Simulation-Based LQR-Trees with Input and State Constraints

Philipp Reist and Russ Tedrake

Abstract— We present an algorithm that probabilistically covers a bounded region of the state space of a nonlinear system with a sparse tree of feedback stabilized trajectories leading to a goal state. The generated tree serves as a lookup table control policy to get any reachable initial condition within that region to the goal. The approach combines motion planning with reasoning about the set of states around a trajectory for which the feedback policy of the trajectory is able to stabilize the system. The key idea is to use a random sample from the bounded region for both motion planning and approximation of the stabilizable sets by falsification; this keeps the number of samples and simulations needed to generate covering policies reasonably low. We simulate the nonlinear system to falsify the stabilizable sets, which allows enforcing input and state constraints. Compared to the algebraic verification using sums of squares optimization in our previous work, the simulation-based approximation of the stabilizable set is less exact, but considerably easier to implement and can be applied to a broader range of nonlinear systems. We show simulation results obtained with model systems and study the performance and robustness of the generated policies.

I. INTRODUCTION

The class of algorithms proposed here and in previous work [1] aims at generating control policies for complicated nonlinear systems (e.g. robotic balancing tasks, walking and flying robots, etc.) when a linear controller is insufficient or the number of states prohibits the application of methods based on dynamic programming. We propose an algorithm that generates a lookup table control policy to stabilize any reachable initial condition within a bounded region of the state space of a nonlinear dynamic system to a goal state. This policy consists of a tree of feedback stabilized trajectories leading to the goal state.

The algorithm generates the tree using a randomized feedback motion planning approach that explores the bounded region using sampling and simulation, adding trajectories to the tree where needed. Simultaneously, it approximates the set of states around a given trajectory that can be stabilized to the goal using the feedback policy along the trajectory. One may think of this set as a volume around a trajectory in state space in which any initial condition can be brought to the goal. We call this volume the 'funnel' of a trajectory, inspired by [2] and [3]. Creating a policy that stabilizes any initial condition within the bounded region to the goal is equivalent to covering the region with funnels of trajectories leading to the goal. Estimating the funnels allows the algorithm to efficiently sample the bounded region and to only add trajectories where needed, resulting in a sparse representation of the policy.

P. Reist and R. Tedrake are with CSAIL at MIT, Cambridge MA, USA.

The algorithm is based on the algorithm presented in [1], which also generates a covering tree of trajectories leading to a goal state by reasoning about the funnels of trajectories. In [1], the funnels are verified using an algebraic method, using a sums of squares (SoS) optimization [4]. The formal method is more exact and provides proper stability guarantees in contrast to the sample-based, non-conservative approximation which only provides probabilistic stability guarantees. However, the simulation-based approach is considerably easier to implement than the formal approach, where many subtleties have to be accounted for. It is straightforward to enforce input constraints and check state constraints in simulation and the approach could also handle hybrid nonlinear systems (e.g. juggling or walking robots with discrete impact events).

In the following, we first review related work. Next, we present the key concepts and a description of the algorithm. Finally, we show the performance of the algorithm in simulation on two model underactuated systems.

A. Related Work

The algorithm is inspired by randomized motion planning, i.e. RRTs [5]. Randomized motion planning demonstrated that it can find feasible trajectories for nonlinear systems in nonconvex, high-dimensional search problems.

The approach is similar to the work of Atkeson [6], who uses trajectories as a sparse representation of the global value function of a system. The authors propose building a library of trajectories leading to a goal state and approximating the global value function with quadratic models along these trajectories. Trajectories are added based on how well two adjacent trajectories agree on the value function in between them. The resulting policy is not time based like in our approach, but transformed to a state dependent policy using a nearest neighbor lookup. The policy of the closest state in the trajectory library is executed, where the closeness is measured using a weighted Euclidean norm [7]. In contrast, the proposed algorithm aims not at producing globally optimal trajectories or estimating the global value function; we are more interested in designing a scalable algorithm that yields feasible policies.

As Atkeson points out, an advantage of using trajectories to represent a policy is that one avoids the problems that a discretization of state space generates. It is remarkable that the proposed algorithm generated policies for the cart-pole (4 states) swing-up with trees of between 20000 and 200000 nodes. The equivalent resolution of a state space discretization would be between 12 and 21 grid points per dimension, which is quite poor.

II. KEY CONCEPTS

A. Stabilizing the Goal State and Verification of the Basin of Attraction

In the following examples, we use goal states $\mathbf{x}_G$ which must be stabilizable. We stabilize a goal state using a linear time-invariant linear quadratic regulator (TILQR) and derive a formal verification of an approximated basin of attraction of the closed-loop nonlinear system at the goal state. The verification produces a description of the set of states around the goal for which the closed-loop nonlinear system is asymptotically stable. It would be costly to approximate this set using simulation as the time horizon to check for convergence approaches infinity. The formally derived basin allows us to simulate for a finite time to test for asymptotic convergence; we just have to check if the simulation ends up in the approximated basin.

We consider sampled-data feedback control of continuous-time dynamical systems and derive the controllers and approximated goal state basin of attraction in discrete-time. This is another point where the presented algorithm differs from [1], where controllers and funnels are derived in continuous-time.

1) TILQR: We derive the goal state controller as presented in [8]. Consider continuous-time, nonlinear system dynamics

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right), \tag{1}$$

where $\mathbf{f}$ is smoothly differentiable, $\mathbf{x} \in \mathbb{R}^n$ is the state of the system and $\mathbf{u} \in \mathbb{R}^m$ is the input to the system. We define

$$\bar{\mathbf{x}} := \mathbf{x} - \mathbf{x}_G, \quad \bar{\mathbf{u}} := \mathbf{u} - \mathbf{u}_G, \tag{2}$$

where $\mathbf{x}_G$, $\mathbf{u}_G$ are the nominal state and input of the system at the goal such that $\mathbf{f}\left(\mathbf{x}_G, \mathbf{u}_G\right) = 0$. We linearize and discretize the system (1) around the goal to obtain the discrete-time, time-invariant linear system dynamics

$$\bar{\mathbf{x}}_{k+1} = \mathbf{A}\bar{\mathbf{x}}_k + \mathbf{B}\bar{\mathbf{u}}_k, \tag{3}$$

where $\bar{\mathbf{x}}_k = \bar{\mathbf{x}}\left(t = k \cdot t_s\right)$ and t_s is the sample time. The system (3) must be stabilizable. The cost-to-go function to be minimized is

$$J\left(\bar{\mathbf{x}}_0\right) = \sum_{n=0}^{\infty} \left[\bar{\mathbf{x}}_n^{\mathrm{T}}\mathbf{Q}\bar{\mathbf{x}}_n + \bar{\mathbf{u}}_n^{\mathrm{T}}\mathbf{R}\bar{\mathbf{u}}_n\right] \tag{4}$$

$$\mathbf{Q} = \mathbf{Q}^{\mathrm{T}} \geq 0, \ \mathbf{R} = \mathbf{R}^{\mathrm{T}} > 0, \tag{5}$$

where $\bar{\mathbf{x}}_0$ is an initial state deviation and $\mathbf{Q}$, $\mathbf{R}$ are penalty matrices on the state and input deviations. The optimal cost-to-go for a linear system is given by

$$J^*\left(\bar{\mathbf{x}}_k\right) = \bar{\mathbf{x}}_k^{\mathrm{T}}\mathbf{S}_G\bar{\mathbf{x}}_k, \tag{6}$$

where $\mathbf{S}_G \geq 0$ is the unique stabilizing solution to the discrete algebraic Riccati equation

$$0 = \mathbf{Q} - \mathbf{S}_G + \mathbf{A}^{\mathrm{T}}\Big(\mathbf{S}_G \tag{7}$$
$$- \mathbf{S}_G\mathbf{B}\left(\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{S}_G\mathbf{B}\right)^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}_G\Big)\mathbf{A}.$$

The optimal feedback policy is given by

$$\bar{\mathbf{u}}_k^* = -\left(\mathbf{R} + \mathbf{B}^{\mathrm{T}}\mathbf{S}_G\mathbf{B}\right)^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{S}_G\mathbf{A}\bar{\mathbf{x}}_k = -\mathbf{K}_G\bar{\mathbf{x}}_k. \tag{8}$$

Both $\mathbf{S}_G$ and $\mathbf{K}_G$ are obtained using the Matlab `dlqr` command.

2) Verification of Goal State Basin of Attraction: In the following, we derive the verification of the approximated basin of attraction of the nonlinear closed-loop system at the goal state. We verify the set of states around the goal state for which the TILQR feedback policy achieves asymptotic stability of the nonlinear system. The presented derivation is analogous to [1], in a discrete-time setting.

Consider discrete-time, nonlinear closed-loop system dynamics of the form

$$\bar{\mathbf{x}}_{k+1} = \mathbf{f}\left(\bar{\mathbf{x}}_k\right)$$
$$\bar{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{x}_G, \tag{9}$$

where $\mathbf{x}_G$ is an equilibrium of the system, i.e. $\mathbf{f}\left(\mathbf{0}\right) = \mathbf{0}$. We define the basin as the largest set of states for which the optimal cost-to-go (6) decreases with every step

$$J^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k\right)\right) - J^*\left(\bar{\mathbf{x}}_k\right) < 0, \tag{10}$$

which represents a discrete-time formulation of a Lyapunov function. Every state within the goal basin should always take a step towards the goal, reducing it's cost-to-go and eventually reaching $\mathbf{x}_G$, implying asymptotic stability. We require this to hold over the hyper-elliptical domain $\mathcal{B}(\rho_G)$ defined as

$$\mathcal{B}(\rho_G) := \left\{\bar{\mathbf{x}} | \bar{\mathbf{x}}^{\mathrm{T}}\mathbf{S}_G\bar{\mathbf{x}} \leq \rho_G\right\}, \tag{11}$$

where $\mathbf{S}_G$ is the TILQR optimal cost-to-go matrix. We now search for the largest ρ_G for which (10) holds, i.e. we are looking for the largest $\mathcal{B}(\rho_G)$ completely contained in the basin of attraction of the nonlinear closed-loop system. To test whether (10) holds for a given ρ_G, we formulate the following sums of squares SoS [4] feasibility program:

$$\begin{aligned} \text{find} \quad & \mathrm{h}(\bar{\mathbf{x}}_k) \\ \text{subject to} \quad & J^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k\right)\right) - J^*\left(\bar{\mathbf{x}}_k\right) \\ & + \mathrm{h}(\bar{\mathbf{x}}_k)\left(\rho_G - \bar{\mathbf{x}}_k^{\mathrm{T}}\mathbf{S}_G\bar{\mathbf{x}}_k\right) < \epsilon\left\|\bar{\mathbf{x}}_k\right\|_2^2, \\ & \mathrm{h}(\bar{\mathbf{x}}_k) \geq 0, \end{aligned} \tag{12}$$

where $\mathrm{h}(\bar{\mathbf{x}}_k)$ is a polynomial of sufficiently large order and ϵ is a margin by which the left hand side of (12) needs to be a negative definite function of $\bar{\mathbf{x}}_k$. If the program is feasible, (10) holds within the hyper-ellipsis defined by $\mathcal{B}(\rho_G)$.

In order to execute the program, (12) has to be a polynomial expression. All summands in (12) except $J^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k\right)\right)$ are already polynomials. To make the whole expression polynomial, we approximate $J^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k\right)\right)$ using a Taylor series expansion

$$\hat{J}^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k\right)\right) = J^*\left(\mathbf{f}\left(\bar{\mathbf{x}}_k = 0\right)\right) + \left.\frac{\partial J^*\left(\mathbf{f}\left(\mathbf{x}\right)\right)}{\partial \mathbf{x}}\right|_{\mathbf{x}=0}\bar{\mathbf{x}}_k + \ldots \tag{13}$$

to sufficiently high order, omitting the higher order terms. We now perform a binary search on ρ_G to find the maximal

ρ_G that results in a feasible SoS program. Since this step does not take input constraints into account, we check if any of the states in $\mathcal{B}(\rho_G)$ violate the input constraints when applying (8) and reduce ρ_G if necessary.

The verification derived here is not straightforward to implement. A Matlab toolbox of the LQR-Tree algorithm using formal funnel verification may be used for this step and is made available at [9].

B. Stabilizing a Trajectory and Funnel Approximation

1) TVLQR: The sampled trajectories in the tree are stabilized using linear time-varying linear quadratic regulator (TVLQR) feedback policies. The derivation is analogous to the TILQR. Consider a sampled nominal trajectory of the system (1)

$$\mathbf{x}_{0k}, \mathbf{u}_{0k}, \quad k = 0, 1, ..., N, \quad (14)$$

where $N + 1$ is the number of elements. We linearize and discretize the system around this trajectory to obtain the discrete-time, time-varying linear system dynamics

$$\bar{\mathbf{x}}_{k+1} = \mathbf{A}_k \bar{\mathbf{x}}_k + \mathbf{B}_k \bar{\mathbf{u}}_k, \quad (15)$$

where

$$\bar{\mathbf{x}}_k := \mathbf{x}_k - \mathbf{x}_{0k}, \quad \bar{\mathbf{u}}_k := \mathbf{u}_k - \mathbf{u}_{0k}. \quad (16)$$

The cost-to-go to be minimized is

$$J(\bar{\mathbf{x}}_k) = \bar{\mathbf{x}}_N^{\mathrm{T}} \mathbf{Q}_N \bar{\mathbf{x}}_N + \sum_{n=k}^{N-1} \left[\bar{\mathbf{x}}_n^{\mathrm{T}} \mathbf{Q} \bar{\mathbf{x}}_n + \bar{\mathbf{u}}_n^{\mathrm{T}} \mathbf{R} \bar{\mathbf{u}}_n \right] \quad (17)$$

$$\mathbf{Q}_N = \mathbf{Q}_N^{\mathrm{T}} \geq 0, \ \mathbf{Q} = \mathbf{Q}^{\mathrm{T}} \geq 0, \ \mathbf{R} = \mathbf{R}^{\mathrm{T}} > 0, \quad (18)$$

where $\mathbf{Q}_N$, $\mathbf{Q}$ and $\mathbf{R}$ are penalty matrices on the final state deviation and state and input deviation from the nominal trajectory, respectively. The optimal cost-to-go is given by

$$J_k^*(\bar{\mathbf{x}}_k) = \bar{\mathbf{x}}_k^{\mathrm{T}} \mathbf{S}_k \bar{\mathbf{x}}_k, \quad (19)$$

where $\mathbf{S}_k \geq 0$ is given by the dynamic programming update

$$\mathbf{S}_k = \mathbf{Q} + \mathbf{A}_k^{\mathrm{T}} \Big(\mathbf{S}_{k+1} \quad (20)$$
$$- \mathbf{S}_{k+1} \mathbf{B}_k \left(\mathbf{R} + \mathbf{B}_k^{\mathrm{T}} \mathbf{S}_{k+1} \mathbf{B}_k \right)^{-1} \mathbf{B}_k^{\mathrm{T}} \mathbf{S}_{k+1} \Big) \mathbf{A}_k,$$

with the boundary condition $\mathbf{S}_N = \mathbf{Q}_N$. The optimal input is given by

$$\bar{\mathbf{u}}_k^* = - \left(\mathbf{R} + \mathbf{B}_k^{\mathrm{T}} \mathbf{S}_{k+1} \mathbf{B}_k \right)^{-1} \mathbf{B}_k^{\mathrm{T}} \mathbf{S}_{k+1} \mathbf{A}_k \bar{\mathbf{x}}_k$$
$$= -\mathbf{K}_k \bar{\mathbf{x}}_k, \quad (21)$$

where $\mathbf{K}_k \in \mathbb{R}^{m \times n}$ is the time-varying compensator matrix.

2) Simulation-Based Funnel Approximation: The funnel of a trajectory is the set of states around the trajectory which the TVLQR policy can get to the approximated goal basin without violating state and input constraints. After calculating the TVLQR policy for a trajectory, each nominal state $\mathbf{x}_{0k}$ of the trajectory has an associated optimal cost-to-go matrix $\mathbf{S}_k$ and compensator matrix $\mathbf{K}_k$. The matrix $\mathbf{S}_k$ together with the funnel parameter $\phi_k \in \mathbb{R}$ describes a hyper-ellipsis around the nominal state $\mathbf{x}_{0k}$. A state $\mathbf{x}$ is inside this ellipsis if

$$(\mathbf{x} - \mathbf{x}_{0k})^{\mathrm{T}} \mathbf{S}_k (\mathbf{x} - \mathbf{x}_{0k}) < \phi_k. \quad (22)$$
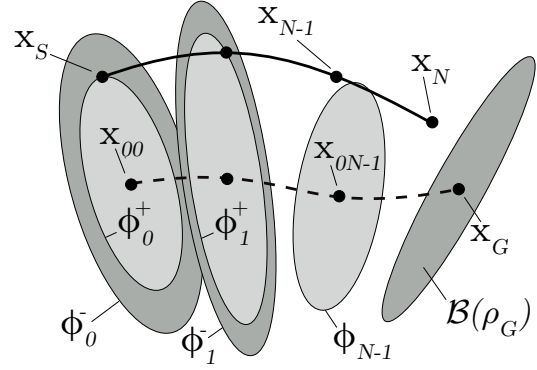

Fig. 1. Adjusting the funnel after a failed simulation. The simulated trajectory (solid black) failed to reach the goal basin $\mathcal{B}(\rho_N)$ using the policy starting at node $q : \mathbf{x}_{00}, \phi_0$. However, the funnel described by the darker grey ellipses defined by $\phi_{0,1}^-$ around the first two nodes of the policy's nominal trajectory (dashed line) predicted a successful simulation. Therefore, we adjust $\phi_{0,1}^-$ to $\phi_{0,1}^+$ according to (25), resulting in the light grey ellipses. The simulated state at time index $N - 1$ was not inside the ellipsis of node $N - 1$ and thus ϕ_{N-1} remains unchanged.

The union of the ellipses around all nominal states in a trajectory is the approximated funnel of the trajectory. We also use an ellipsis to describe the approximated goal basin $\mathcal{B}(\rho_G)$. But there is an important difference between the two parameters ρ_G and ϕ_k: ρ_G is fixed while the ϕ_k of a trajectory may change in an iteration of the algorithm. This is due to the fact that we approximate the funnel of a trajectory by falsification. In the following, we explain this important mechanism of the algorithm.

Consider a random sample $\mathbf{x}_S$ from the bounded region $\mathcal{R}$ to be covered. Assume that $\mathbf{x}_S$ happens to be inside the currently estimated ellipsis of node q of a trajectory in the tree. Node q is the starting point of a nominal trajectory and TVLQR feedback policy

$$\mathbf{u}_k = \mathbf{u}_{0k} - \mathbf{K}_k (\mathbf{x}_k - \mathbf{x}_{0k}), \quad k = 0, 1, ..., N, \quad (23)$$

where $\mathbf{u}_{00} = \mathbf{u}_{0q}$, $\mathbf{x}_{00} = \mathbf{x}_{0q}$, $\mathbf{K}_0 = \mathbf{K}_q$ and $N + 1$ is the number of nodes of the nominal trajectory. The trajectory ends up at the goal state, thus $\mathbf{x}_{0N} = \mathbf{x}_G$. We simulate the nonlinear system using $\mathbf{x}_S$ as initial condition and apply the feedback policy (23) for $t \in [0, N \cdot t_s]$. The simulation generates a sampled trajectory of the system with states $\mathbf{x}_S, \mathbf{x}_1, ..., \mathbf{x}_N$. After the simulation, we check if the system reaches the approximated basin of the goal state $\mathcal{B}(\rho_G)$:

$$(\mathbf{x}_N - \mathbf{x}_G)^{\mathrm{T}} \mathbf{S}_G (\mathbf{x}_N - \mathbf{x}_G) \leq \rho_G. \quad (24)$$

The input constraints are enforced using a saturation on the input; we then only have to check whether the trajectory $\mathbf{x}_S, \mathbf{x}_1, ..., \mathbf{x}_N$ violates any state constraints.

If the system reaches the goal basin and does not violate any state constraints, the simulation is successful. If the simulation fails, we adjust the funnel of the nominal trajectory starting at q setting

$$\phi_k = \bar{\mathbf{x}}_k^{\mathrm{T}} \mathbf{S}_k \bar{\mathbf{x}}_k, \quad k = 0, 1, ..., N - 1 \quad (25)$$

where $\bar{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{x}_{0k}$ and $\mathbf{x}_k$ are the states generated in the simulation. We only adjust ϕ_k if $\phi_k > \bar{\mathbf{x}}_k^{\mathrm{T}} \mathbf{S}_k \bar{\mathbf{x}}_k$. We only shrink the funnels and never expand them. A single simulation may shrink many ellipses ϕ_k of the nominal trajectory. This is illustrated in Fig. 1.

III. THE ALGORITHM

We now combine the key concepts and present the LQR-tree generating algorithm. A pseudocode overview is given in Algorithm 1.

Consider a tree that already consists of a few trajectories. Each node i in the tree corresponds to a node of a trajectory and consists of 6 elements:

1) $\mathbf{x}_{0i}$: Nominal state.
2) $\mathbf{u}_{0i}$: Nominal input.
3) $\mathbf{S}_i$: TVLQR cost-to-go matrix.
4) $\mathbf{K}_i$: TVLQR compensator matrix.
5) ϕ_i: Funnel parameter.
6) p_i: Pointer to parent node (next node in trajectory if time index advances by 1).

For the goal node $\mathbf{x}_G$, $\phi = \rho_G$ and $p = NULL$.

We generate a random sample $\mathbf{x}_S$ drawn uniformly from the bounded region $\mathcal{R}$. The random sample serves as an initial condition for simulating the nonlinear system as described in Section II-B.2. We build the array Γ to determine the simulation priority of the nodes. It's elements are defined as

$$\Gamma_i (\mathbf{x}_S) := \phi_i - (\mathbf{x}_S - \mathbf{x}_{0i})^{\mathrm{T}} \mathbf{S}_i (\mathbf{x}_S - \mathbf{x}_{0i}) , \ i = 1, \ldots, |\mathbf{T}| \tag{26}$$

where $|\mathbf{T}|$ is the total number of nodes in the tree. The Γ_i represent a measure of how 'far' the sample lies within the local elliptical description of the funnel of node i. Any node with a positive Γ_i claims to be able to get the sample to the goal basin $\mathcal{B}(\rho_G)$ when the node's policy is applied.

We simulate the sample if any element of Γ is positive. Only simulating the sample with policies of nodes with $\Gamma_i > 0$ speeds up the simulation step as many nodes that are likely to fail are ignored. If there are no nodes with $\Gamma_i > 0$, we proceed to the motion planning step, see Section III-A. We sort Γ in descending order and start simulating the sample using the policy of node q with the largest Γ_q in Γ, i.e. the first element of Γ.

If the policy of node q results in a successful simulation, we continue with the next node in Γ. After a fixed number (we use 10) of successful simulations, we proceed to the next random sample to test; it would be sufficient to proceed after the first success, however, we set the threshold higher to efficiently sample more funnels with a single sample.

If the policy of node q fails, we adjust the funnel according to Section II-B.2. When a new trajectory is added to the tree, we set it's $\phi_i \to \infty$ such that the trajectory's funnel covers the whole region $\mathcal{R}$. With more and more random samples falsifying the funnel of the trajectory, it approximates the true funnel in a non-conservative way. Using Γ to determine the simulation priority results in an efficient shrinking of the

---

Algorithm 1 Simulation-Based LQR-Trees

1: $[\mathtt{A}, \mathtt{B}] \Leftarrow$ linearization around $\mathbf{x}_G$, $\mathbf{u}_G$ and discretization
2: $[\mathtt{K_G}, \mathtt{S_G}] \Leftarrow \mathtt{dlqr}(\mathtt{A}, \mathtt{B}, \mathtt{Q}, \mathtt{R})$
3: $\rho_G \Leftarrow$ Approximated basin of $\mathbf{x}_G$
4: $\mathtt{T(1)} \Leftarrow \{\mathbf{x}_G, \mathbf{u}_G, \mathtt{S_G}, \mathtt{K_G}, \rho_G, NULL\}$
5: **for** $\mathtt{j} = 1$ to $\mathtt{maxIter}$ **do**
6:     $\mathtt{xSample} \Leftarrow$ random sample
7:     $\Gamma \Leftarrow$ build simulation priority array
8:     **for** $\mathtt{i} = 1$ to $|\Gamma > 0|$ **do**
9:         $\mathtt{xSim} \Leftarrow \mathtt{simTree}(\mathtt{T}, \mathtt{i}, \mathtt{xSample})$
10:         **if** $\mathtt{isInGoalBasin}$ and $\mathtt{constraintsOK}$ **then**
11:             continue; break if enough successes
12:         **else**
13:             $\mathtt{T} \Leftarrow \mathtt{adjustFunnel}(\mathtt{T}, \mathtt{i}, \mathtt{xSim})$
14:         **end if**
15:     **end for**
16:     **if** $\mathtt{noSimSuccessful}$ **then**
17:         $[\mathtt{iNear}] \Leftarrow \mathtt{distanceMetric}(\mathtt{T}, \mathtt{xSample})$
18:         $[\mathtt{T}, \mathtt{foundTraj}] \Leftarrow \mathtt{growTree}(\mathtt{T}, \mathtt{iNear}, \mathtt{xSample})$
19:         **if** $\mathtt{foundTraj}$ **then**
20:             Reset success counter
21:         **end if**
22:     **else**
23:         Increment success counter; terminate if converged
24:     **end if**
25: **end for**

ϕ_i as the value of Γ_i is a measure of how likely it is that the respective ϕ_i needs more shrinking.

If the sample $\mathbf{x}_S$ cannot be brought to the goal using the policies of the existing tree, we add a trajectory to the tree connecting the sample to the tree according to Section III-A.

The goal basin determining the success of a simulation does not need to be an approximated basin. For example, the goal state may not be stabilizable and the verification step is inapplicable. In this case, the user may provide a ρ_G and the algorithm generates a policy stabilizing initial conditions from the region $\mathcal{R}$ to the user defined goal region.

A single random sample is used to both shrink the funnels of existing trajectories and to determine whether the tree needs to be extended using motion planning. It is remarkable that this simulation-based mechanism achieves the same probabilistic coverage guarantees as the formal method [10]. The coverage mechanism for the two approaches remains the same.

A. Growing the Tree

If a random sample $\mathbf{x}_S$ never reaches $\mathcal{B}(\rho_N)$ using the existing policies in the tree, we need to extend the tree. We find the closest node i_{near} to $\mathbf{x}_S$ in the tree using a distance metric and use motion planning to generate a trajectory connecting the sample to i_{near}. In the current implementation, we use the distance metric described in [1] and [11]. It is based on a linearization of the system dynamics around the sample point and calculating the optimal control

cost-to-go from the nodes in the tree. Based on this cost, we choose the closest node to connect to.

The distance metric not only provides the closest node, but also an initial guess of the input $\hat{\mathbf{u}}^*$ to the system to reach the sample from the closest node. We further refine the open-loop trajectory given by $\hat{\mathbf{u}}^*$ with a direct collocation [12] implementation using SNOPT [13]. Direct collocation can handle both input and state constraints. If motion planning fails, we discard the sample as (temporarily) unreachable. As the tree grows, we expect that sample to be included in a funnel of the tree, if it is reachable.

After finding the connecting trajectory, we stabilize it using the discrete TVLQR controller derived in Section II-B.1, setting $\mathbf{Q}_N$ to $\mathbf{S}_{i_{\text{near}}}$. This generates a $\mathbf{K}_j, \mathbf{S}_j$ for every node in the new trajectory and we add the new nodes to the tree, setting $\phi_j \rightarrow \infty$, so that the funnel of the added trajectory covers the whole region $\mathcal{R}$.

B. Algorithm Termination Criteria

We terminate the algorithm after we encountered a fixed number of random samples that either successfully reached the goal or are unreachable, i.e. motion planning fails.

C. Using the Generated Tree

After termination of the algorithm, we obtain a tree $\mathbf{T}$ that probabilistically covers $\mathcal{R}$. To decide which stabilized trajectory to apply to a given initial condition $\mathbf{x}_{IC}$, we search for the node i with

$$i = \underset{j}{\arg\min} \left(\mathbf{x}_{IC} - \mathbf{x}_{0j}\right)^{\mathrm{T}} \mathbf{S}_j \left(\mathbf{x}_{IC} - \mathbf{x}_{0j}\right)$$

$$\text{subject to:} \quad \Gamma_j \left(\mathbf{x}_{IC}\right) > 0 \tag{27}$$

Thus we choose the policy starting at the node with the lowest cost-to-go for the initial condition whose funnel contains the initial condition. Since we approximate the funnel of a trajectory as we generate the tree, we can probabilistically guarantee that the policy we choose gets the initial condition to the goal.

IV. SIMULATION EXAMPLES

A. Simple Pendulum

We illustrate the algorithm using a simple, well visualizable system: the damped simple pendulum with input constraints. The parameters are mass $m = 1.0$ kg, length $l = 0.5$ m, gravity $g = 9.8$ m/s^2 and damping term $b = 0.1$ kgm^2/s. The torque applied at the pivot point is constrained to ± 3 Nm, requiring at least a single pump to swing up. The states are defined as the angle $x_1 := \theta$ and the angular velocity $x_2 := \dot{\theta}$. The pendulum is upright at rest when $x_1 = \pi$, $x_2 = 0$, which is the goal state. We set the cost matrices for the goal state LQR controller to $\mathbf{Q} = \texttt{diag(10,1)}$ and $\mathbf{R} = 15$.

The evolution of the tree is illustrated in snapshots in Fig. 2. Note that wrapping of the coordinate system was not taken into account. The algorithm takes about half an hour to converge, with no attempts made to optimize run time. A large part of this time is due to the high threshold of
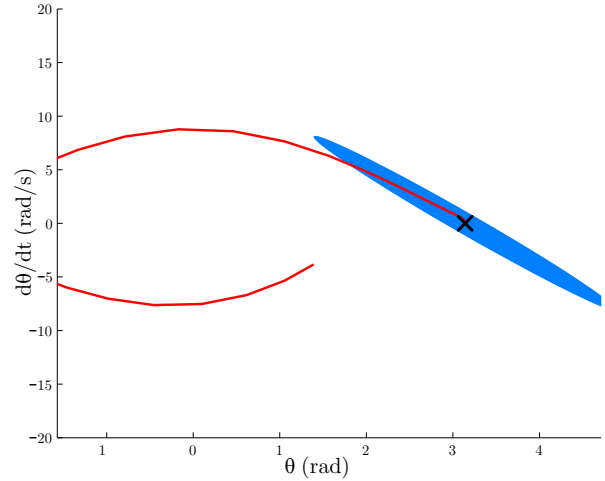
(a) Iteration 2, 1 trajectory, $|\mathbf{T}| = 41$. The final basin is plotted. The black cross is the goal state. The state space shown represents the subspace $\mathcal{R}$. Note that the funnel of the first added trajectory is not plotted, as it would cover the whole subspace.
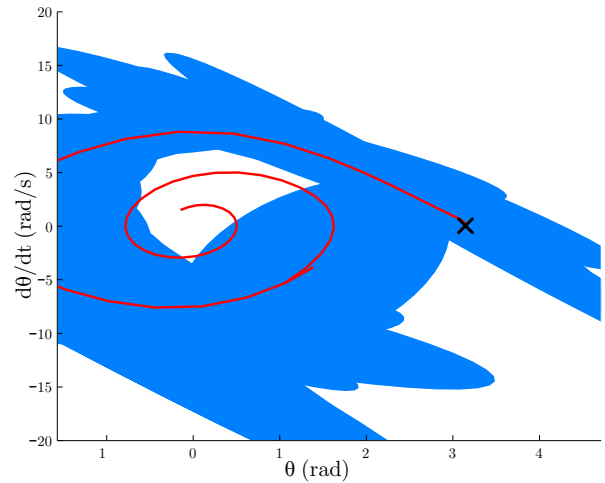
(b) Iteration 28, 2 trajectories, $|\mathbf{T}| = 81$. The funnel of the first trajectory was shrunken by unsuccessful simulations.
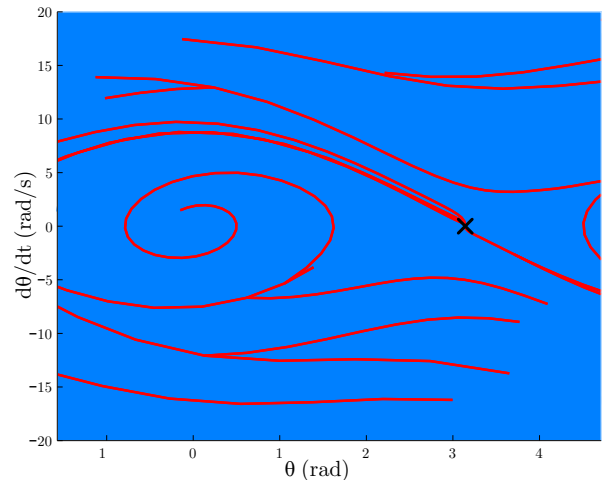
(c) Iteration 6086, 12 trajectories, $|\mathbf{T}| = 477$. 5000 consecutive random initial conditions were successfully brought to the goal state.

Fig. 2.   Tree evolution phase plots for simple pendulum

5000 consecutive successful simulations before the algorithm terminates. The average time for a random sample to be simulated for the whole tree was $0.27\,$s and stays almost constant over all iterations. Major spikes in time to simulate a sample can be observed after the tree has been grown and many funnels are adjusted.
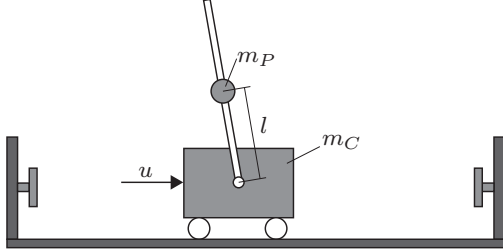
B. Cart-Pole

Fig. 3.   The cart-pole

We show the state constraint capability with the cart-pole, see Fig. 3. It consists of an actuated cart with an undamped simple pendulum attached. The parameters are: $m_C = 1.0\,$kg, $m_P = 1.0\,$kg, $l = 0.5\,$m, $g = 9.8\,$m/s^2 with the input u constrained to $\pm 30\,$N. The states are defined as cart position $x_1 := x$, pendulum angle $x_2 := \theta$, cart velocity $x_3 := \dot{x}$ and angular velocity $x_4 := \dot{\theta}$. The goal state is $x_1 = 0$, $x_2 = \pi$, $x_3 = 0$ and $x_4 = 0$, where the cart is at rest and the pendulum is pointing up. The position x is constrained to $\pm 0.45\,$m, which is given by a limited rail the cart can move on, a constraint often found in laboratory setups of this system. We chose $\mathbf{Q} = \texttt{diag}(50, 5, 40, 4)$ and $\mathbf{R} = 1$.

Note that we chose the region $\mathcal{R}$ to be smaller in horizontal position x than the state constraints. We also set the constraints for motion planning to be a shrunken version of the state constraints. This facilitates the algorithm in the sense that the controller can still take action on trajectories close to the state constraints without violating the constraints.

The algorithm takes significantly longer for the cart-pole than for the simple pendulum. It converged after 74375 iterations generating a tree with 11917 nodes and took about 32 hours on an average PC. We show the resulting tree in Fig. 4. For convergence, we required that the tree successfully gets 5000 consecutive random samples to the goal basin; we ignored unsuccessful samples that failed to connect to the tree. For this specific run, we had seven unreachable samples during the final 5000 iterations that could not be connected to the tree using motion planning. The large number of iterations needed is partly because of the doubled dimensionality compared to the pendulum, but a larger part is due to the increased complexity of adding the state constraints. Running the algorithm with the exact same parameters, only omitting the state constraints, it converged after 5821 iterations, producing a tree of 881 nodes in 2 hours.
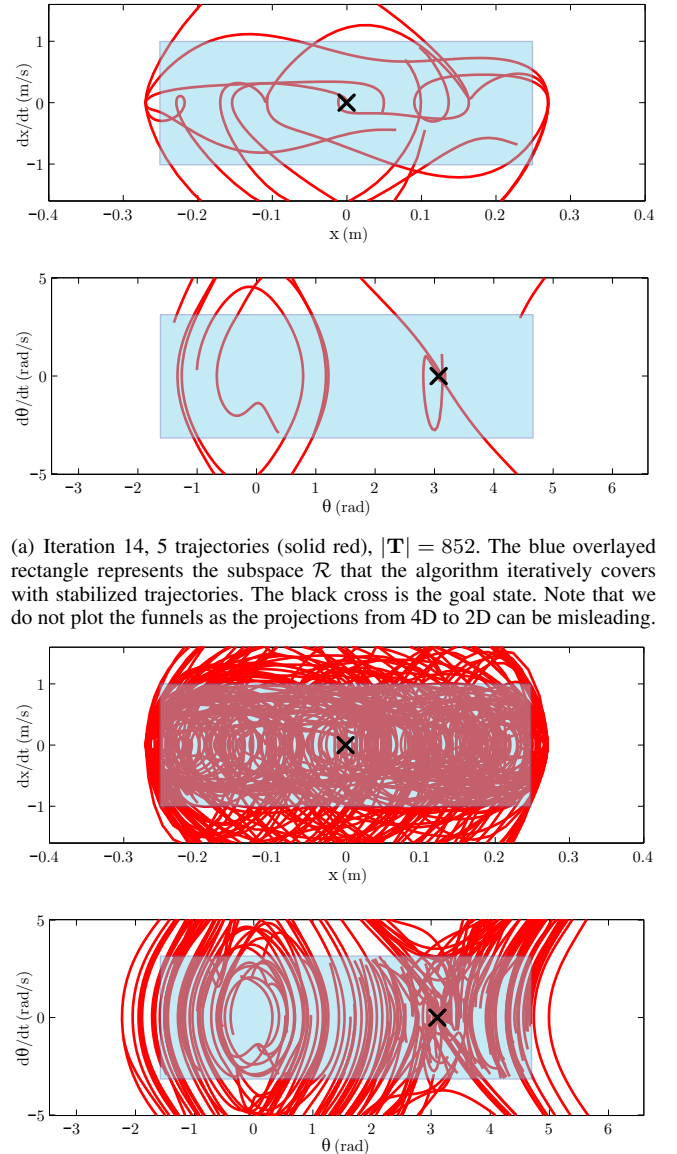
(a) Iteration 14, 5 trajectories (solid red), $|\mathbf{T}| = 852$. The blue overlayed rectangle represents the subspace $\mathcal{R}$ that the algorithm iteratively covers with stabilized trajectories. The black cross is the goal state. Note that we do not plot the funnels as the projections from 4D to 2D can be misleading.

(b) Iteration 74375, 134 trajectories (solid red), $|\mathbf{T}| = 11917$. The algorithm terminated after 5000 consecutive random initial conditions successfully reached the goal state.

Fig. 4.   Generated tree phase plots for the cart-pole

V. BASIC ROBUSTNESS STUDY

We compared the performance of the generated policy for different model parameters than what the policy was designed for. For simplicity, we just introduced a scale factor that scales the mass of the cart and mass and length of the pendulum, e.g. $m_{P,sim} = \kappa \cdot m_P$, $m_{C,sim} = \kappa \cdot m_C$, $l_{sim} = \kappa \cdot l$. We ran simulations with the scaled parameters from 10000 random initial conditions uniformly drawn from the region $\mathcal{R}$ the algorithm was designed for. We further replaced the success criteria of ending up within the final basin by checking the actual simulated state for convergence to the goal state. That means we simulated longer than the nominal time of the executed trajectory's policy and applied the goal state controller for the exceeding simulation time.

TABLE I
BASIC ROBUSTNESS STUDY FOR THE CART-POLE

κ	% Suc. (A)	% Suc. (B)	% Suc. (C)
0.8	2.82	13.57	4.31
0.85	4.92	99.63	14.50
0.9	78.48	100.00	46.15
0.95	99.34	100.00	43.40
1.0	99.75	100.00	46.59
1.05	99.27	100.00	63.07
1.1	95.92	99.98	53.06
1.15	83.56	99.80	25.31
1.2	51.10	97.89	14.67

We show the resulting percentages of successful simulations in Table I for the case of the tree generated with state constraints (A) and the tree without state constraints (B). We also ran the policy generated without state constraints on the constrained setup to compare the performance (C). Since we chose the weights in **Q** for the cart position and velocity to be quite large, it could be that the controllers performed similarly. However, the percentages show that generating the tree with explicitly taking the state constraints into account results in a better performance.

VI. DISCUSSION

It is unlikely that a hyper-ellipsis is the best geometrical primitive to describe the funnel around a trajectory. Simulation-based approximation of the funnel would allow exploring different primitives that could potentially yield tighter fits to the real funnel, further improving the sparsity of the resulting tree. The advantage of the hyper-ellipsis we propose is that they are simple to reason about geometrically and are founded on the TVLQR design, thus are dynamically plausible.

We show a set of simulated trajectories from random initial conditions for the cart-pole in the video accompanying this paper. It is interesting that for some initial conditions the simulation using the tree generated for the unconstrained problem seems to outperform the tree for the constrained problem, even without violating the state constraints. This could be due to effects of the imperfect distance metric and local minima during the motion planning step. A possible way to improve the results is to seed the tree with some carefully designed trajectories from key initial conditions, e.g. a good swing up trajectory for the cart-pole. However, the results we show were obtained without seeding. Currently, we are working on evaluating the performance of the proposed algorithm on a laboratory setup cart-pole system.

It would be interesting to explore the possibility of using the physical system instead of a simulation to test a random sample. This would generate a tree taking all the unmodeled dynamics of the system into account. Another idea is to use a hybrid approach, where we first generate a tree in simulation and then use data of failed initial conditions on the physical system to extend the tree.

REFERENCES

[1] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[2] M. Mason, "The mechanics of manipulation," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, Mar 1985, pp. 544–548.

[3] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential Composition of Dynamically Dexterous Robot Behaviors," *I. J. Robotic Res.*, vol. 18, no. 6, pp. 534–555, 1999.

[4] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo, *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2004.

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

[6] C. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1994, pp. 663–670.

[7] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2006.

[8] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. I*. Athena Scientific, 2005.

[9] LQR-Tree Matlab Toolbox. [Online]. Available: http://groups.csail.mit.edu/locomotion/software.html

[10] R. Tedrake, I. R. Manchester, M. M. Tobenkin, and J. W. Roberts, "LQR-Trees: Feedback motion planning via sums of squares verification," *To appear in the International Journal of Robotics Research*, 2010.

[11] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," *Accepted to ICRA*, 2010.

[12] J. Betts, *Practical Methods for Optimal Control using Nonlinear Programming*. ASME, 2002.

[13] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.