Towards Fast Safe Mission Planning

Debadeepta Dey
Microsoft Research
Redmond, WA 98052
dedey@microsoft.com

Dorsa Sadigh
University of California
Berkeley, CA
dsadigh@berkeley.edu

Ashish Kapoor
Microsoft Research
Redmond, WA 98052
akapoor@microsoft.com

I. Introduction

Robotic and cyber-physical systems are proliferating at a breakneck pace. A key technological hurdle is to ensure safety of such systems especially within proximity of humans. While there has been a push in identifying obstacles and unsafe situations via sensors and machine learned predictors [2, 1], the task of embedding such information to determine safe course while obeying rules-of-the-road is non-trivial [14]. Further, the uncertainty and noise in prediction together with near real-time requirements under bounded computation resources makes this problem very challenging [5].

This work proposes an architecture for fast, safe planning of autonomous missions. We build upon the recent work in Probabilistic Signal Temporal Logic (PrSTL) [18] that synthesizes provably safe controllers that take into account noisy sensors and associated uncertainty in learned classifier/regressor predictions. Currently, solution for PrSTL requires solving Mixed Integer Semi-Definite Programs (MISDPs), which quickly become infeasible to solve in reasonable time as the number of constraints grow. Further, PrSTL needs the description of the mission goal and the required safety invariants as logical formulations and often expressing such objectives and constraints for long horizons and complicated rules-of-the-road remain non-trivial at best.

We alleviate these problems by combining PrSTL with random sampling based planners. We propose using Rapidly-exploring Random Trees (RRT) [11] and associated variants like RRT* [9] to simplify computation by first efficiently sampling feasible points in the robot's configuration space and then generating trajectories by connecting them via safe control. Such fast sampling of the feasible trajectories effectively reduces the optimization from a MISDP to a sequence of Second Order Cone Programs (SOCP), which being convex, can be solved much more efficiently.

II. Background

There has been recent developments in synthesizing control policies, inspired from program verification and artificial intelligence, that satisfy temporal properties, disjunctions, conjunctions or negations of user-specified predicates. Several temporal logic specification languages have been developed and adapted for synthesizing controllers such as Linear Temporal Logic (LTL) [10, 19], Metric Temporal Logic (MTL) [6], Probabilistic Temporal Logic (PTL) [20] and Signal Temporal Logic (STL) [16, 15, 12]. These approaches can be used for task planning [14], where a system designer a priori provides logical specifications composed of disjunctions, conjunctions, negations as well as temporal permutations of those combinations. Previous work has also proposed methods for combining sampling-based motion planners with such specification languages to do joint task and motion planning [7, 6]. However, these approaches are limited in their capacity to both express the constraints as well as the capability to account for uncertainty in sensors and dynamics.

Recent work has proposed probabilistic logical specification (PrSTL) [18] that introduces random variables in logical formulae to express uncertainty in the robot state, environment and other exogenous variables. Such probabilistic formulation enables embedding of Bayesian classifiers and predictors in the specification language, thereby allowing the systems to operate in environments that are only partially observed. We propose a new method that builds upon RRT* and uses PrSTL as a steering function. This method combines the positive aspects of both techniques: (1) PrSTL enables us to specify safety invariants and allows embedding of machine learning predictors operating on real-time signals. (2) The RRT* framework allows fast computation of strategies circumventing the need to solve computationally difficult problems that usually arise in logical specification based control synthesis methods.

III. Probabilistic Signal Temporal Logic

Probabilistic Signal Temporal Logic (PrSTL) allows expressing stochastic properties over real-valued, dense-time signals. We can formally define temporal properties over uncertainties that are present in sensors and classifiers of the system. For example, we can express PrSTL formulas that represent probability that the output of a Bayesian predictor would lie in a desired range for time steps in the future.

Let $x(t)$ denote a real-valued signal at time t, then $(x, t) \models \varphi$ specifies that the signal x satisfies the PrSTL formula φ at time t. A PrSTL formula φ consists of temporal and Boolean properties over atomic predicates represented as $\lambda^{\epsilon_t}_{\alpha_t}$. Such predicates are defined over time-varying random variables α_t drawn from a distribution at every time step. Furthermore, $\epsilon_t \in [0, 0.5]$ represents a tolerance level for satisfaction of the predicate. Therefore, satisfaction of this atomic predicate translates to:

$$(x, t) \models \lambda^{\epsilon_t}_{\alpha_t} \iff P\big(\lambda_{\alpha_t}(x(t)) < 0\big) > 1 - \epsilon_t, \qquad (1)$$

where $\lambda_{\alpha_t}(x(t))$ is a stochastic function of the signal, which can express uncertainties regarding sensors, classifiers, etc. For example, if α_t represent parameters of a classifier then computing the stochastic function simply corresponds to application of the classifier to $x(t)$. Consequently, the atomic predicate described above signifies that only those trajectories for which the condition $\lambda_{\alpha_t}(x(t)) < 0$ holds with a high probability should be considered valid. PrSTL allows nesting of temporal and Boolean properties over the probabilistic predicates. The syntax of PrSTL is defined as follows:

$$\varphi ::= \lambda_{\alpha_t}^{\epsilon_t} \mid \tilde{\neg}\lambda_{\alpha_t}^{\epsilon_t} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{G}_{[a,b]}\psi \mid \varphi\, \mathbf{U}_{[a,b]}\psi \mid \mathbf{F}_{[a,b]}\psi.$$

Here, φ is constructed as a probabilistic predicate $\lambda_{\alpha_t}^{\epsilon_t}$, its negation $\tilde{\neg}\lambda_{\alpha_t}^{\epsilon_t}$, the Boolean conjunction or disjunction of two PrSTL formulae, or temporal operators applied over PrSTL formulae. The temporal operators consist of $\mathbf{G}$ (*globally*), $\mathbf{F}$ (*eventually*) and $\mathbf{U}$ (*until*). For example, $\mathbf{G}_{[5,7]}(P(\lambda_{\alpha_t}(x(t)) < 0) > 0.8)$ is a formula indicating that the stochastic function $\lambda_{\alpha_t}(x(t))$ must be less than zero with 0.8 confidence for all times in the interval $t \in [5, 7]$.

The satisfaction of each temporal or propositional formula is then defined as follows:

$$\begin{array}{lcl}
(\xi,t) \models \lambda_{\alpha_t}^{\epsilon_t} & \Leftrightarrow & P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1-\epsilon_t \\
(\xi,t) \models \tilde{\neg}\lambda_{\alpha_t}^{\epsilon_t} & \Leftrightarrow & P(-\lambda_{\alpha_t}(\xi(t)) < 0) > 1-\epsilon_t \\
(\xi,t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi,t) \models \varphi \wedge (\xi,t) \models \psi \\
(\xi,t) \models \varphi \vee \psi & \Leftrightarrow & (\xi,t) \models \varphi \vee (\xi,t) \models \psi \\
(\xi,t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t+a,t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \mathbf{F}_{[a,b]}\varphi & \Leftrightarrow & \exists t' \in [t+a,t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \varphi\ \mathbf{U}_{[a,b]}\ \psi & \Leftrightarrow & \exists t' \in [t+a,t+b] \text{ s.t. } (\xi,t') \models \psi \\
 & & \wedge \forall t'' \in [t,t'], (\xi,t'') \models \varphi.
\end{array}$$

It is shown that if α_t is drawn from a Gaussian distribution, the control synthesis problem under PrSTL constraints can be solved as a mixed integer semi-definite program (MISDP). While solving an MISDP is NP-complete, there exists a subset of PrSTL called *Convex PrSTL* that is recursively defined over the predicates using only conjunctions or the globally operator. The optimization problem reduces to second order cone programming (SOCP) and is convex. One of the advantages of the proposed framework is that instead of solving a general mission planning task as PrSTL, it uses sampling-based motion planning to decompose the problem into a sequence of simpler convex optimization tasks.

IV. Approach

In this section, we detail our approach for the cenario where a robot is tasked with navigating from a start state to a goal state and an *incomplete* map of the environment is available. There might be additional obstacles and other latent variables on the map, which are unknown in the beginning but as the robot navigates, onboard sensors (noisily) detect them.

The key idea is to first sample the configuration space for valid points that satisfy the safety invariants, and then seek for a safe path or trajectory that would connect these sets of sampled points. Such safe trajectories are determined via safe control synthesis using the PrSTL framework. Given the sampled points and the safe trajectories that connect these, the framework finally chooses the shortest path from start to goal which minimizes the cost criterion of interest.

One big advantage of this framework is that the validity test for random samples does not need a rigid logical specification and can be expressed as an imperative procedure. Such imperative descriptions allows checking of fairly complex safety conditions, which might be very hard to evaluate using PrSTL. For example, the boundaries of a flying arena can be of arbitrary shape, and constraints on such non-parametric boundaries cannot be easily expressed as logical formulae. However, given a map of such arena it is easy to check whether a sample is valid or not.

Algorithm 1 PrSTL-Tree: Safe planning and control to goal.

```
Require: Map of known obstacles $\mathcal{M}$
         Start state $s_{\text{start}}$
         Goal state $s_{\text{goal}}$
         Goal region radius $g_{\text{radius}}$
         Number of vertices in tree $n_{\text{vertices}}$
         Number of steps per planning cycle $n_{\text{steps}}$
Ensure: Path traversed to goal $p = \{s_{\text{start}}, s_1, \ldots, s_{\text{goal}}\}$
 1: $p = \{\}$
 2: $s_{\text{current}} = s_{\text{start}}$
 3: while dist($s_{\text{current}}, s_{\text{goal}}$) $> g_{\text{radius}}$ do
 4:     tree ← BuildSafeTree($\mathcal{M}, s_{\text{current}}, s_{\text{goal}}, n_{\text{vertices}}$)
 5:     $s_{\text{nearest}}$ ← FindNearestNeighbor(tree, $s_{\text{goal}}$)
 6:     $p_{\text{shortest}}$ ← ShortestPathToGoal(tree, $s_{\text{current}}, s_{\text{nearest}}$)
 7:     ($p_{\text{traversed}}, s_{\text{current}}$, obsv) ← TakeNSteps($p_{\text{shortest}}, n_{\text{steps}}$)
 8:     $p \leftarrow p \cup p_{\text{traversed}}$
 9:     UpdateBelief(obsv)
10: end while
```

Algorithm 1 details the main steps of the approach, which we term as PrSTL-Tree. It requires a map $\mathcal{M}$ of the environment which contains known obstacles and also encodes rules-of-the-road like no-fly regions, a start state s_{start}, a goal state s_{goal}, radius g_{radius} which describes the goal region centered around the goal state, the number of vertices to be built into the sampling-based motion planner tree n_{vertices} at each planning cycle and the number of steps n_{steps} that the robot will actually traverse each planning cycle.

Initially the path taken by the robot is set to the empty sequence $p = \{\}$ and the current state is set to s_{current} (lines $1-2$). While the robot is still more than g_{radius} away from the goal region the safe planner is invoked in a receding-horizon style to find a safe path to goal (lines $3-10$). In line 4 the function BuildSafeTree invokes a sampling-based motion planner on the map $\mathcal{M}$ of known obstacles. Suitable choices for sampling-based motion planner include RRT [11] and RRT* [8]. This function creates a tree so that it has n_{vertices} from the current state s_{current} of the robot towards the goal state s_{goal}. Note that it is not a requirement for the tree to reach the goal in n_{vertices}. Approaches like RRT and RRT* build a tree towards the goal state by sampling states at random, checking that they lie in free space and then connecting the sampled state to the nearest node$\{$s$\}$ in the tree using a steering function which is responsible for producing dynamically feasible trajectories. These trajectories are then checked for collision and satisfaction of rules-of-the-road and then added to the tree. We take the approach of constructing dynamically feasible and *high-probability collision-free* trajectories for connecting states in the tree leveraging the PrSTL [18] framework for synthesizing trajectories. PrSTL takes sensor uncertainty and robot dynamics into account to synthesize trajectories which are probabilistically safe up to user-specified confidence. If it is not feasible to construct such a trajectory then the PrSTL routine returns an empty trajectory and the sampled state is rejected. So in line 4 the returned $tree$ has edges (trajectories) which are safe by construction.

In line 5 the nearest state s_{nearest} in the tree to the goal state is found by an efficient nearest neighbor search. Then the shortest path in the tree from root (s_{current}) to s_{nearest} is computed using A* [4] or Dijkstra's shortest path algorithm [3] in line 6 to give a path p_{shortest}.

In line 7 the robot executes n_{steps} of p_{shortest} and ends up in a new current state s_{current}. Along the way it makes observations using onboard (noisy) sensors which then can be used to update the obstacle classifier embedded in the PrSTL framework. If the robot is not in the goal region at this time, it builds a safe tree again using its updated beliefs from its current state.

PRSTL-TREE mitigates the chief limitation of using PRSTL alone for long-horizon mission planning with arbitrarily complicated obstacle maps and rules-of-the-road: it eliminates the use of mixed-integer constraints which are necessary for accounting for obstacles and sequential waypoints in PRSTL. Since mixed-integer SOCPs or SDPs are NP-hard [13] these are usually solved sub-optimally by branch-and-bound based algorithms and have large runtimes for non-trivial problem sizes. By relegating this difficult task of modeling known obstacles and waypoints to a sample-based planner, only the convex subset of PRSTL is needed which gives rise to SOCPs which are convex and can be solved in polynomial time.

We show via initial experiments in simulation the large speedup in time obtained by using PRSTL-TREE as opposed to PRSTL.

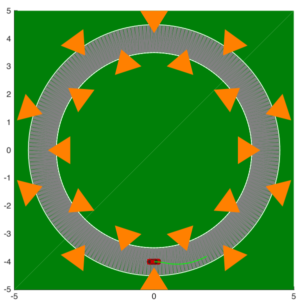
V. Case Study

Our goal in this case study is to find control inputs for a ground robot so it reaches a final goal while avoiding known and unknown obstacles and staying within the road boundaries. Figure 1a shows the initial setting of our experiment. The red car represents our ground vehicle that must stay within the boundaries of the circular road. The orange triangles represent the obstacles present in this scenario. The robot is constrained to travel on the road while avoiding the orange triangles. These obstacles can either be known a priori or only known locally based on uncertainties arising from classifiers.

We then use a mesh of points around the robot that act as range finders that can detect obstacles. Using linear Gaussian Processes [17], we are able to predict if a point in the space is an obstacle or not based on the learned Gaussian distribution. Therefore, the problem of obstacle avoidance translates to probabilistic constraints as follows:

$$\mathbf{G}_{[0,\infty)}\big(Pr(\mathbf{v}\cdot\begin{bmatrix}x & y & 1\end{bmatrix}^\top \leq 0) \geq 1-\epsilon\big) \qquad (2)$$

Here, $\mathbf{v}$ is a Gaussian vector learned by linear Gaussian Processes, and (x, y) are the coordinates of the robot. The inner product of $\mathbf{v}$ and $\begin{bmatrix}x & y & 1\end{bmatrix}^\top$ represents the current belief of coordinates (x, y) being in an obstacle or not. We would like to enforce that the coordinates are outside of obstacles with high probability $1-\epsilon$ at all times $t \in [0,\infty)$. For our experiments, we chose $\epsilon = 0.5$, which allows an easier fit of a prediction line to triangular shaped obstacles. Although


(a) Initial state of the robot.

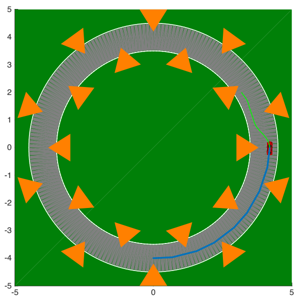
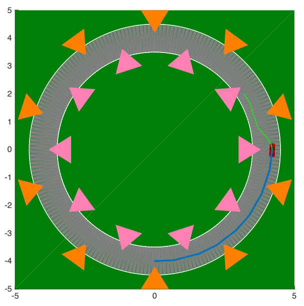

(b) Final state of the robot.

Fig. 1: Autonomous robot reaching a final goal while avoiding obstacles. Here, the red car on the road shows the autonomous robot. The robot's goal is to travel on the circular road while avoiding obstacles and staying within boundaries. The orange triangles represent the obstacles on the road. The green line on both figures shows the computed future trajectory of the robot for the next horizon. The blue line in 1b shows the trajectory computed and taken by the robot to reach its goal.


(a) Reaching the final state by receding horizon optimization for PrSTL under uncertainty.

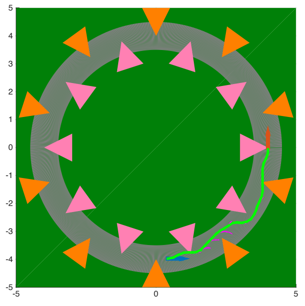

(b) Reaching the final state by following PrSTL-Tree under uncertainty.

Fig. 2: Comparing receding horizon optimization for safe control on the left and PrSTL-Tree approach on the right for reaching the same final goal while staying safe. Here, the orange triangles represent the known obstacles and the pink ones represent the unknown obstacles.

ϵ is large, the resulting trajectories in Figure 2 do not collide with any obstacles.

Note, in this case, the probabilistic constraints can equivalently be written as semi-definite programs which makes the constraints corresponding to uncertain obstacles convex. However, the known obstacles and the boundary conditions of the road are still non-convex constraints. Using PrSTL-Tree we compute the optimal trajectory of traveling a quarter of the circular road in 16.9848 seconds. This is shown in Figure 2a, where the blue line shows the trajectory computed and taken by the robot and the green line is the next horizon's planned trajectory. The computation time for this example is smaller than the same example with known obstacles. Obstacle avoidance under uncertainty results in convex properties, which can help lower the computation time by reducing the number of disjunctions in the formula.

Using PrSTL-Tree, we were able to find the controller in 2.1339 seconds. This is **significantly (approximately 8 times) faster** than only using the optimization based method with PrSTL constraints.

References

[1] Georges S Aoude, Brandon D Luders, Joshua M Joseph, Nicholas Roy, and Jonathan P How. Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. *Autonomous Robots*, 2013.

[2] Debadeepta Dey, Kumar Shaurya Shankar, Sam Zeng, Rupesh Mehta, M. Talha Agcayazi, Christopher Eriksen, Shreyansh Daftry, Martial Hebert, and J. Andrew Bagnell. Vision and learning for deliberative monocular cluttered flight. *Field and Service Robotics*, 2015.

[3] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959.

[4] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 1968.

[5] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 2001.

[6] S Karaman and E Frazzoli. Optimal vehicle routing with metric temporal logic specifications. In *IEEE Conference on Decision and Control*, 2008.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Decision and Control*, 2009.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 2011.

[9] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt*. In *ICRA*.

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

[11] Steven M Lavalle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 2000.

[12] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[13] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. 1982.

[14] Erion Plaku and Sertac Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI Communications*, 2015.

[15] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 81–87. IEEE, 2014.

[16] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM, 2015.

[17] Carl Rasmussen and Chris Williams. Gaussian processes for machine learning. *Gaussian Processes for Machine Learning*, 2006.

[18] Dorsa Sadigh and Ashish Kapoor. Safe control under uncertainty. *Robotics Science and Systems*, 2016.

[19] Paulo Tabuada and George J Pappas. Linear temporal logic control of linear systems. *IEEE Transactions on Automatic Control*, 2004.

[20] Chanyeol Yoo, Robert Fitch, and Salah Sukkarieh. Probabilistic temporal logic for motion planning with resource threshold constraints. 2012.