Optimal Sampling-Based Planning for Linear-Quadratic Kinodynamic Systems

Gustavo Goretkin[1], Alejandro Perez[1], Robert Platt Jr.[2], and George Konidaris[1]

***Abstract*—We propose a new method for applying RRT* to kinodynamic motion planning problems by using finite-horizon linear quadratic regulation (LQR) to measure cost and to extend the tree. First, we introduce the method in the context of arbitrary affine dynamical systems with quadratic costs. For these systems, the algorithm is shown to converge to optimal solutions almost surely. Second, we extend the algorithm to non-linear systems with non-quadratic costs, and demonstrate its performance experimentally.**

I. Introduction

The RRT* algorithm [1] offers a practical and effective method for probabilistically-complete optimal motion planning in many domains. Like all members of the RRT family of planning algorithms [2], it incrementally builds a tree by determining the "closest" vertices in the tree to a random sample, "steering" the system from a nearby vertex toward the sample, and adding this extension to the tree. While developing methods for locating nearby vertices and for steering the system toward new vertices is straightforward for kinematic systems, it is much more challenging for kinodynamic systems (with kinematic and differential constraints) because of the difficulties in calculating locally optimal kinodynamic trajectories. Identifying an appropriate distance metric and extension method are the key elements required to apply RRT* to kinodynamic systems. Moreover, it has been shown that the performance of RRT-based algorithms is sensitive to the distance metric [3] and that the state space is efficiently explored only when this metric reflects the true cost-to-go [4].

To illustrate the class of problems we would like to solve, consider a torque-driven pendulum that starts at rest. We would like to find a control trajectory (torque input) that swings the pendulum up to the upright position, subject to constraints on the control input and on the time. For example, we might require that the pendulum be upright at exactly 10 seconds. We also wish to minimize some cost, such as control effort. In this case it would not be optimal to expend effort getting the pendulum upright earlier than it needs to.

[1]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology {goretkin, aperez, gdk}@csail.mit.edu

[2]Computer Science and Engineering University at Buffalo, The State University of New York robplatt@buffalo.edu

This work was supported in part by the NSF under Grants No. 1117325. Perez was supported by NSF Graduate Fellowship Grant No. 1122374. Goretkin was supported by the MIT EECS - Draper Research and Innovation Scholarship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR MURI grant N00014-09-1-1051, from AFOSR grant FA2386-10-1-4135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

While it is possible to design controllers that work on this specific domain, we search for a general method that is agnostic to the details of the dynamics of the system, the cost metric, and the constraints.

For systems with linear dynamics and quadratic costs, optimal control theory offers methods for computing optimal feedback policies and the corresponding cost-to-go functions using linear quadratic regulation (LQR). However, these methods alone do not provide a way to handle constraints on the control input or in the state. This suggests that we can use LQR as a cost metric and extension method for RRT* to handle linear-quadratic systems with state and control constraints. Additionally, it suggests that we can approximate the cost metric and the extension method for non-linear systems by linearizing system dynamics in a local region of state space and applying LQR.

Recently, several researchers have applied ideas from linear control theory to estimate distance and to calculate locally optimal trajectories (as originally suggested by LaValle and Kuffner [2]). Glassman and Tedrake applied the idea to the standard RRT formulation in order to plan in underactuated domains using an affine version of LQR to estimate kinodynamic 'distances' between a random sample and vertices in the tree [5]. Our prior work extended this idea to the RRT* setting using an infinite-horizon LQR controller that both estimated the cost and calculated trajectories for extending the tree [6]. Due to its use of an infinite-horizon LQR controller, this steering method does not produce locally optimal trajectories for finite-time extensions, even for linear systems. Webb and van den Berg [7] used a finite-horizon optimal controller to calculate the extension trajectories by first computing the optimal finite time-horizon. In conjunction with RRT*, this extension method was proven to converge to optimal solutions for linear systems. However, this approach is only applicable to a limited class of cost functions. Specifically, it supports only those cost functions that trade off between time and control effort and it is not possible to constrain the time of solution trajectories. Therefore it is not suitable for problems that require a fixed trajectory length. Moreover, the extension of this method to non-linear dynamics results in trajectories that do not obey the differential constraints imposed by the dynamics.

We propose a new approach to applying LQR to the problem of finding optimal finite-horizon extension trajectories and costs in the context of RRT*. This new algorithm converges, with probability one, to the optimal plan for problems with affine dynamics and quadratic cost functions. Unlike the method of Webb and van den Berg [7], our method does

not necessitate computing the optimal time horizon for each extension. Instead, we include time as an additional dimension of the space in which the tree grows, an approach commonly used to solve problems in time-varying environments [8], [9].

Because the search tree explicitly represents state-time and explores all possible trajectories in this space, we can set contraints on the length of time of the solutions. This makes the algorithm applicable to a wider range of problems. In particular, we show that for any linear dynamical system with a quadratic cost function, our algorithm satisfies Karaman and Frazzoli's conditions [10] for guaranteeing the probabilistic optimality of the resulting trajectory. Moreover, the algorithm can be directly extended to non-linear systems by linearizing the dynamics at vertices in the tree. These approximate dynamics are, in general, affine, i.e., containing a zero-order term. LQR is typically applied to linear systems, so we also include an extension to LQR which can be applied to affine systems. We present several experiments that suggest that our algorithm obtains good results in these settings.

II. Background

A. Problem Statement

Given a deterministic system with known process dynamics, the optimal kinodynamic motion planning problem is to find a trajectory that begins in a given start state, terminates in a goal region, avoids obstacles, and satisfies differential dynamics constraints. Let the state space and control input space of the system be described by compact sets, $X \subseteq \mathbb{R}^n$ and $U \subseteq \mathbb{R}^m$, respectively. Assume that a start state, $z_0 \in X$, a goal region, $X_{goal} \subset X$, and an obstacle space, $X_{obs} \subset X$, are given. The system process dynamics are described by a continuously differentiable function,

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

where $x(t) \in X$ and $u(t) \in U$. The optimal kinodynamic motion planning problem is as follows.

Problem 1: (Optimal kinodynamic motion planning [10])

Given a state space, $X \subseteq \mathbb{R}^n$, a start state, $z_0 \in X$, an obstacle region, $X_{obs} \subset X$, a goal region, $X_{goal} \subset X$, a control input space, $U \subseteq \mathbb{R}^m$, and function $f : X \times U \to X$ describing the system process dynamics, find a control, $u : [0, T] \to U$, for some $T \in \mathcal{T}_{goal} \subseteq \mathbf{R}_{>0}$ such that the corresponding dynamically feasible trajectory, $x : [0, T] \to X \setminus X_{obs}$, starts at $x(0) = z_0$, avoids the obstacle region, reaches the goal region, $x(T) \in X_{goal}$, and minimizes the cost functional

$$J(x, u) = \int_{t=0}^{t=T} g(x(t), u(t)). \tag{2}$$

B. RRT*

The Optimal Rapidly-Exploring Random Tree (RRT*) [1] is a version of the RRT algorithm [2] that has the *asymptotic optimality* property, i.e., almost-sure convergence to an optimal solution.

The RRT* procedure is presented in Algorithm 1. The basic algorithmic primitives for operation are the following:

- *Random sampling*: The `Sample` procedure provides independent uniformly distributed random samples of states from the configuration space.
- *Nearest nodes*: Given a set V of vertices in the tree and a state z', the $\texttt{Nearest}(V, z')$ procedure calculates the state $z \in V$ from which z' can be reached with the lowest cost.
- *Near nodes*: Given a set V of vertices in the tree and a state z', the $\texttt{Near}(V, z')$ procedure provides a set of states in V from which z' may be reached with a less-than-threshold cost:

$$\texttt{Near}(V, z') = \left\{ z \in V : J(x^*_{z,z'}, u^*_{z,z'}) \leq \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d}} \right\},$$

  where $x^*_{z,z'}$ and $u^*_{z,z'}$ denote the locally optimal trajectory (ignoring obstacles) and control between z and z', 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 z, z', the $\texttt{Steer}(z, z')$ procedure returns a locally optimal path $x^*_{z,z'}$ that connects z and z'.
- *Collision checking*: Given a trajectory x, $\texttt{CollisionFree}(x)$ returns true if the path lies in the obstacle-free portion of configuration space.

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

```
1  for i = 1, ..., N do
2      z_rand ← Sample;
3      z_nearest ← Nearest(V, z_rand);
4      x_new ← Steer(z_nearest, z_rand); z_new ← x_new.end();
5      if CollisionFree(x_new) then
6          X_near ← Near(V, z_new);
7          c_min ← ∞; z_min ← NULL; x_min ← NULL;
8          for z_near ∈ X_near do
9              x ← Steer(z_near, z_new);
10             if Cost(z_near) + Cost(x) < c_min then
11                 c_min ← Cost(z_near) + Cost(x);
12                 z_min ← z_near; x_min ← x;
13         V ← V ∪ {z_new};
14         E ← E ∪ {(z_min, z_new)};
15         (V, E) ← Rewire((V, E), X_near, z_new);
16 return G = (V, E);
```

The algorithm works as follows. The graph G is initialized with $V = \{z_0\}$ and $E = \emptyset$. First, a state is sampled, denoted as z_{rand}, from the configuration space (Line 2), then, the nearest vertex is extended toward this sample (Line 3). The resulting trajectory is denoted as x_{new} and its final state as z_{new} (Line 4). If no collision is found in this trajectory, the `Near` function is invoked to calculate the set of vertices close to z_{new} (Line 6). For each near vertex, the algorithm adds the previously

Algorithm 2: `Rewire`$(\,(V,E), X_{\text{near}}, z_{\text{new}}\,)$

```
for z_near ∈ X_near do
    x ← Steer(z_new, z_near) ;
    if Cost(z_new) + Cost(x) < Cost(z_near) then
        if CollisionFree(x) then
            z_parent ← Parent(z_near);
            E ← E \ {z_parent, z_near};
            E ← E ∪ {z_new, z_near};
return (V, E);
```

stored cost of reaching that vertex to the cost of reaching z_{new} from that vertex. Then, $z_{\min}$, the vertex in the set X_{near} that reaches z_{new} with minimum cost, is returned along with the path (Lines 7-12). The algorithm then connects $z_{\min}$ to z_{new}, and attempts to "rewire" the vertices in X_{near} using the `Rewire` procedure (Algorithm 2). The `Rewire` procedure attempts to connect z_{new} to each vertex in X_{near}. The z_{new} vertex is made the parent of a vertex in X_{near} if the trajectory connecting z_{new} to the vertex does so by incurring less cost than that of its current parent.

C. Affine LQR

Linear quadratic regulation (LQR) is an optimal control technique that efficiently calculates an optimal feedback policy for linear systems with quadratic cost functions. This paper uses finite-horizon LQR, which solves the following problem: given deterministic linear process dynamics,

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{3}$$

find a state and control trajectory, x and u, that minimize the cost functional

$$\begin{aligned} J_{LQR}(x,u) =& x(T)^T Q_F x(T) \\ &+ \int_{t=0}^{t=T} x(t)^T Q x(t) + u(t)^T R u(t), \end{aligned} \tag{4}$$

where we have initial and final value constraints on the state trajectory, $x(0) = z_0$. LQR obtains the optimal solution to the above problem in two steps. First, it calculates the optimal cost-to-go function by integrating the differential Riccati equation (Equation 5) backward in time starting at the final time T with $P(T) = Q_F$ and integrating toward 0

$$-\dot{P}(t) = A^T P(t) + P(t)A - P(t)BR^{-1}B^T P(t) + Q \tag{5}$$

Second, LQR calculates the optimal action (the one that descends the cost-to-go) to take at time $t \leq T$ using:

$$u(t) = -R^{-1}B^T P(t)x(t).$$

The standard LQR formulation requires process dynamics to be linear and imposes a quadratic penalty in x and u centered at the origin. However, our application requires LQR to make use of three coordinate frames: one for the final-state constraint which ensures that the `Steer` function actually reaches z' (represented by Q_F), one for the dynamics, and another for the cost function. Additionally, our non-linear extension requires LQR to handle affine process dynamics. [1] Specifically, suppose that the process dynamics are now

$$\dot{x}(t) = Ax(t) + Bu(t) + c, \tag{6}$$

and instead of minimizing Equation 4, suppose that our goal is now to minimize

$$\begin{aligned} J_{\mathit{aff}}(x,u) =& (x(T) - z^*)^T Q_F (x(T) - z^*) \\ &+ \int_{t=0}^{t=T} (x(t))^T Q (x(t)) \\ &+ 2q^T(x(t)) + u(t)^T R u(t) + d, \end{aligned} \tag{7}$$

where q is the first-order term in the cost function and d is the constant term (which may be used to penalize time). This can be solved by introducing a change of coordinates in the state space. We augment the state vector,

$$\bar{x}(t) = [x^T, 1]^T. \tag{8}$$

Changing to homogenous coordinates allows linear transformations in the augmented space to behave as affine transformations in the original space. The new process dynamics are:

$$\dot{\bar{x}}(t) = \begin{pmatrix} A & c \\ 0 & 0 \end{pmatrix} \bar{x}(t) + \begin{pmatrix} B \\ 0 \end{pmatrix} u(t). \tag{9}$$

It may be verified that these process dynamics are the same as those in Equation 6 but in the form of Equation 3. The bottom rows of the matrices in Equation 9 preserves the augmented value in Equation 8. We can now express the offset cost functional in Equation 7 as:

$$J(\bar{x},u) = \bar{x}\,(t)^T\, \bar{Q}_F \bar{x}\,(t) + \int_{t=0}^{t=T} \bar{x}(t)^T \bar{Q} \bar{x}(t) + u(t)^T R u(t), \tag{10}$$

where $\bar{Q}_F = \begin{pmatrix} Q_F & -q_F \\ -{q_F}^T & (z^*)^T z^* \end{pmatrix}$, and $\bar{Q} = \begin{pmatrix} Q & q \\ q^T & d \end{pmatrix}$

and we solve the affine version of the problem by applying LQR to the augmented system in the standard way. In the rest of this paper, we will refer to this process of solving systems with affine dynamics and second-order costs as LQR.

III. Kinodynamic Planning for Systems with Affine Dynamics and Quadratic Costs

In this section, we propose a version of RRT* that finds optimal motion plans for affine kinodynamic systems with quadratic costs. The algorithm works by constructing a tree where each vertex has an associated state and time. Our method can handle both finite and infinite horizon problems.

[1] While some systems are actually affine, our primary goal for handling affine dynamics is to extend to nonlinear systems.

A. The Kinodynamic Planning Problem with Affine Dynamics and Quadratic Costs

Consider the version of Problem 1 where the system process dynamics are affine (of the form in Equation 6) and where the cost functional is quadratic. The obstacle region, X_{obs}, may be of any shape. We consider two instances of the problem: a) where the time horizon, T, is fixed and given as input, and b) where the time horizon is free to vary in order to minimize cost (*i.e.,* T is an optimization variable).

B. Extensions to the Tree

Rather than sampling points in state space (Step 2 of Algorithm 1), our algorithm samples points from state-time, $v_{rand} = (z_{rand}, t_{rand}) \sim X \times (0, T]$, where T is a maximum final time constraint.

Steps 3 and 4 are implemented using LQR. We integrate the differential Riccati equation for the affine dynamics backward in time to 0 starting from t_{rand} with the quadratic cost function centered at the sample point, $z^* = z_{rand}$ (see Section II-C). The LQR cost function used in the Riccati integration (Equation 10) has Q and R equal to their values in the global quadratic cost function. Q_F is set to a multiple of the identity matrix that is sufficiently large to cause LQR to find trajectories that terminate very close to the random sample.[2] The result of this integration will be written as a function, $P : [0, T] \rightarrow S_+^{n+1}$ (the result of the Riccati integration is always a positive semi-definite matrix). The P function is used to implement Step 3 of Algorithm 1 by evaluating the LQR cost-to-go function for each vertex, $v_i = (x_i, t_i) \in V$, in the tree. Without loss of generality, we assume the time associated with the root vertex to be 0 (non-zero start times can be handled by shifting all times forward). As a result, there is always at least one vertex with a time smaller than the sample time. Once the vertex 'closest' to the sample is found, we generate a trajectory between (z_i, t_i) and (z_{rand}, t_{rand}) (Step 4 of Algorithm 1) using LQR. We then verify if the trajectory intersects an obstacle region and discard it if it does.

C. Tree Rewiring

In the context of a kinodynamic problem, RRT* has two types of extension procedures [10]. In the first (Steps 6 through 14 of Algorithm 1), we identify the set of vertices, $v_i \in V_{near} \subseteq V$, from which z_{new} can be reached such that

$$J(x^*_{v_i, v_{new}}, u^*_{v_i, v_{new}}) \leq \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d}}. \qquad (11)$$

For each vertex in this set, we calculate the global costs by tracing the tree back to the root and summing the costs. After ranking these vertices in order global costs, we start at the top of this list and attempt to connect the vertex to the random sample using LQR (Step 9 of Algorithm 1). If the resulting trajectory intersects an obstacle, that connection is discarded and the algorithm moves down the list to the vertex associated with the next smallest cost. Once a successful trajectory is found, the resulting branch is added to the tree.

Similarly, RRT* searches for vertices that can be reached from the sample and re-wires as necessary (Algorithm 2). Costs are calculated in the same way as above. This time, it is necessary to integrate the differential Riccati equation backward from each vertex in the tree. Instead of requiring this integration to occur for the entire tree on each re-wire step, we store the backward Riccati integration for each vertex in the tree when it is originally added and access the appropriate P function as necessary. Only vertices within the cost threshold of Equation 11 are evaluated.

D. Time Horizon

A key advantage of our algorithm relative to [7] is that we are able to find optimal trajectories for problems with specific constraints on the time horizon. Since we construct the tree in state-time, this amounts to extending the goal region, X_{goal}, to include time. The temporal goal region, $\mathcal{T}_{goal} \subseteq \mathbf{R}_{>0}$, is defined to be the set of times when temporal goal constraints are satisfied. If the problem requires X_{goal} to be reached at a particular time instant, then $\mathcal{T}_{goal}$ is a single instant in time. If the problem requires X_{goal} to be reached between two times, then $\mathcal{T}_{goal}$ is a range. If there is no constraint on final time, then, in principle, $\mathcal{T}_{goal}$ is equal to the positive real numbers. However, rather than sampling from the entire positive number line in this case, we set $\mathcal{T}_{goal}$ equal to a range between zero and some maximum value that we are sure is greater than the optimal time horizon.

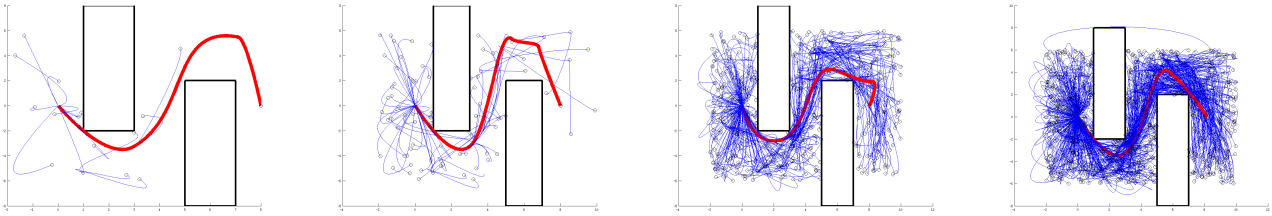
E. Optimality

Karaman and Frazzoli [10] proved that the optimality guarantee for RRT* in kinodynamic systems holds under two conditions.

First, the trajectory found by the `Steer` procedure must be optimal in the absence of obstacles [10]. Similarly, the cost estimate used by `Nearest` and `Near` must reflect the optimal cost. For affine systems, both of these requirements are trivially satisfied by our algorithm because LQR finds optimal policies and costs in this case.

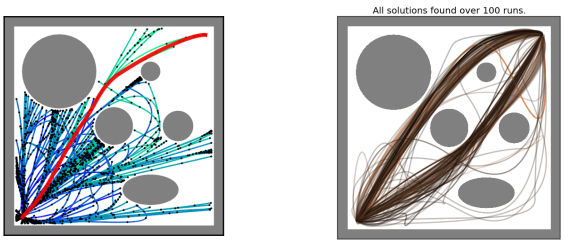
The second requirement is for the domain to satisfy the Weakened Local Controllability criterion (Assumption 3 in Karaman and Frazzoli [10]) and the ϵ-Collision-Free Approximate Trajectories criterion (Assumption 4 in Karaman and Frazzoli [10]). For state that has not been augmented by the time dimension, these conditions are trivially satisfied by any controllable affine system. However, notice that they are no longer technically satisfied when state is augmented by time because it is impossible for the system to go backward in time to reach any state-time in a ball around a given state-time. Nevertheless, it is safe to amend these Assumptions to only require the system to reach future state-times because of the way the assumptions are used in the RRT* proof of optimality. Essentially, these assumptions are needed to ensure that there exists a neighborhood of approximately optimal trajectories

[2] In principle, it is possible to integrate an inverse form of the differential Riccati equation that allows setting Q_F to infinity (*i.e.* $Q_F^{-1} = 0$). However, we have found that setting Q_F to a large value works well in practice.



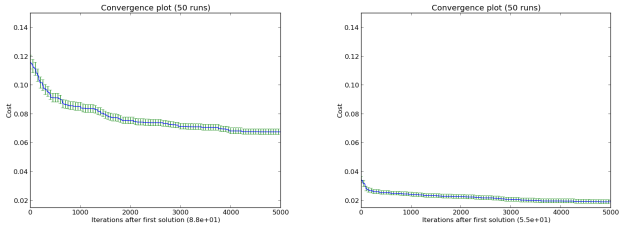
(a) (b) (c) (d)

Fig. 1. Example of using our algorithm to find an optimal solution to the kinodynamic motion planning for the double integrator in the presence of obstacles. The four images show the tree at different points during growing. The blue lines show the tree. The red path denotes the current best path to the goal. The tree in (a) contains 21 vertices; in (b) contains 85 vertices; in (c) contains 485 vertices; and in (d) contains 1049 vertices. The costs of the four best trajectories, from left to right, are 209, 17, 16, and 9.



(a) (b)

Fig. 2. Solution tree generated by our algorithm while solving the double integrator problem. (a) The tree is grown in the domain. The system must move from a stationary position in the lower left to a stationary position in the upper right. The five ellipses denote obstacles, and the tree is color-coded for cost. The thick red line shows the current best trajectory. (b) All candidate solution trajectories found during 100 separate runs of the algorithm, again color coded for cost.



(a) (b)

Fig. 3. The average cost of the best solution found over time, averaged over 50 runs of the algorithm for 200 time steps (left) and 400 time steps (right). Error bars are standard error.

with non-zero measure and to ensure that `Steer` will be able to make local connections with a trajectory that does not leave an obstacle-free neighborhood. Since these requirements are satisfied without requiring the system to go backward in time, we are safe to modify the Assumptions in this way.

F. Experiments

We evaluated our approach to affine systems using a two-dimensional double integrator (four dimensional state space). This system is a unit mass with a damping factor of 0.1. The planning problem is to reach a goal state at $(8, 0)$ with zero velocity 15 seconds after starting at $(0, 0)$ with zero velocity while minimizing a cost function with $Q = 0$ and $R = I$. Similar evaluations of RRT* performance for the double integrator have appeared in Karaman and Frazzoli [10] and Webb and van den Berg [7]. Figure 1 illustrates the progression of the algorithm. Initially, the algorithms finds a trajectory (Figure 1(a)) with cost 209. The algorithm proceeded to generate seven more trajectories to the goal with successively smaller costs: 145, 65, 17, 15, 11, 9. Three of these successive instances of the tree are illustrated in Figures 1(b), (c), and (d).

We also evaluated average performance over multiple runs of the algorithm (Figure 2). Figure 2(a) shows a partial tree constructed by the algorithm in a test environment with different obstacles but the same double-integrator dynamics. Figure 2(b) illustrates different candidate solutions found while running the algorithm 100 times with 5000 iterations on each run. Notice that the algorithm considers many different homotopy classes. Figure 3 illustrates algorithm performance averaged over 100 runs of the algorithm in terms of the mean and standard error of the lowest cost trajectory in the tree.

IV. Non-linear Kinodynamic Planning

This section extends our approach to a class of non-linear systems. First, we define the class of systems and extend the algorithm. Then, we describe experiments that characterize the approach.

A. The Non-linear Planning Problem

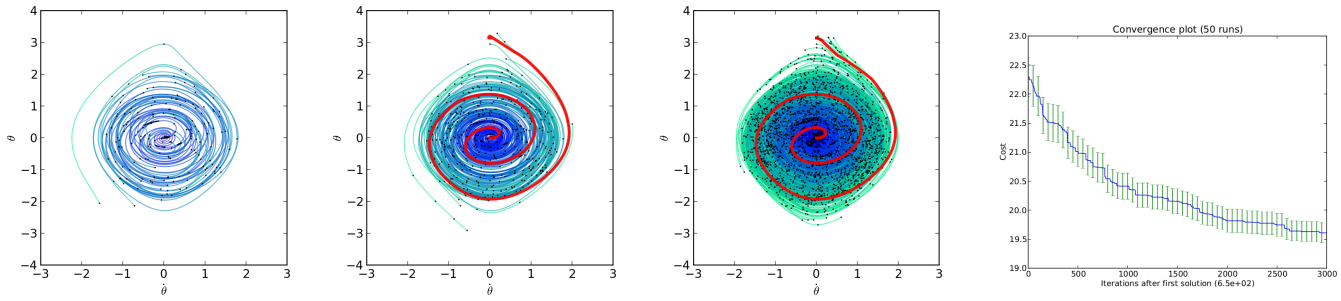
Consider the version of Problem 1 where the system process dynamics (Equation 1) and the cost functional (Equation 2) are both arbitrary functions of state. Define the process dynamics to be

$$\dot{x}(t) = \alpha(x(t)) + \beta(x(t))u(t),$$

where α is a non-linear C^1 continuous function onto state space, β is a non-linear C^1 continuous matrix-valued function. Define the cost to be

$$g(x, u) = h(x(t)) + u(t)^T \Gamma(x) u(t), \tag{12}$$

where h is a non-linear C^2 continuous functional onto the reals and $\Gamma : X \to S^n_+$ is also non-linear and C^2 continuous.



(a) (b) (c) (d)

Fig. 4. Our algorithm applied to the inverted pendulum. (a) through (c) illustrate a phase plot of the RRT tree after 500, 1000, and 1500 iterations, respectively. The red line in each plot shows the lowest cost solution in the tree (after 500 iterations, no solution has been found). Paths are colored according to cost (from dark blue to light cyan). (d) shows performance averaged over 50 runs (average in blue; standard error bars in green.)

B. Local Approximations to Dynamics and Cost

In order to fit problems with non-affine process dynamics and non-quadratic costs into our framework, we need to construct local affine approximations of the dynamics and quadratic approximations of the costs. This approach is similar to standard practice in control where a non-linear system is "linearized" about an operating point and linear control theory is applied. This linearization occurs each time our algorithm calls LQR. Let (z_{lin}, t_{lin}) denote the linearization point. In Steps 3 and 4 of Algorithm 1, the system is linearized about (z_{rand}, t_{rand}). In Steps 6 and 9, we linearize about (z_{new}, t_{new}). We approximate the process dynamics using a first-order Taylor expansion:

$$\begin{aligned}\dot{x}(t) &= \alpha(x(t)) + \beta(x(t))u(t) \\ &\approx \alpha(z_{lin}) + \nabla_z \alpha(z_{lin})^T (x(t) - z_{lin}) \\ &\quad + \beta(z_{lin})u(t) \\ &= A(z_{lin})x(t) + \beta(z_{lin})u(t) + c(z_{lin}), \end{aligned} \quad (13)$$

where

$$A(z_{lin}) = \nabla_z \alpha(z_{lin})^T$$

and

$$c(z_{lin}) = \alpha(z_{lin}) - A(z_{lin}) z_{lin}.$$

We approximate the cost function similarly, this time using a second-order expansion:

$$\begin{aligned} h(x(t)) \approx\ & h(z_{lin}) + \nabla_z h(z_{lin})^T (x(t) - z_{lin}) \\ & + (x(t) - z_{lin})^T \nabla_z^2 h(z_{lin}) (x(t) - z_{lin}) \end{aligned}$$

where $\nabla_z h(z_{lin})$ is the gradient of h and $\nabla_z^2 h(z_{lin})$ is its Hessian. We perform the equivalent approximation for $\Gamma(x)$.

The two approximations above will enable us to formulate an affine LQR problem with the dynamics of Equation 13 and the cost function that can be solved using affine LQR (outlined in Section II-C). However, in order to do this, the quadratic approximation of the cost function must be convex, *i.e.* $\nabla_z^2 h(x_{lin})$ must be positive semi-definite. Since this is not necessarily the case for arbitrary C^2 continuous functionals, h, we will project the Hessian onto the cone of positive semi-definite matrices by setting all non-positive Eigenvalues to zero (using, for example, the Eigen-decomposition of $\nabla_z^2 h(z_{lin})$).

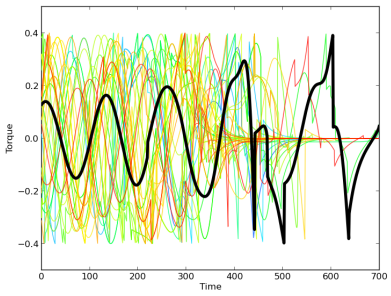


Fig. 5. Several torque trajectories, color-coded by cost, which swing up the pendulum. The torque is limited to $[-0.4, 0.4]$. The best trajectory found after 3000 iterations is shown in black. Because of the quadratic actuation cost, trajectories which bring the pendulum up earlier are more expensive.

C. Tree Extensions and Re-wiring

Given the linear and quadratic approximations described above, the algorithm for the non-linear case is very similar to that used to solve the affine problem. On each iteration of RRT*, after taking the random sample z_{rand}, the algorithm creates an affine approximation of the process dynamics and a quadratic approximation of the cost function in the neighborhood of the sample. These approximations are used to approximate the vertex in the tree from which the sample can be reached with a minimum cost and to compute the LQR policy used to perform the extension toward z_{rand}.

Although we use the approximations to find costs and to calculate trajectories, a key element of our approach is that during re-wiring, we store the true costs of each path in the tree and that each trajectory in the tree satisfies the differential constraint and is realizable with some control trajectory. In particular, we use the cost-to-go function in the implementation of `Near` and `Nearest` to identify the set of vertices for which Equation 11 holds. For each vertex within

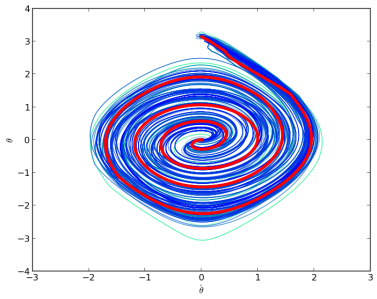


Fig. 6. Phase plot in Figure 4 with goal set at 200 more time steps. Note solutions are of a different homotopy class, i.e., different number of swings.

Fig. 7. All candidate solution trajectories found during 70 separate runs for double integrator with state-dependent actuation cost. Paths are colored-coded according to cost.

this set, we use the linearized LQR controller to calculate a trajectory toward the sample. After finding the trajectory, we integrate its true cost function over the trajectory and store this true cost. Similarly, although the algorithm uses the approximations to calculate trajectories from the sample to other vertices in the tree within the cost radius, the true cost of each of these trajectories is stored in the tree. As a result, the algorithm re-wires paths according to their true costs of executing—not the approximation.

One of the problems with the method as described above is that although we are extending from a vertex in the tree, z_{near}, we are approximating the process dynamics and cost function about z_{rand} instead of about z_{near}. This can result in poor control policies early in the system trajectory. Instead, we would like to create the dynamics and cost approximations with respect to z_{near}. However, this is challenging because it would require computing the LQR cost-to-go once per vertex per random sample. Instead, we calculate the LQR solution to the *inverse* system process dynamics rather than the forward dynamics. In particular, we set

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

and integrate the Riccati equation *forward* from each vertex in the tree. This computation can be cached because its result does not change once a vertex is added to the tree.

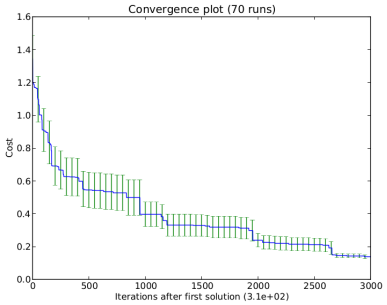


Fig. 8. Convergence plot averaged over 70 runs for the double integrator with state-dependent actuation cost.

D. Experiments

Figure 4 illustrates the application of the algorithm to a torque-limited inverted pendulum with dynamics $\ddot{\theta} = -\sin(\theta) - 0.1\dot{\theta} + u$ (unit mass, length, gravity, and a damping factor of 0.1). Torque is limited to $|u| \leq 0.4$. In our experiment, the problem is to find a trajectory that minimizes a quadratic action cost—the cost function of Equation 2 with:

$$g(x(t), u(t)) = u(t)^T u(t).$$

Since the inverted-pendulum has non-linear process dynamics, it is challenging to find near-optimal solutions. Figures 4(a) – (c) illustrate the tree-growing process. Figure 4(d) shows algorithm performance averaged over 50 runs of the algorithm in terms of average cost and standard error bars as a function of algorithm iteration number. Figure 4(d) clearly shows that, after finding an initial solution, RRT* improves trajectory cost.

Figure 4 exhibits a domain identical to the domain in Figure 2 with the difference that the cost is

$$g(x, u) = \exp(x_y/25) \cdot {u_x}^2 + {u_y}^2.$$

In other words, the penalty on thrust in the x direction becomes steeper as the mass moves up. Notice that while in Figure 2, there is a near symmetry between two of the solution classes, this domain favors trajectories which curve upward (first moving right then up). The penalty on u_x, the thrust in the x direction is not enough for the optimal trajectory to be the one that first moves completely to the right and then completely upward – this trajectory is much longer and so more thrust is needed to get to the same goal within the same time. Figure 8 shows 70 runs of RRT* converging to the same cost.

V. Discussion and Conclusions

We have introduced a new way of using LQR in conjunction with RRT* to find optimal solutions to planning problems involving kinodynamic systems. Our algorithm is provably optimal for affine kinodynamic systems with quadratic cost functions. Relative to other similar approaches [6], [7], our new method is more flexible in that it can handle a more general class of cost functions and various different final time constraints. Moreover, our method can be directly extended to

a broad class of systems with non-affine process dynamics and non-quadratic cost functions. We present experimental results that suggest that the approach works well in these cases.

REFERENCES

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, June 2011.

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

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

[4] 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.

[5] 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.

[6] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2012, pp. 2537–2542.

[7] D. J. Webb and J. V. D. Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints," arXiv:1205.5088v1, submitted.

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

[9] D. Hsu, R. Kindel, J.-C. Latombe, and S. M. Rock, "Randomized kinodynamic motion planning with moving obstacles," *I. J. Robotic Res.*, vol. 21, no. 3, pp. 233–256, 2002.

[10] Karaman and Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.