LQR-Trees with Modified Distance Heuristics

Alex Choi
Dept. of Electrical and Computer Engineering
University of Washington
Seattle, USA
choialex@uw.edu

***Abstract*—The LQR-Trees algorithm is a sample-based planning framework that constructs feedback policies by incrementally covering state space with locally optimal controllers. A critical step in this process involves selecting the nearest tree node for trajectory generation, typically using an affine quadratic regulator (AQR)-based heuristic that computes a cost-to-go from the nearest tree node to a random sample. In this work, we propose a modified "rand-to-near" heuristic that reverses this direction, computing cost-to-go from the random sample to the existing node. We derive the new cost formulation via Pontryagin's Minimum Principle and integrate it into an LQR-Tree implementation for a two-state pendulum. Empirical results across multiple simulations demonstrate that the proposed heuristic yields trees with significantly shorter total trajectory duration (p = 0.03) and fewer nodes, with a modest increase in offline computation time. These findings indicate that the modified heuristic formulation can better exploit system dynamics and improve planning efficiency in offline computation.**

***Index Terms*—LQR-Trees, Motion Planning, Trajectory Optimization**

I. Introduction

The goal of feedback motion planning is to provide a control policy that can stabilize a region of state space towards some goal. This policy ideally stabilizes a large range of initial conditions that fill some, potentially nonconvex, region of state space. This builds upon the classical idea of trajectory stabilization which attempts to stabilize the space around a single nominal trajectory.

For instance, Model Predictive Control (MPC) can be considered a feedback motion planning algorithm, as it generates trajectories online to stabilize towards some goal or nominal trajectory from any region of state space. However, properties such as recursive feasibility and stability are not guaranteed in general for nonconvex state spaces. One solution is to combine the controller with some form of global motion planning algorithm such as Rapidly exploring Random Trees (RRT) or Probabilistic Road Maps (PRM). However, traditional formulations of these planning algorithms do not account for system dynamics, and the paths they propose may not be dynamically favorable.

An early attempt at addressing this global feedback motion planning problem is LQR-Trees [1]. This algorithm iteratively fills a region of state space with judiciously chosen open loop trajectories and continuous time LQR controllers. This algorithm promotes trajectory sparsity while ensuring stability by utilizing regions of attraction (ROAs). These are regions of state space where a given controller is guaranteed to stabilize the system to a state or trajectory of states.

When the tree is extended, a time minimizing affine quadratic regulator (AQR) is used to find the tree node which is dynamically closest to a randomly sampled point. In this paper, we investigate the particular distance heuristic used for finding the nearest tree node and provide an alternative formulation which results in faster open-loop trajectories and fewer nodes on average. We attribute this to the ability of the heuristic to select dynamically favorable tree nodes. We demonstrate this on a simple pendulum which has an easy to visualize state space.

II. Background

A. LQR-Trees

As described in [1], the LQR-Trees algorithm works as follows. An infinite horizon LQR controller is designed for the goal state and goal control located at some equilibrium point in state space. A region of attraction (ROA) is then calculated for this controller and the node is added to the tree. A node is randomly sampled from outside this ROA in the region of state space that is being stabilized. A heuristic is used to find the dynamically nearest tree node to this random point. Some method is then used to extend a trajectory from that nearest node towards the random sample in multiple steps (in an example, [1] splits the trajectory into eight segments). For each segment, a time varying LQR controller is used to stabilize the trajectory, and the regions of attraction are calculated for each segment. These nodes are then added to the tree and a new point is sampled. This is repeated until the user specified region of state space has been covered by the tree's region of attraction.

B. Distance Heuristic

Traditionally, RRT uses the node with the closest euclidean distance. However, in systems with dynamics, this is not necessarily favorable. LQR-Trees takes the approach from [2] which proposes to use the optimal cost-to-go for the following final-time free, affine quadratic regulator problem. Here, a trajectory is defined from the nearest tree node, $\mathbf{x}_{\text{near}}$, to the randomly sampled node, $\mathbf{x}_{\text{rand}}$ such that $\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_{\text{rand}}$.

$$J(\bar{\mathbf{x}}_0, t_0, t_f) = \int_{t_0}^{t_f} \left[1 + \frac{1}{2}\bar{\mathbf{u}}^{\intercal}(t)\mathbf{R}\bar{\mathbf{u}}(t)\right] dt, \tag{1}$$

$$\text{s.t.} \quad \bar{\mathbf{x}}(t_f) = \mathbf{0},\ \bar{\mathbf{x}}(t_0) = \bar{\mathbf{x}}_0,\ \dot{\bar{\mathbf{x}}} = \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}} + \mathbf{c} \tag{2}$$

The solution is fast to compute, even across multiple nodes. This provides an optimal cost-to-go, a candidate optimal control policy, and the corresponding time horizon. The original motivation for this formulation was for constructing an RRT formulation where trajectories branch outward from a single initial node to cover some region of state space. Notably, this is in contrast to the formulation of LQR-Trees which aims to solve the reverse problem, forming many trajectories that point towards a single goal node. This discrepancy will be explored in III-C.

C. SOS Optimization

Sum-of-Squares (SOS) programming provides one method for verifying ROAs by analyzing the validity of Lyapunov functions in some sublevel set. The verification method in LQR-Trees is built on top of the work in [3]. Extensions and alternatives for the ROA verification methods used in LQR-Trees have been described in part in [4] including the use of sampling methods to avoid time dependence. In general, our analysis only requires that the same verification method be used when comparing the alternative RRT distance heuristics. For simplicity, a constant radius ROA is assumed for the remainder of the analysis.

D. Software Packages

The analysis in this paper was done by implementing the LQR-Tree algorithm in Python. Modeling and simulation of the dynamic systems was done using the auto-differentiation framework in Jax [5]. Symbolic manipulation, SOS optimization, and spline manipulation were done using pydrake [6]. The full repository can be found at https://github.com/aechoi/lqrtree.

III. Algorithm Implementation

To test the effect of the alternative distance heuristic, the rest of the LQR-Trees algorithm must be constructed. There are many possible ways to implement this algorithm. We detail our particular implementation before discussing the derivation of the new heuristic.

A. Models

The system under investigation is a simple two-state pendulum. This allows for the state space to be viewed on a two-axis plot where the trees can be directly observed. The exact formulation for the state-space model can be found in A. The goal is to make an LQR-Tree that can complete an underactuated swing up from the state space region defined by $0 \leq \theta \leq 2\pi$ and $-10 \leq \dot{\theta} \leq 10$. Symmetric, hard control limits are enforced to prevent the creation of arbitrarily fast trajectories.

Continuous and discrete formulations of the dynamics are implemented. The discrete time dynamics are used in the trajectory optimization for numeric tractability. The symbolic continuous time formulation is needed for the SOS-based Lyapunov analysis used in calculating the ROAs.

B. Trajectory Generation

When adding a new random sample to the tree, a trajectory must be extended from the nearest node towards the new sample. Work in [1] proposes the use of the control policy derived from the optimal cost-to-go provided by the distance heuristic. However, it also suggests the use of some nonlinear trajectory optimization routine to refine this initial guess, as the derived control is only valid near the point where the dynamics are linearized.

We implement a minimum-time, single shot Sequential Quadratic Programming (SQP) trajectory optimizer. An error coordinate system is constructed with $\bar{\mathbf{x}}_k = \mathbf{x}_k - \mathbf{x}_{\text{near}}$ where $\mathbf{x}_{\text{rand}}$ is the random sample and initial state, and $\mathbf{x}_{\text{near}}$ is the nearest tree node and final state.

$$
\begin{aligned}
\min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & \left(\sum_{k=0}^{T-1} \bar{\mathbf{x}}_k^\intercal \mathbf{Q} \bar{\mathbf{x}}_k + \bar{\mathbf{u}}_k^\intercal \mathbf{R} \bar{\mathbf{u}}_k\right) \\
& \quad + \alpha \left(\sum_{k=0}^{T} \left(\mathbf{x}_k - \mathbf{x}_k^{\text{prev}}\right)^2 + \sum_{k=0}^{T-1} \left(\mathbf{u}_k - \mathbf{u}_k^{\text{prev}}\right)^2\right) \\
\text{s.t.} \quad & \mathbf{x}_0 = \mathbf{x}_{\text{rand}}, \ \mathbf{x}_T = \mathbf{x}_{\text{near}} \\
& \mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k + \mathbf{c}_k \\
& |\mathbf{u}_k| < \mathbf{u}_{\text{lim}}
\end{aligned}
$$

A linear interpolation from the initial states to the goal states is used as the initial $\mathbf{x}^{\text{prev}}[\cdot]$. This provides a re-scalable initialization separate from the trajectory generated by the distance heuristic's candidate open loop control policy. Each instance of SQP is initialized with $\mathbf{u}^{\text{prev}}[\cdot] = \mathbf{0}$.

Each time this optimization is run, a new state and control trajectory is generated, around which the dynamics are linearized. These are used as the next iteration's $\mathbf{x}^{\text{prev}}[\cdot]$ and $\mathbf{u}^{\text{prev}}[\cdot]$. The parameter α adjusts how quickly the nominal trajectory shifts between SQP iterations. This prevents large deviations which can unexpectedly increase the total cost of the trajectory due to changes in the nonlinear dynamics. For this work, the value was hand-tuned to 0.1 to reduce the likelihood that SQP would fail to converge. Each iteration of SQP is run until the total trajectory costs converge to within a precision tolerance (0.1%).

The final-time of the trajectory is minimized iteratively using binary search over the SQP loop. The amount of control effort needed to achieve the two fixed state constraints can change drastically at various time horizons. By minimizing final-time while respecting the control limits, a suitable trajectory is guaranteed.

C. Modifying the AQR-Based Distance Heuristic

The original affine quadratic regulator in [2] was designed in the context of starting at some initial node and branching outwards to fill the state space (one-to-many). However, the LQR-Trees formulation reverses that notion, starting at some final node that the policy tries to stabilize towards and then reaching backwards in time to fill the rest of state space (many-to-one).

As seen in 2, the original LQR-trees formulation defines a cost based on how much time and control effort is needed to reach the randomly sampled node from some tree node. This "near-to-rand" formulation generates and compares control policies that cause states to move away from the tree. This makes sense in an RRT setting where a single initial condition is branching outwards, but does not align with the goals of LQR-trees which funnel trajectories to a single node.

To address this discrepancy, we propose the following modification to the minimum time formulation which swaps the initial and final conditions. This "rand-to-near" construction reframes the problem to calculate the cost-to-go starting from the random point and ending at the nearest tree node. The coordinates used and the point of linearization are kept the same as the original formulation.

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

To understand the impact of this "rand-to-near" formulation has on the distance heuristic, the optimal cost-to-go must be re-derived. Following a similar derivation from [2], we apply Pontryagin's minimum principle. First, the Hamiltonian is defined.

$$H(t) = 1 + \frac{1}{2}\bar{\mathbf{u}}^\intercal(t)\mathbf{R}\bar{\mathbf{u}} + \lambda(t)(\mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\bar{\mathbf{u}} + \mathbf{c})$$

For optimality, the following first order conditions must hold.

$$\frac{\partial H}{\partial \bar{\mathbf{x}}} = -\dot{\lambda}(t) = \mathbf{A}^\intercal\lambda(t)$$
$$\implies \lambda(t) = e^{A^\intercal(T-t)}\lambda(T)$$

$$\frac{\partial H}{\partial \bar{\mathbf{u}}} = 0 = \mathbf{R}\bar{\mathbf{u}} + \mathbf{B}^\intercal\lambda(t)$$
$$\implies \bar{\mathbf{u}}(t) = -\mathbf{R}^{-1}\mathbf{B}^\intercal e^{\mathbf{A}^\intercal(T-t)}\lambda(T)$$

We desire an expression for $\bar{\mathbf{u}}(t)$ so that the cost can be evaluated. To do that, an expression for $\lambda(T)$ must be found. Plugging the expression for $\bar{\mathbf{u}}(t)$ into the dynamics results in the following expression.

$$\dot{\bar{\mathbf{x}}}(t) = \mathbf{A}\bar{\mathbf{x}}(t) - \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\intercal e^{\mathbf{A}^\intercal(T-t)}\lambda(T) + \mathbf{c}$$

Now, an integral is taken so that this relationship can be expressed as a function of the boundary conditions of $\bar{\mathbf{x}}$. Until now, the derivation has been identical to [2]. However, our boundary conditions are swapped, so our resulting expression will now be different. Using the Leibniz integral rule results in the following.

$$\bar{\mathbf{x}}(T) = \bar{\mathbf{x}}_0 = -P(T)\lambda(T) + \int_0^T e^{\mathbf{A}^\intercal(T-\tau)}\mathbf{c}\,d\tau$$
$$P(t) \equiv \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\intercal e^{\mathbf{A}^\intercal(t-\tau)}d\tau$$

Using the notation from [1], the continuous reachability gramian, $P(t)$, remains the same. This gramian is also guaranteed to be invertible due to it's symmetry and the positive definiteness of R. Solving for $\lambda(T)$ results in the following.

$$\begin{aligned}\lambda(T) =& P(T)^{-1}\left(-\bar{\mathbf{x}}_0 + \int_0^T e^{\mathbf{A}^\intercal(t-\tau)}d\tau\right)\\ =& P(T)^{-1}d(\bar{\mathbf{x}}_0, T)\end{aligned}$$

The general structure of $\lambda(T)$ is the same, but the expression for $d(\cdot)$ is modified from its original definition. As a result, the general form of the optimal cost-to-go will also be identical.

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

In order to vectorize the calculation of the nearest node, a decomposition of d can be done such that the tree node position and integration are separated. This means the integration only needs to be done once.

$$d(\bar{\mathbf{x}}_0, T) = -\bar{\mathbf{x}}_0 + r(T)$$
$$\dot{r}(t) = \mathbf{A}r(t) + \mathbf{c},\ r(0) = 0$$

The details of the vectorized block matrix cost calculation can be found in [7].

IV. Results

Simulations were run to generate trees using the two different distance heuristics to stabilize the pendulum at its unstable equilibrium. Figure 1 depicts one such instance of an LQR-Tree for the original "near-to-rand" formulation, while figure 2 depicts an instance of the "rand-to-near" heuristic.

Some metric is needed to quantify the quality of any given tree. Ideally, both heuristics would be directly compared on the same sequence of nodes. However, different nearest-node selections will lead to different generated trajectories. These can lead to differences in feasibility when attempting to add the same nodes. Instead, we assume that for any given tree that covers a desired area of state-space, the most efficient node-selecting heuristic would minimize the time it takes to reach the final goal. To quantify the quality of the tree, we compare its total length (measured in seconds) by adding the duration of each branch between nodes.

Five simulations of each heuristic were conducted, each lasting about 3 minutes. The results are summarized in Table I. A one-sided Welch's t-test claiming the modified heuristic out-performs the original heuristic results in a p-value of 0.03. This suggests that the rand-to-near heuristic results in significantly shorter trees than the original near-to-rand formulation. Additionally, the rand-to-near heuristic required fewer tree nodes on average, again suggesting that the modified heuristic more efficiently spans the relevant region of state space.

The trajectory success rate, defined as the percentage of attempted connections that were feasible trajectories, was slightly lower for the rand-to-near heuristic. The lower this metric is, the longer it takes to generate the offline tree.

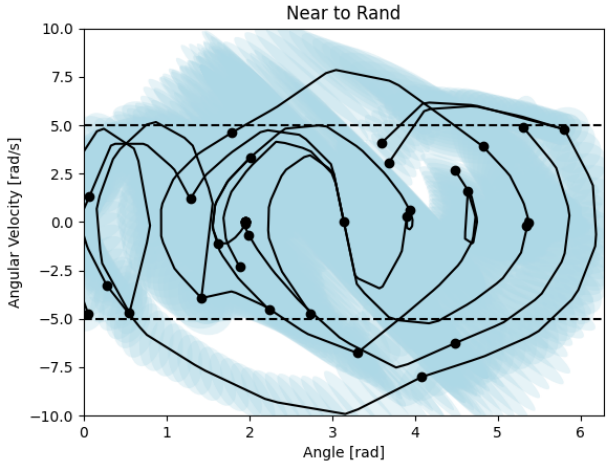

Fig. 1: The original distance metric used in LQR-Trees which is calculated from the nearest tree node to the random sample.

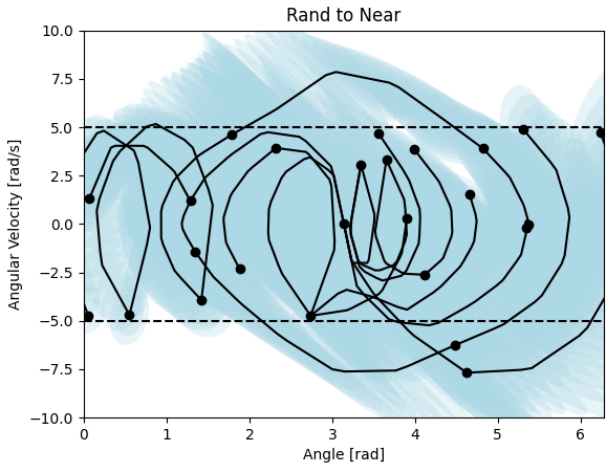

Fig. 2: The modified heuristic which calculates from the random sample to the nearest tree node.

TABLE I: A comparison of the two heuristics

	Near-to-Rand	Rand-to-Near
Avg Total Time [s]	16.48	14.56
Total Time Std Dev [s]	1.31	1.48
Total Nodes [-]	31.8	27.6
Trajectory Success Rate [%]	73.8%	65.9%

V. Discussion

The results presented above suggest that the proposed rand-to-near distance heuristic offers a meaningful improvement over the original near-to-rand formulation in the context of LQR-Trees. On average, the trees generated using the modified heuristic achieved the control objective in less total time and required fewer nodes to span the desired region of state space. These improvements indicate that the new heuristic may better leverage the dynamics of the system by choosing nearest-nodes that are dynamically closer.

This performance gain seems to come at the cost of a lower trajectory success rate. One possible explanation is that the these faster trajectories are generally more aggressive, leading to a higher rate of trajectory infeasibility. The resulting increase in compute time may not be much of a detriment, as it appears to only impact the offline tree generation.

While the original goal was to test this heuristic with time-varying regions of attraction (ROAs), implementation challenges prevented this. The main road block was an inability to get the sum-of-squares optimization to successfully run on the system dynamics. Numeric tests had shown that the Lyapunov function was suitable in a given region, however the solver was unable to find sufficient s-procedure multipliers. Future work should investigate how the heuristic performs when integrated into the full LQR-Trees pipeline. It would also be valuable to see how well the comparative performance of the two heuristics generalize to more complex systems of higher order.

References

[1] R. Tedrake, "Lqr-trees: Feedback motion planning on sparse randomized trees," in *Proceedings of the 5th Robotics: Science and Systems Conference (RSS)*, Seattle, USA, Jun. 2009, june 28–July 1, University of Washington.

[2] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 5021–5028.

[3] P. A. Parrilo, "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization," Ph.D. dissertation, California Institute of Technology, Pasadena, CA, May 2000, ph.D. thesis, advised by John Doyle. [Online]. Available: https://thesis.library.caltech.edu/1647/

[4] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010, originally presented in preliminary form at RSS 2009.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[6] R. Tedrake and the Drake Development Team, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," http://drake.mit.edu, 2016.

[7] E. L. Glassman, "A quadratic regulator-based heuristic for rapidly exploring state space," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2010. [Online]. Available: https://groups.csail.mit.edu/robotics-center/public_papers/Glassman10a.pdf

APPENDICES

A. State Space Model

TABLE II: Model Parameters and Variables

Symbol	Description	Value	Unit
θ	Counter-clockwise angle from down	-	rad
ω	Angular velocity	-	rad/s
m	Mass	1.0	kg
l	Length	0.5	m
g	Gravitational acceleration	9.81	m/s^2
b	Angular damping rate	0.1	-

$$\mathbf{x} = \begin{bmatrix} \theta & \omega \end{bmatrix}^\intercal$$

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ -\frac{g}{l}\sin(\theta) - \frac{b}{ml^2}\omega + \frac{1}{ml^2}u \end{bmatrix}$$