# LQR-Trees: Feedback Motion Planning via Sums of Squares Verification

Russ Tedrake, Ian R. Manchester, Mark Tobenkin, John W. Roberts
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA, 02139
Email: {russt,irm,mmt,jwr}@mit.edu

December 8, 2009

**Abstract**

Recent advances in the direct computation of Lyapunov functions using convex optimization make it possible to efficiently evaluate regions of stability for smooth nonlinear systems. Here we present a feedback motion planning algorithm which uses these results to efficiently combine locally-valid linear quadratic regulator (LQR) controllers into a nonlinear feedback policy which probabilistically covers the reachable area of a (bounded) state space with a region of stability, certifying that all initial conditions that are capable of reaching the goal will stabilize to the goal. We numerically investigate the properties of this systematic nonlinear feedback control design algorithm on model underactuated systems, prove the property of probabilistic coverage, and discuss extensions and implementation details of the basic algorithm.

## 1 Introduction

This paper aims to build on recent advances from systems theory and from randomized motion planning to design efficient and general algorithms for nonlinear feedback control synthesis in constrained nonlinear systems. Specifically, the controls verification community has recently developed a number of efficient algorithms for direct computation of Lyapunov functions for smooth nonlinear systems, using convex optimization[10, 23]. Here we demonstrate that these tools can be used in concert with a motion planning algorithm to automatically compute certificates for local regions of stability around a planned trajectory for even very complicated dynamical systems. Crossover between randomized motion planning algorithms and formal control verification opens up a number of interesting possibilities for algorithm development. In particular, we present the LQR-Tree algorithm, which uses locally optimal linear feedback control policies to stabilize planned trajectories computed by local trajectory optimizers, and

verification based on a sums of squares method to estimate the local regions of stability.

The aim of this work is to generate a class of algorithms capable of computing verified feedback policies for complicated[1] nonlinear systems which are not amenable to feedback linearization and with dimensionality beyond what might be accessible to grid-based algorithms like dynamic programming. The use of local trajectory optimizers and local feedback stabilization scales well to higher-dimensions, and reasoning about the feedback regions allows the algorithm to cover a bounded, reachable subset of state space with a relatively sparse set of trajectories. In addition, the algorithms operate directly on the continuous state and action spaces and perform verification algebraically, and thus are not subject to the pitfalls of discretization which limit accuracy and scalability. By considering feedback during the planning process, the resulting plans are certifiably robust to disturbances and quite suitable for implementation on real robots.

The following sections introduce the LQR-Tree algorithm. After a short background in Section 2, Section 3 starts by describing the design, stabilization, and local verification of a single feasible trajectory. Section 4 then introduces a method by which individual trajectories can be efficiently composed into a tree growing backwards from the goal, yielding a nonlinear feedback policy over the state space. Section 5 demonstrates the algorithm on a few model robotic control problems. Section 6 contains a proof that the algorithm has an important notion of completeness - it provably covers the reachable state space with a stabilizing controller. In the final sections, we discuss a few of the more subtle implementation details, and discuss possibilities for future extensions.

# 2 Background

## 2.1 Feedback motion planning

The last decade has seen rapid progress in motion planning algorithms, spurred by advances in sample-based, randomized algorithms like the probabilistic roadmaps (PRMs) [11] and the rapidly-exploring random trees (RRTs)[14]. These algorithms have demonstrated the ability to produce feasible open-loop trajectories from an initial condition to a goal state in impressively high-dimensional and convincingly non-convex search problems [13, 8, 29].

For implementation on real robots, open-loop trajectories generated by a motion planning system are typically stabilized by a trajectory stabilizing feedback system[2]. While this decoupled approach works for most problems, it is possible that a planned trajectory is not stabilizable, or very costly to stabilize compared to other, more desirable trajectories. *Feedback-motion-planning algorithms*, which explicitly consider the feedback stabilization during the planning

[1] E.g., systems subject to underactuation constraints, input saturations, and/or other kinodynamic constraints

[2] Note that an increasingly plausible alternative is real-time, dynamic re-planning.

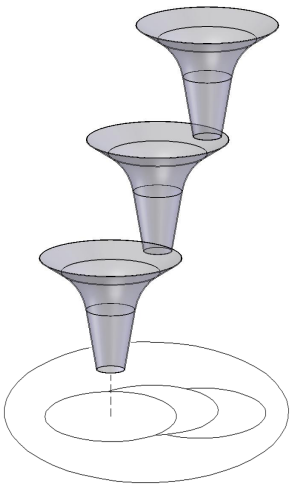


Figure 1: Cartoon of motion planning with funnels in the spirit of [6].

process, can avoid this pitfall, and as we will see, can potentially use a local understanding of the capabilities of the feedback system to guide and optimize the search in a continuous state space.

Mason popularized the metaphor of a "funnel" (see Figure 1) for a feedback policy which collapses a large set of initial conditions into a smaller set of final conditions [16]. In [24] this concept was formalized as "nested Lyapunov functions" and used to analyze the time convergence of nonlinear systems in which the equations of dynamics fould be represented as Fourier series. Burridge, Rizzi, and Koditschek then painted a beautiful picture of feedback motion planning as a sequential composition of locally valid feedback policies, or funnels, which take a broad set of initial conditions to a goal region [6]. At the time, the weakness of this approach was the difficulty in computing, or estimating by trial-and-error, the region of applicability - the mouth of the funnel, or preimage - for each local controller in a nonlinear system. Consequently, besides the particular solution in [6], these ideas have mostly been limited to reasoning about vector-fields on systems without dynamics [15].

## 2.2 Direct computation of Lyapunov functions

Burridge et al. also pointed out the strong connection between Lyapunov functions and these motion planning funnels [6]. For a dynamical system, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with $\mathbf{f}(0) = 0$, a Lyapunov function is a differentiable positive-definite output function, $V(\mathbf{x})$, for which $\frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x})$ is negative-definite. If these conditions are met over some ball in state space, $B_r$, containing the origin, then the origin is asymptotically stable. The ball, $B_r$, can then be interpreted as the preimage of the funnel. Lyapunov functions have played a central role in nonlinear control theory, since they allow verification of a system's stability without ever computing solutions of the system [12]. Indeed, if a dynamical system is asymptotically

stable within a region of attraction, then there exists a Lyapunov function certifying that fact. However, computing such a Lyapunov function for nonlinear systems is a challenging problem, and has been the focus of much of the history of nonlinear control theory.

The last few years has seen the emergence of a number of computational approaches to computing Lyapunov functions for nonlinear systems, often based on convex optimization (e.g., [10, 23]). One of these techniques, which forms the basis of the results reported here, is based on the realization that one can check the uniform positive-definiteness of a polynomial expression (even with constant coefficients as free parameters) using a *sums-of-squares* (SOS) optimization program [23]. Sums-of-squares programs can be recast into semidefinite programs which can be solved efficiently using interior point methods [20]; freely available libraries including SOSTOOLS [26] and the Polynomial Optimization Toolbox [17] makes it quite accessible to perform these computations in MATLAB. As we will see, the ability to check uniform positive (or negative) definiteness will offer the ability to verify candidate Lyapunov functions over a region of state space for smooth nonlinear systems.

## 2.3 Feedback Synthesis by Sums of Squares Optimization

The compatibility of sums of squares methods with semidefinite programming, and the associated interior-point method solvers, has had an notable impact on stability verification (see, e.g., [23, 28, 22, 27, 31, 32, 21] and many others). However, an open and active area of research is focused on extending these approaches to controller synthesis. Given a system, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}$, we wish to simultaneously generate a feedback, $\mathbf{u} = \pi(\mathbf{x})$ and a Lyapunov function, $V(\mathbf{x})$, such that $\frac{\partial V}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\pi(\mathbf{x})]$ is negative definite. This has proven to be a difficult problem because the set of $V(\mathbf{x})$ and $\pi(\mathbf{x})$ satisfying these conditions may not be convex or even connected [28]. To combat this, a number of solutions haven been proposed based on density functions [25], iterative algorithms for model-predictive control design [7].

Region-of-attraction verification for polynomial systems has recently been explored as a search for Lyapunov functions using sum-of-squares techniques [31]. The optimization required is bilinear in the decision variables, and so optimization methods will only find a local solution. Simulation of the system dynamics can help find good initial guesses for the Lyapunov function [32].

This paper takes a different approach to control synthesis. Rather than designing a nonlinear feedback directly in the optimization, we rely on classical linear quadratic regulator (LQR) [1] to design a series of locally-valid controllers and compose these controllers together using feedback motion planning. This has the advantage that it may work for hopelessly non-convex control problems, such as navigation through a complicated obstacle field, robotic manipulation, or legged locomotion over rough terrain, where the randomized motion planning algorithms have demonstrated success. We prove in Section 6 that, under some mild assumptions, the feedback synthesis will eventually stabilize every reachable point for a smooth nonlinear system.

### 2.4 Other related work

In other related work, [2] used local trajectory optimizers and LQR stabilizers with randomized starting points to try to cover the space, with the hope of verifying global optimality (in the infinite resolution case) by having consistent locally quadratic estimates of the value function on neighboring trajectories. The conditions for adding nodes in that work were based on the magnitude of the value function (not the region of guaranteed stability). In the work described here, we sacrifice direct attempts at obtaining optimal feedback policies in favor of computing good-enough policies which probabilistically cover the reachable state space with the basin of attraction. As a result, we have stronger guarantees of getting to the goal and considerably sparser collections of sample paths.

Computing reachable sets of uncertain nonlinear system has been explored using a dynamic game framework [19]. In general this requires computing level sets of a Hamilton-Jacobi-Bellman-Isaacs partial differential equation, which must be approached numerically for all but the simplest systems.

## 3 Linear Feedback Design and Verification

The LQR-Tree algorithm is based on the ability to efficiently design trajectories of the robot through state space, to stabilize those trajectories using linear quadratic regulator (LQR) feedback design, and to estimate the preimage of the feedback controller. In this section, we develop that procedure for a single trajectory.

### 3.1 Stabilizing a Goal State

Consider a controllable, smoothly differentiable, nonlinear system:

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

We first examine stabilizing a goal-state with an infinite horizon LQR controller, and approximating the closed loop basin of attraction. Consider a stabilizable goal state, $\mathbf{x}_G$, with $\mathbf{u}_G$ defined so that $\mathbf{f}(\mathbf{x}_G, \mathbf{u}_G) = 0$. Define

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

Now, linearize the system around $(\mathbf{x}_G, \mathbf{u}_G)$ to yield the dynamics:

$$\dot{\bar{\mathbf{x}}}(t) \approx \mathbf{A}\bar{\mathbf{x}}(t) + \mathbf{B}\bar{\mathbf{u}}(t). \tag{3}$$

Define the quadratic regulator cost-to-go function as

$$J(\bar{\mathbf{x}}') = \int_0^\infty \left[\bar{\mathbf{x}}^T(t)\mathbf{Q}\bar{\mathbf{x}}(t) + \bar{\mathbf{u}}^T(t)\mathbf{R}\bar{\mathbf{u}}(t)\right] dt, \tag{4}$$
$$\mathbf{Q} = \mathbf{Q}^T \geq 0, \mathbf{R} = \mathbf{R}^T > 0, \bar{\mathbf{x}}(0) = \bar{\mathbf{x}}'$$

The optimal cost-to-go function for the linear system is given by

$$J^*(\bar{\mathbf{x}}) = \bar{\mathbf{x}}^T \mathbf{S} \bar{\mathbf{x}}, \tag{5}$$

where $\mathbf{S}$ is the positive-definite solution to the equation:

$$0 = \mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S} \tag{6}$$

(given by the MATLAB `lqr` function). The optimal feedback policy for the linear system is given by

$$\bar{\mathbf{u}}^* = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}\bar{\mathbf{x}} = -\mathbf{K}\bar{\mathbf{x}}. \tag{7}$$

### 3.1.1 Time-Invariant LQR Verification

We acquire an estimate of the basin of attraction of the LQR controller by constructing a function, $V(\mathbf{x})$, which is a valid Lyapunov function for the nonlinear system over some sub-level set:

$$\mathcal{B}_G(\rho) = \{\mathbf{x} | 0 \leq V(\mathbf{x}) \leq \rho\} \tag{8}$$

To demonstrate asymptotic stability, we require:

- $V(\mathbf{x})$ is positive definite in $\mathcal{B}_G(\rho)$,
- $\dot{V}(\mathbf{x})$ is negative definite in $\mathcal{B}_G(\rho)$.

Such a function demonstrates the goal state is stabilized, and furthermore, all initial conditions in $\mathcal{B}_G(\rho)$ will converge to $\mathbf{x}_G$ [30].

Here we use $V(\mathbf{x}) = J^*(\bar{\mathbf{x}})$, the linear optimal cost-to-go function, which is a Lyapunov function for the linear system and hence serves well as a local Lyapunov function candidate for the nonlinear system. By construction, this choice satisfies the positive definite constraint on $V(\mathbf{x})$. For the second condition, first observe that

$$\dot{V}(\mathbf{x}) = \dot{J}(\bar{\mathbf{x}}) = 2\bar{\mathbf{x}}^T\mathbf{S}\mathbf{f}(\mathbf{x}_G + \bar{\mathbf{x}}, \mathbf{u}_G - \mathbf{K}\bar{\mathbf{x}}). \tag{9}$$

Ideally, the constraint on $\dot{J}^*$ can be written as a single inequality:

$$\dot{J}^*(\bar{\mathbf{x}}) - \chi_{\mathcal{B}_G(\rho)}(\mathbf{x}) < 0 \qquad \forall\, \mathbf{x} \neq 0 \tag{10}$$

where $\chi_{\mathcal{B}_G(\rho)}$ is the characteristic function for the approximated basin:

$$\chi_{\mathcal{B}_G(\rho)}(\mathbf{x}) = \begin{cases} 0 & \mathbf{x} \in \mathcal{B}_G(\rho) \\ \infty & \text{o.w.} \end{cases} \tag{11}$$

As is common in the sums-of-squares (SOS) programming literature, we approximate this inequality with the SOS feasibility program [23]:

$$\begin{aligned} \text{find} \quad & h(\bar{\mathbf{x}}) \\ \text{subject to} \quad & \dot{J}^*(\bar{\mathbf{x}}) + h(\bar{\mathbf{x}})\left(\rho - J^*(\bar{\mathbf{x}})\right) \leq -\epsilon\|\mathbf{x}\|_2^2, \\ & h(\mathbf{x}) \geq 0 \end{aligned} \tag{12}$$

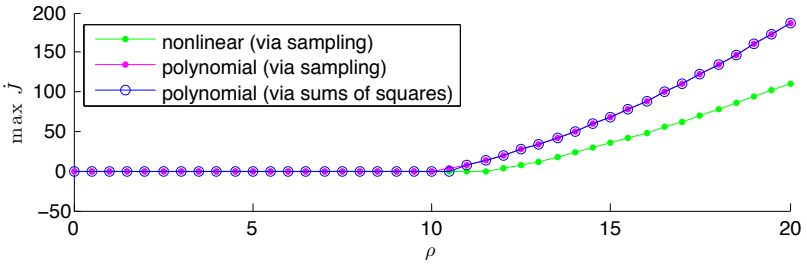



Figure 2: Polynomial verification of LTI feedback on the damped simple pendulum ($m = 1$ kg,$l = 0.5$ m, $b = 0.1$ m$^2$ kg s$^{-1}$, $g = 9.8$ m s$^{-2}$, $\mathbf{Q} =$ diag$([10, 1])$, $\mathbf{R} = 15$, $N_f = 3$, $N_m = 2$)

where $h$ is a polynomial of sufficiently large order, $N_h$, and $\epsilon > 0$ is a margin by which $\dot{V}(\mathbf{x})$ must be negative definite. In the sequel, we will make frequent use of Lagrange multipliers such as $h(\cdot)$ to approximate characteristic functions. In the case where $\mathbf{f}$ is polynomial in $\mathbf{x}$ and $\mathbf{u}$, a feasible $h(\cdot)$ provides exact verification of our desired condition, though it is important to note that SOS programs do not search over all positive polynomials.

For the case where $\mathbf{f}$ is not polynomial, we can approximate this condition using a Taylor expansion of $\mathbf{f}$, with order $> 1$. Let us denote the closed-loop dynamics at

$$\mathbf{f}_c(\mathbf{x}) = \mathbf{f}(\mathbf{x}, \mathbf{u}_G - \mathbf{K}(\mathbf{x} - \mathbf{x}_G)). \tag{13}$$

We denote the truncated Taylor expansion of $\mathbf{f}_c$ to order $N_f > 1$:

$$\mathbf{f}_c(\mathbf{x}_G + \bar{\mathbf{x}}) \approx \hat{\mathbf{f}}_c(\mathbf{x}_G + \bar{\mathbf{x}}). \tag{14}$$

For example, when $N_f = 2$, we have

$$\hat{\mathbf{f}}_c(\mathbf{x}_G + \bar{\mathbf{x}}) = f_c(\mathbf{x}_G) + \sum_i \bar{x}_i \left[\frac{\partial f_c(\mathbf{x})}{\partial x_i}\right]_{\mathbf{x}=\mathbf{x}_G} + \frac{1}{2}\sum_{i,j} \bar{x}_i \bar{x}_j \left[\frac{\partial^2 f_c(\mathbf{x})}{\partial x_i \partial x_j}\right]_{\mathbf{x}=\mathbf{x}_G}, \tag{15}$$

where we also have that $f_c(\mathbf{x}_G) = 0$. When the polynomial approximation of the dynamics is used, we replace the $J^*(\bar{\mathbf{x}})$ in the SOS program above with $\hat{J}^*(\bar{\mathbf{x}})$. Figure 2 presents a numerical exploration of such a certificate for the simple pendulum. For increasing values of $\rho$, the maximum value of $\hat{\dot{J}}^*(\bar{\mathbf{x}})$ in $\mathcal{B}_G$ is plotted. Also plotted is the maximum value achieved by $\hat{\dot{J}}(\bar{\mathbf{x}})$ sampling the boundary of this ellipse, and the maximum value of the true (non-polynomial) $\dot{J}^*(\bar{\mathbf{x}})$ sampled similarly.

Finally, we formulate a convex optimization to find the largest region $\mathcal{B}_G(\rho)$ over which the second condition is also satisfied:

$$\begin{aligned} \text{maximize} \quad & \rho \\ \text{subject to} \quad & \hat{\dot{J}}^*(\bar{\mathbf{x}}) + h(\bar{\mathbf{x}}) \left( \rho - \hat{J}^*(\bar{\mathbf{x}}) \right) \leq \epsilon \|\mathbf{x}\|_2^2 \\ & \rho > 0 \\ & h(\bar{\mathbf{x}}) \geq 0 \end{aligned} \tag{16}$$

This optimization can be performed via a line-search on $\rho$. If the corresponding SOS feasibility program succeeds, $\rho$ is increased, otherwise, $\rho$ is reduced.

Although we use Taylor expansions here for generality, many problems of interest can be verified exactly without this approximation using other basis functions besides polynomials; for robotic manipulators, the trigonemetric polynomails are a logical choice[18]. It is possible to bound the error of the Taylor approximation for the purposes of verification[3], though we do not treat explore that possibility here. Although the polynomial approximation of the nonlinear dynamics need not be conservative, we find that the remainder of the proposed verification procedure is conservative enough in practice (by approximating the true basin with only the largest verified ellipse, and by using Lagrange multipliers of small, finite order).

### 3.2 Trajectory Optimization

In order to increase the set of states which can reach the goal beyond the time-invariant LQR design, we will design and stabilize a feasible trajectory for the system with an initial condition outside of the verified region of attraction of the time-invariant controller. Trajectory design can be accomplished readily by trajectory optimization algorithms including shooting methods, multiple shooting methods, and direct collocation methods [4], all of which are quite mature, and can perform well on even very complicated nonlinear systems. Given the nonlinear system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, we solve for a finite-length feasible trajectory of the system $\mathbf{x}_0(t)$, $\mathbf{u}_0(t)$ over the interval $[t_0, t_f]$ which (locally) optimizes a cost function of the form

$$J = \int_{t_0}^{t_f} \left[ 1 + \mathbf{u}^T \mathbf{R} \mathbf{u} \right] dt, \tag{17}$$

subject to the constraint that $\mathbf{x}(t_f) = \mathbf{x}_G$. Throughout the paper, we perform trajectory optimization with a multiple shooting method with the initial control tape initialized to a random input of random duration; this constrained nonlinear optimization is implemented with sequential quadratic programming, using SNOPT [9].

### 3.3 Time-Varying LQR

Given a nominal trajectory, $\mathbf{x}_0(t)$, $\mathbf{u}_0(t)$, (a solution of equation 1) over a finite time interval, $t \in [t_0, t_f]$, we stabilize the trajectory using a time-varying LQR

controller. Linearizing the system around the trajectory, we obtain:

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_0(t), \quad \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_0(t), \tag{18}$$

$$\dot{\bar{\mathbf{x}}}(t) \approx \mathbf{A}(t)\bar{\mathbf{x}}(t) + \mathbf{B}(t)\bar{\mathbf{u}}(t) \tag{19}$$

and define a quadratic regulator (tracking) cost function:

$$J(\bar{\mathbf{x}}', t') = \bar{\mathbf{x}}^T(t_f)\mathbf{Q}_f\bar{\mathbf{x}}(t_f) + \int_{t'}^{t_f} \left[\bar{\mathbf{x}}^T(t)\mathbf{Q}\bar{\mathbf{x}}(t) + \bar{\mathbf{u}}^T(t)\mathbf{R}\bar{\mathbf{u}}(t)\right] dt, \tag{20}$$

$$\mathbf{Q}_f = \mathbf{Q}_f^T > \mathbf{0}, \mathbf{Q} = \mathbf{Q}^T \geq \mathbf{0}, \mathbf{R} = \mathbf{R}^T > \mathbf{0}, \bar{\mathbf{x}}(t') = \bar{\mathbf{x}}'.$$

In general, $\mathbf{Q}$ and $\mathbf{R}$ could easily be made a function of time as well. With time-varying dynamics, the resulting cost-to-go is time-varying. It can be shown that the optimal cost-to-go, $J^*$, is given by

$$J^*(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T\mathbf{S}(t)\bar{\mathbf{x}}, \quad \mathbf{S}(t) = \mathbf{S}^T(t) > \mathbf{0}. \tag{21}$$

where $\mathbf{S}(t)$ is the solution to

$$-\dot{\mathbf{S}} = \mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S}, \quad \mathbf{S}(t_f) = \mathbf{Q}_f, \tag{22}$$

and the optimal feedback policy is given by

$$\bar{\mathbf{u}}^*(t) = -\mathbf{R}^{-1}\mathbf{B}^T(t)\mathbf{S}(t)\bar{\mathbf{x}}(t) = -\mathbf{K}(t)\bar{\mathbf{x}}(t) \tag{23}$$

### 3.3.1 Time-Varying LQR Verification

In order to describe the stabilization of a finite-length nomimal trajectory, rather than aymptotic stability, we specify a bounded goal region of state space, $\mathcal{B}_f$, and search for a time-varying region, $\mathcal{B}(t)$, for which the closed-loop system obeys:

$$\mathbf{x}(t) \in \mathcal{B}(t) \implies \mathbf{x}(t_f) \in \mathcal{B}_f \qquad t \in [t_0, t_f] \tag{24}$$

We define the goal region in terms of the positive definite quadratic form $\mathbf{Q}_f$ - the terminal cost from the LQR design - and scalar $\rho_f > 0$:

$$\mathcal{B}_f = \{\mathbf{x} | (\mathbf{x} - \mathbf{x}_0(t_f))^T\mathbf{Q}_f(\mathbf{x} - \mathbf{x}_0(t_f)) \leq \rho_f\} \tag{25}$$

We now search for a time-varying region $\mathcal{B}(t)$ for $t \in [t_0, t_f]$ satisfying:

$$\mathbf{x}(t) \in \mathcal{B}(t) \implies \mathbf{x}(t_f) \in \mathcal{B}_f. \tag{26}$$

We search for such a region in terms of a (now time-varying) positive definite, radially unbounded function $V(\mathbf{x}, t)$. At a moment in time, the region is determined as a sub-level-set:

$$\mathcal{B}(\rho(\cdot), t) = \{\mathbf{x} | 0 \leq V(\mathbf{x}, t) \leq \rho(t)\}. \tag{27}$$

Where $\rho(t)$ is chosen to ensure (26) holds. We also define the goal region similarly:

$$\mathcal{B}_f = \{\mathbf{x} | 0 \leq V(\mathbf{x}, t_f) \leq \rho_f\}. \tag{28}$$

Naturally, this implies that $\rho(t_f) \leq \rho_f$. When $\mathbf{x}(t_f) = \mathbf{x}_G$, a stabilizable goal, one can choose $\mathbf{Q}_f$ and $\rho_f$ to ensure trajectories land in an infinite-horizon LQR controller's basin of attraction.

To ensure (26), we choose the certificate function $\rho(t) : [t_0, t_f] \mapsto \mathbb{R}^+$ with the following conservative properties:

- $\rho(t_f) \leq \rho_f$
- $\rho(t)$ is discontinuous at finitely many points, $\tau_1, \ldots, \tau_M$, and:

$$\lim_{t \uparrow \tau_m} \rho(t) \leq \lim_{t \downarrow \tau_m} \rho(t) \tag{29}$$

- The right derivative, $\partial_+ \rho(t)$, exists and:

$$V(\mathbf{x}, t) = \rho(t) \implies \partial_+ V(\mathbf{x}, t) \leq \partial_+ \rho(t) \tag{30}$$

This ensures that the value $V(\mathbf{x}, t)$ decreases faster along trajectories than our level-set $\rho(t)$. It is useful to note that given two such certificates, $\rho_1(t)$ and $\rho_2(t)$, their point-wise maximum $\rho(t) = \max\{\rho_1(t), \rho_2(t)\}$ is also a certificate, which verifies that $\mathcal{B}(\rho_1(\cdot), t) \cup \mathcal{B}(\rho_2(\cdot), t)$ satisfies (26).

Again, we choose here to use $V(\mathbf{x}, t) = J^*(\mathbf{x}, t)$; which is positive definite by the LQR derivation which ensures $\mathbf{S}(t)$ is uniformly positive definite. Now we have

$$\dot{J}^*(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T \dot{\mathbf{S}}(t) \bar{\mathbf{x}} + 2\bar{\mathbf{x}}^T \mathbf{S}(t) \mathbf{f}\left(\hat{\mathbf{x}}_0(t) + \bar{\mathbf{x}}, \hat{\mathbf{u}}_0(t) - \mathbf{K}(t)\bar{\mathbf{x}}\right) \tag{31}$$

Here, even if $\mathbf{f}$ is polynomial in $\mathbf{x}$ and $\mathbf{u}$ and the input tape $\mathbf{u}_o(t)$ was polynomial, our analysis must make use of $\mathbf{x}_0(t)$, $\mathbf{S}(t)$, and $\mathbf{K}(t)$ which are the result of numerical integration (e.g., with `ode45` in Matlab). We will approximate this temporal dependence with (elementwise) piecewise polynomials using splines of order $N_t$, where $N_t$ is often chosen to be 3 (cubic splines), with the knot points at the timesteps output by the variable step integration, which we denote $t_0, t_1, ..., t_N$, with $t_N = t_f$, e.g.:

$$\forall t \in [t_k, t_k + 1], \quad S_{ij}(t) \approx \sum_{m=0}^{N_t} \alpha_{ijm} (t - t_k)^m = \hat{S}_{ij}(t), \tag{32}$$

$$\hat{J}^*(\bar{\mathbf{x}}, t) = \sum_i \sum_j \bar{x}_i \bar{x}_j \hat{S}_{ij}(t). \tag{33}$$

To approximate $\dot{J}^*(\bar{x}, t)$, we first take the Taylor expansion in $\bar{\mathbf{x}}$ to arrive at a polynomial at each knot, $t_k$, and then interpolate these coefficients of the resulting polynomials as piece-wise polynomials of time.

### 3.3.2 Optimizing for $\rho(t)$

In general, we would desire the largest possible $\rho(t)$, perhaps measured by the volume of state-space included in $\mathcal{B}(\rho(\cdot), t)$. However, searching over a suitably

parameterized $\rho(t)$ over all time-segments while verifying invariance appears intractable. Instead, we find a piecewise-linear[3] $\rho(t)$ over the intervals $[t_k, t_{k+1})$:

$$\rho_k(t) = \beta_{1k} t + \beta_{0k} \tag{34}$$

$$\rho(t) = \begin{cases} \rho_k(t), & \forall t \in [t_k, t_{k+1}) \\ \rho_f, & t = t_f, \end{cases} \tag{35}$$

which we construct backwards in time, starting with $k = N-1$. For a given $\rho_k$, it is is easy to test if $\rho_k(t_{k+1}) = \beta_{1k} t_{k+1} + \beta_{0k} \leq \rho(t_{k+1})$. The more complicated step of verification requires:

$$\hat{J}^*(\bar{\mathbf{x}}, t) = \rho_k(t) \implies \dot{\hat{J}}^*(\bar{\mathbf{x}}, t) \leq \dot{\rho}_k(t) = \beta_{1k} \quad \forall t \in [t_k, t_{k+1}) \tag{36}$$

which can be tested by the SOS feasibility program:

$$\begin{aligned} \text{find} \quad & h_1, h_2, h_3 \\ \text{subject to} \quad & \dot{\hat{J}}^*(\bar{\mathbf{x}}, t) - \dot{\rho}_k(t) + h_1(\bar{\mathbf{x}}, t)(\rho_k(t) - \hat{J}^*(\bar{\mathbf{x}}, t)) \\ & \qquad + h_2(\bar{\mathbf{x}}, t)(t - t_k) + h_3(\bar{\mathbf{x}}, t)(t_{k+1} - t) \leq 0 \\ & h_2(\bar{\mathbf{x}}, t) \geq 0 \\ & h_3(\bar{\mathbf{x}}, t) \geq 0 \end{aligned} \tag{37}$$

Choosing the three Lagrange multipliers, $h_1, h_2, h_3$ to be polynomials of sufficient order. This feasibility enables a search for $\beta_{\cdot k}$. The feasible region is generally non-convex, but there are some guarantees as to the local minima.

Our assumptions about the system dynamics ensure that:

$$\exists\, \epsilon_k > 0 \;\text{ s.t. }\; \forall\, t \in [t_k, t_{k+1})\; V(\mathbf{x}, t) \leq \epsilon_k \implies \partial_+ \dot{V}(\mathbf{x}, t) \leq 0. \tag{38}$$

As a result, searching for a non-zero constant $\rho_k(t)$ should always be feasible. We formulate the optimization:

$$\begin{aligned} \underset{\alpha_k}{\text{maximize}} \quad & \rho_k(t) = \alpha_k \\ \text{subject to} \quad & \alpha_k \leq \rho(t_{k+1}) \\ & \text{SOS Program (37)} \end{aligned} \tag{39}$$

This program is amenable to line-search, similar to the LTI basin. In general, the optimal value will not be found, but a value, $\alpha_k^*$, approaching or exceeding any $\epsilon_k$ can be found. A piece-wise constant $\rho(t)$ would be defined by these values, but has the disadvantage that $\rho(t)$ will monotonically decrease.

This value, $\alpha_k^*$, can be used as an initial guess to optimizing $\rho_k(t)$ over a class of polynomials. A particularly successful optimization strategy has been

[3]Higher order polynomials are of course possible, but for small time-steps, piecewise linear appears to perform well.

to:

$$\begin{aligned} \underset{\beta_{\cdot k}}{\text{maximize}} \quad & \rho_k(t_k) = \beta_{1k} t_k + \beta_{0k} \\ \text{subject to} \quad & \rho_k(t_{k+1}) \leq \rho(t_{k+1}) \\ & \text{SOS Program (37)} \end{aligned} \tag{40}$$

Provides coefficients $\beta^*$ such that:

$$\rho_k(t) = \max\left\{\alpha_k^*, \beta_{1k} t + \beta_{0k}\right\} \quad k = N-1, \ldots, 1 \tag{41}$$

form a valid, piece-wise linear certificate, which avoids locally maximizing $\rho_k$ at the expense of later steps.

### 3.4 Verification with Input Saturation

This section describes a modified conditions for certificates covering systems with saturated inputs. We examine the single-output case where the control law $\mathbf{u}(t) = \mathbf{u}_0(t) - \mathbf{K}(t)\bar{\mathbf{x}}(t)$ is mapped through:

$$g(\mathbf{u}(t)) = \begin{cases} \mathbf{u}_+ & \mathbf{u}(t) \geq \mathbf{u}_+ \\ \mathbf{u}_- & \mathbf{u}(t) \leq \mathbf{u}_- \\ \mathbf{u}(t) & \text{o.w.} \end{cases}$$

We begin by calculating the smallest level-set $\rho$ for which either constraint becomes active. e.g.:

$$\underset{\bar{\mathbf{x}}'}{\text{minimize}} \quad J^*(\bar{\mathbf{x}}') \tag{42}$$

$$\text{subject to} \quad \mathbf{u}_+ = \mathbf{u}_0 - \mathbf{K}\bar{\mathbf{x}}' \tag{43}$$

Let $\rho_{\max}$ be the solution to this is a convex QP. Similarly, $\rho_{\min}$ can be computed as the smallest level-set for which $\mathbf{u}^*(t) = \mathbf{u}_-$. These values allow for a case-wise analysis of the saturation.

We then compute a separate SOS condition for each active constraint, and choose the largest $\rho$ which satisfies all.

Let

$$\dot{J}_-(\bar{\mathbf{x}}) = \frac{\partial}{\partial \bar{\mathbf{x}}} J(\mathbf{x}) f(\bar{\mathbf{x}}, \mathbf{u}_-) + \frac{\partial}{\partial t} J(\bar{\mathbf{x}}) \tag{44}$$

$$\dot{J}_K(\bar{\mathbf{x}}) = \frac{\partial}{\partial \bar{\mathbf{x}}} J(\bar{\mathbf{x}}) f(\mathbf{x}, \mathbf{u}_0 - \mathbf{K}\bar{\mathbf{x}}) + \frac{\partial}{\partial t} J(\bar{\mathbf{x}}) \tag{45}$$

$$\dot{J}_+(\bar{\mathbf{x}}) = \frac{\partial}{\partial \bar{\mathbf{x}}} J(\bar{\mathbf{x}}) f(\bar{\mathbf{x}}, \mathbf{u}_+) + \frac{\partial}{\partial t} J(\bar{\mathbf{x}}) \tag{46}$$

Then the verification inequality (13) can be replaced by

$$\dot{J}_-(\bar{\mathbf{x}}) + h_1(\bar{\mathbf{x}})(\rho - J(\bar{\mathbf{x}})) + h_2(\bar{\mathbf{x}})(\mathbf{u}_- - \mathbf{u}_0 + \mathbf{K}\bar{\mathbf{x}}) \leq -\epsilon\|\mathbf{x}\|_2^2 \tag{47}$$

$$\begin{aligned} \dot{J}_K(\bar{\mathbf{x}}) + h_3(\bar{\mathbf{x}})(\rho - J(\mathbf{x})) + h_4(\bar{\mathbf{x}})(\mathbf{u}_0 - \mathbf{K}\bar{\mathbf{x}} - \mathbf{u}_-) & \\ + h_5(\mathbf{u}_+ - \mathbf{u}_0 + \mathbf{K}\mathbf{x}) \leq -\epsilon\|\mathbf{x}\|_2^2 & \end{aligned} \tag{48}$$

$$\dot{J}_+(\bar{\mathbf{x}}) + h_6(\bar{\mathbf{x}})(\rho - J(\bar{\mathbf{x}})) + h_7(\bar{\mathbf{x}})(\mathbf{u}_0 - \mathbf{K}\bar{\mathbf{x}} - \mathbf{u}_+) \leq -\epsilon\|\mathbf{x}\|_2^2 \tag{49}$$

with each $h_i(\bar{\mathbf{x}})$ sum-of-squares. The terms with the $h_i(\bar{\mathbf{x}})$ multipliers serve to focus the attention of the verification on the regions of interest. Analogous additions can be made to the LTV verification inequality (37).

Similar SOS programs can be set up for higher input dimensions, however in general there will be $3^m$ different combinations of saturated and unsaturated inputs, where $m$ is the control dimension, which could quickly become computationally prohibitive for systems with many inputs.

# 4 The LQR-Tree Algorithm

Given the ability to design, stabilize, and verify a trajectory from an initial condition to the goal, the LQR-Tree algorithm proceeds by growing a randomized tree of stabilizing controllers to reach the goal. Because both the feedback design and verification work backwards in time, we grow the tree backwards, starting from the goal. The result is that the tree becomes a large web of stabilizing controllers which grab initial conditions and pull them towards the goal (with formal certificates of stability for the nonlinear, continuous state and action system).

The goal of the algorithm is to cover an entire region of interest, the set of points from which the goal state is reachable, or a specified bounded subset of this region, with this stabilizing controller. In Section 6, we define this property as "probabilistic feedback coverage". To achieve this, we grow our tree in the fashion of a RRT, where new subgoals are chosen at random from a uniform distribution over the state space. Unlike the RRTs, we have additional information from the estimated regions of stability (funnel), and we can immediately discard sample points which are sampled inside the previously verified region. Define $\mathcal{C}_k$ as the estimated region of stability for the entire tree after $k$ iterations of the algorithm. Discarding random samples from $\mathcal{C}_k$ occurs rapidly, due in part to our choice of a simple ellipsoidal estimate for the funnels, and causes a dramatic improvement in the sparsity of the randomized tree.

One consequence of this sampling strategy is the the algorithm which attempts to connect the random sample back to the goal must be capable of establishing this connection over some finite distance through state space. For this reason, we use the trajectory optimization algorithms described briefly in Section 3.2 to grow a trajectory from the sampled point forward until it connects to the existing tree. These algorithms are local, and are subject to local minima, but we prove in Section 6 that they are sufficient to obtain the desired coverage property of the algorithm. If the trajectory optimization fails to connect to the tree, the sampled point is discarded in order to allow for the possibility that the tree is not reachable from that sampled point; if the tree is reachable then this region will be sampled again in the future with a more extensive tree available to connect to. The trajectory optimization is set to timeout with a failure after some relatively small number of major iterations of the SQP algorithm, in order to facilitate fast sampling and discarding of new samples.

The most important component of the trajectory optimization which at-

tempts to connect to the tree is the "final tree constraint", which specifies that the optimized trajectory must connect back to the tree, but is free to connect to any part of the continuous manifold of points which form the tree. On the $k$th iteration of the algorithm, given a candidate trajectory terminating in $\mathbf{x}_0(t_f)$, we enforce the vector constraint that

$$\left[\mathbf{x}_0(t_f) - \underset{\mathbf{x}' \in \mathcal{T}_{k-1}}{\operatorname{argmin}} ||\mathbf{x}_0(t_f) - \mathbf{x}'||_2\right] = \mathbf{0}, \tag{50}$$

where $\mathcal{T}_k$ is the set of all points in the tree after $k$ iterations of the algorithm. A well implemented optimization algorithm can satisfy this constraint by designing a trajectory which connects to the tree, then continue to "walk" the terminal condition along the constraint manifold of the tree to maximize the objective.

## 4.1 The Algorithm

The algorithm proceeds by producing a tree, $T$, with nodes containing the tuples, $\{\mathbf{x}, \mathbf{u}, \mathbf{S}, \mathbf{K}, \rho_c, i\}$, where $J^*(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T \mathbf{S} \bar{\mathbf{x}}$ is the local quadratic approximation of the value function, $\bar{\mathbf{u}}^* = -\mathbf{K}\bar{\mathbf{x}}$ is the feedback controller, $J^*(\bar{\mathbf{x}}, t) \leq \rho(t)$ is the funnel where $\rho(t)$ is described by the vector of polynomial coefficients $\rho_c$, and $i$ is a pointer to the parent node.

**Algorithm 1** LQR-Tree ($\mathbf{f}$,$\mathbf{x}_G$,$\mathbf{u}_G$,$\mathbf{Q}$,$\mathbf{R}$)

1: $[\mathbf{A}, \mathbf{B}] \Leftarrow$ linearization of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ around $\mathbf{x}_G, \mathbf{u}_G$
2: $[\mathbf{K}, \mathbf{S}] \Leftarrow \text{LQR}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$
3: $\rho_c \Leftarrow$ level-set computed as described in section 3.1.1
4: T.init($\{\mathbf{x}_g, \mathbf{u}_g, \mathbf{S}, \mathbf{K}, \rho_c$, NULL$\}$)
5: **for** $k = 1$ to K **do**
6: $\quad \mathbf{x}_{rand} \Leftarrow$ random sample
7: $\quad$ **if** $\mathbf{x}_{rand} \in \mathcal{C}_k$ **then**
8: $\quad\quad$ continue
9: $\quad$ **end if**
10: $\quad [t, \mathbf{x}_0(t), \mathbf{u}_0(t)]$ from trajectory optimization with a "final tree constraint"
11: $\quad$ **if** $\mathbf{x}_0(t_f) \notin \mathcal{T}_k$ **then**
12: $\quad\quad$ continue
13: $\quad$ **end if**
14: $\quad [\mathbf{K}(t), \mathbf{S}(t)]$ from time-varying LQR
15: $\quad \rho_c \Leftarrow$ level-set computed as in section 3.3.1
16: $\quad i \Leftarrow$ pointer to branch in $T$ containing $\mathbf{x}_0(t_f)$
17: $\quad$ T.add-branch($\mathbf{x}_0(t), \mathbf{u}_0(t), \mathbf{S}(t), \mathbf{K}(t), \rho_c, i$)
18: **end for**

### 4.2 Executing LQR-Tree feedback

The resulting LQR-Tree policy is a function,

$$\mathbf{u} = \pi_T(\mathbf{x}, t_T, b), \tag{51}$$

with internal controller state $t_T$ - the time corresponding to the current execution - and $b$ - a unique identifier to the currently active branch of the tree. At the onset of control from an initial condition $\mathbf{x}_i$, these controller variables are initialized as

$$[t_T, b] = \underset{b \in T, t \in [t_0^b, t_f^b]}{\operatorname{argmax}} \; c(\mathbf{x}_i, t, b) \tag{52}$$

$$c(\mathbf{x}, t, b) = \rho^b(t) - \left(\mathbf{x}_i - \mathbf{x}_0^b(t)\right)^T \mathbf{S}^b(t) \left(\mathbf{x} - \mathbf{x}_0^b(t)\right), \tag{53}$$

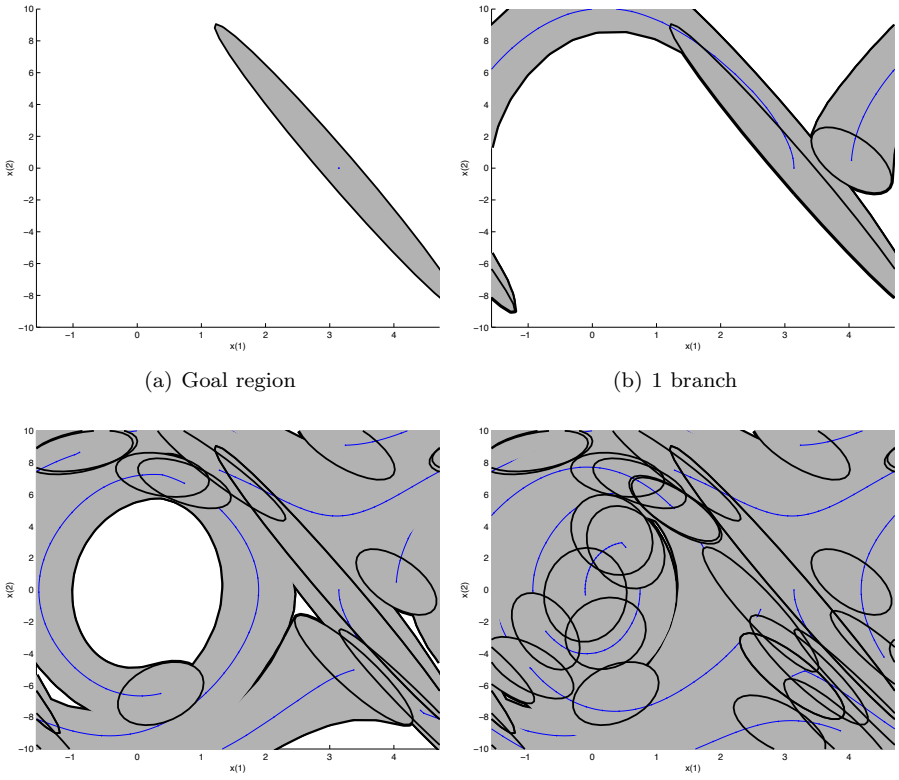
where superscript $b$ denotes the nominal times, trajectories, and funnels on branch $b$ of the tree (where $b$ is from an index set of branches). We refer to the quantity, $c(t)$, as the *confidence*. The policy, of course, can only be guaranteed to succeed if the certificate is valid ($c(t_F) \geq 0$). Although it is tempting to continually search over $t_T$ as the policy is executed, this can lead to undesirable properties such as chattering. Therefore, in practice, we execute the policy with time evolving naturally ($\dot{t}_T = 1$) unless a disturbance occurs which takes the state outside of the funnel, in which case we re-evaluate $t_T$ and $b$. Similarly, $b$ is held constant until the point in which execution of a finite-length tape concludes and $b$ is set to the parent trajectory of the previous branch, closer to the goal.

## 5 Simulations

We illustrate the operation of the algorithm on a two-dimensional toy problem, for which we can carefully plot the entire state space and covered set, $\mathcal{C}_k$, on each iteration of the algorithm. Here we use the swing-up task for a torque-limited simple pendulum, $I\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = \tau$, with $m = 1, l = .5, b = .1, I = ml^2, g = 9.8$. Here $\mathbf{x} = [\theta, \dot{\theta}]^T$ and $\mathbf{u} = \tau$. The parameters of the LQR-tree algorithm were $\mathbf{x}_G = [\pi, 0]^T$, $\mathbf{u}_G = 0$, $\mathbf{Q} = diag([10, 1])$, $\mathbf{R} = 20$, $N_f = 3$, $N_m = 2$, $N_x = 3$, $N_S = 3$.

Figure 3(a) shows the basin of attraction (gray shaded region) after computing the linear time-invariant (LTI) LQR solution around the unstable equilibrium. Figure 3(b) shows the entire nominal trajectory (blue) to the first random sample point, and the funnels that have been computed for this trajectory. Note that the state-space of the pendulum lives on a cylinder, and that the trajectory (and basin of attraction) wraps around from the left to the right. Plots (c-d) show the basin of attraction as it grows to fill the state space.

In this example, the entire space is probabilistically covered (1000 random points chosen sequentially were all in the basin of attraction) after the tree contained just 13 branches in the tree. On average, the algorithm terminates after 13.25 branches have been added for the simple pendulum with these parameters. For contrast, [5] shows a well-tuned single-directional RRT for the




(a) Goal region

(b) 1 branch

(c) 6 branches

(d) 13 branches

Figure 3: An LQR-tree for the simple pendulum. The $x$-axis is $\theta \in [-\pi/2, 3\pi/2]$ (note that the state wraps around this axis), and the $y$-axis is $\dot{\theta} \in [-10, 10]$. The gray shaded region represents the covered set, $\mathcal{C}_k$, (aka, the "funnels"), after $k$ branches have been added to the tree.

simple pendulum which has 5600 nodes. However the cost of adding each node is considerably greater here than in the traditional RRT, dominated by the line search used to maximize the estimated region of stability.

# 6 LQR-Trees achieve "Probabilistic Feedback Coverage"

Consider a smooth system

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{54}$$

with $\mathbf{x}(t) \in \mathcal{X}$ and $\mathbf{u}(t) \in \mathcal{U}$ where $\mathcal{X}$ and $\mathcal{U}$ are open subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. We assume that for all piecewise continuous functions $u : [0, \infty) \to \mathcal{U}$,

a unique solution of (54) exists, and solutions have continuous dependence on controls and initial conditions. Note that there are many polynomial differential equations for which this is not true, however for well-modelled physically-motivated systems these are natural assumptions.

We wish to consider the performance of an iterative motion-planning algorithm for the task of planning motions to a particular goal state $\mathbf{x}_f \in \mathcal{X}$.

We define the following sets:

- $\mathcal{R}(\mathbf{x}_f)$: the set of points from which a state $\mathbf{x}_f$ is reachable, possibly in infinite time. I.e. $\mathbf{x}_0 \in \mathcal{R}$ if and only if there exists a piecewise continuous control signal $\mathbf{u}^\star : [0, \infty) \to \mathcal{U}$ such that the system (54) with initial condition $\mathbf{x}_0$ and input $\mathbf{u}^\star$ asymptotically approaches $\mathbf{x}_f$.

- $\mathcal{C}_k$: The set of points covered by the feedback motion planning algorithm after $k$ iterations. That is, the union of all the funnels so far added.

- $\mathcal{C}_\infty$: the limit set of the coverage sets $\mathcal{C}_k$:

$$\mathcal{C}_\infty = \lim_{k \to \infty} \mathcal{C}_k. \tag{55}$$

  That is, $\mathbf{x} \in \mathcal{C}_\infty$ if and only if there exists a sequence of points $\{\mathbf{x}_k\}$ with $\mathbf{x}_k \to \mathbf{x}$ as $k \to \infty$, and $\mathbf{x}_k \in \mathcal{C}_k$ for all $k$.

Note that the limit $\mathcal{C}_\infty$ is well-defined since $\mathcal{C}_k \subseteq \mathcal{C}_{k+1}$ for all $k$, which implies that $\limsup \mathcal{C}_k = \liminf \mathcal{C}_k$.

We will be interesting in proving the following property, which plays an analagous role in feedback-motion-planning as probabilistic completeness does in open-loop motion planning [14]:

**Definition 1** *A feedback motion planning algorithm achieves* probabilistic feedback coverage *for the goal state* $\mathbf{x}_f$ *if* $\mathcal{C}_\infty = \mathcal{R}(\mathbf{x}_f)$ *with probability one.*

The LQR-Trees algorithm works by randomly sampling states, and attempting to link them back to the tree. We make the following assumption:

**Assumption 1** *Each new state to be linked back to the tree is sampled from a distribution with non-zero probability density everywhere on* $\mathcal{X}$*.*

For bounded $\mathcal{X}$ a uniform probability density is adequate, however for unbounded $\mathcal{X}$ more care will need to be taken to ensure the density is integrable.

**Assumption 2** *(a) For the goal point* $x_f$*, the LTI system is completely controllable, (b) for any feasible trajectory* $(\mathbf{x}^\star, \mathbf{u}^\star)$ *of (54), the associated LTV Riccati differential equation has a positive-definite stabilizing solution from any positive-definite final-time condition.*

Assumption 2(b) is an assumption on local exponential stabilizability of the system dynamics by linear feedback. Taken together, these imply that $\mathcal{C}_k$ has non-empty interior for every $k$ and hence $\mathcal{R}(\mathbf{x}_f)$ has non-empty interior, since each $\mathcal{C}_k \subset \mathcal{R}(\mathbf{x}_f)$.

Let $\mathcal{T}_k$ be the set of states on the branches of the LQR-tree after $k$ nodes have been added. We make the following assumption on the availability of an open-loop motion-planning algorithm:

**Assumption 3** *The open-loop motion-planner used to reconnect to the tree has the following property: given any point $\mathbf{x}_0 \in \mathcal{R}(\mathbf{x}_f)$, then the motion-planner will, with non-zero probability, find a finite time $t_f$ and a control signal $\mathbf{u}^\star(t)$ defined on $[0, t_f]$ such that the solution of $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}^\star)$ with $\mathbf{x}(0) = \mathbf{x}_0$ achieves $\mathbf{x}(t_f) \in \mathcal{T}_k$.*

In Subsection 6.1 we will discuss this assumption, and how it can be satisfied in practice.

We are now ready to state the main result of this section:

**Theorem 1** *For any system and goal state satisfying Assumptions 1, 2, 3, the LQR-Trees algorithm achieves probabilistic feedback coverage.*

**Proof:** For a particular instance of the randomized tree, define

$$\mathcal{F} = \bigcap_{k=1}^{\infty} \{\mathcal{R}(\mathbf{x}_f)/\mathcal{C}_k\}, \tag{56}$$

i.e. the set of all states from which the goal is reachable that fail to be added to the tree. Note that $\mathcal{F}$ is the complement of $\mathcal{C}_\infty$ relative to $\mathcal{R}(\mathbf{x}_f)$. Now, $C_\infty$ equals $\mathcal{R}(\mathbf{x}_f)$ if and only if $\mathcal{F}$ has empty interior. We will prove that this must be the case by contradiction.

Suppose $\mathcal{F}$ has non-empty interior, with the interior denoted by $\text{int}(\mathcal{F})$. In this case the Lebesgue measure of $\text{int}(\mathcal{F}) > 0$ and hence by Assumption 1, there is a non-zero probability of sampling from $\text{int}(\mathcal{F})$. Furthermore, by Assumption 3, for any such sample there is a non-zero probability that it can be connected back to the tree. Since all the samplings and trajectory searches are independent events, this implies that with probability one a sample $\mathbf{x}_i$ will eventually be found in $\text{int}(\mathcal{F})$ which connects back to the tree with a feasibile trajectory $\mathbf{x}^\star(t)$, i.e. $\mathbf{x}^\star(t_i) = \mathbf{x}_i$, $\mathbf{x}^\star(t_f) \in \mathcal{T}_k$ for some $t_i, t_f$.

Now, if such an $\mathbf{x}_i$ were found then by Assumption 2 the solution of Riccati equation associated with $\mathbf{x}^\star(t)$ is positive-definite for all $t \in [t_i, t_f]$. Since $\mathcal{X}$ and $\mathcal{U}$ are open sets, and $f(\mathbf{x}, \mathbf{u})$ is smooth, this implies that around each point on the trajectory $x^\star(t)$ there is an ellipsoid of positive radius which will be added to $\mathcal{C}_{k+1}$. Since $\mathbf{x}_i \in \mathcal{F}^i$, some of these ellipsoids will have non-zero measure intersection with $\text{int}(\mathcal{F})$. However, this would contradict the definition $\mathcal{F}$, hence there is zero probability that $\mathcal{F}$ has non-empty interior. Therefore with probability one $\mathcal{C}_\infty$ equals $\mathcal{R}$.

□

Note that we have not assumed anywhere that $\mathcal{X}$,$\mathcal{U}$, or $\mathcal{R}(\mathbf{x}_f)$ are bounded, however unbounded sets will require careful selection of sampling density, and in practice may be problematic for satisfaction of Assumption 3.

## 6.1 Discussion on the Motion Planning Assumption

We briefly argue that Assumption 3 is reasonable in practice. Firstly, due to the smoothness of (54) and the complete controllability of the linearized system about $\mathbf{x}_f$, if from a sampled point the system can be made to asymptotically approach $\mathbf{x}_f$ then it can be made to reach $\mathbf{x}_f$ in finite time.

In the present implementation of LQR-Trees, a path back to the tree is sought by randomly sampling a piecewise-constant control signal $\mathbf{u}$, where the time horizon $t_f$ is randomly sampled from a distribution which has non-zero density everywhere on $[0, T]$ for some large $T$ and the sampling interval is random variable with non-zero density on $[0, t_f]$. Therefore, for sufficiently large $T$, arbitrarily fine approximations of any finite-length piece-wise continuous control signal can be sampled.

Using the sampled $\mathbf{u}$ as an initial guess, a search is performed using sequential quadratic programming (SQP) over input signals with the cost function (17) subject to the constraints that $\mathbf{x}(t_2) \in \mathcal{T}_k$ and $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. The optimization is posed in direct collocation or multiple-shooting form, and is solved numerically via a sequential quadratic programming method in which nonlinear equality constraints are handled by $l_1$ regularized elastic variables [9].

Now, suppose a sampled $\mathbf{x}_0 \in \mathcal{R}(\mathbf{x}_f)$, i.e., there exists a piecewise continuous function $u_0 : [0, t_1] \to \mathcal{U}$ which drives the state from $\mathbf{x}_0$ to a point on the tree $\mathbf{x}(t_f) \in \mathcal{T}_k$. Since $\mathcal{U}$ is open and solutions of (54) are continuous with respect to controls, there exists an open neighborhood of piecewise continuous functions $[0, t_2] \to \mathcal{U}$, $t_2 > t_1$ containing $u_0$, which is the region of convergence to a local optimum for the optimization (17) considered over piecewise-continuous functions. Since our sampling can produce arbitrarily fine approximations of such functions, it can be argued that there is a non-zero probabilty of numerically finding a feasible solution of (17).

In practice, if the optimization fails to find a solution given the initial seed, the sampled point is discarded as temporarily unreachable, however after the tree has grown a nearby point will eventually be sampled from which the search is successful.

Another choice for open-loop motion-planner could be a randomized-sampling based planner such as the RRT [14].

## 6.2 Reachability-limited example

The performance of the algorithm when searching near a boundary of reachability in state space is an important consideration, particularly when applying it to complicated, underactuated systems in which the location and form of such boundaries is unknown. Figure 4 demonstrates the performance of the algorithm when applied to the following second-order polynomial dynamic system:

$$\ddot{x} = x + u^2 - u - 3/4. \tag{57}$$

This system has a known reachable set defined by $\dot{x} < 1 - x$ (whose boundary is shown as a red line in the figure). Intuitively, the force applied to the system

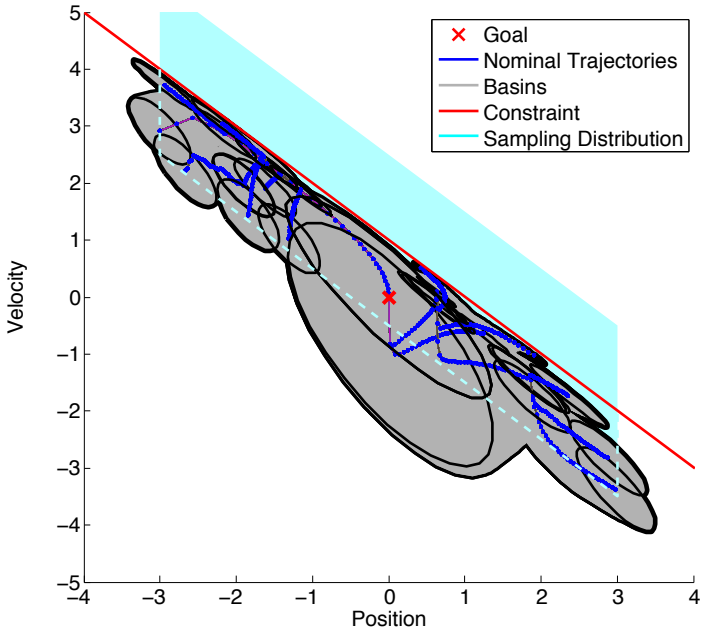



Figure 4: Reachability-limited example

is bounded to be in the set $[-1, \infty)$. From this realization, energy considerations clearly show that the boundary of the reachable set is defined by the line $\dot{x} = 1 - x$. The algorithm will provably fill the sampled space (shown in light blue) which is capable of reaching the goal. As the figure demonstrates, this is achieved by placing ever smaller funnels closer and closer to the boundary, until the boundary is approximated by the surfaces of the funnels. In essence, the algorithm will approximate the the boundary of the reachable set through a fractal-like structure of ever smaller funnels.

# 7 Variations of the algorithm

- **Discrete-time formulation**. Although the presentation here focused on the continuous time case, the approach has a natural dual in a discrete-time formulation. In many ways, the discrete time formulation is simpler, mitigating many of the numerical subtleties to be described in Section 8, and may in fact be preferable for implementation on a sampled-data control system. The primary drawbacks are that (1) it is more difficult to do a careful time-discretization of a polynomial system than a linear one, and (2) the tree will now consist of a collection of points instead of a smooth manifold, complicating any efforts to create a final tree constraint which can "walk along the tree".

- **Compatible with optimal trajectories**. The LQR-tree algorithm provides a relatively efficient way to fill the reachable state space with funnels, but does not stake any claim on the optimality of the resulting trajectories. If tracking particular trajectories, or optimal trajectories, is important for a given problem, then it is quite natural to seed the LQR-tree with one or more locally optimal trajectories (e.g., using [2]), then use the random exploration to fill in any missing regions.

- **Early termination**. For higher dimensional problems, covering the reachable state space may be unnecessary or impractical. Based on the RRTs, the LQR-trees can easily be steered towards a region of state space (e.g., by sampling from that region with slightly higher probability) containing important initial conditions. Termination could then occur when some important subspace is covered by the tree.

- **Bidirectional trees**. Although LQR-trees only grow backwards from the goal, a partial covering tree (from an early termination) could also serve as a powerful tool for real-time planning. Given a new initial condition, a forward RRT simply has to grow until it intersects with the *volume* defined by the basin of attraction of the backwards tree.

- **Finite-horizon trajectories**. The LQR stabilization derived in section 3.1 was based on infinite horizon trajectories. This point was necessary in order to use the language of basins of attraction and asymptotic stabilization. Finite-horizon problems can use all of the same tools (though perhaps not the same language), but must define success as being inside some finite volume around the goal state at $t_G$. Funnels connecting to this volume are then computed using the same Riccati backup.

- **Multi-query algorithms**. Another very interesting question is the question of reusing the previous computational work when the goal state is changed. In the pendulum example, consider having a new goal state, $\mathbf{x}_G = [\pi + 0.1, 0]^T$ - this would of course require a non-zero torque to stabilize. To what extent could the tree generated for stabilizing $\mathbf{x}_G = [\pi, 0]^T$ be used to stabilize this new fixed point? If one can find a trajectory to connect up the new goal state near the root of the tree, then the geometry of the tree can be preserved, but naively, one would think that all of the stabilizing controllers and the verification would have to be re-calculated. Interestingly, there is also a middle-road, in which the existing feedback policy is kept for the original tree, and the estimated funnels are not recomputed, but simply scaled down to make sure that the funnels from the old tree transition completely into the funnel for the new tree. This could be accomplished very efficiently, by just propagating a new $\rho_{max}$ through the tree, but might come at the cost of losing coverage.

# 8 Implementation details

The LQR-Tree algorithm presented here, which relies on a continuous time formulation and optimization over polynomials, requires the practioner to address a number of potential numerical issues. We list a few of these here:

- **Balancing coordinates**. One drawback of working with polynomials is that their associated optimization problems are often poorly conditioned (high order polynomials blow up quickly), and solutions to the Riccati differential equation can easily have a large range of eigenvalues. An essential step for getting good performance out of the semidefinite programming was finding a coordinate transformation:

$$\mathbf{x}_b = \mathbf{T}\mathbf{x}, \tag{58}$$

  where $\mathbf{T}$ was selected to numerically condition the problem by making the matrices $\mathbf{T}'\mathbf{S}\mathbf{T}$ and $\mathbf{T}'\left(2\mathbf{S}\frac{\partial \mathbf{f}(\mathbf{x_0}+\bar{\mathbf{x}},\mathbf{u_0}-\mathbf{K}\bar{\mathbf{x}})}{\partial \mathbf{x}} + \dot{\mathbf{S}}\right)\mathbf{T}$ are as close as possible to the identity matrix. Without this balancing, the semidefinite solvers often return errors with infeasible programs.

- **Piecewise polynomial approximations of numerical integration.** For the continuous time formulation, it was essential to carefully manage the piecewise polynomial interpolations of solutions from trajectory optimization and from the Riccati differential equation. Simple cubic spline representations often caused even the nominal trajectory to appear to be unstable. In the current implementation, we represent the output of the trajectory optimization with a cubic piecewise polynomial that, for every segment, matches the knot points and the derivatives of the knot points which are known from the dynamics function. The Riccati differential equation is solved using MATLAB `ode45`, interpolations of that solution are done using `deval`, and calculations of $\dot{\mathbf{S}}$ are done by explicitly re-evaluating the Riccati equation.

- **Time dependence on $\dot{J}$.** Finally, one must be careful when performing verification on the time-varying Lyapunov candidate. Naively, if one used a cubic spline representation of the nominal trajectory and of $\mathbf{S}$, then $\dot{\mathbf{J}}$ quickly becomes an unnaturally high order. Instead, we fit the best cubic spline directly to $\dot{J}$.

## 8.1 A Software Distribution

Due to the potential complexity of getting a robust and high performance implementation of the LQR-Tree algorithm, the authors are making a MATLAB toolbox available at `http://groups.csail.mit.edu/locomotion/software.html` .

# 9 Summary and Conclusions

Recent advances in direct computation of Lyapunov functions have enabled a new class of feedback motion planning algorithms for complicated dynamical systems. This paper presented the LQR-Tree algorithm which uses Lyapunov computations to evaluate the basins of attraction of randomized trees stabilized with LQR feedback. We prove, and demonstrate through simulation examples, that this algorithm has the property of "probabilistic feedback coverage". Furthermore, initial experimental results suggest that this coverage occurs efficiently, requiring only a small number of stabilized trajectories to cover the reachable space.

Further investigation of this algorithm will likely result in a covering motion planning strategy for underactuated systems with dimensionality greater than what is accessible by discretization algorithms like dynamic programming, and early termination strategies which provide targeted coverage of state space in much higher dimensional systems. The resulting policies will have certificates guaranteeing their performance on the system model.

# Acknowledgements

The authors wish to thank Alexandre Megretski and Pablo Parrilo for their invaluable assistance on the formulations and numerical implementations of the sums of squares methods. We also wish to thank Phillip Reist and Steve LaValle for valuable discussions. This work was supported by NSF awards 0746194 and 0915148, and AFRL contract FA8650-05-C- 7262.

# References

[1] B. D. O. Anderson and J. B. Moore. *Optimal control: linear quadratic methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[2] Christopher G. Atkeson and Benjamin Stephens. Random sampling of states in dynamic programming. In *Advances in Neural Information Processing Systems*, 2008.

[3] M. Berz and G. Hoffstatter. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing*, 4(1):8–97, Feb 1998.

[4] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics, 2001.

[5] Michael Branicky and Michael Curtiss. Nonlinear and hybrid control via RRTs. *Proc. Intl. Symp. on Mathematical Theory of Networks and Systems*, 2002.

[6] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, June 1999.

[7] G. Franzè, A. Casavola, D. Famularo, and E. Garone. An off-line mpc strategy for nonlinear systems based on sos programming. In L. Magni, D. M. Raimondo, and F. Allgower, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 491–499. Springer-Verlag, 2009.

[8] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-Time Motion Planning for Agile Autonomous Vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, JanuaryFebruary 2002.

[9] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[10] Tor A. Johansen. Computation of lyapunov functions for smooth nonlinear systems using convex optimization. *Automatica*, 36(11):1617 – 1626, 2000.

[11] L.E. Kavraki, P. Svestka, JC Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[12] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 3rd edition, December 2001.

[13] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 1:692–698 vol.1, 2001.

[14] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2000.

[15] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] M.T. Mason. The mechanics of manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 544–548. IEEE, 1985.

[17] A. Megretski. Polynomial optimization toolbox, available online: `http://web.mit.edu/ameg/www/`.

[18] A. Megretski. Positivity of trigonometric polynomials. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 4, pages 3814–3817 vol.4, Dec. 2003.

[19] Mitchell, I.M., Bayen, A.M., Tomlin, and C.J. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, July 2005.

[20] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics. Society for Industrial Mathematics, 1995.

[21] A. Papachristodoulou, M. M. Peet, and S. Lall. Analysis of polynomial systems with time delays via the sum of squares decomposition. *IEEE Transactions on Automatic Control*, 54(5):1058–1064, May 2009.

[22] A. Papachristodoulou and S. Prajna. On the construction of lyapunov functions using the sum of squares decomposition. In *Proceedings of the 41st IEEE Conference onDecision and Control*, Dec 2002.

[23] Pablo A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, May 18 2000.

[24] N. Peterfreund and Y. Baram. Convergence analysis of nonlinear dynamical systems by nested Lyapunov functions. *IEEE Transactions on Automatic Control*, 43(8):1179–1184, Aug 1998.

[25] S. Prajna, P.A. Parrilo, and A. Rantzer. Nonlinear control synthesis by convex optimization. *Automatic Control, IEEE Transactions on*, 49(2):310–314, Feb. 2004.

[26] Stephen Prajna, Antonis Papachristodoulou, Peter Seiler, and Pablo A. Parrilo. *SOSTOOLS: Sum of Squares Optimization Toolbox for MATLAB Users guide*, 2.00 edition, June 1 2004.

[27] Stephen Prajna and Anders Rantzer. Convex programs for temporal verification of nonlinear dynamical systems. *SIAM Journal of Control and Optimization*, 46(3):999–1021, 2007.

[28] Anders Rantzer. A dual to lyapunov's stability theorem. *Systems & Control Letters*, 42(3):161 – 168, 2001.

[29] Alexander Shkolnik and Russ Tedrake. Path planning in 1000+ dimensions using a task-space Voronoi bias. In *Proceedings of the IEEE/RAS International Conference on Robotics and Automation (ICRA)*. IEEE/RAS, 2009.

[30] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, October 1990.

[31] W. Tan and A. Packard. Stability region analysis using polynomial and composite polynomial lyapunov functions and sum-of-squares programming. *IEEE Transactions on Automatic Control*, 53(2):565–571, March 2008.

[32] Ufuk Topcu, Andrew Packard, and Peter Seiler. Local stability analysis using simulations and sum-of-squares programming. *Automatica*, 44(10):2669 – 2675, 2008.