LQR-RRT*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics

Alejandro Perez, Robert Platt Jr., George Konidaris, Leslie Kaelbling and Tomas Lozano-Perez
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology
{aperez, rplatt, gdk, lpk, tlp}@csail.mit.edu

Abstract— **The RRT* algorithm has recently been proposed as an optimal extension to the standard RRT algorithm [1]. However, like RRT, RRT* is difficult to apply in problems with complicated or underactuated dynamics because it requires the design of a two domain-specific extension heuristics: a distance metric and node extension method. We propose automatically deriving these two heuristics for RRT* by locally linearizing the domain dynamics and applying linear quadratic regulation (LQR). The resulting algorithm, LQR-RRT*, finds optimal plans in domains with complex or underactuated dynamics without requiring domain-specific design choices. We demonstrate its application in domains that are successively torque-limited, underactuated, and in belief space.**

I. Introduction

The problem of planning dynamically feasible trajectories from an initial state to a goal state has recently been the subject of extensive study. Unfortunately, the general motion planning problem is known to be PSPACE-hard [2], [3]. In an effort to solve problems of practical interest, much recent research has been devoted to developing sampling-based planners such as the Rapidly-exploring Randomized Tree (RRT) [4] and the Probabilistic Road Map (PRM) [5]. These algorithms only achieve *probabilistic completeness*—that is, the probability that a solution is found (if one exists) converges to one as the number of samples approaches infinity; however, they have good average-case performance and work well on many problems of practical interest.

One of the key drawbacks of these sampling-based algorithms is that they often produce suboptimal solutions that "wander" through state space. In fact, Karaman and Frazzoli have recently shown that RRT almost surely does not find an optimal solution [1]. They introduced RRT*, an algorithm which extends RRT to achieve the *asymptotic optimality* property, i.e., almost-sure convergence to the optimal solution. Compared with the RRT, RRT* requires only a constant factor more computation. Like RRT, RRT* requires the user to specify two domain-specific extension heuristics: a distance metric and a node extension procedure. These components play a significant role in the algorithm: naive implementations (e.g., using the Euclidean distance) may not reflect the dynamics of particular domain, and therefore result in poor exploration and slow convergence to the optimal solution. Indeed, it has been shown that the performance of RRT-based algorithms is sensitive to the distance metric used [6].

Prior research has characterized the effect of the distance metric component on the convergence of RRT. Cheng and LaValle [7] showed that RRTs efficiently explore the state space only when this metric reflects the true cost-to-go. LaValle and Kuffner have previously noted that cost-to-go functions could be based on, among other options, optimal control for linearized systems [4]. Glassman and Tedrake [8] derived a cost-to-go pseudometric based on linear quadratic regulators (LQR), which was used to grow a standard RRT in domains with constrained and underactuated dynamics.

We propose an LQR-based heuristic to be used with the RRT* algorithm similar to the one proposed by Glassman and Tedrake [8] for the RRT. As in Glassman and Tedrake [8], we linearize the system dynamics about newly sampled points and use the LQR cost function to select the vertex in the tree to extend. However, in addition, we use the infinite horizon LQR policy to perform the extension, ultimately reaching a state that is potentially different from the sampled state. Finally, we evaluate the finite horizon LQR value function about this state and use it to evaluate distance during the RRT* "rewiring" step. The results are demonstrated on three underactuated robotics problems that are difficult to solve using naive extension heuristics: the torque-limited inverted pendulum, the acrobot, and a belief space planning problem.

II. Background

A. Problem Statement

Given a system with known (possibly non-linear) process dynamics, the optimal motion planning problem is to find a minimum cost feasible trajectory from an initial configuration to a goal configuration. Let *configuration space* be a compact set, $X \subseteq \mathcal{X}$, that describes all possible configurations of the system. Let $U \subseteq \mathcal{U}$ be a compact control input space. We assume that we are given the non-linear process dynamics:

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

A *dynamically feasible trajectory* is a continuous function, $\sigma : [0,1] \to X$, for which there exists a control function, $u : [0,1] \to U$, such that $\forall t \in [0,1], \dot{\sigma}(t) = f(\sigma(t), u(t))$. The set of all dynamically feasible trajectories is denoted by Σ_{traj}. Given an initial configuration, x_{init}, and a goal region X_{goal}, the motion planning problem is to find a dynamically feasible trajectory, σ, and a control, u, that starts in the initial configuration and reaches the goal region: $\sigma(0) = x_{\text{init}}$ and $\sigma(1) \in X_{\text{goal}}$. Let $c : \Sigma_{\text{traj}} \to \mathbb{R}^+$ be a *cost functional* that maps each feasible trajectory to a non-negative cost. The

optimal motion planning problem is to find a solution to the motion planning problem that minimizes $c(\sigma)$.

B. RRT*

Rapidly exploring random trees (RRTs) are a standard randomized approach to motion planning [4]. RRT* [1], shown in Algorithm 1, is a variant of this algorithm that has the *asymptotic optimality* property, i.e., almost-sure convergence to an optimal solution.

Algorithm 1: $\texttt{RRT}^*((V,E),N)$

1 **for** $i = 1, \ldots, N$ **do**
2 $\quad x_{\text{rand}} \leftarrow \texttt{Sample}$;
3 $\quad X_{\text{near}} \leftarrow \texttt{Near}(V, x_{\text{rand}})$;
4 $\quad (x_{\min}, \sigma_{\min}) \leftarrow \texttt{ChooseParent}(X_{\text{near}}, x_{\text{rand}})$;
5 $\quad$ **if** $\texttt{CollisionFree}(\sigma)$ **then**
6 $\quad\quad V \leftarrow V \cup \{x_{\text{rand}}\}$;
7 $\quad\quad E \leftarrow E \cup \{(x_{\min}, x_{\text{rand}})\}$;
8 $\quad\quad (V,E) \leftarrow \texttt{Rewire}((V,E), X_{\text{near}}, x_{\text{rand}})$;
9 **return** $G = (V,E)$;

Algorithm 2: $\texttt{ChooseParent}(X_{\text{near}}, x_{\text{rand}})$

1 $\text{minCost} \leftarrow \infty$; $x_{\min} \leftarrow \texttt{NULL}$; $\sigma_{\min} \leftarrow \texttt{NULL}$;
2 **for** $x_{\text{near}} \in X_{\text{near}}$ **do**
3 $\quad \sigma \leftarrow \texttt{Steer}(x_{\text{near}}, x_{\text{rand}})$;
4 $\quad$ **if** $\texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma) < \text{minCost}$ **then**
5 $\quad\quad \text{minCost} \leftarrow \texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma)$;
6 $\quad\quad x_{\min} \leftarrow x_{\text{near}}$; $\sigma_{\min} \leftarrow \sigma$;
7 **return** $(x_{\min}, \sigma_{\min})$;

Algorithm 3: $\texttt{Rewire}((V,E), X_{\text{near}}, x_{\text{rand}})$

1 **for** $x_{\text{near}} \in X_{\text{near}}$ **do**
2 $\quad \sigma \leftarrow \texttt{Steer}(x_{\text{rand}}, x_{\text{near}})$;
3 $\quad$ **if** $\texttt{Cost}(x_{\text{rand}}) + \texttt{Cost}(\sigma) < \texttt{Cost}(x_{\text{near}})$ **then**
4 $\quad\quad$ **if** $\texttt{CollisionFree}(\sigma)$ **then**
5 $\quad\quad\quad x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
6 $\quad\quad\quad E \leftarrow E \setminus \{x_{\text{parent}}, x_{\text{near}}\}$;
7 $\quad\quad\quad E \leftarrow E \cup \{x_{\text{rand}}, x_{\text{near}}\}$;
8 **return** (V,E);

The major components of the algorithm are:

- *Random sampling*: The `Sample` procedure provides independent uniformly distributed random samples of states from the configuration space.
- *Near nodes*: Given a set V of vertices in the tree and a state x, the $\texttt{Near}(V,x)$ procedure provides a set of states in V that are close to x. Here, closeness is measured with respect to some metric as follows:

$$\texttt{Near}(V,x) := \left\{ x' \in V \ : \ \|x - x'\| \leq \gamma \left(\frac{\log n}{n} \right)^{1/d} \right\},$$

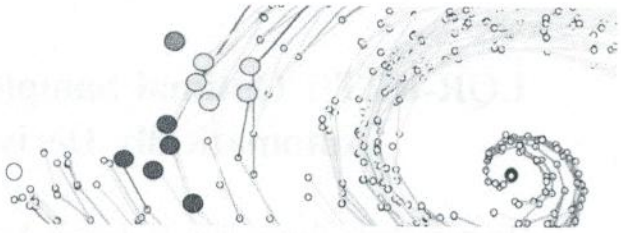


Fig. 1. "Near" neighborhood of nodes for x_{new} (red circle) on an iteration of the torque-limited pendulum. Nodes found using Euclidean distance as the cost metric (green circles) are closer to x_{new}. However, extension is not feasible from any of the considered states as these did not account for the dynamics of the system. The `LQRNear` calculation provides a neighborhood of nodes (blue circles) that are able to reach x_{new} with minimum effort. Policies in the tree are false-colored (red through blue) as cost increases.

where $\|\cdot\|$ is the distance using the metric, n is the number of vertices in the tree, d is the dimension of the space, and γ is a constant.

- *Steering*: Given two states x, x', the $\texttt{Steer}(x,x')$ procedure returns a path σ that connects x and x'. Most implementations connect two given states with a straight path in configuration space.
- *Collision checking*: Given a path σ, $\texttt{CollisionFree}(\sigma)$ returns true if the path lies in the obstacle-free portion of configuration space.

The algorithm proceeds as follows. Firstly, a state is sampled, denoted as x_{rand}, from the configuration space (Line 2), and near neighbors are evaluated using the `Near` function (Line 3).[1] Next, RRT* calls `ChooseParent` (Algorithm 2) to find a candidate for a parent node to x_{rand} (Line 4). `ChooseParent` searches through the nodes in the set X_{near} of near nodes and returns the node, denoted as $x_{\min}$, that reaches x_{rand} with minimum cost along with the path $\sigma_{\min}$ connecting $x_{\min}$ to x_{rand}. After that node is found, the algorithm checks $\sigma_{\min}$ for collision (Line 5). If the path is obstacle-free, the algorithm adds x_{rand} to the tree and connects $x_{\min}$ to x_{rand}, and attempts to "rewire" the nodes in X_{near} using the `Rewire` procedure (Algorithm 3). The `Rewire` procedure attempts to connect x_{rand} with each node in the set X_{near} of near nodes. If the path that connects x_{rand} with a near node x_{near} reaches x_{near} with cost less than that of its current parent, then the x_{near} is "rewired" to x_{rand} by connecting x_{rand} and x_{near}.

C. LQR

Linear quadratic regulation (LQR) is an approach to computing optimal control policies for linear systems with Gaussian process noise. In general, the problem of finding optimal control policies for arbitrary stochastic systems is has polynomial computational complexity in the number of value function parameters. However, when the system dynamics are linear, it is possible to solve for the optimal value function efficiently in closed form. Furthermore, it is possible to apply this approach to non-linear systems by linearizing the non-linear process dynamics about an operating point.

[1] Although it is not explicitly noted here, we assume that the algorithm proceeds to the next iteration if the `Near` function returns an empty set.

Suppose that we are given a continuous-time non-linear system,

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

and we are interested in computing a local policy that approximately optimizes a quadratic cost function defined about a point, (x_0, u_0). Taking the first-order Taylor expansion of f about (x_0, u_0), we have:

$$\dot{x} \approx f(x_0, u_0) + \frac{\partial f(x_0, u_0)}{\partial x}(x - x_0) + \frac{\partial f(x_0, u_0)}{\partial u}(u - u_0).$$

If we notice that $\dot{x}_0 = f(x_0, u_0)$ and define a change of variables, $\bar{x} = x - x_0$ and $\bar{u} = u - u_0$, then this becomes:

$$\dot{x} \approx \dot{x}_0 + A(x_0, u_0)\bar{x} + B(x_0, u_0)\bar{u},$$

or simply

$$\dot{\bar{x}} \approx A(x_0, u_0)\bar{x} + B(x_0, u_0)\bar{u},$$

where

$$A(x_0, u_0) = \frac{\partial f(x_0, u_0)}{\partial x}$$

and

$$B(x_0, u_0) = \frac{\partial f(x_0, u_0)}{\partial u}.$$

Given a cost function of the form:

$$J = \int_0^\infty \bar{x}^T Q \bar{x} + \bar{u}^T R \bar{u},$$

we can compute the locally optimal policy by solving the algebraic Riccati equation for the matrix, S:

$$A^T S + SA - SBR^{-1}B^T S + Q = 0,$$

where $A = A(x_0, u_0)$ and $B = B(x_0, u_0)$. It is possible to solve this equation for S efficiently, and many software algebra packages provide standard functions to do so, including the `lqr` function provided by Matlab. Given a solution, S, the optimal value function is:

$$V(\bar{x}) = \bar{x}^T S \bar{x}.$$

The optimal policy is

$$\pi^*(\bar{x}) = -K\bar{x}, \tag{1}$$

where

$$K = R^{-1}B(x_0, u_0)^T S$$

is known as the LQR *gain matrix*. In this paper, it will be important to recognize that the value function can be used to compute a cost to go from any point to the goal. Following Glassman and Tedrake [8], we note that this cost can be used as a measure of "distance": states with a high cost to go are further from the goal than those with a lower cost to go. Using cost to go as a distance metric results in efficient exploration of the state space [7]. It is these locally linearized dynamics that we use in LQR-RRT* to estimate distances and perform extensions. Initially in our algorithm, when the distances between the new node and the "near" nodes in the tree are large, this distance measure will be inaccurate. However, as the tree covers the state space more densely, the distances shrink and the local LQR policy becomes more closely optimal. In the following, we will use the notation

$$[K, S] = LQR(A, B, Q, R),$$

to denote the function that calculates the LQR gain matrix, K, and the cost matrix, S.

III. LQR AS AN EXTENSION HEURISTIC FOR RRT*

We present a variation of the RRT* algorithm that incorporates LQR for "near" and "nearest" node calculations as well as for extensions among nodes, given in Algorithm 4. In each iteration, the algorithm first samples a state, denoted as x_{rand}, from the configuration space (Line 2), and finds its nearest neighbor using the `LQRNearest` function (Line 5). Then, the algorithm calls the `LQRSteer` function to create a x_{new} state. This state represents the end of a trajectory executed with the LQR-policy generated from the `LQRNearest` function with a pre-specified step size based on the cost. The `LQRNear` procedure is then called on x_{new} to search for a neighborhood of nodes around this state. Then, `ChooseParent` finds a candidate for a parent node to x_{new} (Line 6). The `ChooseParent` procedure, presented in detail in Algorithm 5, simply searches through all the nodes in the set X_{near} of near nodes and returns the node, denoted as $x_{\min}$, that reaches x_{new} with minimum LQR S cost along with the trajectory $\sigma_{\min}$ that connects $x_{\min}$ to x_{new}. After that node is found, the algorithm checks $\sigma_{\min}$ for collision. If the path is obstacle-free, the algorithm adds x_{new} to the tree and connects $x_{\min}$ to x_{new}, and it attempts to "rewire" the nodes in X_{near} using the `Rewire` procedure, which is provided in Algorithm 6. The `Rewire` procedure attempts to connect x_{new} with each node in the set X_{near} of near nodes. If the trajectory that connects x_{new} with a near node x_{near} reaches x_{near} with cost less than that of its current parent, then the x_{near} is "rewired" to x_{new} by connecting x_{new} and x_{near}.

The algorithm primitives are outlined in detail below:

- *Nearest Vertex*: Given a set of vertices in the tree, V, and a state x, the `LQRNearest`(V, x) function returns the nearest node in the tree in terms of the LQR cost-to-go function. We locally linearize the system about x and the zero action, $u = 0$, and evaluate the LQR gain matrix, $K(x)$, and cost matrix, $S(x)$:

  $$[K(x), S(x)] = LQR(A(x, 0), B(x, 0), Q, R).$$

  We use the cost matrix to evaluate the distance between vertices in the tree and the sample point:

  $$x_{\text{nearest}} = \arg\min_{v \in V} (v - x)^T S (v - x).$$

- *Near Vertices*: This function returns the set of vertices that are within a certain distance of a state, x, calculated by evaluating the LQR cost function about that state:

  `LQRNear`(V, x)

  $$= \left\{ v \in V : (v - x)^T S (v - x) \leq \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d}} \right\}$$

where n is the number of vertices in the tree, d is the dimension of the space, and γ is a constant.

- *Steering*: Given two states, x, x', the $\texttt{LQRSteer}(x, x')$ function "connects" state x with x' using the local LQR policy calculated by linearizing about x.

Algorithm 4: $\texttt{LQR} - \texttt{RRT}^*((V, E), N)$

for $i = 1, \ldots, N$ **do**
  $x_{\text{rand}} \leftarrow \texttt{Sample}$;
  $x_{\text{nearest}} \leftarrow \texttt{LQRNearest}(V, x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{LQRSteer}(x_{\text{nearest}}, x_{\text{rand}})$;
  $X_{\text{near}} \leftarrow \texttt{LQRNear}(V, x_{\text{new}})$;
  $(x_{\min}, \sigma_{\min}) \leftarrow \texttt{ChooseParent}(X_{\text{near}}, x_{\text{new}})$;
  if $\texttt{CollisionFree}(\sigma)$ **then**
    $X \leftarrow X \cup \{x_{\text{new}}\}$;
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    $(V, E) \leftarrow \texttt{Rewire}((V, E), X_{\text{near}}, x_{\text{new}})$;
return $G = (V, E)$;

Algorithm 5: $\texttt{ChooseParent}(X_{\text{near}}, x_{\text{new}})$

$\text{minCost} \leftarrow \infty$; $x_{\min} \leftarrow \texttt{NULL}$; $\sigma_{\min} \leftarrow \texttt{NULL}$;
for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $\sigma \leftarrow \texttt{LQRSteer}(x_{\text{near}}, x_{\text{new}})$;
  if $\texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma) < \text{minCost}$ **then**
    $\text{minCost} \leftarrow \texttt{Cost}(x_{\text{near}}) + \texttt{Cost}(\sigma)$;
    $x_{\min} \leftarrow x_{\text{near}}$; $\sigma_{\min} \leftarrow \sigma$;
return $(x_{\min}, \sigma_{\min})$;

Algorithm 6: $\texttt{Rewire}((V, E), X_{\text{near}}, x_{\text{new}})$

for $x_{\text{near}} \in X_{\text{near}}$ **do**
  $\sigma \leftarrow \texttt{LQRSteer}(x_{\text{new}}, x_{\text{near}})$;
  if $\texttt{Cost}(x_{\text{new}}) + \texttt{Cost}(\sigma) < \texttt{Cost}(x_{\text{near}})$ **then**
    if $\texttt{CollisionFree}(\sigma)$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow E \setminus \{x_{\text{parent}}, x_{\text{near}}\}$;
      $E \leftarrow E \cup \{x_{\text{new}}, x_{\text{near}}\}$;
return (V, E);

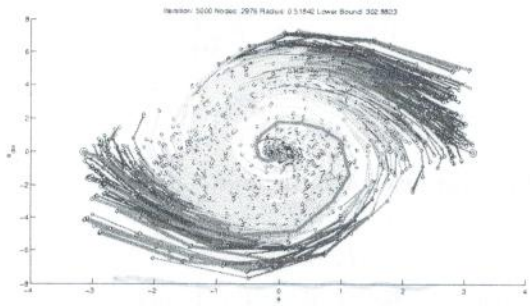
Although not explicitly noted in the algorithm descriptions, we also employ a branch-and-bound technique. In Karaman et al. [9], branch-and-bound is implemented to prune parts of the tree that are evaluated with a cost-to-go function and found to result in a higher cost than that of current solutions available in the tree. In this paper, a cost-to-go function is not implemented. Instead, the cost of the current best solution in the tree is used as a lower bound, and all nodes with a total cost above this value are removed. The branch and bound method concentrates the search on finding a trajectory that reaches the goal optimally.

IV. Results

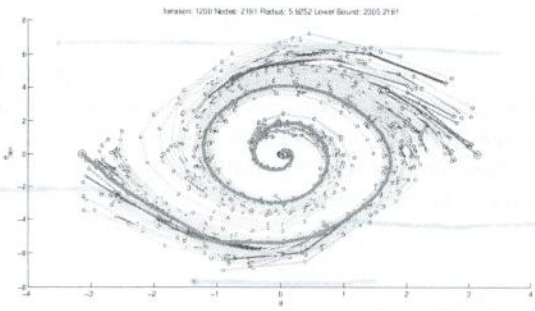
In this section we evaluate LQR-RRT* in three domains with complex dynamics: one which is torque-limited, one which is underactuated, and one which requires planning in belief-space.

A. Torque-limited Pendulum

Our first domain is a torque-limited pendulum. The objective is to swing up the pendulum from rest to a vertically balanced position. This domain has a two-dimensional state space, $(\theta, \dot{\theta})$, where θ denotes the orientation of the pendulum. The equation of motion for this system is: $\ddot{\theta} = u - b\dot{\theta} - g\cos(\theta)$, where gravity is $g = 9.81$, the constant of viscous friction is $b = 0.1$, torque is constrained to $u \in [-3, 3]$, and we have assumed unit mass and unit length. Since this problem is low dimensional, we are able to compare the solution found using our approach with the solution found using dynamic programming (which is optimal for a given discretization) using a state-space grid of 100×100 and 10 discrete actions.



(a) $R = 1$

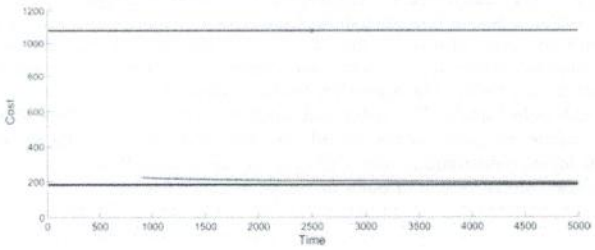


(b) $R = 50$

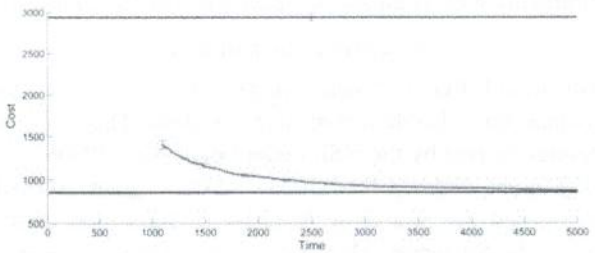
Fig. 2. The search tree and solution trajectories (thick red lines) found by our approach after 5000 iterations. (a) shows the solution found by the algorithm when torque is cheap. (b) shows the solution found when the action costs are high. Policies in the tree are false-colored (yellow through magenta) as cost increases.

We tested our algorithm in two scenarios where different cost functions were used in LQRNearest, Near, and LQRSteer. In the first, we set Q equal to identity and $R = 1$. In the second, we set Q equal to identity and $R = 50$. Figure 2 illustrates the results. In Figure 2(a), we see that the solution trajectory found by LQR-RRT* when action cost is small involves only three swings to reach the upright position. Figure 2(b) shows that the resulting trajectory for a higher action cost involves many more swings. The algorithm finds a trajectory that takes longer to reach the upright position, but expends less action cost in the process. All

solutions were simulated and we observed that the goal was reached without violating the torque constraint during all runs. Figure 3 shows the solution cost over time for both problem instances. We also show the average solution cost found by RRT (which is cost-insensitive) using the same extension heuristics and the solution cost found by dynamic programming. These graphs show that the solution cost found by our approach decreases rapidly over time, improving significantly on the solution cost found by the RRT and converging to the approximately optimal solution cost as obtained by dynamic programming.



(a) Torque-limited Pendulum $R = 1$



(b) Torque-limited Pendulum $R = 50$

Fig. 3. Solution cost over time for LQR-RRT (blue) and LQR-RRT* (red) for the torque-limited simple pendulum with $R = 1$ (a) and $R = 50$ (b). Averages are over 100 runs, and error bars are standard error. An approximately optimal solution cost, obtained by dynamic programming, is shown in black. Average times to initial solutions for LQR-RRT and our approach were 4.8 and 22.2 seconds, respectively.

Although pendulum is a low-dimensional domain, solving it using RRT* requires good extension heuristics: our attempt to solve it using extensions based on Euclidean distance failed, running out of memory before a solution was found.

B. Acrobot

In the second domain, the planner must bring an acrobot (a double inverted pendulum actuated only at the middle joint) from its initial rest configuration to a balanced vertical pose. The two joints are denoted q_1 and q_2. The state space is four-dimensional: $(q_1, q_2, \dot{q}_1, \dot{q}_2)$. Since the first joint is completely un-actuated, the acrobot is a mechanically underactuated system. The equations of motion for the acrobot are described in detail in Murray and Hauser [10].

Figure 4 shows the solution found using our approach, and Figure 5 shows the solution cost over time. Again, average cost rapidly improves, starting from a cost similar to that of RRT (with the same extension heuristic) and rapidly improving. The Acrobot is a difficult domain for which to design extension heuristics because it is dynamic and

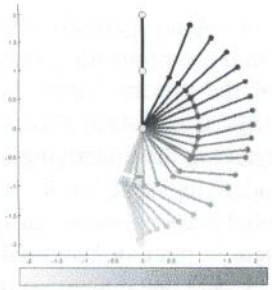


Fig. 4. A solution found after 5000 iterations of LQR-RRT* on the acrobot. Earlier state configurations are shown in lighter gray. Torque is presented as a false-colored arc where red represents higher values.

underactuated. Attempts to use standard Euclidean distance-based heuristics ran out of memory before a solution could be found.

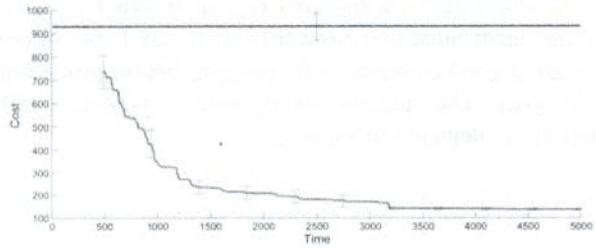


Fig. 5. Acrobot solution cost over time for LQR-RRT (blue) and our approach (red). Error bars indicated standard error over an average of 50 runs. Average times to initial solutions for LQR-RRT and our approach were 109 and 136 seconds respectively.

C. Light-Dark Domain

The Light-Dark domain (adapted from Platt *et al.* [11]) is a partially observable problem where the agent must move into a goal region with high confidence. Initially, the agent is uncertain of its true position. On each time step, the agent makes noisy state measurements. The covariance of these measurements is a quadratic function of the minimum distance to each of two light sources located at $(5, 2)$ and $(5, -2)$ (see Figure 6): $w(\mathbf{x}) = \min((\mathbf{x} - \mathbf{b}_1)^2, (\mathbf{x} - \mathbf{b}_2)^2)$, where $\mathbf{b}_1$ and $\mathbf{b}_2$ are the locations of the first and second light sources, respectively.

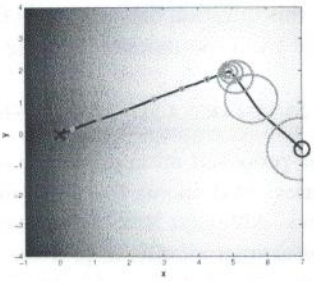


Fig. 6. The Light-Dark domain, where the noise in the agent's location sensing depends upon the amount of light present at its location. Here the agent moves from its start location (marked by a black circle) to its goal (a black X), first passing through a well-lit area to reduce its localization uncertainty (variance shown using gray circles).

A key concept in solving partially observable problems is that of *belief state*, a probability distribution over the underlying state of the system. Since the agent is unable to sense state directly, it is instead necessary to plan in the space of beliefs regarding the underlying state of the system rather than the underlying state itself. In this example, it is assumed that belief state is always an isotropic Gaussian such that belief state is three-dimensional: two dimensions describe the mean, $\mathbf{x}$, and one dimension describes variance, v. The dynamics of the belief system are derived from the Kalman filter equations and describe how belief state is expected to change as the system gains more information: $\dot{\mathbf{s}} = (u_1, u_2, \dot{v})^T$, where $\dot{v} = -v^2/(v + w(\mathbf{x}))$.

The goal of planning is to move from an initial high-variance belief state to a low-variance belief state where the mean of the distribution is at the goal. This objective corresponds to a situation where the agent is highly confident that the true state is in the goal region. In order to achieve this, the agent must move toward one of the lights in order to obtain a good estimate of its position before proceedings to the goal. The domain, along with a sample solution trajectory, is depicted in Figure 6.

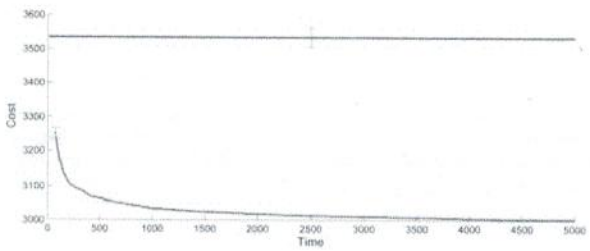


Fig. 7. Light-Dark solution cost over time for LQR-RRT (blue) and our approach (red). Error bars indicated standard error over an average of 100 runs. Average times to initial solutions for LQR-RRT and our approach were 0.2 and 2.1 seconds respectively. All algorithms were implemented in MATLAB and tested on a Intel Core 2 Duo (2.26 GHz) computer.

Figure 7 shows the solution cost for the Light-Dark domain over time. As in the other examples, our approach finds a solution quickly and rapidly improves. Figure 8 shows an example tree. Notice that it would be difficult to create heuristics by hand that would be effective in this domain: it is unclear how much weight should be attached to differences in variance against differences in real-world coordinates. Our approach avoids this question because it obtains its extension heuristics directly from the dynamics of the problem.

V. Summary and Conclusion

This paper has proposed an approach to using LQR with RRT* in the context of domains that have underactuated, or complex dynamics. Although RRT* is guaranteed to find the optimal solutions asymptotically, in practice its performance is crucially dependent on the careful design of appropriate extension heuristics. By contrast, LQR-RRT* solved all three problems *using the same automatically derived extension heuristics*—without the need for significant design effort. The use of such automatically derived heuristics eliminates a major obstacle to the application of sampling-based planners to problems with complex or underactuated dynamics.

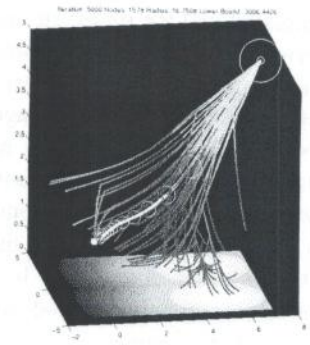


Fig. 8. The search tree for Light-Dark after 5000 iterations of planning. The belief space is represented in 3-dimensions, where the mean of the agent's location estimate is the "floor", and the vertical axis represents the variance of the agent's belief distribution; lower points represent less location uncertainty. The algorithm builds a search tree to find a trajectory through belief-space. The optimized solution is shown as a thick yellow line, where the agent moves towards the goal while lowering the variance of its location distribution (descending the vertical axis). Policies in the tree are false-colored (green through magenta) as cost increases.

Acknowledgements

We would like to thank Gustavo Goretkin and Elena Glassman for valuable technical discussions. This work was supported in part by the NSF under Grant No. 019868, ONR MURI grant N00014-09-1-1051, AFOSR grant AOARD-104135, and the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

References

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research (to appear)*, June 2011.

[2] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, pp. 298–351, 1983.

[3] S. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[5] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. M. Lavalle, "From dynamic programming to RRTs: Algorithmic design of feasible trajectories," in *Control Problems in Robotics*. Springer-Verlag, 2002.

[7] P. Cheng and S. M. Lavalle, "Reducing metric sensitivity in randomized trajectory design," in *In IEEE International Conference on Intelligent Robots and Systems*, 2001, pp. 43–48.

[8] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2010.

[9] S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *IEEE International Conference on Robotics and Automation*, 2011.

[10] R. Murray and J. Hauser, "A case study in approximate linearization: The acrobot example," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M91/46, 1991.

[11] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," in *Proceedings of Robotics: Science and Systems*, June 2010.