An Exploration of Global Planning in LQR-Trees

David DePauw
University of Pennsylvania
Philadelphia, PA, USA
daviddep@seas.upenn.edu

Raymond Bjorkman
University of Pennsylvania
Philadelphia, PA, USA
raybjork@seas.upenn.edu

***Abstract*—The planning class of LQR-Trees seeks to provide an answer to the problem of robust stabilizing control of nonlinear underactuated systems, but has not been widely studied yet. This algorithm generates a library of trajectories with corresponding linear quadratic regulator (LQR) controllers and regions of attraction (ROAs) which are guaranteed through Sums of Squares (SOS) optimization. These trajectories can be linked through a probabilistic global planner to potentially cover a system's controllable state space. We aim to build off the existing research into this type of planner by attempting to address one of its shortcomings - the suboptimality of the global trajectories it generates.**

I. Introduction

Over the past couple decades the robotics community has made great strides in the field of global planning for high dimensional systems through the probabilistically generated Rapidly-exploring Random Trees (RRTs). However, the benefits of this research did not immediately translate to improvements for highly dynamical and underactuated systems. With uncertainty in model and unpredictability in our world it is difficult to trust open loop trajectories created through this method. The success of a robotic system should not hinge entirely around a single trajectory being executed as intended. Our research was motivated by the need for a contingency plan when things don't go as planned.

Model predictive control (MPC) exists as a solution to this problem, but its limitations lie in its limited prediction horizon and the high computational cost that this online computation incurs. Explicit MPC ameliorates this issue by providing a precomputed network of polyhedral control but require a set of simplifying assumptions to ensure that the optimization problem is a quadratic program (QP).

LQR-Trees are a newer invention with a very promising upside. They leverage the benefits from probabilistic planning by precomputing locally optimized trajectories and controllers with conservative approximations for their ROA. These trajectories can then be composed into a tree. A controllable state will exist inside one of these ROAs ensuring positive invariance, and guaranteeing that this state will eventually travel to the goal position [9]. If this plan should fail, the algorithm provides a built in contingency plan by recalculating the system's state, finding an ROA that encompasses it and following its corresponding trajectory. The ability to globally plan trajectories for underactuated systems with controllability guarantees motivated us to further investigate improvements to LQR-Trees.

The problem with precomputed controllers such as explicit MPC or LQR-Trees is that they scale poorly with system dimensionality. Since this limitation is somewhat inherent to this philosophy of control, we did not explore improving them in this aspect. Instead we chose to focus on improving their global planning capabilities by introducing a new method of growing the tree by looking at both local cost to connect to a node and global cost of the resultant trajectory. This paper will first walk through the decisions we made in our MATLAB implementation of the LQR-Trees algorithm and then will go more into depth on our proposed changes to it.

II. Background

A. LQR-Tree Algorithm

The algorithm builds a tree in which each of its nodes is a structure storing data defining a trajectory through state space and relevant information relating to its corresponding ROA. In each loop it samples the state space and tries to create a trajectory to connect this node to the tree, which is initialized a goal state and input x_G, u_G. In this way the global planning properties of LQR-Trees are similar to RRTs. Where they differ lies within the way in which trajectories are generated and nodes are connected.

Algorithm 1 outlines this method at the highest level as originally proposed by Tedrake et al [9]. Subroutines referenced in capital letters could have many possible implementations and the choices we made for our own implementation will be explained in further detail in Section III.

Algorithm 1: LQR-Tree(x, u)

```
[A, B] ⇐ linearize(x, u)
[K, L] ⇐ solution to infinite horizon lqr(A, B, Q, R)
ρ ⇐ CALCULATE_ROA(S)
Tree.add_Node(x, u, A, B, K, L, ρ)
while state space is not filled do
    x_s ⇐ randomly sample state space
    if not inside an ROA then
        x_near ⇐ FIND_NEAREST_NODE(x_s)
        [x_d, u_d] ⇐ FIND_TRAJECTORY(x_near, x_s)
        [A(t), B(t), K(t), L(t)] ⇐ tvlqr(x_d, u_d)
        ρ(t) ⇐ CALCULATE_ROA(S)
        Tree.add_Node(x_d, u_d, A(t), B(t), K(t), L(t), ρ(t))
    end
end
```

B. SOS Optimization

LQR-Trees make use of SOS optimization to calculate and verify the ROA around a nominal point or trajectory. This is done by converting linear matrix inequality (LMI) constraints into a semidefinite program (SDP), and finding the minimum objective value that lies along the semidefinite constraint cone [2]. This means that each inequality will be represented as a matrix with the constraint on it being that it must remain positive semidefinite (throughout this paper we will refer to such matricies as being "SOS"). These constraints are generally derived from logical inequalities that guarantee some property of the system dynamics. For that reason, this paper will outline the inequality constraints used in our SOS programs, followed by their conversion into corresponding SOS constraints. In order to tie these inequalities together into logical implications, the semialgebraic conditions can be removed through the introduction of slack variables via the S-procedure [7].

III. Implementation Details

A. Direct Collocation

In order to build an LQR-Tree, nodes within the tree must be linked together with valid trajectories through state-space with arbitrary endpoints $[x_0, x_N]$ provided by the global planner. For our local trajectory generation routine we decided to use Hermite-Simpson collocation to find an optimal trajectory for the system [5]. In order to generate these trajectories we solved the optimization program corresponding to (1) using MATLAB's `fmincon`. While direct collocation may introduce small errors from approximating trajectories as cubic splines, it is often better numerically conditioned than multiple shooting as direct collocation features a sparse constraint matrix. In addition to penalizing input we also chose to make the trajectory time step a decision variable and add cost τ that scales proportional to it in order to incentivize shorter, more direct paths. This allows for a trade-off between time and energy efficiency.

$$\begin{aligned}
\min_{z} \quad & \sum_{i=0}^{N-1} \frac{\Delta t}{2}\left(u_i^2 + u_{i+1}^2 + \tau\right) \\
\text{s.t.} \quad & x_{i+\frac{1}{2}} = \frac{1}{2}(x_i + x_{i+1}) + \frac{\Delta t}{8}(f_i - f_{i+1}) \\
& x_{i+1} - x_i = \frac{\Delta t}{6}(f_i + 4f_{i+\frac{1}{2}} + f_{i+1}) \\
& 0 \le u \le u_{max} \\
& \Delta t_{min} \le \Delta t \le \Delta t_{max} \\
& z = [x_1, u_1, \ldots, x_{N-1}, u_{N-1}, x_N, \Delta t]
\end{aligned} \tag{1}$$

B. Trajectory Stabilization via Time-Varying LQR

We used a time-varying linear quadratic regulator (TVLQR) to track trajectories created by direct collocation as outlined in the LQR-Tree algorithm. This controller has guaranteed convergence to a nominal trajectory within the neighborhood of that trajectory. The control policy of TVLQR directly depends upon cost-to-go which is found by solving the continous-time Ricatti differential equation as outlined in (2).

To improve numerical stability of the solution, we used the Cholesky factorization of the solution S such that $S(t) = L(t)^T L(t)$. This differential equation is solved by integrating backwards in time to match the nominal trajectory's solution at the point where the trajectories intersect, $L_i(t_i)$. To perform this integration we used the MATLAB function `ode45`.

$$\begin{aligned}
\dot{L}(t) = &-\frac{1}{2}QL(t)^{-T} - A(t)^T L(t) \\
&+ \frac{1}{2}L(t)L(t)^T B(t) R^{-1} B(t)^T L(t) \\
L(t_f) &= L_i(t_i)
\end{aligned} \tag{2}$$

A Lyapunov function for the system can be created from the solution to this equation.

$$V(x,t) = \frac{1}{2}\bar{x}L(t)^T L(t)\bar{x} \tag{3}$$

with $\bar{x} = x - x_d$ expressing the error from the trajectory. This function locally approximates the cost-to-go from a given point around the nominal trajectory x_d. We will use this function as a basis for implementing ROA verification through SOS.

Furthermore, finding the solution to this equation gives us the optimal controller for following this trajectory $u^* = -K(t)\bar{x}$ with

$$K(t) = R^{-1}B(t)^T L(t) L(t)^T \tag{4}$$

C. Polynomial Approximation of Dynamics

Since SOS programs make use of constraints that must be polynomial, we must then be able to express all conditions captured by the SOS program as polynomials. The dynamics of the systems we interrogated using LQR-Trees were nonlinear, so we first attempted to write represent the trigonometric functions they depended on as polynomials [2] subject to the constraint

$$\phi(x)(1 - s^2 + c^2) \quad \text{is SOS} \tag{5}$$

Properly numerically conditioning our SOS program became difficult with the introduction of this constraint, so to avoid slack variable bloat we substituted $\sin$ and $\cos$ terms with second-order Taylor Series linearizations given by

$$s(x, x_0) = \sin x_0 + (x - x_0)\cos x_0 - (x - x_0)^2 \sin x_0 \tag{6}$$

$$c(x, x_0) = \cos x_0 - (x - x_0)\sin x_0 - (x - x_0)^2 \cos x_0 \tag{7}$$

We tested our algorithm with two underactuated systems, a severely torque limited pendulum and a cartpole. The pendulum dynamics were then formulated through the equation

$$ml^2\ddot{\theta} + b\dot{\theta} + mgls(\theta, \theta_0) = u \tag{8}$$

To capture the fact that this system is underactuated torque limits on input will be enforced such that $u \le u_{max}$

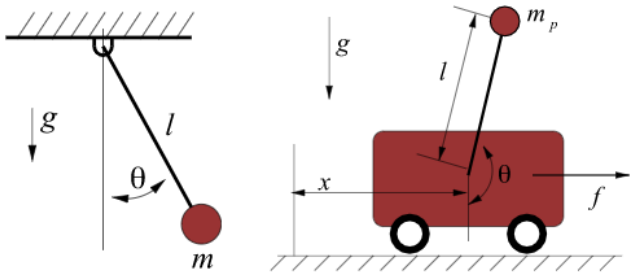

Fig. 1. Free body diagrams depicting the pendulum and cartpole systems Source: Adapted from [10]

Similarly cartpole dynamics we used were also written in terms of $s(\theta, \theta_0)$ and $c(\theta, \theta_0)$.

$$\begin{bmatrix} c(\theta,\theta_0) & l \\ m_c + m_p & m_p l c(\theta,\theta_0) \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -gs(\theta,\theta_0) \\ u + m_p l \dot{\theta}^2 s(\theta,\theta_0) \end{bmatrix} \quad (9)$$

D. Infinite Horizon LQR ROA Verification

A SOS program can be used to find the ROA for a TVLQR controller about a nominal trajectory as well as a for an infinite horizon LQR controller around a nominal position. Finding these ROA is a critical step in constructing an LQR-Tree as the controller can only guarantee convergence to a nominal position if the entirety of its controllable state space is filled by overlapping ROAs. This is done by implementing a SOS program to find the largest ρ-sublevel set, of the Lyapunov function. This region is positively invariant - no trajectory will exit this set upon entering it. This calculation can be done both for a infinite horizon LQR controller, or for a time varying Lyapunov function along a nominal trajectory.

For the infinite horizon LQR case, a point can be considered within the ROA if $\dot{V}$ is negative at that point. Therefore conditions for a valid ρ-sublevel set are defined as:

$$V(x) \leq \rho \Rightarrow \dot{V} \leq 0 \quad (10)$$

Since the system must also obey the imposed torque limits that make this pendulum underactuated, these limits must also be represented in the formulation of our SOS program. We will expand the ROA sublevel set ρ until input saturation is reached at any point that lies on the sublevel set. This is a conservative approximation for the region of attraction for a torque limited pendulum, as it does not attempt to determine if the sublevel set conditions could also be valid for other areas of input saturation than the boundary.

$$V(x) \leq \rho \Rightarrow Kx \leq u_{max} \quad (11)$$

$$V(x) \leq \rho \Rightarrow -Kx \leq u_{max} \quad (12)$$

Where K is the control gain from the infinite horizon LQR controller. We can now formulate these inequality constraints as SOS constraints to formulate the optimization program, the solution for which is illustrated by Figure 2.

$$\begin{aligned} \max_{\sigma_1,\sigma_2,\sigma_3} \quad & \rho \\ \text{s.t.} \quad & V - \rho - (1+\sigma_1)\dot{V} \text{is SOS} \\ & V - \rho - (1+\sigma_2)(-K\bar{x} - u_{max}) \text{is SOS} \\ & V - \rho - (1+\sigma_3)(K\bar{x} - u_{max}) \text{is SOS} \\ & \sigma_1, \sigma_2, \sigma_3 \text{is SOS} \end{aligned} \quad (13)$$

To prepare this and other SOS programs in this paper we utilized the MATLAB distribution of SPOTLESS which subsequently called on MOSEK to solve the underlying SDP.

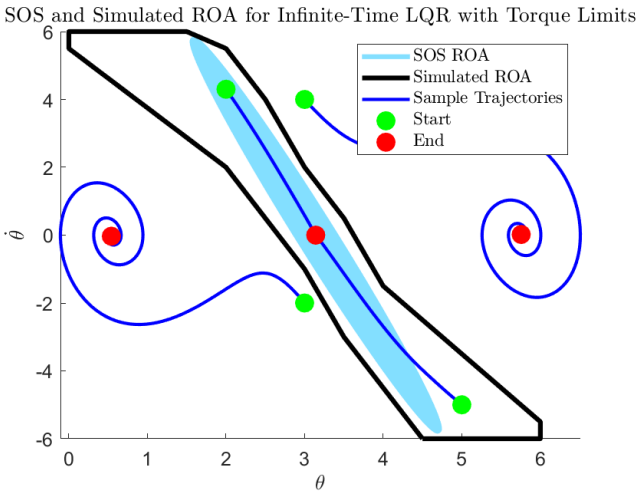

Fig. 2. Simulated ROA vs. SOS ROA for upright pendulum position. Trajectories are added as examples for each regime.

E. ROA Verification of TVLQR Controllers

Transitioning from ROA verification for an infinite horizon LQR controller to a TVLQR controller requires additional considerations. In addition to the constraints imposed by equations (10) - (12), additional constraints must be added to ensure a point within the ROA stays within the ρ-sublevel set for the duration of the trajectory. This can be accomplished by requiring that the derivative of the sublevel set, $\dot{\rho}(t)$, be greater than $\dot{V}(x,t)$ at all points along the boundary of the sublevel set to ensure that the Lyapunov function is contracting faster than the valid ROA. We also need to impose that the final calculated ROA of a trajectory should lie completely within the ROA of the connecting node. Doing so means that transitions between trajectories that leave the ROA.

$$V(x,t) = \rho(t) \Rightarrow \dot{V}(x,t) \leq \dot{\rho(t)} \quad (14)$$

$$\rho(t_f) \leq \rho(t)_{next} \quad (15)$$

These inequalities are captured in the following SOS program that maximizes $\rho(t)$ along each time step, dt, in the trajectory. We found that better results could be achieved if a new $\rho(t)$

was calculated for each point using a separate SOS program starting with the end of the trajectory. $\dot{\rho}(t)$ is calculated using forward difference between the $\rho(t)$ decision variable in the current SOS program, and since we are iterating backwards through time $\rho(t+dt)$ is calculated in the previous SOS iteration. The SOS program is shown below

$$\begin{aligned}\max_{\rho,\sigma_1,\sigma_2,\sigma_3,\sigma_4,\phi_1,\phi_2} \quad & \rho \\ \text{s.t.} \quad & \sigma_1(\dot{\rho}-\dot{V})+\phi(V-\rho) \quad \text{is SOS} \\ & V-\rho-(1+\sigma_2)(-K\bar{x}-u_{max}) \quad \text{is SOS} \\ & V-\rho-(1+\sigma_3)(K\bar{x}-u_{max}) \quad \text{is SOS} \\ & \sigma_4(\rho-V)-\phi_2(\rho(N)-\rho_0) \quad \text{is SOS} \\ & \sigma_1,\sigma_2,\sigma_3,\sigma_4 \quad \text{is SOS}\end{aligned} \tag{16}$$

A bilinearity is produced between $p(t)$ and σ_4 in the following SOS program definition, which requires the SOS program to be reformulated to take advantage of into a bilinear alternations . The idea behind bilinear alternations involves holding half of the bilinear variables constant while a SOS program finds the optimal values for the other half. A second SOS program then takes the output of the first and holds those variables constant while searching for the variables originally held constant. These two SOS programs alternate and will converge to an optimal solution [6].

$$\begin{aligned}\max_{\gamma,\sigma_1,\sigma_2,\sigma_3,\sigma_4,\phi_1,\phi_2} \quad & \gamma \\ \text{s.t.} \quad & \sigma_1(\dot{\rho}-\dot{V})+\phi(V-\rho)-\gamma \quad \text{is SOS} \\ & V-\rho-(1+\sigma_2)(-K\bar{x}-u_{max}) \quad \text{is SOS} \\ & V-\rho-(1+\sigma_3)(K\bar{x}-u_{max}) \quad \text{is SOS} \\ & \sigma_4(\rho-V)-\phi_2(\rho(N)-\rho_0)-\gamma \quad \text{is SOS} \\ & \sigma_1,\sigma_2,\sigma_3,\sigma_4 \quad \text{is SOS}\end{aligned} \tag{17}$$

The SOS program (17) is the first alternation in our routine. ρ is initialized to the previous ρ in the trajectory and held constant while a feasibility program runs and finds acceptable values for the slack variables. The objective of this function γ is maximized to help address numerical conditioning issues that arise when performing these alternations. This addition to the algorithm will be discussed in more depth in Section VI

$$\begin{aligned}\max_{\rho} \quad & \rho \\ \text{s.t.} \quad & \sigma_1(\dot{\rho}-\dot{V})+\phi(V-\rho)-\gamma \quad \text{is SOS} \\ & V-\rho-(1+\sigma_2)(-K\bar{x}-u_{max}) \quad \text{is SOS} \\ & V-\rho-(1+\sigma_3)(K\bar{x}-u_{max}) \quad \text{is SOS} \\ & \sigma_4(\rho-V)-\phi_2(\rho(N)-\rho_0)-\gamma \quad \text{is SOS} \\ & \sigma_1,\sigma_2,\sigma_3,\sigma_4 \quad \text{is SOS}\end{aligned} \tag{18}$$

The second alternation takes in the slack variables which were just found in alternation A and holds them constant while searching for the maximum ρ for which the constraints still hold.

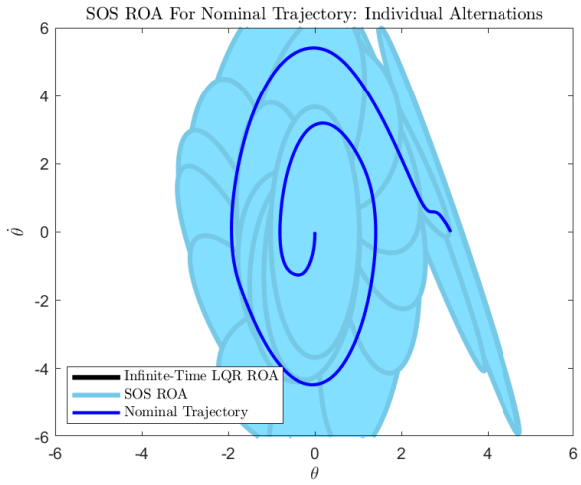

Fig. 3. Pendulum swingup trajectory with ROA verified through SOS. Individual ROAs are calculated sequentially starting from the goal node.

IV. Heuristic for Improved Tree Growth

A. LQR Cost-To-Go-Based Distance Metric

When globally linking trajectories there is a need for a good measure of "closeness" between points in state space. Traditionally RRT methods employ Euclidean distance as this metric, but this is not able to fully capture the dynamics of the system and can be inefficient as a result [9].

Glassman et al. proposes a heuristic for closeness by leveraging an LQR cost based distance metric that approximates the cost incured by a finite-horizon LQR controller as it drives the system from a newly sampled point, x_{new}, to a point on an existing trajectory, x_e [3]. To find this cost, a search must be done over a range of finite-horizon times to get the horizon that incurs the lowest cost. Glassman defines the cost and constraints of the LQR controller over a know time horizon as:

$$J(\hat{x},t_0,t_f)=\int_{t_0}^{t_f}[1+\frac{1}{2}\hat{u}^T(t)R\hat{u}(t)]dt \tag{19}$$

$$\hat{x}=x-x_{new} \tag{20}$$

$$\hat{x}(t_0)=x-x_e \tag{21}$$

$$\hat{x}(t_f)=0 \tag{22}$$

$$\tag{23}$$

where t_f is the final time horizon, $\hat{u}(t)$ is the control input over the finite-horizon controller, and R is the input penalty. The inverse of the cost-to-go of this LQR control can be found using Pontryagins minimum principle as:

$$\dot{P}(t)=AP(t)+P(t)A^T+BR^{-1}B^T s.t. P(t_f)=0 \tag{24}$$

The final cost-to-go for a given t_f and x_e can then be defined as:

$$J(\hat{x}, t_f) = t_f + \frac{1}{2} d^T(\hat{x}, t_f) P^{-1}(t_f) d(\hat{x}, t_f) \quad (25)$$

where

$$d(x, t_f) = e^{At_f} x + \int_0^{t_f} e^{A(t_f - \tau)} c d\tau \quad (26)$$

This method can be used to search for the point on an existing trajectory, x_e, that has the lowest cost to reach the newly sampled point. With this distance heuristic, the LQR-Tree can be extended in such a way as to always allow new points to connect to the tree in the least input expensive manner as possible [3].

B. Collocation-LQR Cost Hybrid Approach

Using this heuristic alone does not fully consider the cost of the global trajectory that results from connecting in this way and can lead to trajectories chaining off one another inefficiently. In practice when this is simulated it causes the pendulum or cartpole system to take unnecessarily long to swing up.

Our proposal aims to mitigate this issue by introducing collocation cost along a trajectory as a secondary measure of closeness to the goal by searching points along existing trajectories. By evaluating the objective function in (1) over the course of the trajectory x_e is sampled from, we can chart collocated cost as a function of state space. This mapping will be referred to as $G(x)$

For each queried point $x_e \in \mathcal{X}$ both $J(x_e)$ and $G(x_e)$ are calculated. We then borrow a method from operations engineering and define the Pareto front [4] of this set of points $\mathcal{X}^* \subseteq \mathcal{X}$ with the statement

$$x_e \in \mathcal{X}^* \Rightarrow \neg \exists x \in \mathcal{X} (V(x) < V(x_e) \wedge G(x) < G(x_e)) \quad (27)$$

Set $\mathcal{X}^*$ then represents a set of optimal solutions to the multiple-objective optimization problem posed by minimizing both G and V.

Visually we can understand this by plotting each x in "cost space" with coordinates $(V(x_e), G(x_e))$. The Pareto front corresponds to the highlighted region in Figure 4.

To select a the optimal solution from $\mathcal{X}^*$ we seek to minimize the norm of each point to find the point with the overall lowest cost. However, a small problem arises in that the costs of V and G are not scaled in the same way. To compensate for this we introduce a design variable α into the equation.

$$\min_{x \in \mathcal{X}^*} \quad V(x)^2 + \alpha G(x)^2 \quad (28)$$

By choosing a high value for α this heuristic will favor choosing nearby points (ones with low LQR cost) which will in turn create largely short trajectories that link to each other. In turn, choosing a low value for α will cause the trajectories to prioritize linking closer to the goal location. This means that each trajectory will be longer but will have a more optimal path to the goal. Thus selecting an appropriate α represents a trade-off between space complexity and optimality. Depending on the priorities for the system being designed either end of this spectrum could be a viable choice.

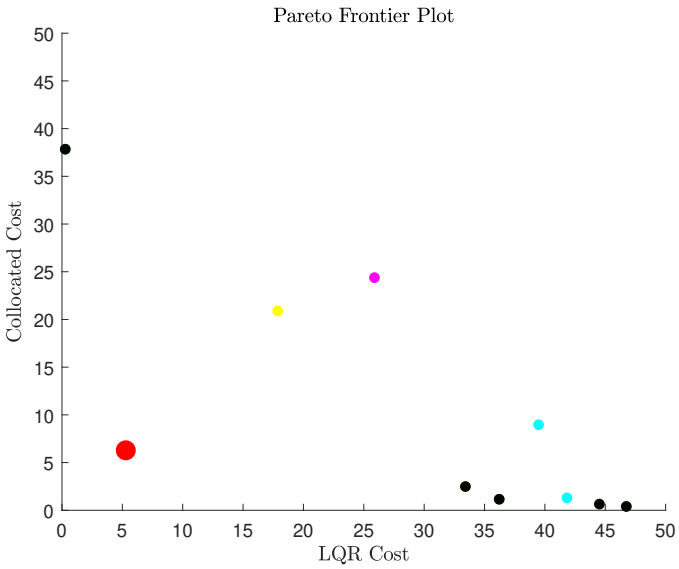

Fig. 4. A plot of the Pareto front for a growing pendulum LQR-Tree with five nodes. The black points demarcate the Pareto front, and the red point is the optimal choice to connect to. Each other point represents a potential connection point that has been dominated by the points making up the Pareto front, with each color representing a different trajectory.

V. Results

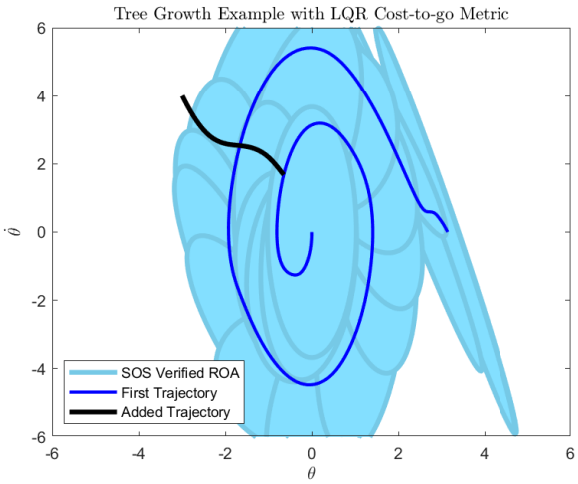

Fig. 5. Example LQR-Tree Growth for the LQR Cost-to-go distance metric. This trajectory connects to an easy to get to point from the sample position, even though it connects a point very early in the first trajectory

We have now successfully completed all the steps needed to construct an LQR-Tree as seen in Figure 7. As previously mentioned, our implementation can be expanded to include higher dimensional systems. To verify this, we tested our algorithm on a cartpole system and had success in building up an LQR-Tree to control it in the upright position.

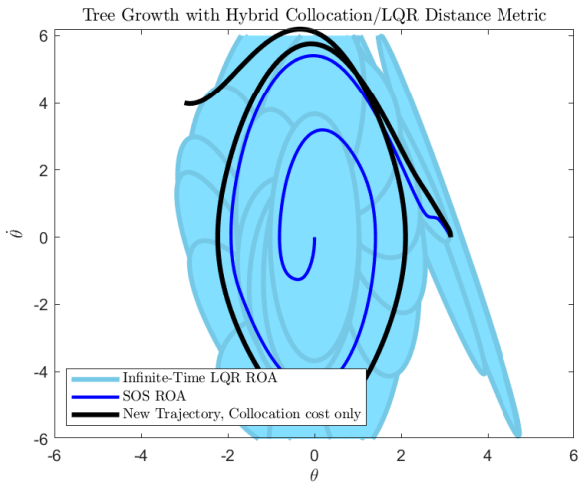

Fig. 6. LQR Tree Growth with hybrid distance metric.

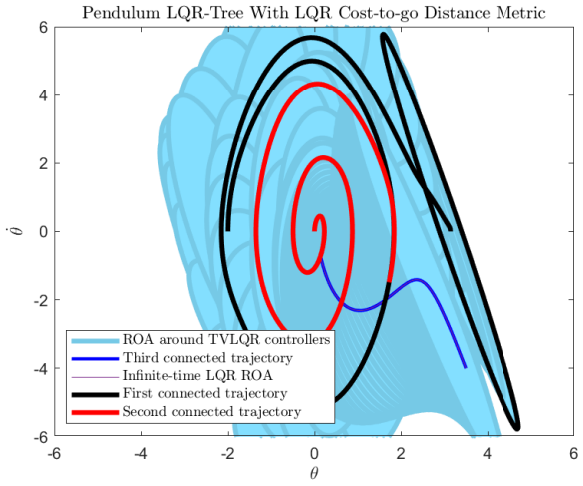

Fig. 7. Near complete LQR Tree using the LQR cost-to-go distance metric.

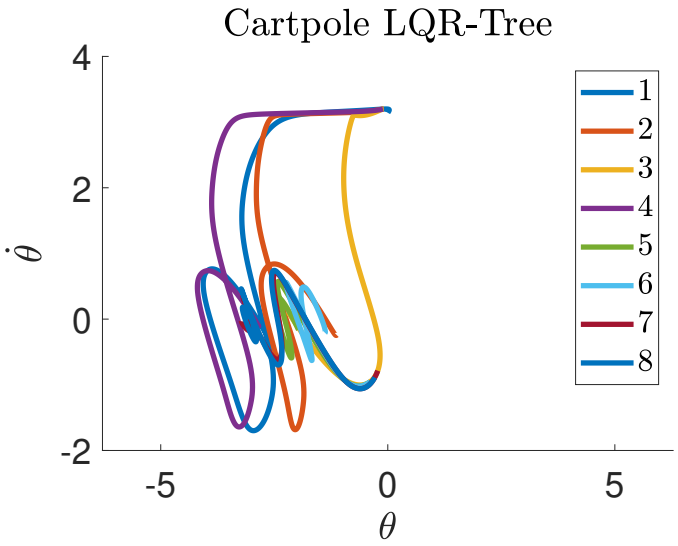

Fig. 8. Cartpole LQR-Tree projection on θ dimensions. The SOS ROA's are not included due to misleading projections from 4D to 2D.

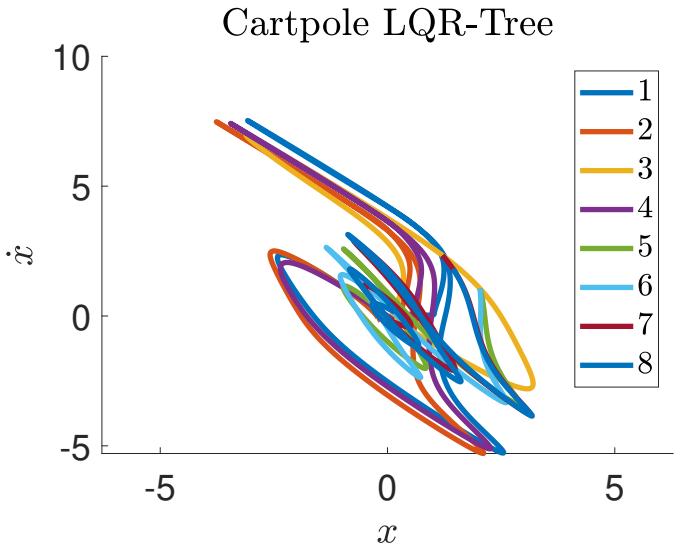

Fig. 9. Cartpole LQR-Tree projection on x dimensions corresponding with the LQR-Tree in Figure 8

While our collocation-LQR hybrid heuristic appeals to the logic connecting to points on the LQR-Tree that are closer to the setpoint than can be found with the LQR cost-to-go distance metric, we found this solution to be very difficult to tune in order to work generally. The majority of the attempts to use our hybrid distance metric ended with the trajectory simply trying to collocate toward the setpoint immeadiately. This is likely because the setpoint always has a collocation cost of 0, so unless it is very difficult for the sampled position to get there directly (resulting in a high LQR cost-to-go distance metric), the sample point will attempt to attach to the setpoint.

VI. Discussion and Conclusions

A. Implementation Problems and Solutions

The process of implementing LQR-Trees taught us about the practical limitations of the optimization methods we were using and troubleshooting methods to debug and work through problems. Foremost among the problems we encountered in this project was the issue of poor numerical conditioning for the optimization programs. In particular, bilinear alternations required extra attention to detail. Before the addition of γ as the objective function, Alternation A was providing feasible solutions on the edge of the constraint cone. When passed into Alternation B, these poorly conditioned slack variables would cause the optimization program to become "ill-posed". γ functions to pull the feasible slack variables found in Alternation A away from this boundary, where the objective function of Alternation B can be minimized more efficiently. This effect is illustrated in Figure 10.

A second concern we learned to pay attention to when implementing SOS programs was monitoring the degree of each polynomial in the SOS constraints. If one polynomial

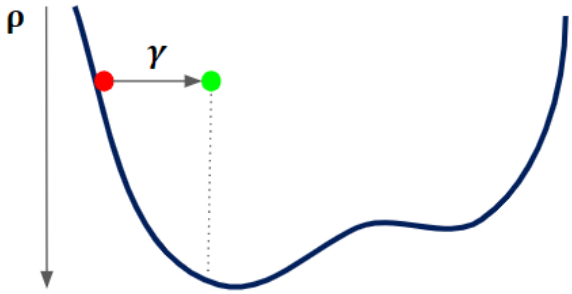

Fig. 10. The red dot represents a set of feasible slack variables from Alternation A that exist on the edge of the constraint boundary. The addition of γ serves to bring this solution closer into the center of the cone of possible solutions (represented by the green dot). The dotted line illustrates that it can be pulled further down than before to more efficiently maximize ρ.

term is of a comparatively higher order, that term will grow exponentially faster than the others and the program will be either badly conditioned or outright infeasible.

One of the ways we were able to improve over the SOS formulation as outlined by Majumdar and Tedrake was by simplifying the SOS program to find ROAs sequentially rather than all at once. We found that this method ensured better numerical conditioning across a wide variety of test cases. This is demonstrated through Figure 11. The drawbacks of taking this approach is that it is a computationally less efficient way of making the same calculation. Secondly, this was made possible by fact that we were using LQR as the Lyapunov function for SOS whereas in the literature it is typically found alongside ρ [6].

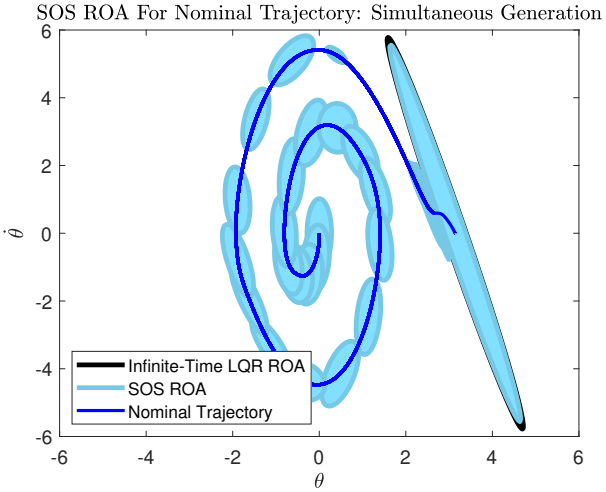

Fig. 11. Pendulum swingup trajectory calculated through a SOS program simultaneously finding the ROA of each trajectory sample point. When compared to Figure 3 which used a sequential ROA calculation in its SOS program, it is clear that this version has smaller ROA ellipses and is even missing regions entirely due to infeasibility of the program.

B. Future Work

There are still many new research directions that one could take to further investigate LQR-Trees. Firstly, since we were able to write generalized code which was agnostic of system dimensionality we were interested in trying to create an LQR-Tree to stabilize a compass-gait limit cycle.

In a similar vein, we are interested in implementing collocation for a pendulum on a circular manifold to avoid discontinuity problems with bounding θ between 0 and 2π. This in turn will make the heuristic approach we use for global trajectory planning more robust.

AWe are also interested in was applying the new innovations of DSOS and SDSOS to create LQR-Trees with potentially lower computational cost to calculate due to their status as linear programs [1].

References

[1] Ahmadi, Amir Ali, and Anirudha Majumdar. "DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization." SIAM Journal on Applied Algebra and Geometry 3.2 (2019): 193-230.

[2] Blekherman, Grigoriy. Semidefinite Optimization and Convex Algebraic Geometry. 2013.

[3] Glassman, Elena, and Russ Tedrake. "A Quadratic Regulator-based Heuristic for Rapidly Exploring State Space." IEEE International Conference on Robotics and Automation (ICRA), 2010

[4] Jahan, A., Edwards, K. and Bahraminasab, M. (n.d.). Multi-criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design.

[5] Kelly, Matthew. "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation." SIAM Review 59 (2017): 849-904.

[6] Majumdar, Anirudha, and Russ Tedrake. "Funnel Libraries for Real-Time Robust Feedback Motion Planning." The International Journal of Robotics Research, vol. 36, no. 8, July 2017, pp. 947–982.

[7] Posa, Michael et al. "Balancing and Step Recovery Capturability via Sums-of-Squares Optimization." Robotics: Science and Systems (2017).

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

[9] Tedrake, R., Manchester, I. R., Tobenkin, M., Roberts, J. W. (2010). LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification. The International Journal of Robotics Research, 29(8), 1038–1052.

[10] Russ Tedrake. Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832). Downloaded on 12/10/2019 from http://underactuated.mit.edu/