Bounded-Error LQR-Trees

Barrett Ames[1] George Konidaris[2]

***Abstract*— We present a feedback motion planning algorithm, Bounded-Error LQR-Trees, that leverages reinforcement learning theory to find a policy with a bounded amount of error. The algorithm composes locally valid linear-quadratic regulators (LQR) into a nonlinear controller, similar to how LQR-Trees constructs its policy, but minimizes the cost of the constructed policy by minimizing the Bellman Residual, which is estimated in the overlapping regions of LQR controllers. We prove a sample-based upper bound on the true Bellman Residual, and demonstrate a five-fold reduction in cost over previous methods on a simple underactuated nonlinear system.**

I. Introduction

Consider optimizing the control policy for a high-performance robot arm in an industrial setting. At low and medium speeds, the arm's design keeps it rigid and kinematic plans work well. However, at high speeds, dynamics begin to affect its ability to follow a desired trajectory. In many cases, the dynamics come in the form of nonlinear constraints. These problem characteristics make the LQR-Tree algorithm [13] a reasonable choice for constructing a policy. LQR-Tree is able to scale and handle nonlinearities due to its use of trajectory optimization. In addition, the stability guarantees provide assurance that the arm can execute the trajectories robustly. However, it does not optimize the resulting policy in a global sense.

We introduce an algorithm that bounds the amount of error present in a LQR-Tree policy by leveraging reinforcement learning theory and distribution-free regression theory. From reinforcement learning we use a generalization of the Bellman Residual [16] (Fig. 1) to provide a measure of how well the current set of controllers accomplishes the task for a given reward function. We then upper bound this theoretical quantity with a sampled version using tools from distribution-free regression theory [4].

The algorithm developed in this paper inherits all of the favorable properties of LQR-Trees. Primarily, it is capable of constructing controllers as the state space scales, operates directly on the continuous state and action spaces, has verifiable stability, and considers feedback in the planning process. This work adds the ability to bound the error from optimal, an essential aspect for many tasks.

II. Background

LQR-Tree [13] is a prominent feedback motion planning algorithm. It uses advances in automatic controller construction to form a set of controllers for underactuated nonlinear systems. LQR-Trees is a sampling-based planner [6] but the local planner is replaced with a method that creates provably stable controllers. It continues to add controllers, stably sequencing them, until they cover a bounded region of configuration space. The components of the algorithm follow.

[1]Duke University Computer Science, cbames@cs.duke.edu
[2]Brown University Computer Science, gdk@cs.brown.edu

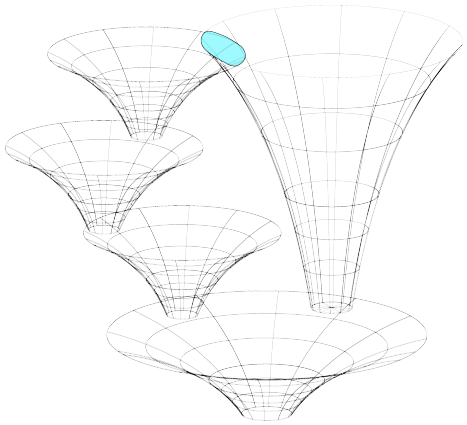


Fig. 1: An illustration of controller overlap. The cyan region highlights the overlap between funnels, which can be used for estimating the Bellman Residual. The residual is used to determine whether the two funnels are consistent in their value estimate.

A. Stable Controller Construction

The first major component of the algorithm constructs a stable controller that connects two states. First a reference trajectory between the two states must be found. Trajectory optimization finds a sequence of states and actions that will take the robot from the start point to the goal while obeying constraints enforced by the physics of the robot and attempts to minimize the cost function. However, it can only be guaranteed to find a local minimum. The output of trajectory optimization is a single trajectory and if the robot ever deviates from it, there is no defined control behavior. Therefore, the LQR-Tree algorithm wraps each trajectory in a Time-Varying Linear-Quadratic Regulator (TV-LQR) [1]. Before we review TV-LQR, we will briefly touch on the Linear-Quadratic Regulator.

1) Linear-Quadratic Regulator (LQR): The Linear-Quadratic Regulator (LQR) [1] assumes the state dynamics are linear in the state and action variables, and that the cost function is quadratic in the state and action variables. When these assumptions are satisfied a continuous problem of high dimension can be solved directly. The transition function of the LQR is defined as:

$$\dot{s}(s,a) = As + Ba. \tag{1}$$

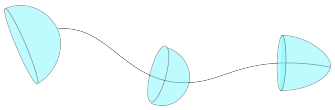


Fig. 2: Time Varying LQR is an extension of LQR to handle trajectory following. This figure depicts three different value functions for three different time points along the trajectory. Since the LQR problem is solved in relation to the trajectory the value function solutions will change in time. Thus some portions may have larger regions of stability than others.

A describes how an agent's state would evolve passively over time and B describes how an action changes the state of the agent. s is the state of the system and a is the action taken. The cost function of LQR is defined as:

$$c(s, a) = s^\intercal Q s + a^\intercal R a, \tag{2}$$

where Q weights the how much the state affects the cost incurred and R is the cost associated with taking an action. LQR is heavily used in control theory because an optimal solution is known for the continuous case. The value function of an LQR problem is found via the Discrete Algebraic Riccati equation:

$$P = A^\intercal P A + (A^\intercal P B)(R + B^\intercal P B)^{-1}(B^\intercal P A) + Q. \tag{3}$$

The value function for this control problem is:

$$V(s) = s^\intercal P s. \tag{4}$$

The ease of computing the solution is balanced by the rather stringent requirements that the state dynamics are linear, and the cost is quadratic. While this might seem to limit the applicability of LQR, locally linearizing has proved to be an effective solution for many systems [12].

2) Time-Varying LQR (TV-LQR): Time-Varying LQR [1] is an extension of LQR which handles a time-varying goal point, i.e., a trajectory. TV-LQR approximates the region around a trajectory as an LQR problem and provides feedback commands for states that are off of the trajectory.

The state dynamics for the time-varying problem are linearized as:

$$\dot{s}(s, a) = A(t)s + B(t)a. \tag{5}$$

The value function and control are then determined in relation to the goal trajectory, $\tilde{s}$, given:

$$V(s, a, t) = \int_t^T \big(a^\intercal R a + (s - \tilde{s}(t))^\intercal Q (s - \tilde{s}(t))\big) dt.$$

The feedback control is then a linear term, K, summed with the command selected by the trajectory optimizer, a_{traj}:

$$a_{TV} = K^\intercal s + a_{traj}. \tag{6}$$

Combining TV-LQR with trajectory optimization allows for a small region of acceptable states around a trajectory to be stabilized. Figure 2 depicts how the value function changes along a trajectory at three different time points.

While the TV-LQR feedback controller is defined for all states off of the trajectory, it is not stable for all of those points. In order to determine the region of stability around the trajectory, the LQR-Tree algorithm uses a convex optimization approach which constructs an ellipse that is an inner approximation of the region of stability. Any point inside the region is stable in the sense of Lyapunov.

3) Lyapunov Stability Analysis: Lyapunov stability analysis is a method for ensuring that a dynamical system reaches a region around an equilibrium point. If a controller can be constructed whose value function has a derivative that is negative everywhere but at the goal, then that controller is guaranteed to converge to the goal region:

$$\dot{V} = \nabla V f(s(t), a(t)) \leq 0. \tag{7}$$

Intuitively, the negative derivative of the value function for a controller ensures that every step of every trajectory decreases in cost. The region around the goal point for which the value function has a negative derivative is the region of attraction. Any point in the region of attraction is guaranteed, asymptotically, to converge to the goal.

There are several ways to construct Lyapunov functions and their associated regions of attraction. The Sum-of-Squares (SOS) [14] approach uses semi-definite programming and a polynomial description of the dynamical system to construct a provably conservative estimate of the Lyapunov function. However, it does not scale well to high dimensions. This is in part due to the explosion of polynomial terms as the dimension of the system increases. There are also simulation-based approaches [10] which rely on probabilistic arguments for defining their regions of attraction. Most recently, the simulation-based approaches were paired with function approximators [11] to learn oddly-shaped regions of attraction.

B. Space Covering

The LQR-Trees algorithm continues to construct these stability region bounded controllers until a bounded region of the state space has been covered in controllers. The addition of controllers happens in a method similar to RRT [7]. A random point, not contained in any current controller's stability region, is selected. If a stable controller can be constructed from that point to any point already on the tree then that controller is added. If a controller cannot be constructed then a new point is selected. Figure 1 provides an idea of how the tree might grow to cover the configuration space.

While the LQR-Trees algorithm provides a way to construct provably stable policies for underactuated nonlinear systems it is not capable of providing a bound on the error of the policies generated.

III. Bounded-Error LQR-Trees

We use a combination of optimal control methods and bounds from reinforcement learning to construct a bounded-error policy for nonlinear underactuated systems. The bounded error allows for the policies that are generated to be

optimized with respect to a cost function. In addition, having bounded error provides some assurance that the system will perform as expected, which is becoming increasingly important as robotics begins to be applied in close proximity to humans and in safety-critical applications.

The method described here combines several existing methods in order to construct a set of controllers that cover the configuration space with verifiable and approximately optimal controllers. Trajectory optimization provides trajectories that are locally optimal and respect constraints. TV-LQR and Lyapunov analysis provide approximate methods for determining the value of states around the trajectory, and the boundaries of those estimates. Sampling-based motion planning provides a method for covering the space, as well as guidance on which neighbors to connect to. Finally, reinforcement learning theory provides a way to verify globally that the controllers are ϵ-optimal. Figure 3 provides an overview of the whole process.

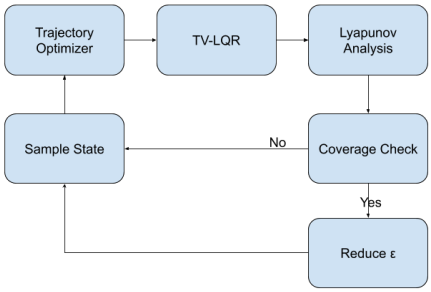


Fig. 3: A flow diagram of the system detailed in Section III. The start point for the diagram is Sample State.

A. Dynamic Programming and Trajectory Optimization

Trajectory optimization has significant benefits over dynamic programming as the dimensionality of the problem grows. However, trajectory optimization only finds locally optimal trajectories. We combine these two approaches to construct bounded cost controllers. Atkeson [2] notes that a Bellman Residual can be calculated at any point where two trajectories overlap. The Bellman Residual is a measure of the inconsistency in value function estimates. Each trajectory contains an estimate of the value function at that point, thus the difference of the value functions at that point is a sample of the Bellman Residual. This is accomplished by constructing a value function for every trajectory returned by the trajectory optimizer. The extent of the value function is limited by Lyapunov analysis. Lastly, this cycle is repeated until all controllers overlap with at least one other controller, and all samples returned are within the threshold on the Bellman residual.

1) Trajectory-Based Approximate Value Function: The first step is to approximate the value function of a task using a set of trajectories returned by the trajectory optimizer. The trajectory τ returned by the trajectory optimizer is the reference for a TV-LQR policy, which defines a value function centered on the trajectory. The region of stability for

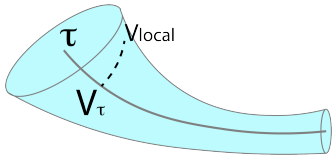


Fig. 4: The trajectory τ and its associated TV-LQR funnel are used to approximate the value at V_{local}.

the TV-LQR defines the extent to which the value function is considered valid. To be clear, the TV-LQR could approximate the value of any point, however here the approximation is restricted to the stable region around the trajectory. Only using the values from inside the stability region defined by the Lyapunov analysis ensures that only stable controllers are constructed.

In order to estimate the value of a particular state, first the value of the closest point, s_0, on a trajectory is found by backing up along the trajectory:

$$V_\tau(s_0) = \sum_{i=0}^{T} \gamma^i r(s_i, \tau(s_i)) + \gamma^{T+1} V(s_{T+1}), \quad (8)$$

where $V(s_{T+1})$ is the value of the end point the trajectory optimizer used, and γ is the discount factor, which weights how a reward received now should be traded with a reward in the future. The process of selecting the end point will be discussed in Section III-A.2. Next, the value function constructed by TV-LQR for that point on the trajectory is used to estimate the local value of the sample point:

$$V_{local}(s) = s^\intercal P_{local} s. \quad (9)$$

This local value is then combined with the backup from equation 8 to provide a global action-value estimate:

$$V(s) = V_{local}(s) + V_\tau(s_0). \quad (10)$$

This approximation is visualized in Figure 4. The local value estimate, V_{local}, performs two duties here. First, it provides an approximate value for the value function from a particular point on the trajectory. Note that, the value function, V_{local} returned by TV-LQR for a trajectory is not the same as the value function for the task, but rather it is the value function for stabilizing the system around the trajectory. Freeman and Primbs [3] demonstrate that the Lyapunov value function, in this case V_{local} is an upper bound on the optimal value function for some meaningful reward function. Primbs et al. [9] relate V_{local} and the optimal value function of a known reward function by a scalar multiple:

$$V_{Task}(s) \approx \lambda(s) V_{local}(s). \quad (11)$$

Knowledge of λ allows an approximation of the optimal value function to be calculated from the TV-LQR value function. Primbs et al. [9] demonstrate that λ can be derived from Sontag's formula [3], which results in the following:

$$\lambda(s) = 2\left(\frac{V_s A + \sqrt{(V_s A)^2 + Q(s)(V_s B B^t V_s^t)}}{V_s B B^t V_s^t}\right), \quad (12)$$

where V_s is the derivative of the local value function, V_{local}, with respect to the state space, A is the passive state dynamics, B is the dynamics resulting from control, and Q is the cost due to state only. This approximation works well in areas where the shape of the local value function and the optimal value function are similar. The Lyapunov analysis ensures that the local value function and the optimal value function are of similar shape in the stability region.

The second way the local value function is used is to determine the boundary around the trajectory:

$$\{s|V_{local}(s) < \epsilon\}. \tag{13}$$

Here the level set of the local value function is used to determine the region of stability for the controller. The combination of these two aspects permits a value function approximation to be constructed any time a point is contained within a funnel.

2) Neighbor Selection: In addition to refining the trajectories that a particular controller takes, the neighbor that a particular start state is connected to will also affect the optimality of the solution. Following Hauser and Zhou [5], we only consider neighbors with a decreasing cost as iterations increase. Thus, in the limit, the trajectory optimization will be using the approximately optimal value function, $\hat{V}^*$, to connect to the goal.

3) Termination Criteria: Convergence is determined by evaluating the Bellman Residual at sample points drawn from a uniform distribution over the state space. Since a point may belong to more than two funnels, only the two lowest cost estimates are used. If the lowest two approximations are within ϵ of each other then we bound the error from optimal using a technique from Williams and Baird [16]:

$$||V^* - V|| = \frac{\epsilon}{1-\gamma}. \tag{14}$$

We continue adding trajectories until all sampled points have two funnels with approximate values within ϵ of each other.

One difficulty with this approach is that the Williams and Baird bound is based on an ϵ that is defined over the entire value function of a task, and here it is being used to determine the convergence of a small number of funnels. However, because each funnel is a partial policy, and all partial policies are being held to this bound then the whole policy will also obey it. Another difficulty is that the Bellman Residual is defined over the whole function as opposed to samples. We address the validity of the sample bound in Section III-C.

B. Bounded-Error LQR-Trees Algorithm

The algorithm combines the two previous sections into an alternating algorithm which first constructs an LQR-Tree that probabilistically covers the configuration space. Lines 6-12 implement the same probabilistic coverage mechanism that was used in the original LQR-Trees paper [13]. The *ValidValue* function implements the bound from equation 14. *InFunnel* checks whether a point is contained in any of the regions of attraction for nodes of the LQR-Tree. The

Algorithm 1 Bounded-Error LQR-Trees

1: **procedure** CONSTRUCTLQRTREE($c, g, \alpha, \epsilon_{max}$)
2: ▷ The cost function, the goal, the contraction coefficient, and the maximum amount of error
3: $T \leftarrow \emptyset$
4: $T \leftarrow$ *LQR*(g) ▷ Add LQR solution for goal to tree
5: *coverThresh* $\leftarrow 459$
6: *coverSum* $\leftarrow 0$
7: $\epsilon = \frac{R_{max}}{2(1-\gamma)}$
8: **while** $\epsilon > \epsilon_{max}$ **do**
9: $\epsilon = \alpha * \epsilon$
10: **while** *coverSum* $>$ *coverThresh* **do**
11: $s \leftarrow$ *RandomConfiguration*
12: **if** *ValidValue*(s, ϵ) **then**
13: *coverSum* $\leftarrow$ *coverSum*$+1$
14: **else**
15: *coverSum* $\leftarrow 0$
16: **if** $\neg$*InFunnel*(s) **then**
17: $n \leftarrow$*SelectNeighbor*(s, T)
18: $\tau \leftarrow$ *ConstructFunnel*(s, n, c)
19: $T \leftarrow$ *AddFunnel*(τ, T)
20: **return** T

selectNeighbor function implements the method described in the Neighbor Selection subsection. *ConstructFunnel* uses a trajectory optimizer to connect s to s_n, where s_n is the zero point defined by the Lyapunov function at node n, such that the cost according to c is minimized. It then passes that trajectory to a TV-LQR method which constructs a candidate Lyapunov function that is then scaled by a Sum-of-Squares Lyapunov analysis. *AddFunnel* takes the previously constructed funnel and adds representative Lyapunov functions to the tree.

C. Empirical Bellman Residual Analysis

The algorithm presented in the previous section depends critically on the existence of a sample-based upper bound on the Bellman Residual. We call this the Empirical Bellman Residual:

Definition 1.

$$B_n = \frac{1}{n}\sum_{i}^{n}(f_a(x_i) - f_b(x_i))^2, \tag{15}$$

where f_a is the value function from one funnel, and f_b is the value function from another funnel.

Now we prove that the Empirical Bellman Residual is an upper bound for the value function approximation that we have constructed. The approach taken here is similar to Maillard et al. [8].

Theorem 1. The Bellman Residual is upper bounded by:

$$B < B_n + 32\sqrt{\frac{2\log(\frac{16}{\delta})}{n}}L^2. \tag{16}$$

L is the upper bound on the discounted value function, and $1-\delta$ is the probability that this bound holds.

Proof: Györfi et al. [4], in Theorem 9.1, provide the following general Hoeffding bound:

$$P\big(\sup_{g\in\mathcal{G}}\big|\frac{1}{n}\sum^{n} g(X_i) - \boldsymbol{E}\{g(X)\}\big| > \epsilon\big) < 8\boldsymbol{E}\{\mathcal{N}_1(\frac{\epsilon}{8},\mathcal{G},X^n)\}e^{\frac{-n\epsilon^2}{128M^2}}, \quad (17)$$

where $\mathcal{G}$ is the class of functions that are fit to samples X^n. M is a bound on the value that samples drawn from X can take, $-M < x < M | x \sim X$. n is the number of samples drawn. $\mathcal{N}_1$ is the ϵ-covering number for functions in $\mathcal{G}$ w.r.t. the 1-norm, and $\frac{\epsilon}{8}$ distance between them. In order to determine M a bound for:

$$(f_a(x) - f_b(x))^2,$$

must be determined. We know that the maximum value a value function can reach in the discounted case is $\frac{R_{max}}{1-\gamma}$, where R_{max} is the maximum reward achievable from the reward function. In order to be consistent with Györfi et al. [4] we refer to $\frac{R_{max}}{1-\gamma}$ as L:

$$(f_a(x) - f_b(x))^2 < (L - (-L))^2 = 4L^2.$$

This allows us to substitute $M = 4L^2$ into equation 17. Next we bound the ϵ-covering number for piecewise polynomial function approximators that fit LQR solutions to each region. From Györfi et al. [4], Lemma 9.3, we know that the ϵ-covering number for the *l2*-norm can be upper bounded:

$$\mathcal{N}_2 \leq \left(\frac{4W+\epsilon}{\epsilon}\right)^D.$$

W, which must be > 0, bounds the squared average value of the function. D is the dimension of the function class.

However, we need a bound for the covering number with respect to the *l1*-norm. This is obtained by multiplying by the number of *l1*-norm ϵ balls that can fit inside a a unit *l2*-norm ball:

$$\mathcal{N}_1 \leq (\frac{2}{\epsilon})^D \mathcal{N}_2 = \left(\frac{8W+2\epsilon}{\epsilon^2}\right)^D.$$

For LQR-Trees the dimension of the function class must be the number of polynomial basis functions plus one, times the number of partitions, which is k below. This holds because the function approximator constructed is a piecewise polynomial approximation where every partition is fit with the same degree polynomial. This can be combine with equation 17 by setting $W = \frac{\epsilon(2^{\frac{1}{k}}\epsilon - 2)}{8}$. Thus we can upper bound equation 17 by the following:

$$\begin{aligned} &P\big(\sup_{g\in\mathcal{G}_{LQR-Tree}}\big|\frac{1}{n}\sum^{n} g(X_i) - \boldsymbol{E}\{g(X)\}\big| > \epsilon\big) \\ &<8\left(\frac{\epsilon(2^{\frac{1}{k}}\epsilon - 2) + 2\epsilon}{\epsilon^2}\right)^k e^{\frac{-n\epsilon^2}{2048L^4}} \\ &=8\left(2^{\frac{1}{k}}\right)^k e^{\frac{-n\epsilon^2}{2048L^4}}. \end{aligned}$$

The function class, $\mathcal{G}_{LQR-Tree}$, is the piecewise polynomial class that contains all representations built by LQR-Trees. Next we set $\epsilon = 32\sqrt{\frac{2\log(\frac{16}{\delta})}{n}}L^2$:

$$\begin{aligned} &P\big(\sup_{g\in\mathcal{G}_{LQR-Tree}}\big|\frac{1}{n}\sum^{n} g(X_i) - \boldsymbol{E}\{g(X)\}\big| > \epsilon\big) \\ &<8\left(2^{\frac{1}{k}}\right)^k e^{\frac{-n\left(32\sqrt{\frac{2\log(\frac{16}{\delta})}{n}}L^2\right)^2}{2048L^4}} \\ &=8\left(2^{\frac{1}{k}}\right)^k \frac{\delta}{16} \\ &=\delta. \end{aligned}$$

This implies that with probability $1-\delta$ equation 16 holds. ■

D. Experiments

In order to validate the model we evaluate its performance on the constrained inverted pendulum problem. The constrained inverted pendulum problem was chosen because it is an underactuated nonlinear control problem, which makes it a reasonable first test for robotics control frameworks. The approach detailed in the previous section is compared to two baselines: a Value Iteration dynamic programming solution that uses state space discretization, and LQR-Trees.

Comparisons between the baselines and the proposed approach were performed by randomly selecting a start state from the configuration space of the pendulum. The goal in all experiments was to reach the inverted position, 3.14, with velocity 0. Trajectories from all the different baselines were re-sampled so that they used the same time step. Then a trapezoidal integration was used on the re-sampled trajectories to calculate the cost. This was repeated for 100 samples to gather sufficient statistics on max cost and average cost. All experiments were carried out in MATLAB using Drake [15].

E. Results

In order to compare bounded error LQR-Trees to the two baselines the value function created by the value iteration baseline was considered to be V^*. In Figure 5 several different bounded error LQR-Trees are compared to the upper bound that they were given. Every tree respects the upper bound it was provided, as we would expect from our proof. The horizontal yellow line is the maximum observed error from the baseline LQR-Tree. In summary, Figure 5 demonstrates that the bound from equation 14 holds. This validates that equation 14 holds if there are multiple partial policies being concatenated. It also validates that the approximation of the value function via the Primbs multiplier (eqn. 12) and a quadratic Lyapunov function are an upper bound on the optimal value function.

Next we compare the value functions produced by Value Iteration and the first iteration of the bounded LQR-Trees approach. Figure 6a is the approximate value function constructed by the proposed method after one iteration. Figure 6b is the value function created by Value Iteration. The

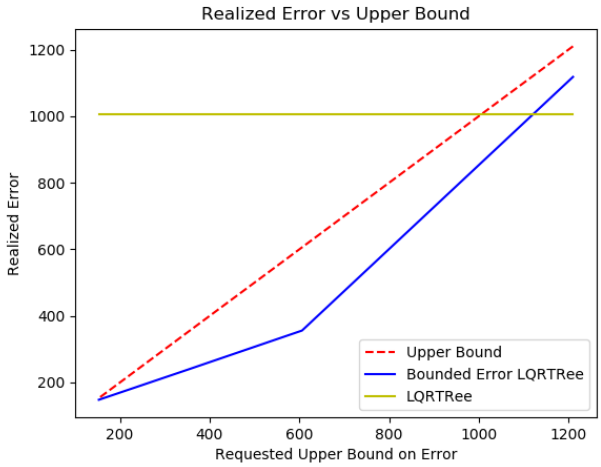
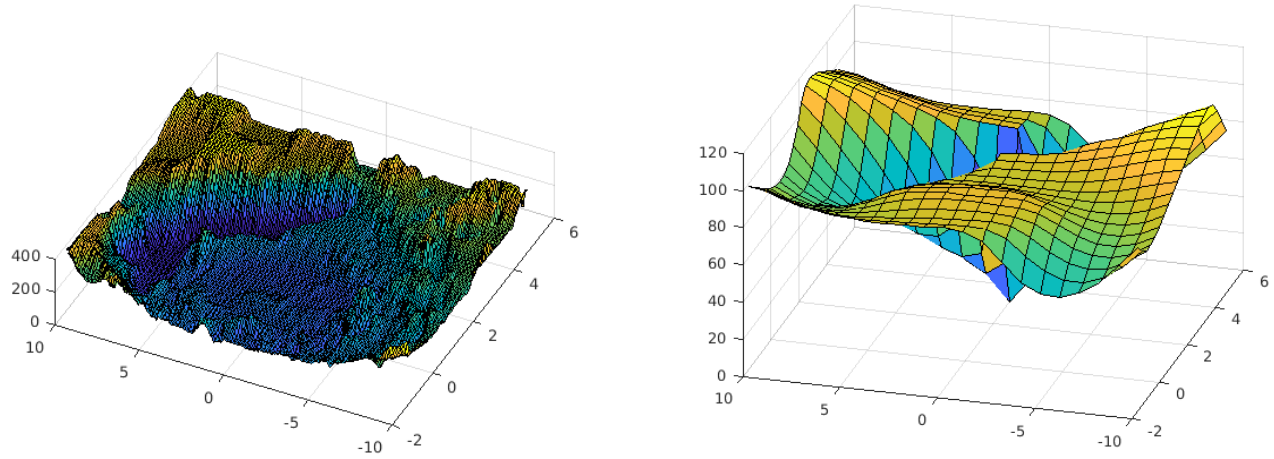


Fig. 5: A comparison of Bounded Error LQR-Trees and LQR-Trees. The red line is the error upper bound. The yellow line is the maximum observed error of the LQR-Trees method. The blue line is the observed maximum error for the Bounded-Error LQR-Trees method described in Section III. All the bounded error trees constructed stay under the requested error bound.



(a) The approximate value function constructed by sampling from the Lyapunov stability regions.

(b) The value function constructed by Value Iteration for the same problem.

Fig. 6: The LQR-Trees value function is similar in structure to the optimal value function from Value Iteration.

overall structures of the two value functions are similar. Both contain a deep valley near the solution, with large ridges near the corners $(4.71, 10)$ and $(4.71, -10)$. In addition, they both contain a small bump centered on $(0, 0)$. The major difference between the two is the sharpness of the creases. This is due to the application of Lyapunov stability analysis to the approximation of values. In regions where the dynamics might change rapidly, equation 7 may no longer hold true, and thus the region of attraction will have boundaries on or near rapid changes in the dynamics. This prevents the value approximation from smoothing out creases as is seen in Figure 6b. For example, the regions of attraction found in the valley around the solution, $(3.14, 0)$, are aligned with the long axis of the valley, because the dynamics of the system rapidly changes once a certain velocity threshold is surpassed. This threshold corresponds to the top of the ridge that defines the solution valley. In addition, since the trajectory optimizer is discretizing in time as opposed to space, there is no lower bound on how fine the representation can be in regions of rapidly changing dynamics. Lastly, the value function constructed according to the proposed method estimates the cost-to-go of states to be much higher; this is to be expected as the trajectory optimizer is a nonlinear optimization. This will be corrected as more iterations are performed and poorly performing funnels are replaced.

IV. Related Work

Trajectory Centric Reinforcement Learning [2] and LQR-Trees [10, 13] are the two pieces of work closest to our method. While Trajectory Centric Reinforcement Learning

(TCRL) uses the Bellman Residual to construct optimal value functions from sampled trajectories, it does not provide a principled way for sampling new trajectory start points. It also assumes that all trajectories end at the goal. While this is a reasonable assumption for the inverted pendulum, trajectory optimizers will have an easier time reaching subgoals in high dimension nonlinear dynamics. Further, TCRL uses the iLQR [12] to approximate the value function of the trajectory. While this is reasonable for approximation, it says nothing about the stability of points off the trajectory. LQR-Trees is the direct predecessor to this work, sharing the most in common with the approach outlined here. However, it is missing any notion of optimality: the policies returned by it are stable and will reach the goal, but may be sub-optimal. The proposed approach tackles this by combining Primbs' multiplier with the Bellman Residual approach of TCRL to provide an upper bound on the error of the LQR-Tree.

V. Conclusion

We introduced a new algorithm that constructs bounded error policies for underactuated nonlinear systems. The performance of the resulting policies were bounded by the Bellman Residual, for which a sample based upper bound was constructed. Constructing funnels which minimized this Empirical Bellman Residual resulted in LQR-Tree policies that performed significantly better than vanilla LQR-Trees, as was demonstrated on the inverted pendulum. This work represents a significant step towards optimal and scalable nonlinear control.

ACKNOWLEDGMENTS

This research was supported in part by DARPA under agreement number D15AP00104, and the ONR under the PERISCOPE MURI Contract N00014-17-1-2699. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The content is solely the responsibility of the authors and does not necessarily represent the official views of DARPA. Barrett Ames was supported by a NDSEG Fellowship. A special thanks to Jiayue Fan for her insightful comments on drafts of this paper.

References

[1] B.D.O. Anderson and J.B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice Hall, 1990.

[2] C. Atkeson and B. Stephens. Random Sampling of States in Dynamic Programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(4):924–929, 8 2008.

[3] R.A. Freeman and J.A. Primbs. Control Lyapunov Functions: New Ideas From an Old Source. In *Proceedings of 35th IEEE Conference on Decision and Control*, 1996.

[4] L. Györfi, M. Kohler, A. Krzyzak, and H. Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer, 2002.

[5] K. Hauser and Y. Zhou. Asymptotically Optimal Planning by Feasible Kinodynamic Planning in State-Cost Space. *IEEE Transactions on Robotics*, 32(6):1431–1443, 2016.

[6] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[7] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[8] O-A. Maillard, R. Munos, A. Lazaric, M. Ghavamzadeh, M. Sugiyama, and Q. Yang. Finite-Sample Analysis of Bellman Residual Minimization. In *Proceedings of the 2nd Asian Conference on Machine Learning*, pages 299–314, 2010.

[9] J.A. Primbs, V. Nevistić, and J.C. Doyle. Nonlinear Optimal Control: A Control Lyapunov Function and Receding Horizon Perspective. *Asian Journal of Control*, 1(1):14–24, 1999.

[10] P. Reist, P. Preiswerk, and R. Tedrake. Feedback-motion-planning with simulation-based LQR-trees. *The International Journal of Robotics Research*, 35(11):1393–1416, 2016.

[11] S.M. Richards, F. Berkenkamp, and A. Krause. The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamical Systems. In *Proceedings of the International Conference on Robot Learning*, 2018.

[12] Y. Tassa, N. Mansard, and E. Todorov. Control-Limited Differential Dynamic Programming. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, pages 1168–1175, 2014.

[13] R. Tedrake. LQR-Trees: Feedback Motion Planning on Sparse Randomized Trees. In *Robotics: Science and Systems V*, 2009.

[14] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. 2018. URL http://underactuated.csail.mit.edu/.

[15] Russ Tedrake and Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016. URL https://drake.mit.edu.

[16] R.J. Williams and L.C. Baird. Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions. Technical Report NU-CCS-93-14, Northeastern University, Boston MA, 1993.