Submitted, 2015 International Conference on Hybrid Systems: Computation and Control (HSCC)
http://www.cds.caltech.edu/~murray/papers/lwm15-hscc.html

Cross-entropy Temporal Logic Motion Planning

Scott C. Livingston
Department of Control and Dynamical Systems
California Institute of Technology
Pasadena, CA, USA
slivingston@cds.caltech.edu

Eric M. Wolff
nuTonomy LLC
Cambridge, MA, USA
eric@nutonomy.com

Richard M. Murray
Department of Control and Dynamical Systems
California Institute of Technology
Pasadena, CA, USA
murray@cds.caltech.edu

ABSTRACT

This paper presents a method for optimal trajectory generation for discrete-time nonlinear systems with linear temporal logic (LTL) task specifications. Our approach is based on recent advances in stochastic optimization algorithms for optimal trajectory generation. These methods rely on estimation of the rare event of sampling optimal trajectories, which is achieved by incrementally improving a sampling distribution so as to minimize the cross-entropy. A key component of these stochastic optimization algorithms is determining whether or not a trajectory is collision-free. We generalize this collision checking to efficiently verify whether or not a trajectory satisfies a LTL formula. Interestingly, this verification can be done in time polynomial in the length of the LTL formula and the trajectory. We also propose a method for efficiently re-using parts of trajectories that only partially satisfy the specification, instead of simply discarding the entire sample. Our approach is demonstrated through numerical experiments involving Dubins car and a generic point-mass model subject to complex temporal logic task specifications.

1. INTRODUCTION

This paper presents a method for computing optimal trajectories for discrete-time nonlinear systems subject to temporal logic specifications. We are motivated by the need for autonomous robots to efficiently execute tasks with complex temporal constraints. Such robots often have nonlinear dynamics and configuration spaces that are difficult to represent analytically. Examples include robotic manipulators that must repeatedly perform intricate assembly tasks, or autonomous cars that must navigate traffic. In these types of demanding settings, near-optimal control policies are typically required.

To concisely and precisely specify a wide range of complex tasks, we use linear temporal logic (LTL). LTL is an expressive task-specification language that can be used to specify safety requirements, acceptable response to the environment, desired goal visitation, periodic motions, and stability [2]. These properties generalize classical robotic motion planning [15].

Informally, traditional methods for motion planning with LTL specifications rely on first constructing a labeled graph (i.e., a *finite abstraction*) that represents possible behaviors of the dynamical system (see [1, 4, 12, 24]). Given a finite abstraction and an LTL specification, controllers can be automatically constructed using an automata-based approach [2, 7, 12] inspired by techniques developed for discrete systems [2]. However, this approach is limited to low-dimensional systems (less than 6 continuous dimensions) as the time to compute a discrete abstraction is exponential in the number of dimensions. Additionally, the size of an appropriate automaton may be exponential (or worse) in the length of the LTL formula [2].

To avoid the expensive computations of a discrete abstraction and an automaton, we directly sample trajectories from a distribution over trajectories. Each sampled trajectory is then checked to determine if it satisfies a given temporal logic specification. Surprisingly, this check can be done time polynomial in the length of the specification and trajectory. The sampling distribution is then updated based on the quality of the sampled trajectories, and this procedure repeats until convergence. Our approach generalizes the cross-entropy motion planning method [13] by incorporating LTL task constraints.

The contributions of this paper are twofold. First, we present a stochastic optimization technique for optimal trajectory generation of nonlinear systems operating in complex configuration spaces with linear temporal logic specifications. Each iteration of this algorithm runs in time polynomial in the size of the system and specification, and does not require the costly computation of a discrete abstraction or automaton. Furthermore, the approach is straightforward to implement in a parallel manner. Second, we provide a method for re-sampling parts of trajectories that are promising, in the sense that they satisfy a relaxed specification. We empirically demonstrate that this reuse improves convergence rates, improving on state-of-the-art techniques. The latter contribution of incremental trajectory construction is important for the types of highly-constrained problems that are common in robotics, where sampling-based methods are expected to generate many infeasible points.

Related work
Some recent work avoids the computationally expensive construction of an abstraction and an automaton by directly encoding a temporal logic formula as mixed-integer linear constraints on the system [10, 14, 22, 23]. However, these approaches assume that the free configuration space can be represented as a union of polytopes, which is not the case for many robotic systems.

Our work is closely related to sampling-based motion planning techniques for temporal logic planning [5, 9, 18]. These approaches iteratively build a finite abstraction of the system using sampling- based motion planners, which can handle nonlinear dynamics and complicated configuration spaces. Our approach is different in that we iteratively refine a sampling distribution over "good" trajectories, instead of iteratively building and model checking a graph of feasible trajectories.

Statistical approaches have previously been used for model checking of hybrid systems. Monte Carlo algorithms have been developed for verifying finite-state systems [8], discrete event systems [25], and linear hybrid systems [17]. Additionally, cross-entropy approaches have been used to falsify metric temporal logic properties for hybrid systems [20], and finite-time properties for black-box hybrid models [6]. Instead, we consider synthesizing optimal plans with infinite-time temporal properties, leverage efficient LTL satisfaction algorithms, and exploit more detailed knowledge of the system dynamics.

2. PRELIMINARIES
In this section we give background on dynamical systems and linear temporal logic. Our treatment is brief and intended primarily to fix notation.

Notation: An *atomic proposition* is an indivisible statement that is either *True* or *False*. The cardinality of a set X is denoted by $|X|$.

2.1 System model
We consider discrete-time nonlinear systems of the form

$$x_{t+1} = f(x_t, u_t), \tag{1}$$

with time indices $t = 0, 1, \ldots$, states $x \in \mathcal{X}$, control inputs $u \in \mathcal{U}$, and initial state $x_0 \in \mathcal{X}$. As will be shown later, our method applies to any system that admits sampling of trajectories based on a parameterization, and in particular $\mathcal{X}$ and $\mathcal{U}$ can have the structure of $\mathbb{R}^n$, SE(2), or countable spaces like $\mathbb{Z}$.

Let AP be a finite set of atomic propositions, which indicate basic properties, such as occupancy of a goal region. The labeling function $\mathcal{L} : \mathcal{X} \to 2^{AP}$ maps states to subsets of atomic propositions that are *True*.

A *trajectory* (run) $\mathbf{x} = x_0 x_1 x_2 \ldots$ of system (1) is an infinite sequence of its states, where $x_t \in \mathcal{X}$ is the state of the system at index t, and for each $t = 0, 1, \ldots$, there exists a control input $u_t \in \mathcal{U}$ such that $x_{t+1} = f(x_t, u_t)$. Given an initial state x_0 and a control input sequence $\mathbf{u}$, the resulting trajectory $\mathbf{x} = \mathbf{x}(x_0, \mathbf{u})$ is unique. A *word* is an infinite sequence of labels $\mathcal{L}(\mathbf{x}) = \mathcal{L}(x_0)\,\mathcal{L}(x_1)\,\mathcal{L}(x_2)\ldots$ where $\mathbf{x}$ is a trajectory. Let $\mathbf{x}_i = x_i x_{i+1} x_{i+2} \ldots$ denote the trajectory $\mathbf{x}$ from index i, and let $\mathcal{L}(\mathbf{x}_i) = \mathcal{L}(x_i)\,\mathcal{L}(x_{i+1})\,\mathcal{L}(x_{i+2})\ldots$ denote the word from index i.

2.2 Linear temporal logic
We use linear temporal logic (LTL) to concisely and precisely specify permitted system behavior. LTL is powerful language that can be used to specify a wide range of important system behaviors for robots and other cyberphysical systems. We briefly state the syntax and semantics of LTL; see [2] for a detailed treatment.

Syntax: LTL syntax consists of (a) a set of atomic propositions AP, (b) Boolean operators: $\wedge$ (and) and $\neg$ (not), and (c) temporal operators: $\bigcirc$ (next) and $\mathcal{U}$ (until). An LTL formula is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2,$$

where $p \in AP$ is an atomic proposition.

Semantics: The semantics of LTL are defined inductively over a word $\mathcal{L}(\mathbf{x})$ as follows:

$$\begin{aligned}
&\mathcal{L}(\mathbf{x}_i) \models p \text{ if and only if } p \in \mathcal{L}(x_i)\\
&\mathcal{L}(\mathbf{x}_i) \models \neg\varphi \text{ if and only if } \mathcal{L}(\mathbf{x}_i) \not\models \varphi\\
&\mathcal{L}(\mathbf{x}_i) \models \varphi_1 \vee \varphi_2 \text{ if and only if } \mathcal{L}(\mathbf{x}_i) \models \varphi_1 \vee \mathcal{L}(\mathbf{x}_i) \models \varphi_2\\
&\mathcal{L}(\mathbf{x}_i) \models \varphi_1 \wedge \varphi_2 \text{ if and only if } \mathcal{L}(\mathbf{x}_i) \models \varphi_1 \wedge \mathcal{L}(\mathbf{x}_i) \models \varphi_2\\
&\mathcal{L}(\mathbf{x}_i) \models \bigcirc\varphi \text{ if and only if } \mathcal{L}(\mathbf{x}_{i+1}) \models \varphi\\
&\mathcal{L}(\mathbf{x}_i) \models \varphi_1 \,\mathcal{U}\, \varphi_2 \text{ if and only if } \exists j \geq i \text{ such that}\\
&\qquad \mathcal{L}(\mathbf{x}_j) \models \varphi_2 \text{ and } \mathcal{L}(\mathbf{x}_n) \models \varphi_1 \;\; \forall i \leq n < j
\end{aligned} \tag{2}$$

Definition 1. A word $\mathcal{L}(\mathbf{x})$ *satisfies* φ, denoted by $\mathcal{L}(\mathbf{x}) \models \varphi$, if $\mathcal{L}(\mathbf{x}_0) \models \varphi$. A trajectory $\mathbf{x}$ satisfies φ if $\mathcal{L}(\mathbf{x}) \models \varphi$.

Remark 1. The Boolean operators $\vee$ (or) and $\implies$ (implies) can be defined in the usual way. Informally, the notation $\bigcirc\varphi$ means that φ is true at the next step, $\varphi_1 \,\mathcal{U}\, \varphi_2$ means that φ_1 is true until φ_2 is true, $\square\varphi$ means that φ is always true, $\diamond\varphi$ means that φ is eventually true, and $\square\diamond\varphi$ means that φ is true repeatedly [2].

3. PROBLEM STATEMENT
In this section, we formally state the main problem treated in this paper.

Let the generic *cost function* $J(\mathbf{x}, \mathbf{u})$ map from trajectories and control input sequences to nonnegative real numbers.

Problem 1. Given a dynamical system of the form (1), an initial state $x_0 \in \mathcal{X}$, and an LTL formula φ, compute a control input sequence $\mathbf{u}$ that minimizes $J(\mathbf{x}(x_0, \mathbf{u}))$ subject to $\mathcal{L}(\mathbf{x}(x_0, \mathbf{u})) \models \varphi$.

Problem 1 is a challenging nonconvex optimization problem due to the nonlinear dynamic constraints, and the combinatorial temporal logic constraints. The core of our solution to Problem 1 is the cross-entropy method, a stochastic optimization algorithm that has been used to successfully solve challenging point-to-point motion planning problems [13].

The remainder of this paper details how to extend the cross-entropy method to handle complex LTL specifications. We present our approach in two parts: a basic algorithm in Section 4, and a more efficient version in Section 5. The latter algorithm reuses infeasible trajectories to significantly increase the computational efficiency of the method.

Remark 2. While we give a discrete-time formulation of the problem, continuous-time systems can be handled using the same cross-entropy framework by defining appropriate semantics for LTL over continuous trajectories, e.g., [12].

4. CROSS-ENTROPY LTL PLANNING

In this section we present our first main contribution: a method for stochastic optimization of trajectories subject to LTL constraints. Our solution is based on the cross-entropy motion planning framework introduced in [13].

The basic idea behind applying the cross-entropy approach to motion planning [13] is to repeat the following two steps: 1) generate sample trajectories from a distribution and compute their costs, and 2) update the distribution using a subset of "good" samples, until the sampling distribution converges to a delta function (hopefully) over an optimal trajectory. Although convergence to a globally optimal solution cannot be guaranteed (as with nonconvex optimization in general), the approach does explore the entire state space.

We first give a high-level overview of the cross-entropy method in Section 4.1. In Section 4.2, we introduce a finite trajectory parameterization that is amenable to computation. We extend the cross-entropy method to complex temporal logic tasks in Section 4.3. Finally, we discuss efficient methods for determining if a sampled trajectory satisfies an LTL formula in Section 4.4, and detail the complexity of our approach in Section 4.5.

4.1 Cross-entropy optimization

The cross-entropy method estimates rare events (e.g., sampling an optimal trajectory) using importance sampling. Let Z denote a random variable defined over a space $\mathcal{Z}$. The rare event of interest is finding a parameter z with a real-valued cost $J(z)$ which is near the cost of an optimal parameter z^*. This rare-event estimation is equivalent to the global optimization of $J(z)$. Our development closely follows [13] and is based on [19].

Rare-event estimation

Consider the problem of estimating the probability that a parameter $z \in \mathcal{Z}$ sampled from the probability density function $p(\cdot;\bar{v})$ (with parameter $v \in \mathcal{V}$) has a cost $J(z)$ smaller than a given constant γ. This probability is defined as

$$l = \mathbb{P}_{\bar{v}}(J(Z) \leq \gamma) = \mathbb{E}_{\bar{v}}\left[I_{\{J(Z)\leq\gamma\}}\right],$$

which can be approximated by

$$\hat{l} = \frac{1}{N}\sum_{i=1}^{N} I_{\{J(Z_i)\leq\gamma\}}\frac{p(Z_i;\bar{v})}{p(Z_i;v)},$$

where $Z_1,\ldots,Z_N$ are i.i.d. samples from the *importance density* $p(\cdot,v)$. The issue is that when $\{J(Z)\leq\gamma\}$ is a rare event, $\hat{l}$ will be incorrectly estimated as zero.

The idea behind the cross-entropy method is to employ a multi-level approach using a sequence of parameters $\{v_j\}_{j\geq 0}$ (parameterizing the importance density) and levels $\{\gamma_j\}_{j\geq 1}$. The sequence converges to the optimal v^*, which then can be used to estimate the integral l.

The procedure starts by drawing N samples $Z_1,\ldots,Z_N$ using a sampling distribution with an initial parameter v_0, e.g., $v_0 = \bar{v}$. The value γ_1 is set to the ρth quantile of $I_{\{J(Z)\leq\gamma\}}$, where ρ is a small scalar, e.g., $\rho \leq 10^{-1}$. The level γ_1 can be approximated by sorting the costs of the samples $J(Z_1),\ldots,J(Z_N)$ in an increasing order, and setting $\hat{\gamma}_1 = J_{\lceil\rho N\rceil}$.

The optimal sampling parameter v_1 for level γ_1 is then approximated numerically by

$$\hat{v}_1 = \arg\max_{v\in\mathcal{V}} \frac{1}{N}\sum_{i=1}^{N} I_{\{J(Z_i)\leq\hat{\gamma}_1\}} \ln p(Z_i,v),$$

where $Z_1,\ldots,Z_N$ are i.i.d. samples from the previous distribution $p(\cdot,v_0)$.

The procedure then iterates to compute the next γ_i and v_i, terminating when $\gamma_i \leq \gamma$. The probability of $J(Z)\leq\gamma$ is then computed using $v = v_i$. In summary, each iteration of the algorithm performs two steps, starting with v_0:

1. **Sampling and updating of** γ_j: Sample $Z_1,\ldots,Z_N$ from $p(\cdot,\hat{v}_{i-1})$ and compute the ρth quantile $\hat{\gamma}_t$.
2. **Adaptive updating of** v_j: Compute $\hat{v}_j$ such that
$$\hat{v}_j = \arg\min_{v\in\mathcal{V}} \frac{1}{|\mathcal{E}_j|}\sum_{Z_k\in\mathcal{E}_j} \ln p(Z_k;v), \qquad (3)$$
where $\mathcal{E}_j$ is the set of samples for which $J(Z_k)\leq\hat{\gamma}_j$.

Computing an optimal trajectory

An optimal trajectory can be computed by iterating the steps above until the level γ_j approaches the optimal cost γ^*. Typically, $p(\cdot,v_j)$ will approach a delta distribution, signifying that a local optimum has been found. Note that although the method explores the state space globally it may converge to a local optimum if there were no samples near the global optimum.

4.2 A finite trajectory parameterization

As LTL specifications are defined over infinite time, one must encode an infinite trajectory using a finite representation that is amenable for computation. We encode an infinite trajectory using a "lasso," i.e., a finite prefix followed by a finite suffix that is repeated. Precisely, we consider trajectories of the *prefix-suffix* form

$$\mathbf{x} = x_0x_1\cdots x_{\tau-1}(x_\tau\cdots x_T)^\omega = \mathbf{x}_{\text{pre}}(\mathbf{x}_{\text{suf}})^\omega, \qquad (4)$$

where $\mathbf{x}_{\text{pre}} = x_0x_1\cdots x_{\tau-1}$ is the *trajectory prefix*, $\mathbf{x}_{\text{suf}} = x_\tau\cdots x_T$ is the *trajectory suffix*, and ω denotes infinite repetition. Besides step-wise consistency with the dynamics (1), loop closure must be enforced, i.e., $x_\tau = f(x_T,u_T)$ for some control input $u_T\in\mathcal{U}$. If $\tau = 0$, then the prefix is empty, i.e., the trajectory is a loop.

Only considering trajectories in prefix-suffix form is restrictive, as there may exist trajectories that satisfy the specification, but are not eventually periodic. This behavior is possible (unlike for finite, discrete systems [2]) due to the continuous state space. While this restriction is potentially an issue for analysis, it is of no practical limitation for trajectory synthesis like we are considering.

4.3 Algorithm

Let $\Theta = \mathbb{R}^n$ denote the *parameter space*, and fix the scalar *loop index* l in the range $1, \ldots, n+1$. Given an initial state x_0 and loop index l, we assume there exists a procedure $\textsc{GenTraj}_{x_0,l}(\theta)$ that takes as input parameter values $\theta \in \Theta$ and returns a trajectory

$$\mathbf{x}(\theta) = x_0 x_1 \cdots x_{\tau-1}(x_\tau \cdots x_T)^\omega \tag{5}$$

where τ depends on p as follows. Intuitively, each of the n components of $\theta = (\theta^1, \theta^2, \ldots, \theta^n)$ determines a finite segment of the trajectory obtained from GenTraj and moreover, there are $n+1$ such segments. The first segment is from the initial state x_0 to some state that is obtained using θ^1. From this state, another sequence of states is obtained using θ^2, and so on. The loop index l, which we fix before performing the stochastic optimization described below, is the end point of these fragments at which the loop of the suffix is closed, thereby forming the desired prefix-suffix structure. If $l = 1$, then there is no prefix. For $l > 1$, the suffix is obtained by connecting after the $(l-1)$-th fragment, as obtained after using parameter component θ^{l-1}.

We now list some concrete realizations of the parameter space Θ. Parameter values $\theta \in \Theta$ could be control inputs $\mathbf{u}$, in which case GenTraj would integrate (1) while applying these inputs to obtain a trajectory. Alternatively, the parameters θ could define a sequence of waypoints in $\mathcal{X}$ and GenTraj could solve the corresponding boundary-value problems by using an appropriate local planner [15]. For example, in Section 6, trajectories for Dubins car are parameterized using finite sequences of time durations and turning rates.

Besides trajectory parametrization, the other important part of our method is defining and updating the sampling distribution. Following the notation of Section 4.1, this distribution is assumed to be taken from a parametrized family of probability density functions $\{p(\cdot; v) \mid v \in \mathcal{V}\}$. In order to simplify notation in the remainder of the paper, we introduce a routine $\textsc{Update}(v, \theta_1, \ldots \theta_m)$ that returns parameters for a new distribution (see equation (3)) more appropriate for trajectories with the parameters $\theta_1, \ldots \theta_m$, given the distribution $p(\cdot; v)$.

We are now ready to express the CE-LTL method in Algorithm 1. Details for several lines are as follows:

- Line 7: Trajectories are obtained by sampling parameter vectors. When a trajectory is infeasible (i.e., does not satisfy the LTL specification) as checked on Line 9, it is re-sampled. An extension for re-using promising trajectories is described in Section 5.
- Line 8: Control inputs $\mathbf{u}$ are obtained from θ_i. E.g., θ_i itself can be a sequence of control inputs, or it can be a collection of waypoints in $\mathcal{X}$, local planners for which yield control inputs.

Algorithm 1 CE-LTL

```
1: INPUT: LTL formula $\varphi$, cost function $J$, initial state $x_0$,
   initial sampling distribution $p(\cdot; v_0)$, number of trajectories per iteration $N$, threshold cost $\alpha$
2: OUTPUT: parameters $\theta_*$ of best trajectory $\mathbf{x}_{\theta_*}$ found
3: $j := 0$ //Iteration counter
4: repeat
5:    for all $i = 1, \ldots, N$ do
6:       repeat
7:          $\theta_i \sim p(\cdot; v_j)$
8:          $\mathbf{x}_{\theta_i} = \textsc{GenTraj}(\theta_i)$ //Compute trajectory
9:       until $\mathcal{L}(\mathbf{x}_{\theta_i}) \models \varphi$
10:   end for
11:   Sort $\theta_1, \ldots, \theta_N$ according to cost, such that
      $J(\mathbf{x}_{\theta_1}) \le J(\mathbf{x}_{\theta_2}) \le \cdots J(\mathbf{x}_{\theta_N})$
12:   $j := j + 1$
13:   $v_j := \textsc{Update}(v_{j-1}, \theta_1, \ldots, \theta_{\lceil \rho N \rceil})$
14: until $J(\mathbf{x}_{\theta_1}) < \alpha$
15: return $\theta_1$ //Parameters of trajectory with least cost
```

- Line 9: A thorough discussion about checking whether traces of trajectories satisfy φ is in Section 4.4.
- Line 13: The probability density function according to which trajectoriesare sampled is adjusted so as to approach the (unknown) optimal distribution using the set of best trajectories $\mathbf{x}_{\theta_1}, \ldots, \mathbf{x}_{\theta_{\lceil \rho N \rceil}}$ found on the current iteration. We abbreviate this step using the routine Update, which is implemented using equation (3).
- Line 14: Any of several termination conditions may be used, including a fixed number of iterations, or the sampling distribution approaching a delta distribution.

4.4 Model checking sampled trajectories

An important part of the method presented in Section 4.3 is checking whether sampled trajectories satisfy the LTL formula φ, as on Line 9 of Algorithm 1. Indeed, the time required to do this is a major contributor to the total execution time, besides the time entailed by discarding the many infeasible trajectories and re-sampling (for which we present an extension in Section 5). Here we outline three approaches to checking trajectories, which taken together demonstrate the modularity of our method.

The length of an LTL formula φ is the number of symbols, and is denoted by $|\varphi|$. The length of a trajectory $\mathbf{x}$ in prefix-suffix form is $|\mathbf{x}_{\text{pre}}| + |\mathbf{x}_{\text{suf}}|$, and is denoted by $|\mathbf{x}|$.

4.4.1 Polynomial-time checking

The first approach for determining if a sampled trajectory satisfies an LTL formula exploits the fact that, the semantics of LTL and CTL (computational tree logic) coincide over paths [16]. Thus, one can use efficient CTL model checking algorithms to verify that a trajectory satisfies an LTL formula. The standard CTL model checking algorithm uses dynamic programming to solve this problem in time time bilinear in the length of the system and specification, i.e., $O(|\mathbf{x}| \times |\varphi|)$ [2].

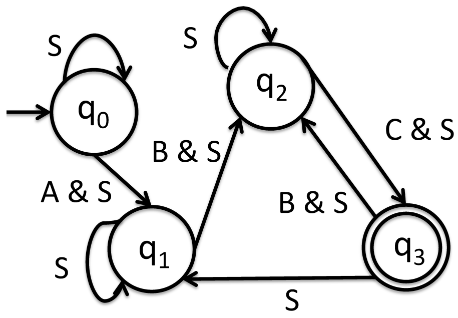

Figure 1: A (simplified) Büchi automaton corresponding to the LTL formula $\varphi = \Diamond A \wedge \Box\Diamond B \wedge \Box\Diamond C \wedge \Box S$. Informally, the system must visit A, repeatedly visit B and C, and always remain in S. Here $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{A, B, C, S\}$, $Q_0 = \{q_0\}$, $F = \{q_3\}$, and transitions are represented by labeled arrows.

4.4.2 Automata-based checking

Another approach to determining if $\mathcal{L}(\mathbf{x}) \models \varphi$ is to check whether $\mathcal{L}(\mathbf{x})$ is in the language of a finite automaton that recognizes φ.

Nondeterministic Büchi automata: Any LTL formula φ can be automatically translated into a corresponding Büchi automaton $\mathcal{A}_\varphi$ of size $2^{O(|\varphi|)}$ [2]. Figure 1 shows an example of a Büchi automaton.

Definition 2. A nondeterministic Büchi automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ consisting of (i) a finite set of states Q, (ii) a finite alphabet Σ, (iii) a transition relation $\delta \subseteq Q \times \Sigma \times Q$, (iv) a set of initial states $Q_0 \subseteq Q$, (v) and a set of accepting states $F \subseteq Q$.

Let Σ^ω be the set of infinite words over Σ. A word $L(\sigma) = \Sigma_0\Sigma_1\Sigma_2 \ldots \in \Sigma^\omega$ induces an infinite sequence $q_0q_1q_2\ldots$ of states in $\mathcal{A}$ such that $q_0 \in Q_0$ and $(q_i, \Sigma_i, q_{i+1}) \in \delta$ for $i \geq 0$. Run $q_0q_1q_2\ldots$ is *accepting (accepted)* if $q_i \in F$ for infinitely many indices $i \in \mathbb{N}$ appearing in the run.

Deterministic Rabin automata: Similarly, any LTL formula can be translated into a corresponding deterministic Rabin automaton $\mathcal{A}_\varphi$ of size $2^{2^{O(|\varphi|)}}$ [2]. Figure 2 shows an example of a deterministic Rabin automaton.

Definition 3. A *deterministic Rabin automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ consisting of (i) a finite set of states Q, (ii) a finite alphabet Σ, (iii) a transition function $\delta : Q \times \Sigma \to Q$, (iv) an initial state $q_0 \in Q$, (v) and a set of accepting pairs $\mathcal{F} = \{(L_1, U_1), \ldots, (L_J, U_J)\}$.

Let Σ^ω be the set of infinite words over Σ. A word $\mathcal{L}(\sigma) = \Sigma_0\Sigma_1\Sigma_2 \ldots \in \Sigma^\omega$ is an infinite sequence $q_0q_1q_2\ldots$ of states in $\mathcal{A}$ such that $\delta(q_i, \Sigma_i) = q_{i+1}$ for $i \geq 0$. Run $q_0q_1q_2\ldots$ is *accepting (accepted)* if there is a pair $(L_j, U_j) \in \mathcal{F}$ such that $q_i \in L_j$ for infinitely many indices $i \in \mathbb{N}_0$ appearing in the run and $q_i \in U_j$ for only finitely many i (possibly none).

For either type of automaton, the set of words for which the corresponding run is accepting is denoted by $\mathcal{L}(\mathcal{A})$. Let

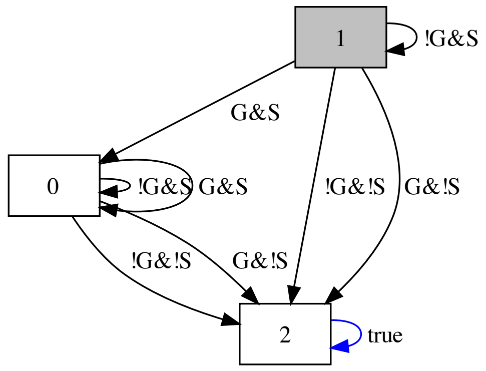

Figure 2: A deterministic Rabin automaton for the LTL formula $\Diamond$G $\wedge$ $\Box$S. States are shown in rectangles numbered 0, 1, and 2. The initial state is 1 and is shaded gray. Edges are labeled with the input values that would cause the transition, written as a Boolean formula in terms of G and S (the inputs to the automaton). There is one acceptance pair (L, U), where $L = \{0\}$ and $U = \{1, 2\}$.

$\mathcal{A}_\varphi$ be an automaton (either deterministic Rabin or nondeterministic Büchi) that recognizes φ. Then, checking that a trajectory satisfies the LTL formula φ is equivalent to checking if $\mathcal{L}(\mathbf{x}) \in \mathcal{L}(A_\varphi)$.

For nondeterministic Büchi automata, this condition can be checked in time bilinear in the size of $\mathcal{A}_\varphi$ and the trajectory, i.e., $O(|\mathbf{x}| \times |\mathcal{A}_\varphi|)$. Due to the nondeterminism in the automaton, this check must use graph search to explore different branches. For deterministic Rabin automata, this condition can be checked in $O(|\mathbf{x}|)$ time, since the automaton is deterministic.

4.5 Complexity

While determining rates of convergence for cross-entropy method is difficult [19], it is possible to precisely state the complexity of generating each sample trajectory and determining if it satisfies the LTL specification.

The complexity of generating each sample trajectory is highly dependent on the parameterization that is used. For example, if one parameterizes the trajectory by a sequence of states, and uses a local planner [15] to connect these states, the complexity is dependent on the local planner used. Note, that a simple parameterization consisting of a sequence of control inputs can be used to generate a dynamically feasible trajectory in time linear in the length of the trajectory. However, a local planner might still be necessary, as loop closure involves the solution of a two-point boundary value problem.

We now summarize the complexity of determining whether or not a trajectory in prefix-suffix form satisfies an LTL formula. The reduction to CTL model checking described in Section 4.4.1 gives an $O(|\mathbf{x}| \times |\varphi|)$ algorithm. Additionally, this does not require the initial construction of an automaton. The automaton-based approaches described in Section 4.4.2 require the initial construction of an automaton of size $2^{O(|\varphi|)}$ (Büchi) or $2^{2^{O(|\varphi|)}}$ (Rabin). However, this worst-case behavior is rarely encountered in practice. Once the appropriate automaton has been constructed, the cost to check each trajectory is $O(|\mathbf{x}| \times |\mathcal{A}_\varphi|)$ (Büchi) and $O(|\mathbf{x}|)$ (Rabin). Thus, there is a problem-dependent trade-off between using the CTL model checking algorithm, constructing a non-deterministic Büchi automaton, or constructing a deterministic Rabin automaton.

5. RE-USING TRAJECTORIES

In this section we present our second main contribution: a method for re-using sampled trajectories that satisfy a relaxed specification, in a sense made precise below. For motivation, consider the classical motion planning setting of going to a goal region while avoiding collisions with obstacles. Incorporating information about the obstacle positions and shapes into the sampling distribution over the parameter space may be relatively difficult, i.e., not far from solving the entire Problem 1. However, while it may be easier to begin with a distribution that does not incorporate application-specific information, it comes at the practical cost of sampling a large number of infeasible trajectories, which using the basic approach of Section 4 would be subsequently rejected. For general LTL formulae, the problem of finding feasible trajectories is strictly more difficult than the classical setting, and so we are motivated to try to re-sample parts of trajectories that appear to be nearly feasible.

To make the intuitive motivation above precise, let φ be the given LTL specification. Suppose that ψ is another LTL formula such that $\mathcal{L}(\varphi) \subset \mathcal{L}(\psi)$ (notice the subset relation is proper). Everything else being equal, any trajectory that is feasible with respect to φ is also feasible with respect to ψ. This implies a subset relation over the set of trajectories for a given dynamics (1), from which it follows that a sampled trajectory satisfies ψ with at least the probability of satisfying φ. Intuitively, sampling trajectories for ψ is easier than for φ. For a carefully chosen formula ψ, the trajectories feasible for ψ can be made feasible for φ by adjusting only some of the control inputs.

5.1 Trajectory re-use algorithm

Suppose that the given LTL specification φ can be decomposed into two LTL formulae ψ and ζ such that $\varphi \equiv \psi \wedge \zeta$. Furthermore suppose that words not in $\mathcal{L}(\psi)$ can be identified after a finite number of steps. That is, certificates of violations of ψ are finite. In the context of the present work, we can decide whether $\mathcal{L}(\mathbf{x}) \models \psi$ for a trajectory $\mathbf{x}$ without having to search for cycles (cf. procedures for checking feasibility in Section 4.4.). Furthermore, since $\varphi \implies \psi$, a trajectory $\mathbf{x}$ is feasible with respect to φ only if it is also feasible with respect to ψ. A trajectory $\mathbf{x}$ is called *promising* if $\mathcal{L}(\mathbf{x}) \models \zeta$ but $\mathbf{x}$ is infeasible with respect to the desired LTL formula φ, in particular $\mathcal{L}(\mathbf{x}) \notin \mathcal{L}(\psi)$.

Recalling the notation and parameterization introduced in Section 4, let $\theta = (\theta^1, \ldots, \theta^n)$ be the parameter values used to construct $\mathbf{x}_\theta$. Since ψ admits finite violation certificates, it is possible to find a first state of the trajectory at which step it must be that $\mathcal{L}(\mathbf{x}) \notin \mathcal{L}(\psi)$. Because sampling is in terms of parameters θ, we want to find a value s in $\{1, \ldots, n\}$ such that the trajectory fragment constructed from $\theta^1, \ldots, \theta^s$ is the largest such fragment not providing a certificate of violation of ψ. It is assumed there is a routine named LastSafe that finds s. E.g., LastSafe could be based on a runtime monitor for LTL as described in [3].

Algorithm 2 Trajectory re-use

```
1: INPUT: $\mathbf{x}_\theta$, $\theta$, $\psi$ such that $\varphi \implies \psi$, max_attempts
2: OUTPUT: $\hat{\theta}$ or **invalid**
3: $s :=$ LastSafe$_\psi(\mathbf{x}_\theta, \theta)$
4: counter := 0
5: **while** counter < max_attempts **do**
6:     Sample $\hat{\theta}^{s+1}, \ldots, \hat{\theta}^n$ from restricted $p(\cdot; v_j)$
7:     $\hat{\theta} := \left(\theta^1, \ldots, \theta^s, \hat{\theta}^{s+1}, \ldots, \hat{\theta}^n\right)$
8:     Compute trajectory $\mathbf{x}_{\hat{\theta}}$ from $\hat{\theta}$
9:     **if** $\mathcal{L}(\mathbf{x}_{\hat{\theta}}) \models \varphi$ **then**
10:        **return** $\hat{\theta}$
11:    **end if**
12:    counter := counter + 1
13: **end while**
14: **return invalid**
```

In the context of the $(j+1)$-th iteration of Algorithm 1, once s is found, the current probability density function $p(\cdot; v_j)$ is restricted to dimensions $s+1, \ldots, n$ of the parameter space Θ and sampling is attempted using this restricted distribution. Denoting this new sample by $\hat{\theta}^{s+1}, \ldots, \hat{\theta}^n$, a new trajectory is constructed using $\theta^1, \ldots, \theta^s, \hat{\theta}^{s+1}, \ldots, \hat{\theta}^n$ and checked for feasibility.

Algorithm 2 is intended to be inserted into Algorithm 1 immediately following Line 8, together with an if-clause to check whether it returns **invalid**, in which case the trajectory is entirely discarded and a new one is sampled (Line 7).

5.2 Comparison with existing work

In classical motion planning, the basic problem is to move from an initial state to a set of goal states while avoiding obstacles. As an LTL formula in Problem 1, this may be expressed by

$$\varphi = \Box\neg\text{Obs} \wedge \Diamond G, \tag{6}$$

where Obs and G are atomic propositions corresponding to unions of polygons in the state space $\mathcal{X}$, known respectively as the obstacles and goal. However, as an LTL formula, any solution trajectory is necessarily infinite, whereas classically the trajectory terminates upon reaching the set labeled G. In this paper we treat LTL specifications as appropriate for Problem 1 and as such, our methods produce trajectories of infinite duration. However, it is not difficult to adjust the results herein for other specification languages. In particular we expect that the method introduced in this section for re-using promising trajectories (cf. Algorithm 2) could improve the original application of the cross-entropy method to point-to-point motion planning, which we briefly demonstrate by numerical experiments.

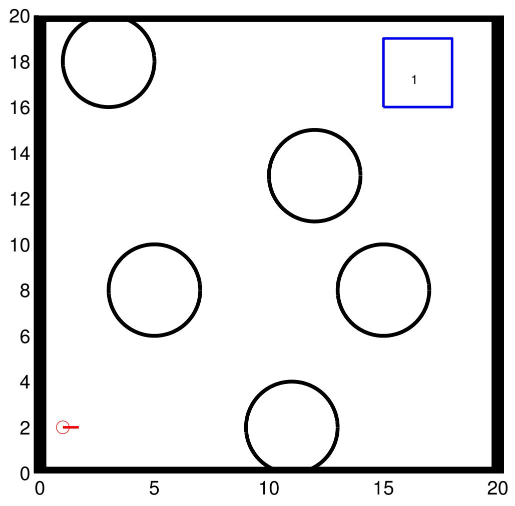

Figure 3: **The labeled workspace in which the reachability specification (6) is applied in our comparison of prior work and our method for re-use in a classical point-to-point problem.**

Table 1: **Run times for 7 trials of CE (prior work), 10 trials of CE with re-use, each with 20 iterations, and 20 feasible trajectories per iteration**

Method	min (s)	mean (s)	max (s)
CE (prior work)	354.2	466.1	582.5
CE with re-use	133.0	174.7	226.5

Using the specification (6) but allowing satisfying trajectories to have finite length, we compared the method described in [13] against a modified version in which our Algorithm 2 is applied to re-use parts of promising trajectories. The workspace providing labels is shown in Figure 3. The task specification (6) is decomposed by selecting the subformula $\psi := \Box\neg\mathrm{Obs}$, which clearly admits finite violation certificates—these are just the part of the trajectory from the initial state x_0 to the first state in collision with an obstacle. Hence a sampled trajectory $\mathbf{x}$ is promising if it reaches G (i.e., $\mathcal{L}(\mathbf{x}) \models \Diamond G$) but there is some obstacle with which it collides. The subroutine $\textsc{LastSafe}_\psi$ is then implemented by finding the last parameter from which the promising trajectory was constructed before having the collision. In our experiment we used the dynamics of Dubins car, as treated in a thorough example in Section 6 below. We also used the same trajectory parameters as described there, namely a finite sequence of time duration and control inputs. The maximum number of re-use attempts before returning **invalid** in Algorithm 2 is 10. Timing results after repeated trials are listed in Table 5.2. A substantial advantage from applying our method for trajectory re-use is apparent. The convergence time improves by approximately a factor of 3.

6. EXAMPLES

In this section we apply the presented methods to two examples: Dubins car and a point-mass model. For both, the

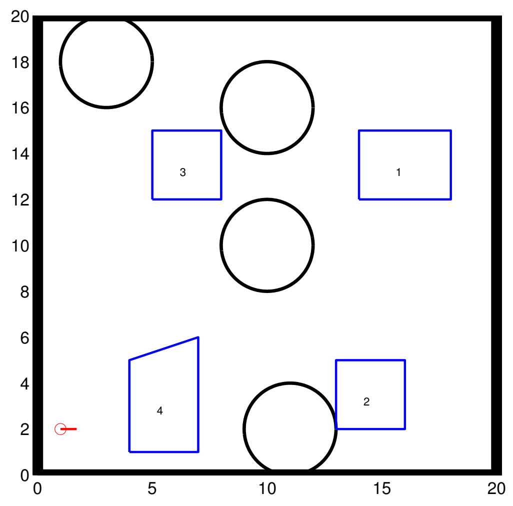

Figure 4: **The workspace $\mathcal{W}$ used in the Dubins car example. The initial state is indicated by a small red circle in the lower-left together with a line segment based at it. The polygons correspond to atomic propositions $P_1, \dots, P_4$, and the union of the circles is the collective obstacle and has atomic proposition Obs, all as used in the example specification (8).**

cost function is defined to be

$$J(\mathbf{x}(x_0, \mathbf{u})) = \sum_{t=0}^{T-1} \|x_{t+1} - x_t\|_2 + \|x_\tau - x_T\|_2, \qquad (7)$$

for a trajectory $x_0 x_1 \cdots x_{\tau-1}(x_\tau \cdots x_T)^\omega$, which is of the prefix-suffix form as introduced in Section 4.2. Ignoring the infinite repetition of the suffix, (7) is just the path length.

Algorithms 1 and 2 and the examples described in this section were implemented in the MathWorks MATLAB. LTL specifications are converted to deterministic Rabin automata using the tool LTL2DSTAR [11], which is freely available online at `http://www.ltl2dstar.de`. We created a Python script that parses output from LTL2DSTAR and generates a MATLAB function that returns true if a given word in prefix-suffix form is accepted by the automaton.

6.1 Dubins car

Dubins car has trajectories that are solutions of the system of ordinary differential equations

$$\begin{aligned}\dot{x} &= \cos\theta\\ \dot{y} &= \sin\theta\\ \dot{\theta} &= \omega,\end{aligned}$$

where the only control input is turning rate ω. Intuitively it describes a unicycle moving in the plane at constant speed and that is instantaneously oriented in the direction θ. Here we take the forward speed to be constant 1, but the treatment is easily modified for other magnitudes. The Dubins car model is both well-studied and well-motivated, e.g., since it captures basic requirements for aircraft to maintain lift.

The labeling function is defined according to the labeling shown in Figure 4, which we call the *workspace*. Notice that this defines $\mathcal{L}$ only in terms of position $(x, y) \in \mathbb{R}^2$, i.e., labels are independent of orientation. This matches our intuitive description of tasks in terms of the vehicle visiting or avoiding certain locations. The specification is

$$\Box \mathcal{W} \wedge \Box \neg \mathrm{Obs} \wedge \Box \Diamond P_1 \wedge \Box \Diamond P_2 \wedge \Box \Diamond P_3 \wedge \Box \Diamond P_4, \qquad (8)$$

where the labeled polygons in the figure have corresponding propositions $P_1, \ldots, P_4$. The union of the circles in the figure is the obstacle, collectively represented by the atomic proposition Obs. Occupancy of the workspace is enforced by $\Box \mathcal{W}$ where the atomic proposition is true only for positions inside the range $[0, 20] \times [0, 20]$.

We parameterize trajectories as sequences of 10 pairs of time durations and turning rates. The loop index is 5. (Recall the terminology of Section 4.3.) E.g., the first three trajectory parameter components (one per column)

$$\begin{pmatrix} 1 & 1 & 0.5 \\ 0 & \pi/4 & -\pi/4 \end{pmatrix}$$

describe motion that proceeds by steering forward for 1 second, turning left by $\pi/4$ radians (i.e., turning at a rate of $\pi/4$ radians per second for a total of 1 second), and finally turning right by $\pi/8$ radians.

The sampling distribution is a multivariate Gaussian. When generating samples, we discard those in which the time duration is less than zero (because negative input durations are not meaningful) or the absolute turning rate is greater than 1. Because control inputs are bounded, trajectories are of bounded curvature, which in addition to the constant forward speed renders the example nontrivial.

Providing some of the details for Line 8 of Algorithm 1 in this example, sampled trajectories are constructed as follows. First, sample a matrix of durations and turning rates. Then apply it as a piecwise constant steering input. Let x_τ be the state reached after applying the first four parameter components (i.e., first four columns of the sampled matrix), and let x_T be the state reached after applying all of the sampled inputs. (Recall that $p = 5$ is the loop index in this example.) The desired prefix-suffix is finally created by steering from x_T to x_τ. While our current implementation always makes this connection by a hard left turn, note that analytic solutions for optimal point-to-point steering ignoring obstacles is known for Dubins car [15].

Algorithms 1 and 2 were applied for 30 iterations. Example feasible trajectories found at various times during the optimization are shown in Figure 5. A plot showing the minimum path length (recall the cost function (7)) trajectory found for each iteration is Figure 6. Notice the drastic improvement following the first few iterations. Such jumps may arise from changes in the best homotopy class of trajectories found thus far. The plateau of minimum path lengths beginning at iteration 21 suggests that a local minimum was found.

6.2 Sampling waypoints

The example in this section demonstrates the case of sampling waypoints and then using local planners to create a full

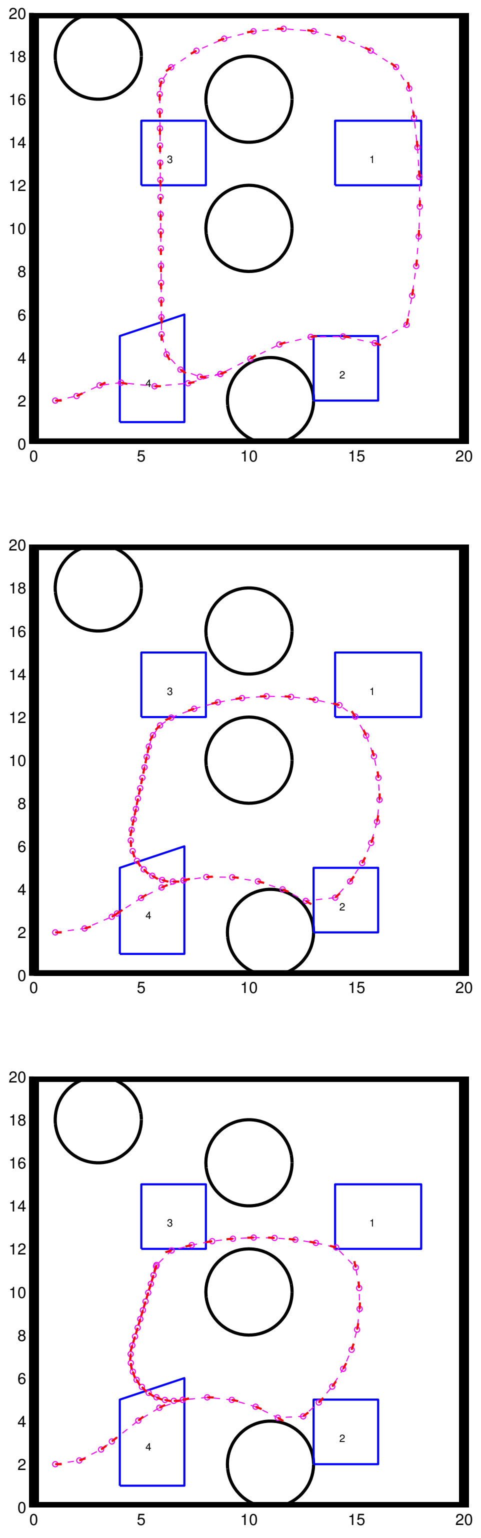

Figure 5: Demonstration of the improvement of trajectories. From top to bottom, feasible trajectories are shown from iterations 1, 15, and 30, respectively. The small line segments drawn along the path indicate orientation of Dubins car.

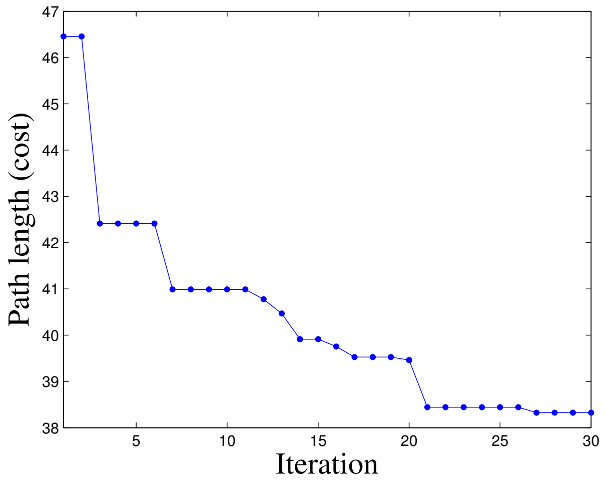

Figure 6: For each iteration of Algorithm 1 applied in the example of Section 6.1, the minimum path-length (cost) among feasible sampled trajectories is plotted. Compare with the sampled trajectories shown in Figure 5.

trajectory. This is distinct from the previous example because, rather than apply a finite sequence of control inputs and obtain the resulting states, we must solve a collection of boundary-value problems in order to obtain the trajectory. As apparent from Line 8 of Algorithm 1, this is a crucial step, and it has the advantage that manual design of the initial sampling distribution is more intuitive because the LTL specification in Problem 1 depends on a labeling function defined over states, not control inputs. (Recall the system model from Section 2.1.)

The workspace used here is the same as in the previous example, as shown in Figure 4. The same LTL specification (8) is also used.

We do not declare a particular dynamics (1) here. Instead, the crucial aspect is the manner of constructing trajectories, which proceeds as follows. Samples are obtained from a multivariate gaussian distribution. Each sample is a 2×10 matrix, where each column corresponds to a position in the workspace, and the sample as whole is a sequence of 10 waypoints. Means of the sampling distribution were initialized to locations selected manually as indicated by bold red asterisks in the top plot of Figure 7.

The initial state x_0 is $(1, 2)$, and the tie index is 3. For this example, the tie index has the easy interpretation as the second waypoint, i.e., the suffix of a trajectory is formed by connecting back to the second waypoint. Trajectories are constructed by cubic spline interpolation between waypoints and the initial state.

7. CONCLUSIONS AND FUTURE DIRECTIONS

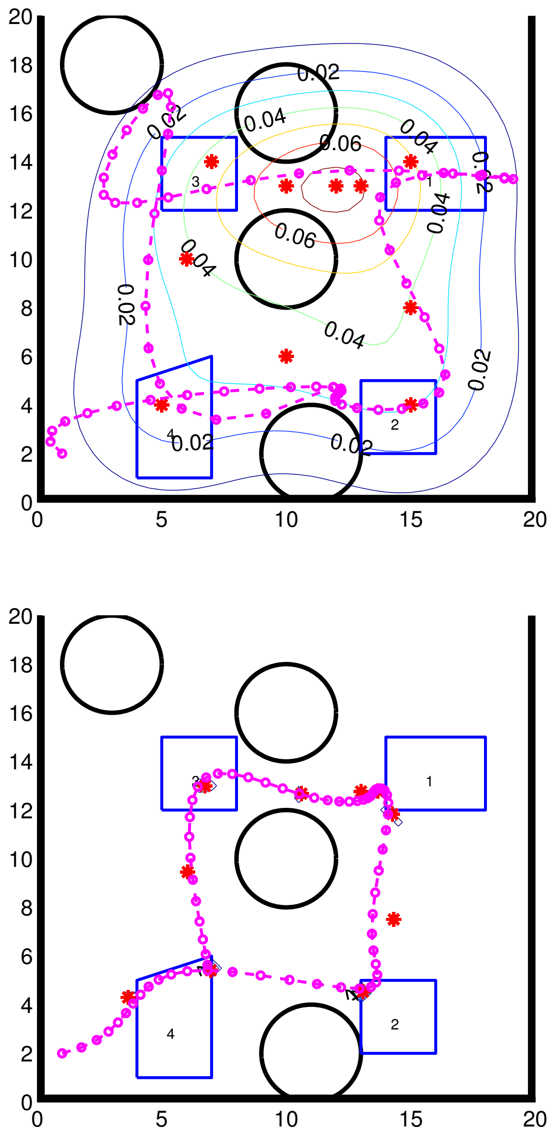

Figure 7: Demonstration of the improvement of trajectories for the example of Section 6.2. Both plots are displayed in the workspace from Figure 4, which is used in both examples. The sampling distribution is multivariate Gaussian, and the means of the distribution used in the respective iteration are indicated in each plot by bold red asterisks. Furthermore the covariance sublevels are indicated by a iso-curves like a topographic map. The top plot is from the first iteration. One of the feasible trajectories found is shown as a magenta dashed curve. The bottom plot is from iteration 15, and it also includes a feasible trajectory found at that iteration. The sampling distribution is very tight and the covariance levels are mostly occluded.

We presented a stochastic optimization algorithm for optimal trajectory generation for nonlinear systems operating in complex configuration spaces with linear temporal logic specifications. Importantly, each iteration of our algorithm runs in time polynomial in the size of the system and specification, and does not require the computation of a discrete abstraction or automaton. However, it may be beneficial to pre-compute an appropriate automaton to practically increase the efficiency of the algorithm. Additionally, we demonstrated how re-sampling parts of partially infeasible trajectories can result in empirically faster convergence compared to a state-of-the-art method.

One promising avenue for future work is planning for stochastic systems. A chance-constrained approach could take into account disturbances during the planning stage to compute a "robust" or "risk- aware" open-loop trajectory. Additionally, feedback control policies could be computed by considering a sampling distribution over value functions instead of trajectories.

Another direction is to generalize the notion of trajectory reuse. The notion of *edit distance* between strings, has been used to compute minimal corrections to strings so that they belong to a regular language [21]. Similar techniques for omega-regular languages could be used to develop a more principled approach to trajectory reuse in stochastic optimization.

We are in the process of performing detailed experimental comparisons with similar state-of-the-art methods, e.g., [9].

Acknowledgements

The authors thank Marin Kobilarov for providing source code implementing the CE method in MATLAB. This work was partially supported by United Technologies Corporation and IBM, through the industrial cyberphysical systems (iCyPhy) consortium.

8. REFERENCES

[1] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7):971–984, 2000.

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

[3] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol*, 40(4), September 2011.

[4] C. Belta and L. C. G. J. M. Habets. Controlling of a class of nonlinear systems on rectangles. *IEEE Trans. on Automatic Control*, 51:1749–1759, 2006.

[5] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics and Automation Magazine*, 18:55–64, 2011.

[6] E. M. Clarke and P. Zuliani. Statistical model checking for cyber-physical systems. In *Proc. of Automated Technology for Verification and Analysis*, pages 1–12, 2011.

[7] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45:343–352, 2009.

[8] R. Grosu and S. A. Smolka. Monte Carlo model checking. In *In Proc. of Tools and Algorithms for Construction and Analysis of Systems*, pages 271–286, 2005.

[9] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proc. of American Control Conf.*, 2012.

[10] S. Karaman, R. G. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *Proc. of IEEE Conf. on Decision and Control*, pages 2117–2122, 2008.

[11] J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363:182–195, 2006.

[12] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. on Automatic Control*, 53(1):287–297, 2008.

[13] M. Kobilarov. Cross-entropy motion planning. *Int. J. of Robotics Research*, 31:855–871, 2012.

[14] Y. Kwon and G. Agha. LTLC: Linear temporal logic for control. In *Proc. of HSCC*, pages 316–329, 2008.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

[16] N. Markey and P. Schnoebelen. Model checking a path. In *Proc. of the Int. Conf. on Concurrency Theory*, 2003.

[17] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas. Monte-Carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proc. of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 211–220, 2010.

[18] E. Plaku. Planning in discrete and continuous spaces: from LTL tasks to robot motions. In *Advances in Autonomous Robotics*. Springer, 2012.

[19] R. Y. Rubinstein and D. P. Kroese. *The Cross-entropy Method: A Unified Approach to Combinatorial Optimization*. Springer, 2004.

[20] S. Sankaranarayanan and G. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *Proc. of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, pages 125–134, 2012.

[21] R. A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, 1974.

[22] E. M. Wolff and R. M. Murray. Optimal control of nonlinear systems with temporal logic specifications. In *Proc. of Int. Symposium on Robotics Research*, 2013.

[23] E. M. Wolff, U. Topcu, and R. M. Murray. Optimization-based control of nonlinear systems with linear temporal logic specifications. In *Proc. of Int. Conf. on Robotics and Automation*, 2014.

[24] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Trans. on Automatic Control*, 2012.

[25] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *In Proc. 14th International Conference on Computer Aided Verification*, pages 223–235, 2002.