times and the set $\mathcal{B}_i$ only finitely many times. Formally, this means that $\vartheta_\infty(\mathbf{s}) \cap \mathcal{F}_i \neq \emptyset$ and $\vartheta_\infty(\mathbf{s}) \cap \mathcal{B}_i = \emptyset$.

B. Computing transition and intersection probability

Given a transition $e = (B_u, B_v)$ and its associated local controller ec_e, Alg. 2 computes the transition distribution from an initial DRA state s_u to a some random DRA state, and a set of intersection distributions associated with each pair $(\mathcal{F}_i, \mathcal{B}_i)$ of the acceptance set of $\mathcal{R}$. These distributions are hard to compute analytically. Therefore, we estimate them from sample trajectories of the closed-loop system enforcing edge e. In Alg. 2, the function $sampleBeliefSet(S)$ returns a random sample from a uniform distribution over the belief set S.

The distribution $\pi^{S_\mathcal{R}}$ captures the probability that s_v is the state of $\mathcal{R}$ at the end of closed-loop trajectory generated by controller ec_e to steer the system from belief node B_u and DRA state s_u to belief node B_v: $\pi^{S_\mathcal{R}} = Pr[s_v \mid e, s_u, ec_e]$, where $s_v \in S_\mathcal{R}$, $s_u \xrightarrow{\sigma_{0:T-1}} s_v$, $b^{0:T} = ec_e(b_u)$, $b_u \in B_u$, and $\sigma_k \leftarrow \{f \mid f(b^k) \leq 0, \forall f \in F_\phi\}$.

Each intersection distribution represents the probability that edge e intersects $\mathcal{F}_i$, $\mathcal{B}_i$ or neither, where $(\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R}$, and the controller ec_e was used to drive the system along the edge e starting from the DRA state s_u:

$$\pi^{\Omega_\mathcal{R}} = \left\{ \begin{Bmatrix} Pr[\mathbf{s} \cap \mathcal{F}_i \mid e, s_u, ec_e] \\ Pr[\mathbf{s} \cap \mathcal{B}_i \mid e, s_u, ec_e] \\ Pr[\mathbf{s} \cap (\mathcal{F}_i \cup \mathcal{B}_i) \mid e, s_u, ec_e] \end{Bmatrix} \;\middle|\; \forall (\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R} \right\} \quad (6)$$

For convenience, we use the following notation $\pi^{\Omega_\mathcal{R}}(e, X) = Pr[\mathbf{s} \cap X \mid e, s_u, ec_e]$, where $X \in \{\mathcal{F}_i, \mathcal{B}_i, \mathcal{F}_i \cup \mathcal{B}_i\}$.

Algorithm 1: $ConstructTS(x_0, \phi, \varepsilon)$

Input: initial state x^0, GDTL specification ϕ, and lower bound ε

Output: belief transition system $\mathcal{T}$, product MDP $\mathcal{P}$, and satisfying policy μ^*

convert GDTL formula ϕ to LTL formula φ over the set of atomic propositions $AP = F_\phi$
compute DRA $\mathcal{R} = (S_\mathcal{R}, s_0^\mathcal{R}, 2^{AP}, \delta, \Omega_\mathcal{R})$ from φ
$ec_0, P_0^\infty \leftarrow localController(x^0)$
$B_0 \leftarrow N_\delta(x^0, P_0^\infty)$
$e_0 = (B_0, B_0)$
$\pi_0^{S_\mathcal{R}}, \pi_0^{\Omega_\mathcal{R}} \leftarrow computeProb(e_0, s_0, ec_0, \mathcal{R})$
initialize belief TS $\mathcal{T} = (\mathfrak{B}_\mathcal{T} = \{B_0\}, B_0, \Delta_\mathcal{T} = \{e_0\}, \mathcal{C}_\mathcal{T} = \{(e_0, ec_0)\})$
construct product MDP $\mathcal{P} = \mathcal{T} \times \mathcal{R} = (S_\mathcal{P} = \mathfrak{B}_\mathcal{T} \times S_\mathcal{R}, (B_0, s_0), Act = \mathfrak{B}_\mathcal{T}, \delta_\mathcal{P} = \{\pi_0^{S_\mathcal{R}}\}, \Omega_\mathcal{P} = \{\pi_0^{\Omega_\mathcal{R}}\})$
for $index = 1$ **to** N **do**
  $x^n \leftarrow extend(\mathcal{X}, \mathcal{T})$
  $ec_n, P_n^\infty \leftarrow localController(x^n)$
  $B_n \leftarrow N_\delta(x^n, P_n^\infty)$
  $\mathcal{N}_n \leftarrow near(B_n, \mathfrak{B}_\mathcal{T}, \gamma)$
  $\Delta_n \leftarrow \{(B_i, B_n) | x^n = steer(x^i, x^n), B_i \in \mathcal{N}_n\}$
    $\cup \{(B_n, B_i) | x^i = steer(x^n, x^i), B_i \in \mathcal{N}_n\}$
  $\mathfrak{B}_\mathcal{T} \leftarrow \mathfrak{B}_\mathcal{T} \cup \{B_n\}$, $\Delta_\mathcal{T} \leftarrow \Delta_\mathcal{T} \cup \Delta_n$
  $S_\mathcal{P} \leftarrow S_\mathcal{P} \cup (\{B_n\} \times S_\mathcal{R})$
  foreach $e = (B_u, B_v) \in \Delta_n$ **do**
    $\mathcal{C}_\mathcal{T} \leftarrow \mathcal{C}_\mathcal{T} \cup \{(e, ec_v)\}$
    foreach $s_u \in S_\mathcal{R}$ *s.t.* $(B_u, s_u) \in S_\mathcal{P}$ **do**
      $\pi_e^{S_\mathcal{R}}, \pi_e^{\Omega_\mathcal{R}} \leftarrow computeProb(e, s_u, ec_v, \mathcal{R})$
      $\delta_\mathcal{P} \leftarrow \delta_\mathcal{P} \cup \{\pi_e^{S_\mathcal{R}}\}$
      $\Omega_\mathcal{P} \leftarrow \Omega_\mathcal{P} \cup \{\pi_e^{\Omega_\mathcal{R}}\}$
  $\Delta_\mathcal{P}^n = \{(p, p') \in \Delta_\mathcal{P} \,|\, (p, p')\lfloor_\mathcal{T} \in \Delta_n\}$
  foreach $(\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R}$ **do** `// update ECs`
    $\Gamma_i = \{(p, p') \in \Delta_\mathcal{P}^n \,|\, \pi^{\Omega_\mathcal{R}}(e, \mathcal{F}_i) = 0 \wedge \pi^{\Omega_\mathcal{R}}(e, \mathcal{B}_i) > 0, e = (p, p')\lfloor_\mathcal{T}\}$
    $c_i.update(\Delta_\mathcal{P}^n \setminus \Gamma_i)$
  if $existsSatPolicy(\mathcal{P})$ **then**
    solve DP (7) and compute policy μ^* with probability of satisfaction p
    if $p \geq \varepsilon$ **then return** $(\mathcal{T}, \mathcal{P}, \mu^*)$
return $(\mathcal{T}, \mathcal{P}, \emptyset)$

Algorithm 2: $computeProb(e = (B_u, B_v), s_u, ec_e, \mathcal{R})$

Input: transition between belief nodes $e = (B_u, B_v)$, starting DRA state s_u, controller enforcing e ec_e, and deterministic Rabin automaton $\mathcal{R}$

Output: transition distribution $\pi^{S_\mathcal{R}}$, and intersection distribution $\pi^{\Omega_\mathcal{R}}$

Parameter: NP – number of particles

$t \leftarrow \mathbf{0}_{|S_\mathcal{R}|,1}$
$ra_i \leftarrow \mathbf{0}_{3,1}$, $\forall (\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R}$
for $p = 1 : NP$ **do**
  $b_u \leftarrow sampleBeliefSet(B_u)$
  $b^{0:T} \leftarrow ec_e(b_u)$
  for $k = 0$ **to** $T - 1$ **do**
    $\sigma_k \leftarrow \{f \,|\, f(b^k) \leq 0, \forall f \in F_\phi\}$
  $\mathbf{s} = s_{0:T} \leftarrow (s_u \overset{\sigma_{0:T-1}}{\rightarrow} s_T)$
  $t[s_T] \leftarrow t[s_T] + 1$
  for $(\mathcal{F}_i, \mathcal{B}_i) \in |\Omega_\mathcal{R}|$ **do**
    if $\mathcal{F}_i \cap \mathbf{s} \neq \emptyset$ **then** $ra_i[1] \leftarrow ra_i[1] + 1$
    if $\mathcal{B}_i \cap \mathbf{s} \neq \emptyset$ **then** $ra_i[2] \leftarrow ra_i[2] + 1$
    if $(\mathcal{F}_i \cup \mathcal{B}_i) \cap \mathbf{s} = \emptyset$ **then** $ra_i[3] \leftarrow ra_i[3] + 1$
return $\left(\pi^{S_\mathcal{R}} = \frac{t}{NP}, \pi^{\Omega_\mathcal{R}} = \left\{\frac{ra_i}{NP} \,|\, 1 \leq i \leq |\Omega_\mathcal{R}|\right\}\right)$

C. GDTL-FIRM Product MDP

In this section, we define a construction procedure of the product MDP between the (belief) TS $\mathcal{T}$ and the specification DRA $\mathcal{R}$.

Definition 5 (GDTL-FIRM MDP)**.** Given a DTS $\mathcal{T} = (\mathfrak{B}_\mathcal{T}, B_0, \Delta_\mathcal{T}, \mathcal{C}_\mathcal{T})$, a Rabin automaton $\mathcal{R} = (S_\mathcal{R}, s_0^\mathcal{R}, \Sigma = 2^{AP}, \delta, \Omega_\mathcal{R})$, and the transition and intersection probabilities $\pi^{S_\mathcal{R}}$, $\pi^{\Omega_\mathcal{R}}$, their product MDP, denoted by $\mathcal{P} = \mathcal{T} \times \mathcal{R}$, is a tuple $\mathcal{P} = (S_\mathcal{P}, s_0^\mathcal{P}, Act, \delta_\mathcal{P}, \Omega_\mathcal{P})$ where $s_0^\mathcal{P} = (B_0, s_0^\mathcal{R})$ is the initial state; $S_\mathcal{P} \subseteq \mathfrak{B}_\mathcal{T} \times S_\mathcal{R}$ is a finite set of states which are reachable from the initial state by run of positive probability (see below); $Act = \mathfrak{B}_\mathcal{T}$ is the set of actions available at each state; $\delta_\mathcal{P} : S_\mathcal{P} \times Act \times S_\mathcal{P} \to [0, 1]$ is the transition probability defined by $\delta_\mathcal{P}((B_i, s_i), B_j, (B_j, s_j)) = \pi^{S_\mathcal{R}}(s_j; e_{ij}, s_i, \mathcal{C}_\mathcal{T}(e_{ij}))$, $e_{ij} = (B_i, B_j)$; and $\Omega_\mathcal{P}$ is the set of tuples of good and bad transitions in the product automaton.

Denote the set of edges of positive probability by $\Delta_\mathcal{P} = \left\{\left((B_i, s_i), (B_j, s_j)\right) \,|\, \delta_\mathcal{P}((B_i, s_i), B_j, (B_j, s_j)) > 0\right\}$. A transition in $\mathcal{P}$ is also denoted by $p_i \to_\mathcal{P} p_j$ if $(p_i, p_j) \in \Delta_\mathcal{P}$. A trajectory (or run) of *positive probability* of $\mathcal{P}$ is an infinite sequence $\mathbf{p} = p_0 p_1 \ldots$, where $p_0 = s_0^\mathcal{P}$ and $p_k \to_\mathcal{P} p_{k+1}$ for all $k \geq 0$.

The acceptance condition for a trajectory of $\mathcal{P}$ is encoded in $\Omega_\mathcal{P}$, and is induced by the acceptance condition of $\mathcal{R}$. Formally, $\Omega_\mathcal{P}$ is a set of pairs $(\mathcal{F}_i^\mathcal{P}, \mathcal{B}_i^\mathcal{P})$, where $\mathcal{F}_i^\mathcal{P} = \{e \in \Delta_\mathcal{P} \,|\, \pi^{\Omega_\mathcal{R}}(e, \mathcal{F}_i) > 0\}$, $\mathcal{B}_i^\mathcal{P} = \{e \in \Delta_\mathcal{P} \,|\, \pi^{\Omega_\mathcal{R}}(e, \mathcal{B}_i) > 0\}$, and $(\mathcal{F}_i, \mathcal{B}_i) \in \Omega_\mathcal{R}$.

A trajectory of $\mathcal{P} = \mathcal{T} \times \mathcal{R}$ is said to be accepting if and only if there is a tuple $(\mathcal{F}_i^\mathcal{P}, \mathcal{B}_i^\mathcal{P}) \in \Omega_\mathcal{P}$ such that the trajectory intersects the sets $\mathcal{F}_i^\mathcal{P}$ and $\mathcal{B}_i^\mathcal{P}$ infinitely and finitely many times, respectively. It follows by construction that a trajectory $\mathbf{p} = (B_0, s_0)(B_1, s_1) \ldots$ of $\mathcal{P}$ is accepting if and only if the trajectory $\mathbf{s}_{0:T_0-1}^0 \mathbf{s}_{0:T_1-1}^1 \ldots$ is accepting in $\mathcal{R}$, where $\mathbf{s}_{0:T_i}^i$ is the random trajectory of $\mathcal{R}$ obtained by traversing the transition $e = (B_i, B_{i+1})$ using the controller $\mathcal{C}_\mathcal{T}(e)$ and $s_0^i = s_i$ for all $i \geq 0$. Note that $\mathbf{s}_{T_i}^i = \mathbf{s}_0^{i+1}$. As a result, a trajectory of $\mathcal{T}$ obtained from an accepting trajectory of $\mathcal{P}$ satisfies the given specification encoded by $\mathcal{R}$ with

positive probability. We denote the projection of a trajectory $\mathbf{p} = (B_0, s_0)(B_1, s_1)\ldots$ onto $\mathcal{T}$ by $\mathbf{p}\lfloor_{\mathcal{T}} = B_0 B_1 \ldots$. A similar notation is used for projections of finite trajectories.

Remark 1. Note that the product MDP in Def. 5 is defined to be amenable to incremental operations with respect to the growth of the DTS, i.e., updating and checking for a solution of positive probability. This property is achieved by requiring the states of $\mathcal{P}$ to be reachable by transitions in $\Delta_{\mathcal{P}}$. The incremental update can be performed using a recursive procedure similar to the one described in [21].

Remark 2. The acceptance condition for $\mathcal{P}$ is defined by its transitions and not in the usual way in terms of its states, due to the stochastic nature of transitions between belief nodes in $\mathcal{T}$. We only record the initial and end DRA states of the DRA trajectories induced by the sample paths obtained using the local controllers.

D. Finding satisfying policies

The existence of a satisfying policy with positive probability can be checked efficiently on the product MDP $\mathcal{P}$ by maintaining end components EC[1] for induced subgraphs of $\mathcal{P}$ determined by the pairs in the acceptance condition $\Omega_{\mathcal{P}}$. For each pair $\mathcal{F}_i^{\mathcal{P}}, \mathcal{B}_i^{\mathcal{P}}$, let c_i denote the ECs associated with the graphs $G_i^{\mathcal{P}} = (S_{\mathcal{P}}, \Delta_{\mathcal{P}} \setminus \Gamma_i)$, where $\Gamma_i = \{(p, p') \in \Delta_{\mathcal{P}} \,|\, \pi^{\Omega_{\mathcal{R}}}(e, \mathcal{F}_i) = 0 \wedge \pi^{\Omega_{\mathcal{R}}}(e, \mathcal{B}_i) > 0, e = (p, p')\lfloor_{\mathcal{T}}\}$. Given c_i, checking for a satisfying trajectory in procedure $existsSatPolicy(\mathcal{P})$ becomes trivial. We test if there exists an EC that contains a transition (p, p') such that $\pi^{\Omega_{\mathcal{R}}}(e, \mathcal{F}_i) > 0$, where $e = (p, p')\lfloor_{\mathcal{T}}$. Note that we do not need to maintain $\Omega_{\mathcal{P}}$ explicitly, we only need to maintain the c_i. Efficient incremental algorithms to maintain these ECs were proposed in [29].

E. Dynamic program for Maximum Probability Policy

Given a GDTL-FIRM MDP, we can compute the optimal switching policy to maximize the probability that the given formula ϕ is satisfied. In other words, we find a policy that maximizes the probability of visiting the states in $\mathcal{F}_i$ infinitely often and avoiding $\mathcal{B}_i$. To find this policy, we first decompose $\mathcal{P}$ into a set of end components and find the accepting components. Since any sample path that satisfies ϕ must end in an accepting component, maximizing the probability of satisfying ϕ is equivalent to maximizing the probability of reaching such a component. The optimal policy is thus given by the relationship

$$\begin{aligned} J^{\infty}(s) &= \begin{cases} 1, & s \in c_i \\ \max\limits_{a \in Act(s)} \sum_{s'} \delta(s, a, s') J^{\infty}(s') & \text{else} \end{cases} \\ m(s) &= \arg\max_{a \in Act(s)} \sum_{s'} \delta(s, a, s') J^{\infty}(s') \end{aligned} \tag{7}$$

This can be solved by a variety of methods, including approximate value iteration and linear programming [23].

[1]An EC of an MDP is a sub-MDP such that there exists a policy such that each node in the EC can be reached from each other node in the EC with positive probability.

F. Complexity

The overall complexity of maintaining the ECs used for checking for satisfying runs in $\mathcal{P}$ is $O(|\Omega_{\mathcal{R}}|\,|S_{\mathcal{P}}|^{\frac{3}{2}})$. The complexity bound is obtained using the algorithm described in [29] and is better by a polynomial factor $|S_{\mathcal{P}}|^{\frac{1}{2}}$ than computing the ECs at each step using a linear algorithm. Thus, checking for the existence of a satisfying run of positive probability can be done in $O(|\Omega_{\mathcal{R}}|)$ time. The dynamic programming algorithm is polynomial in $|S_{\mathcal{P}}|$ [30].

V. Case Studies

In this section, we apply our algorithm to control a unicycle robot moving in a bounded planar environment. To deal with the non-linear nature of the robot model, we locally approximate the robot's dynamics using LTI systems with Gaussian noise around samples in the workspace. This heuristic is very common, since the non-linear and non-Gaussian cases yield recursive filters that do not in general admit finite parametrization. Moreover, the control policy is constrained to satisfy a rich temporal specification. The proposed sampling-based solution opvercomes these difficulties due to its randomized and incremental nature. As the size of the GDTL-FIRM increases, we expect the algorithm to return a policy, if one exists, with increasing satisfaction probability. Since it is very difficult to obtain analytical bounds on the satisfaction probability, we demonstrate the performance of our solution in experimental trials.

1) Motion model: The motion model for our system is a unicycle. We discretize the system dynamics using Euler's approximation. The motion model becomes:

$$x_{k+1} = f(x_k, u_k, w_k) = x_k + \begin{bmatrix} \cos(\theta_k) & 0 \\ \sin(\theta_k) & 0 \\ 0 & 1 \end{bmatrix} \cdot u_k + w_k \tag{8}$$

where $x_k = \begin{bmatrix} p_k^x & p_k^y & \theta_k \end{bmatrix}^T$, p_k^x, p_k^y and θ_k are the position and orientation of the robot in a global reference frame, $u_k = \begin{bmatrix} v_k' & \omega_k' \end{bmatrix}^T = \Delta t \begin{bmatrix} v_k & \omega_k \end{bmatrix}^T$, v_k and ω_k are the linear and rotation velocities of the robot, Δt is the discretization step, and w_k is a zero-mean Gaussian process with covariance matrix $Q \in \mathbb{R}^{3\times 3}$. Next, we linearize the system around a nominal operating point (x^d, u^d) without noise,

$$x_{k+1} = f(x^d, u^d, 0) + A\,(x_k - x^d) + B\,(u_k - u^d) + w_k, \tag{9}$$

where $A = \frac{\partial f}{\partial x_k}(x^d, u^d, 0)$ and $B = \frac{\partial f}{\partial u_k}(x^d, u^d, 0)$ are the process and control Jacobians, $x^d = \begin{bmatrix} p^{x\,d} & p^{y\,d} & \theta^d \end{bmatrix}^T$, and $u^d = \begin{bmatrix} v_k'^d & \omega_k'^d \end{bmatrix}^T$.

In our framework, we associate with each belief node $B_g \in \mathfrak{B}_{\mathcal{T}}$ centered at $(\hat{x}^g, P)$ an LTI system obtained by linearization (9) about $(\hat{x}^g, u^g)$, where $u^g = [0.1,\, 0]^T$ corresponds to 0.1 m/s linear velocity and 0 angular velocity.

2) Observation Model: We localize the robot with a multiple camera network. This reflects the real world constraints of sensor networks, e.g. finite coverage, finite resolution, and improved accuracy with the addition of more sensors. The network was implemented using four TRENDnet Internet Protocol (IP) cameras with known pose with respect to the

global coordinate frame of the experimental space. Each 640×400 RGB image is acquired and segmented, yielding multiple pixel locations that correspond to a known pattern on the robot. The estimation of the planar position and orientation of the robot in the global frame is formulated as a least squares problem (*structure from motion*) [31]. The measurement, $y_k \in \mathcal{Y}$, is given by the discrete observation model: $y_k = Cx_k + v_k$. The measurement error covariance matrix is defined as $R = \text{diag}(r_x, r_y, r_\theta)$, where the value of each scalar is inversely proportional to the number of cameras used in the estimation, i.e. the number of camera views that identify the robot. These values are generated from a camera coverage map (Fig. 2(b)) of the experimental space.

3) Specification: The specification is given over belief states associated with the measurement y of the robot as follows: "Visit regions A and B infinitely many times. If region A is visited, then only corridor D_1 may be used to cross to the right side of the environment. Similarly, if region B is visited, then only corridor D_2 may be used to cross to the left side of the environment. The obstacle Obs in the center must always be avoided. The uncertainty must always be less than 0.9. When passing through the corridors D_1 and D_2 the uncertainty must be at most 0.6."

The corresponding GDTL formula is:

$$\begin{aligned}
\phi_1 &= \phi_{avoid} \wedge \phi_{reach} \wedge \phi_{u,1} \wedge \phi_{u,2} \wedge \phi_{bounds} \qquad (10)\\
\phi_{avoid} &= \square\, \neg\phi_{Obs}\\
\phi_{reach} &= \square\, \big(\, \diamondsuit\, (\phi_A \wedge \neg\phi_{D_2} \mathcal{U} \phi_B) \diamondsuit\, (\phi_B \wedge \neg\phi_{D_1} \mathcal{U} \phi_A)\big)\\
\phi_{u,1} &= \square\, (tr(P) \leq 0.9)\\
\phi_{u,2} &= \square\, \big((\phi_{D_1} \vee \phi_{D_2}) \Rightarrow (tr(P) \leq 0.6)\big)\\
\phi_{bounds} &= \square\, (box(\hat{x}, x_c, a) \leq 1),
\end{aligned}$$

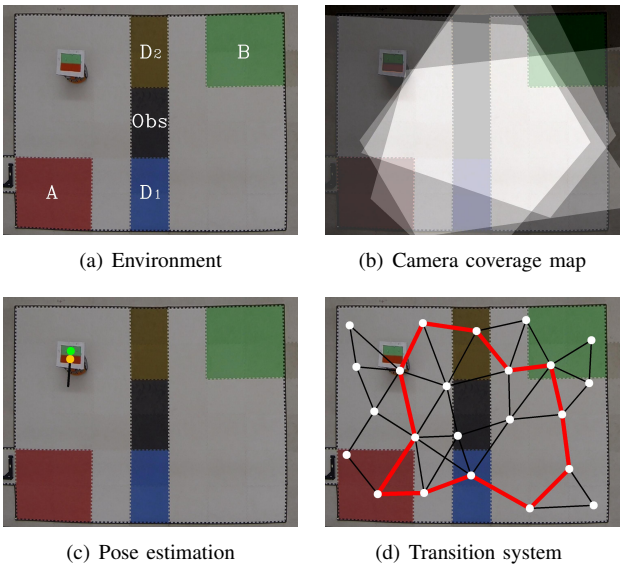
where $(\hat{x}, P)$ is a belief state associated with y, $a = \begin{bmatrix} \frac{2}{l} & \frac{2}{w} & 0 \end{bmatrix}$ so that $\hat{x}$ must remain within a rectangular $l \times w$ region with center $x_c = \begin{bmatrix} \frac{l}{2} & \frac{w}{2} & 0 \end{bmatrix}$, $l = 4.13\,m$ and $w = 3.54\,m$. The 5 regions in the environment are defined by GDTL predicate formulae $\phi_{Reg} = (box(\hat{x}, x_{Reg}, r_{Reg}) \leq 1)$, where x_{Reg} and r_{Reg} are the center and the dimensions of region $Reg \in \{A, B, D_1, D_2, Obs\}$, respectively.

4) Local controllers: We used the following simple switching controller to drive the robot towards belief nodes:

$$u_{k+1} = \begin{cases} \begin{bmatrix} k_D \left\| \alpha^T (x^g - \hat{x}_k) \right\|_2 & k_\theta (\theta_k^{los} - \hat{\theta}_k) \end{bmatrix}^T & \text{if } \left| \theta_k^{los} - \hat{\theta}_k \right| < \frac{\pi}{12} \\ \begin{bmatrix} 0 & k_\theta (\theta_k^{los} - \hat{\theta}_k) \end{bmatrix}^T, & \text{otherwise} \end{cases},$$

where $k_D > 0$ and $k_\theta > 0$ are proportional scalar gains, x^g is the goal position, θ_k^{los} is the line-of-sight angle and $\alpha = [1\ 1\ 0]^T$. We assume, as in [18], that the controller is able to stabilize the system state and uncertainty around the goal belief state (x^g, P^∞), where P^∞ is the stationary covariance matrix.

5) Experiments: The algorithms in this paper were implemented in Python2.7 using *LOMAP* [32] and *networkx* [33] libraries. The *ltl2star* tool [28] was used to convert the LTL specification into a Rabin automaton. All computation was performed on a Ubuntu 14.04 machine with an Intel Core i7 CPU at 2.4 Ghz and 8GB RAM.



(a) Environment (b) Camera coverage map

(c) Pose estimation (d) Transition system

Fig. 2. Fig. (a) shows an environment with two regions A and B, two corridors D_1 and D_2 and an obstacle Obs. Fig. (b) shows the coverage of the cameras. Fig. (c) shows the pose of the robot computed from the images taken by the 4 cameras. Fig. (d) shows the transition system computed by Alg. 1.

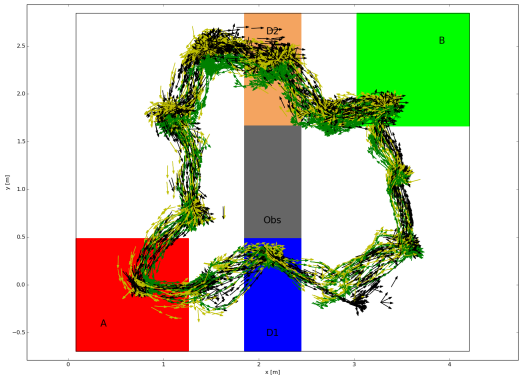


Fig. 3. The figure shows the trajectory of the robot over 10 surveillance cycles. At each time step, the pose of the robot is marked by an arrow. The true trajectory of the robot is shown in green. The trajectory obtained from the camera network is shown in yellow, while the trajectory estimated by the Kalman filter is shown in black.

A switched feedback policy was computed for the ground robot described by (8) operating in the environment shown in Fig. 2(a) with mission specification (10) using Alg. 1. The overall computation time to generate the policy was 32.739 seconds and generated a transition system and product MDP of sizes (23, 90) and (144, 538), respectively. The Rabin automaton obtained from the GDTL formula has 7 states and 23 transitions operating over a set of atomic propositions of size 8. The most computationally intensive operation in Alg. 1 is the computation of the transition and intersection probabilities. To speed up the execution, we generated trajectories for each transition of the TS and reused them whenever Alg. 2 is called for a transition of the product MDP. The mean execution time for the probability computation was 0.389 seconds for each transition of $\mathcal{T}$.

We executed the computed policy on the ground vehicle over 9 experimental trials for a total of 24 surveillance cycles. The specification was met in all of surveillance cycles. A trajectory of the ground robot over 10 surveillance cycles (continuous operation) is shown in Fig. 3.

VI. Conclusion

In this paper, we presented a sampling-based algorithm that generates feedback policies for stochastic systems with temporal and uncertainty constraints. The desired behavior of the system is specified using *Gaussian Distribution Temporal Logic* such that the generated policy satisfies the task specification with maximum probability. The proposed algorithm generates a transition system in the belief space of the system. A key step towards the scalability of the automata-based methods employed in the solution was breaking the *curse of history* for POMDPs. Local feedback controllers that drive the system within belief sets were employed to achieve history independence for paths in the transition system. Also contributing to the scalability of our solution is a construction procedure for an annotated product Markov Decision Process called GDTL-FIRM, where each transition is associated with a "failure probability". GDTL-FIRM captures both satisfaction and the stochastic behavior of the system. Switching feedback policies were computed over the product MDP. Lastly, we showed the performance of the computed policies in experimental trials with a ground robot tracked via camera network. The case study shows that properties specifying the temporal and stochastic behavior of systems can be expressed using GDTL and our algorithm is able to compute control policies that satisfy the specification with a given probability.

References

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

[2] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99 – 134, 1998.

[3] J. van den Berg, P. Abbeel, and K. Goldberg, "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[4] K. Hauser, "Randomized belief-space replanning in partially-observable continuous spaces," in *Algorithmic Foundations of Robotics IX*. Springer, 2011, pp. 193–209.

[5] A. Bachrach, S. Prentice, R. He, P. Henry, A. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[6] M. P. Vitus and C. J. Tomlin, "Closed-loop belief space planning for linear, gaussian systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 2152–2159.

[7] K. Lesser and M. Oishi, "Finite State Approximation for Verification of Partially Observable Stochastic Hybrid Systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. New York, NY, USA: ACM, 2015, pp. 159–168.

[8] M. Zamani, P. M. Esfahani, R. Majumdar, A. Abate, and J. Lygeros, "Symbolic control of stochastic systems via approximately bisimilar finite abstractions," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3135–3150, Dec 2014.

[9] A. Jones, M. Schwager, and C. Belta, "Distribution temporal logic: Combining correctness with quality of estimation," in *IEEE 52nd Conference on Decision and Control (CDC)*, 2013, pp. 4719–4724.

[10] K. Leahy, A. Jones, M. Schwager, and C. Belta, "Distributed information gathering policies under temporal logic constraints," in *54th IEEE Conference on Decision and Control*, 2015, pp. 6803–6808.

[11] M. Svorenova, I. Cerna, and C. Belta, "Optimal control of mdps with temporal logic constraints," in *IEEE 52nd Annual Conference on Decision and Control (CDC)*, 2013, pp. 3938–3943.

[12] A. M. Ayala, S. B. Andersson, and C. Belta, "Formal Synthesis of Control Policies for Continuous Time Markov Processes From Time-Bounded Temporal Logic Specifications," *IEEE Transactions on Automatic Control*, vol. 59, no. 9, pp. 2568–2573, Sept 2014.

[13] S. Patil, G. Kahn, M. Laskey, J. Schulman, K. Goldberg, and P. Abbeel, "Scaling up Gaussian Belief Space Planning Through Covariance-Free Trajectory Optimization and Automatic Differentiation," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 515–533.

[14] B. Burns and O. Brock, "Sampling-based motion planning with sensing uncertainty," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 3313–3318.

[15] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 723–730.

[16] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *The International Journal of Robotics Research*, 2009.

[17] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[18] A. Agha-mohammadi, S. Chakravorty, and N. Amato, "Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 268–304, 2014.

[19] S. Karaman and E. Frazzoli, "Sampling-based Motion Planning with Deterministic μ-Calculus Specifications," in *IEEE Conference on Decision and Control (CDC)*, Shanghai, China, December 2009.

[20] ——, "Sampling-based Optimal Motion Planning with Deterministic μ-Calculus Specifications," in *American Control Conference*, 2012.

[21] C. Vasile and C. Belta, "Sampling-Based Temporal Logic Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, 2013.

[22] ——, "Reactive Sampling-Based Temporal Logic Path Planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 2014.

[23] D. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2012.

[24] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[25] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[26] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[27] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

[28] J. Klein and C. Baier, "Experiments with deterministic ω-automata for formulas of linear temporal logic," in *Implementation and Application of Automata*. Springer, 2006, pp. 199–212.

[29] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. Tarjan, "Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance," *ACM Trans. Algorithms*, vol. 8, no. 1, pp. 1–33, 2012.

[30] C. Papadimitriou and J. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[31] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3D Vision: From Images to Geometric Models*. Springer, 2004.

[32] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints," *International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[33] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.