LQR-Trees: Feedback motion planning on sparse randomized trees

Russ Tedrake. Paper-ID 116

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 carefully investigate the algorithm on a two-dimensional model system and discuss the potential for the control of more complicated underactuated control problems like bipedal walking.**

I. Introduction

Randomized algorithms for motion planning have become incredibly popular in recent years due to their ability to efficiently solve relatively high-dimensional configuration space planning problems with kinematic constraints[13]. Randomized approaches have dominated because complex geometric constraints can be difficult to reason about analytically, or design heuristics for, but can often be sampled very efficiently with a fast collision-checker. Work on sample-based *feedback* motion planning attempts to scale this intuition to planners which consider feedback stabilization while generating the motion plan. So far, however, sample-based feedback planning algorithms have mostly ignored problems with dynamics[13], because it can dramatically complicate the design and evaluation of even local feedback policies.

Another class of problems where motion planning plays an essential role is in the control of underactuated systems. Underactuated systems have second-order dynamic constraints; the system cannot be controlled to follow arbitrary trajectories (fully-actuated systems, by contrast, can be feedback-linearized and commanded to follow arbitrary trajectories, mitigating the need for advanced motion planning machinery). Although there have been a number of very elegant domain-specific instances of feedback motion planning for underactuated systems (e.g., [6]), by contrast to configuration space planning, there is a clear paucity of general algorithms.

This paper aims to build on recent advances from control theory to design efficient and general algorithms for feedback motion planning in underactuated systems. Specifically, the controls community have recently developed a number of efficient algorithms for direct computation of Lyapunov functions for smooth nonlinear systems, using convex optimization [10, 18]. These tools can plug into motion planning algorithms to automatically compute planning "funnels" for even very complicated dynamical systems, and open 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 computational Lyapunov 'certificates' based on a sum-of-squares method to create the funnels.

The aim of this work is to generate a class of algorithms capable of computing covering feedback policies for underactuated systems with dimensionality beyond that of dynamic programming. The use of local trajectory optimizers and local feedback stabilization scales well to higher-dimensions, and reasoning about the feedback "funnels" 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 thus are not subject to the pitfalls of discretization. By considering feedback during the planning process, the resulting plans are certifiably robust to disturbances and quite suitable for implementation on real robots. Although scaling is the driving motivation of this approach, this paper focuses on the coverage properties of the LQR-Tree algorithm by carefully studying a simple 2D example (the torque-limited simple pendulum), which reveals the essential properties of the algorithm in a space that can be easily visualized.

II. Background

A. Feedback motion planning

For implementation on real robots, trajectories generated by a motion planning system are commonly stabilized by a feedback control system[1]. 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. Algorithms which explicitly consider the feedback stabilization during the planning 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 for a feedback policy which collapses a large set of initial conditions into a smaller set of final conditions[16]. 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] (see Figure 1). At the time, the weakness

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

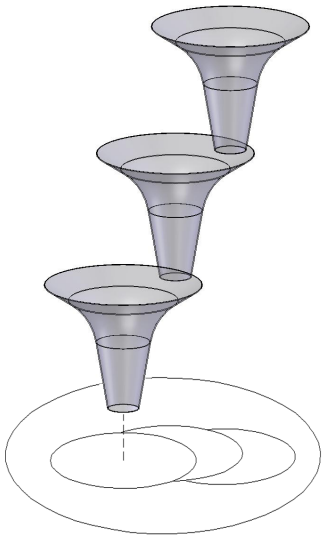


Fig. 1: Cartoon of motion planning with funnels in the spirit of [6].

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[13].

B. Direct computation of Lyapunov functions

Burridge et al. also pointed out the strong connection between Lyapunov functions and these motion planning funnels[6]. A Lyapunov function is a differentiable positive-definite output function, $V(\mathbf{x})$, for which $\dot{V}(\mathbf{x}) \leq 0$ as the closed-loop dynamics of the system evolve[22]. If these conditions are met over some ball in state space, B_r, containing the origin, then the origin is said to be locally stable in the sense of Lyapunov. The ball, B_r, can then be interpreted as the preimage of the funnel. Lyapunov functions have played an incredibly important role in nonlinear control theory, but can be difficult to discover analytically for complicated systems.

The last few years has seen the emergence of a number of computational approaches to discovering Lyapunov functions for nonlinear systems, often based on convex optimization(e.g., [10, 18]). 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[18]. Sums of squares programs can be recast into semidefinite programs and solved using convex optimization solvers (such as interior point methods); the freely available SOSTOOLS library makes it quite accessible to perform these computations in MATLAB[19]. As we will see, the ability to check uniform positive (or negative) definiteness will infer the ability to verify candidate Lyapunov functions over a region of state space for smooth (nonlinear) polynomial systems.

These tools make it possible to automate the search for Lyapunov functions. Many researchers have used this capability to find stability proofs that didn't previously exist for nonlinear systems, or even to estimate the basins of attraction of stable nonlinear systems[18]. In this paper, we begin to explore the implications for planning of being able to efficiently compute planning funnels.

C. Other related work

The ideas presented here are very much inspired by the randomized motion planning literature, especially rapidly-exploring randomized trees (RRTs)[12] and probabilistic roadmaps (PRMs)[11]. This work was also inspired by [15] and [20] who point out a number of computational advantages to using sample-paths as a fundamental representation for learning policies which cover the relevant portions of state space.

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.

III. The LQR-Tree Algorithm

Let us first consider the decoupled approach to planning and control. Given an initial condition in state space and an open-loop trajectory of control inputs (the result of a motion plan), one natural way to stabilize that trajectory is by designing a time-varying, linear quadratic regulator (LQR). LQR, iterative LQR (iLQR)[23], and the closely related differential dynamic programming (DDP)[9] are quietly becoming a common tool for roboticists[2, 1], and have demonstrated success in a number of applications. The design of a time-varying LQR controllers along a trajectory involves linearization along the trajectory, and proceeds by integrating a Riccati equation[3] backwards in time to compute the time-varying cost-to-go function and time-varying linear optimal feedback policy. It turns out that, if one grows a randomized tree backwards from the goal, then it is efficient and natural to recursively compute the LQR stabilizers along all branches of the tree. The result is that the backwards tree becomes a large stabilizing controller which grabs trajectories and pulls them towards the goal.

Now consider a feedback motion planning algorithm which grows a randomized tree backwards from the goal, computing LQR feedback gains for each node in the tree as it is added, and reasoning about those feedback controllers as the tree grows. Conveniently, the LQR derivation also provides a quadratic approximation of the cost-to-go function along the

trajectory; this cost-to-go function happens to be a Lyapunov function for the linearized system. Exploiting this, we can formulate a sum of squares program to discover a region of state space over which the quadratic cost-to-go function is still a valid Lyapunov function for the nonlinear system. The result is an efficient optimization to obtain a Lyapunov certificate.

Finally, we use the computed certificates, and resulting basin of attraction to influence the way that our tree grows, with the goal of probabilistically filling the reachable state space with the basin of attraction of the goal state. Subgoals are sampled uniformly from a bounded region of state space which is currently outside the basin of attraction of the current stabilized tree.

A. Essential Components

1) Time-varying LQR feedback stabilization: Let us first consider the subproblem of designing a time-varying LQR feedback based on a time-varying linearization along a nominal trajectory. Consider a controllable, smoothly differentiable, nonlinear system:

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

with a stabilizable goal state, $\mathbf{x}_G$. Define a nominal trajectory (a solution of equation 1) which reaches the goal in a finite time: $\mathbf{x}_0(t), \mathbf{u}_0(t)$, with $\forall t \geq t_G, \mathbf{x}_0(t) = \mathbf{x}_G$ and $\mathbf{u}_0(t) = \mathbf{u}_G$. Define

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

Now linearize the system around the trajectory, so that we have

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

Define a quadratic regulator (tracking) cost function as

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

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

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

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

$$-\dot{\mathbf{S}} = \mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S} \tag{2}$$

and the boundary condition $\mathbf{S}(t_G)$ is the positive-definite solution to the equation:

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

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

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

2) Computing certificates: It is well-known that $J^*(\bar{\mathbf{x}}, t)$ is a Lyapunov function for the linearized system. Our goal is to provide a certificate for the closed-loop nonlinear system of the form[17]:

$$\frac{d}{dt}J^*(\bar{\mathbf{x}}(t), t) \leq 0, \quad \text{for } t \in [t_k, t_{k+1}], J^*(\bar{\mathbf{x}}, t) \leq \rho_k. \tag{3}$$

In order to impose the domain from Equation (3), we introduce a Lagrange multiplier:

$$\frac{d}{dt}J^*(\bar{\mathbf{x}}, t) + h(\bar{\mathbf{x}}, t)\left(\rho_0 - J^*(\bar{\mathbf{x}}, t)\right) \leq 0, \quad h(\bar{\mathbf{x}}, t) \geq 0. \tag{4}$$

(h can make the left-hand terms arbitrarily more negative over the region where $\rho_k < J^*$). If we are willing to perform a polynomial expansion of J^* and $\frac{d}{dt}J^*$, then we can solve this problem (finding an h which satisfies the negative semi-definiteness) as a sum-of-squares (SOS) problem[18].

First, approximate $\mathbf{x}_0(t)$ as an N_xth order spline, $\hat{\mathbf{x}}_0(t)$, and $\mathbf{S}(t)$ as an N_Sth order (element-wise) spline, $\hat{\mathbf{S}}(t)$, based on the ODE integration results over the domain $[t_0, t_1]$, and $\dot{\bar{\mathbf{x}}}$ is approximated with an N_fth order Taylor expansion. Then we have

$$J^*(\bar{\mathbf{x}}, t) \approx \bar{\mathbf{x}}\hat{\mathbf{S}}(t)\bar{\mathbf{x}},$$
$$\dot{J}^*(\bar{\mathbf{x}}, t) \approx 2\bar{\mathbf{x}}^T\hat{\mathbf{S}}(t)\dot{\hat{\bar{\mathbf{x}}}} + \bar{\mathbf{x}}^T\dot{\hat{\mathbf{S}}}(\mathbf{t})\bar{\mathbf{x}}.$$

Note that in our implementation, $\mathbf{u}_0(t)$ and $\mathbf{K}(t)$ are held in a zero-order hold over the domain $t \in [t_0, t_1]$. Similarly, we will search for Lagrange multipliers of a quadratic form:

$$h(\bar{\mathbf{x}}, t) = \mathbf{m}(t, \mathbf{x})^T\mathbf{H}_0\mathbf{m}^T(t, \mathbf{x}),$$

where $\mathbf{m}$ is a vector of all monomials from order 0 to order N_m, and $\mathbf{H}_0$ is a matrix of free variables (the constant coefficients). If the sum of squares optimization returns a feasible solution to the problem, then $J^*(\mathbf{x}, t)$ is a valid Lyapunov function for the system over the domain defined by $t \in [t_k, t_{k+1}]$, $J^*(\mathbf{x}, t) \leq \rho_k$. In order to compute the Lyapunov certificates recursively backwards from the goal, we require that $\rho_k \leq \rho_{k+1}$; this ensures that the funnel at time k leads into the funnel at time $k+1$. We use a line search on ρ_k to find the largest certificate possible for every node of the tree.

The certificate is conservative in every way, except that the nonlinear dynamics are approximated by the polynomial expansion (therefore limiting it to smooth nonlinear systems). In practice, the algorithm acquires conservative, but impressively tight approximations of the basin of attraction for the system in initial tests with the pendulum and Acrobot.

3) Growing the tree: Another essential component of the LQR tree algorithm is the method by which the backwards tree is extended. Following the RRT approach, we select a sample at random from some distribution over the state space, and attempt to grow the tree towards that sample. Unfortunately, RRTs typically do not grow very efficiently in differentially constrained (e.g., underactuated) systems, because simple distance metrics like the Euclidean distance are inefficient in determining which node in the tree to extend from. Further

embracing LQR as a tool for motion planning, in this section we develop an affine quadratic regulator around the sample point, then use the resulting cost-to-go function to determine which node to extend from, and use the open-loop optimal policy to extend the tree.

Choose a random sample (not necessarily a fixed point) in state space, $\mathbf{x}_s$ and a default $\mathbf{u}_0$, and use $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}_s$, $\bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}_0$.

$$\begin{aligned}\dot{\bar{\mathbf{x}}} =& \frac{d}{dt}(\mathbf{x}(t) - \mathbf{x}_s) = \dot{\mathbf{x}}(t) \\ \approx& \mathbf{f}(\mathbf{x}_s, \mathbf{u}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}(t) - \mathbf{x}_s) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u} - \mathbf{u}_0) \\ =& \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}} + \mathbf{c}.\end{aligned}$$

Now define an affine quadratic regulator problem with a hard constraint on the final state, but with the final time, t_f, left as a free variable[14]:

$$J(\bar{\mathbf{x}}_0, t_0, t_f) = \int_{t_0}^{t_f} \left[1 + \frac{1}{2}\bar{\mathbf{u}}^T(t)\mathbf{R}\bar{\mathbf{u}}(t)\right] dt,$$
$$\text{s.t.} \quad \bar{\mathbf{x}}(t_f) = \mathbf{0},\ \bar{\mathbf{x}}(t_0) = \bar{\mathbf{x}}_0,\ \dot{\bar{\mathbf{x}}} = \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}} + \mathbf{c}.$$

Without loss of generality (since the dynamics are autonomous), we will use $J(\bar{\mathbf{x}}_0, t_f - t_0)$ as a shorthand for $J(\bar{\mathbf{x}}_0, t_0, t_f)$. It can be shown that the optimal (open-loop) control is

$$\bar{\mathbf{u}}^*(t) = -\mathbf{R}^{-1}\mathbf{B}^T e^{\mathbf{A}^T(t_f - t)}\mathbf{P}^{-1}(t_f)\mathbf{d}(\bar{\mathbf{x}}(t_0), t_f),$$

where

$$\dot{\mathbf{P}}(t) = \mathbf{A}\mathbf{P}(t) + \mathbf{P}(t)\mathbf{A}^T + \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T, \quad \mathbf{P}(t_0) = \mathbf{0}$$
$$\mathbf{d}(\bar{\mathbf{x}}, t) = \mathbf{r}(t) + e^{\mathbf{A}t}\bar{\mathbf{x}}, \quad \dot{\mathbf{r}}(t) = \mathbf{A}\mathbf{r}(t) + \mathbf{c}, \quad \mathbf{r}(\bar{\mathbf{x}}, t_0) = \mathbf{0}$$

and the resulting cost-to-go is

$$J^*(\bar{\mathbf{x}}, t_f) = t_f + \frac{1}{2}\mathbf{d}^T(\bar{\mathbf{x}}, t_f)\mathbf{P}^{-1}(t_f)\mathbf{d}(\bar{\mathbf{x}}, t_f).$$

Thanks to the structure of this equation, it is surprisingly efficient to compute the cost-to-go from many initial conditions (here the existing vertices in the tree) simultaneously. For each $\bar{\mathbf{x}}$ the horizon time, $t_f^* = \text{argmin}_{t_f} J^*(\bar{\mathbf{x}}, t_f)$, is found by selecting the minimum after integrating $\mathbf{P}(t)$ and $\mathbf{r}(t)$ over a fixed horizon. This cost-to-go function provides a relatively efficient *dynamic* distance metric[2] for the RRT expansion which performs much better than Euclidean metrics for underactuated systems[7].

Once the "closest" node in the existing tree is identified, by this LQR distance metric, the tree is extended by applying a series of actions backwards in time from the closest node. The initial guess for this series of actions is given by $\bar{\mathbf{u}}*(t)$ from the LQR distance metric, but this estimate (which is only accurate in the neighborhood of the sample point) can be further refined by a fast, local, nonlinear trajectory optimization routine. In the current results, we use a direct collocation[24, 4] implementation using the formulation from equation III-A.3, but with the nonlinear dynamics. If the direct collocation method cannot satisfy the final value constraint, then the point is considered (temporarily) unreachable, and is discarded. Interestingly, using the LQR open-loop control to initialize the nonlinear optimization appears to help overcome many of the local minima in the nonlinear optimization process.

4) A sampling heuristic: Finally, we take advantage of the Lyapunov certificates by changing the sampling distribution. Adding branches of the tree that will be contained by the existing basin of attraction has little value. The sampling heuristic used here is implemented by sampling uniformly over the desired subset of state space, then rejecting any sample which are already in the basin of attraction of any of the tree branches. This "collision checking" is very inexpensive; it is far more expensive to add a useless node into the tree. Other sampling distributions are possible, too. One interesting alternative is sampling from states that are just at the edges of the basin of attraction, e.g, $\forall_i J^*(\mathbf{x} - \mathbf{x}_0^i, t) > \rho_i, \exists_j J^*(\mathbf{x} - \mathbf{x}_0^i, t) \leq 1.5\rho_j$.

B. The Algorithm

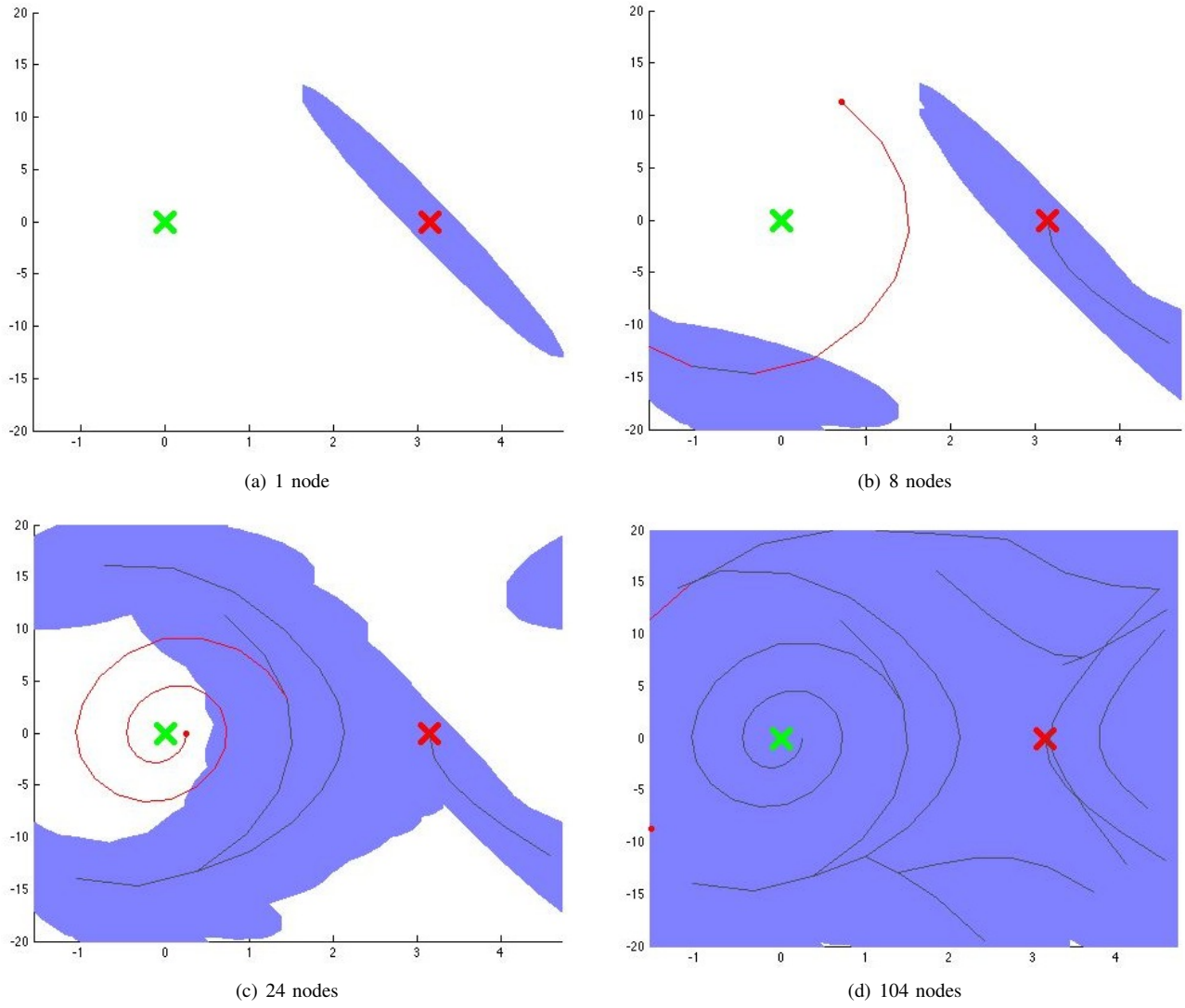
The algorithm proceeds by producing a tree, T, with nodes containing the tuples, $\{\mathbf{x}, \mathbf{u}, \mathbf{S}, \mathbf{K}, \rho, i\}$, where $J^*(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T\mathbf{S}\bar{\mathbf{x}}$ is the local quadratic approximation of the value function, $\bar{\mathbf{u}}^*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

Algorithm 1 LQR-Tree ($\mathbf{x}_g$,$\mathbf{Q}$,$\mathbf{R}$)

$[\mathbf{A}, \mathbf{B}] \Leftarrow$ linearization of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ around $\mathbf{x}_G, \mathbf{u}_G$
$[\mathbf{K}, \mathbf{S}] \Leftarrow$ LQR($\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$)
$\rho \Leftarrow$ certificate computed as described in section III-A.2
T.init($\{\mathbf{x}_g, \mathbf{u}_g, \mathbf{S}, \mathbf{K}, \rho$, NULL$\}$)
for k = 1 to K **do**
  $\mathbf{x}_{rand} \Leftarrow$ random sample as described in section III-A.4; if no samples are found, then FINISH
  $\mathbf{x}_{near}$ from cost-to-go distance metric described in section III-A.3
  $\mathbf{u}_{tape}$ from extend operation described in section III-A.3
  for each $\mathbf{u}$ in $\mathbf{u}_{tape}$ **do**
    $\mathbf{x} \Leftarrow$ Integrate backwards from $\mathbf{x}_{near}$ with action $\mathbf{u}$
    $[\mathbf{K}, \mathbf{S}]$ from LQR derivation in section III-A.1
    $\rho \Leftarrow$ certificate computed as described in section III-A.2
    $i \Leftarrow$ pointer to node containing $\mathbf{x}_{near}$
    T.add-node($\mathbf{x}, \mathbf{u}, \mathbf{S}, \mathbf{K}, \rho, i$)
    $\mathbf{x}_{near} \Leftarrow \mathbf{x}$
  end for
end for

Execution of the LQR tree policy is accomplished by selecting any node in the tree with a basin of attraction which contains the initial conditions, $\mathbf{x}(0)$, and following the time-varying feedback policy along that branch all of the way to the goal.

[2]Note that it is not technically a distance metric, since it is not symmetric, but works very well.



(a) 1 node

(b) 8 nodes

(c) 24 nodes

(d) 104 nodes

Fig. 2: 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 $\dot{\theta} \in [-20, 20]$. The green X (on the left) represents the stable fixed point; the red X (on the right) represents the unstable (upright) fixed point. The blue ovals (darker, smaller) represent the "funnels", sampled at every node.

IV. Simulations

Simulation experiments on a two-dimensional toy problem have proven very useful for understanding the dynamics of the algorithm. Figure 2 tells the story fairly succinctly. The algorithm was tested on a simple pendulum, $I\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = \tau$, with $m = 1, l = .5, b = .1, I = \frac{ml}{12}, g = 9.8$. Here $\mathbf{x} = [\theta, \dot{\theta}]^T$ and $\mathbf{u} = \tau$. The parameters of the LQR tree algorithm were $\mathbf{x}_G = [\pi, 0]^T$, $\mathbf{u}_G = 0$, $\mathbf{Q} = diag([10, 1])$, $\mathbf{R} = 15$, $N_f = 3$, $N_m = 2$, $N_x = 3$, $N_S = 3$.

Figure 2(a) shows the basin of attraction (blue oval) after computing the linear time-invariant (LTI) LQR solution around the unstable equilibrium. Figure 2(b) shows the entire trajectory to the first random sample point (red dot), and the funnels that have been computed so far for the second-half of the 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. The final tree in Figure 2(d) also reveals three instances where the trajectories on the tree cross - this is a result of having an imperfect distance metric.

Note that state $\mathbf{x} = [0, 0]^T$, corresponding to the stable fixed-point of the unactuated pendulum, is covered by the basin of attraction after 32 nodes have been added. The algorithm was not biased in any way towards this state, but this bias can be added easily. The entire space is probabilistically covered (1000 random points chosen sequentially were all in the basin of attraction) after the tree contained just 104 nodes. On average, the algorithm terminates after 146 nodes for the simple pendulum with these parameters. For contrast, [5] shows a well-tuned single-directional RRT for the 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 compute the Lyapunov certificates. The entire algorithm runs in about two minutes on a laptop, without any attempt to optimize the code.

V. Discussion

A. Properties of the algorithm

Recall that for nonlinear systems described by a polynomial of degree $\leq N_f$, the estimated Lyapunov certificates are only conservative; the true basin of attraction completely contains the estimated stability region. In practice, this is often (but not provably) the case for more general smooth nonlinear systems.

Proposition 1: For nonlinear systems described by a polynomial of degree $\leq N_f$, the LQR tree algorithm probabilistically covers the sampled portion of the reachable state space with a stabilizing controller and a Lyapunov function, thereby guaranteeing that all initial conditions which are capable of reaching the goal will stabilize to the goal.

Proving proposition 1 carefully requires a proof that the local trajectory optimizer is always capable of solving a trajectory to a reachable point in the state space that is in an ϵ-region outside the existing basin of attraction. This is likely the case, seeing as the nonlinear optimizer is seeded by a linear optimal control result which will be accurate over some region of similar size to the basin of attraction ellipse. However, the full proof is left for future work.

Perhaps even more exciting is the fact that, in the model explored, this coverage appears to happen rapidly and allow for fast termination of the algorithm. The pendulum is a surprisingly rich test system - for example, as key parameters such as $\mathbf{R}$ or b change, the size of the funnels can change dramatically, resulting in quite different feedback policy coverings of the state space, and always resulting in rapid coverage.

It is also worth noting that the trajectories out of a more standard RRT are typically smoothed. Trajectories of the closed-loop system which result from the LQR algorithm are (qualitatively) quite smooth, despite coming from a randomized algorithm. The LQR stabilizing controller effectively smoothes the trajectory throughout state space.

B. Why randomness?

During this study, we investigated a number of deterministic concepts for efficiently extending the basin of attraction. While these approaches may still bear fruit, the problem does seem to have the essential quality that often motivates randomized planning. While it is tractable to reason about the basin of attraction of a single trajectory, it is difficult and expensive to reason analytically, or computationally about the volumes and boundaries of the basin of attraction of the entire tree (especially when branches of the tree overlap or intersect). However, it is relatively inexpensive and quite easy to evaluate whether a random sample is inside or outside of the basin of attraction.

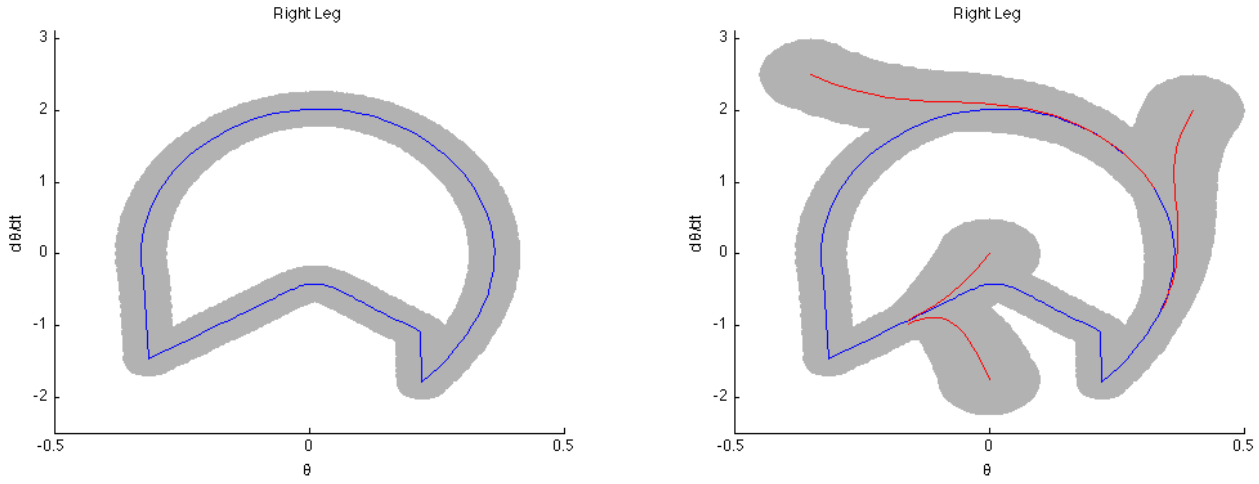
C. Straight-forward variations in the algorithm

- **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 some small 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 III-A.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.

D. Controlling walking robots

A feedback motion planning algorithm like the LQR tree algorithm could be a very natural control solution for walking robots, or other periodic control systems. In this case, rather than the goal of the tree being specified as a point, the goal would be a periodic (limit cycle) trajectory. This could be implemented in the tree as a set of goal states, which happen to be connected, and the Lyapunov certificates about this goal would emerge from the periodic steady-state solution of the Riccati equation and certification process on the limit cycle. Limit cycles for walking systems in particular are often described as a hybrid dynamics punctuated by discrete impacts. These discrete jump events must be handled with care in the feedback and certificate derivation, but are not fundamentally incompatible with the approach[21].

Figure 3 cartoons the vision of how the algorithm would play out for the well-known compass gait biped[8]. On the left is a plot of the (passively stable) limit cycle generated by the compass gait model walking down a small incline. This trajectory can be stabilized using a (periodic) time-varying linearization and LQR feedback, and the resulting basin of attraction might look something like the shaded region in



(a) The initial tree

(b) After expanding four random nodes

Fig. 3: Sketch of LQR trees on the compass gait biped.

Figure 3(a). The goal of the LQR tree algorithm would then be to fill the remaining portion of state space with transient "maneuvers" to return the system to the nominal limit cycle. A potential solution after a few iterations of the algorithm is cartooned in Figure 3(b). In these hybrid models, the distance metric will be especially important since it will also be used to determine the mode of the hybrid system.

E. 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 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 Lyapunov certificates 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 LQR certificates 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 ρ_{max} through the tree, but might come at the cost of losing coverage.

One reason why this multi-query question is so exciting is that the problem of controlling a robot to walk on rough terrain could be nicely formulated as a multi-query stabilization of the limit cycle dynamics from Figure 3. If the control system could quickly adapt the motion plan from steady-state periodic walking to a modified plan which allowed one or more steps with different foot-placement, then this would provide a very elegant solution to a very state-of-the-art problem.

VI. 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. Careful investigations on a torque-limited simple pendulum revealed that, by modifying the sampling distribution to only accept samples outside of the computed basin of attraction of the existing tree, the result was a very sparse tree which covered the state space with a basin of attraction.

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.

Acknowledgment

The author gratefully acknowledges Alexandre Megretski for many helpful discussions and tutorials.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the Neural Information Processing Systems (NIPS '07)*, volume 19, December 2006.

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

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[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] Elena Glassman and Russ Tedrake. Rapidly exploring state space. *In Progress*, 2009.

[8] A. Goswami, B. Espiau, and A. Keramane. Limit cycles and their stability in a passive bipedal gait. pages 246 – 251. IEEE International Conference on Robotics and Automation (ICRA), 1996.

[9] David H. Jacobson and David Q. Mayne. *Differential Dynamic Programming*. American Elsevier Publishing Company, Inc., 1970.

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

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

[14] Frank L. Lewis. *Applied Optimal Control and Estimation*. Digital Signal Processing Series. Prentice Hall and Texas Instruments, 1992.

[15] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Technical Report TR-2006-35, University of Massachusetts, Department of Computer Science, July 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] Alexandre Megretski. Personal communication, 2008.

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

[19] 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.

[20] Khashayar Rohanimanesh, Nicholas Roy, and Russ Tedrake. Towards feature selection in actor-critic algorithms. Technical report, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, 2007.

[21] A.S. Shiriaev, L.B. Freidovich, and I.R. Manchester. Can we make a robot ballerina perform a pirouette? orbital stabilization of periodic motions of underactuated mechanical systems. *Annual Reviews in Control*, 2008.

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

[23] Emanuel Todorov and Weiwei Li. Iterative linear-quadratic regulator design for nonlinear biological movement systems. volume 1, pages 222–229. International Conference on Informatics in Control, Automation and Robotics, 2004.

[24] Oskar von Stryk. Numerical solution of optimal control problems by direct collocation. In *Optimal Control, (International Series in Numerical Mathematics 111)*, pages 129–143, 1993.