2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Kinodynamic RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics

Dustin J. Webb Jur van den Berg

Abstract— **We present Kinodynamic RRT*, an incremental sampling-based approach for asymptotically optimal motion planning for robots with linear dynamics. Our approach extends RRT*, which was introduced for holonomic robots [10], by using a fixed-final-state-free-final-time controller that optimally connects any pair of states, where the cost function is expressed as a trade-off between the duration of a trajectory and the expended control effort. Our approach generalizes earlier work on RRT* for kinodynamic systems, as it guarantees asymptotic optimality for any system with controllable linear dynamics, in state spaces of any dimension. In addition, we show that for the rich subclass of systems with a nilpotent dynamics matrix, closed-form solutions for optimal trajectories can be derived, which keeps the computational overhead of our algorithm compared to traditional RRT* at a minimum. We demonstrate the potential of our approach by computing asymptotically optimal trajectories in three challenging motion planning scenarios: (i) a planar robot with a 4-D state space and double integrator dynamics, (ii) an aerial vehicle with a 10-D state space and linearized quadrotor dynamics, and (iii) a car-like robot with a 5-D state space and non-linear dynamics.**

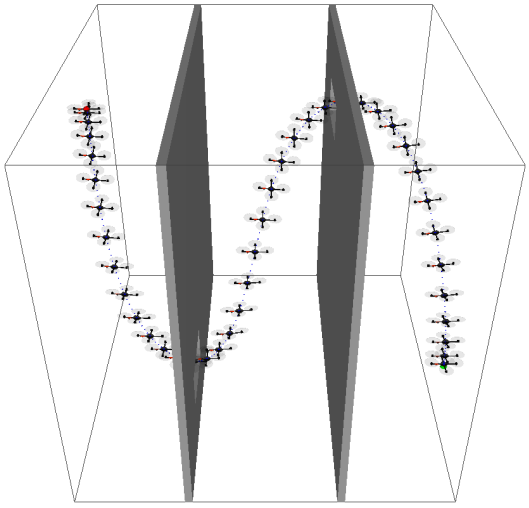

Fig. 1. An asymptotically optimal trajectory computed by our algorithm for a quadrotor helicopter with linearized dynamics in a 10-D state space.

I. Introduction

Much progress has been made in the area of motion planning in robotics over the past decades, where the basic problem is defined as finding a trajectory for a robot between a start state and a goal state without collisions with obstacles in the environment. The introduction of incremental sampling-based planners, such as probabilistic roadmaps (PRM) [11] and rapidly-exploring random trees (RRT) [13] enabled solving motion planning problems in high-dimensional state spaces in reasonable computation time, even though the problem is known to be PSPACE-hard [12]. PRM and RRT are probabilistically complete, which means that a solution will be found (if one exists) with a probability approaching 1 if one lets the algorithm run long enough. More recently, an extension of RRT called RRT* [10] was developed that achieves *asymptotic optimality*, which means that an *optimal* solution will be found with a probability approaching 1.

A key limitation of RRT* is that it is applicable only to systems with simple dynamics, as it relies on the ability to connect any pair of states with an optimal trajectory (i.e. solving the *two-point boundary value problem* [14]). For holonomic robots, straight lines through the state space represent optimal trajectories, but for *kinodynamic* systems such trajectories are typically not feasible due to the system's *differential constraints*. Finding an optimal trajectory between two states for differentially constrained systems is non-trivial in general. Prior works on extending RRT* for kinodynamic systems have therefore focused on simple specific instances of kinodynamic systems [9], or have used approximate solutions to the two-point boundary value problem [5], [18]. Such approximate solutions may not reach the target state exactly (but only within a bounded neighborhood) and may not be optimal with respect to the specified cost function (but only satisfy a notion of bounded suboptimality). This requires the RRT* algorithm to repropagate part of the tree in each iteration of the algorithm, which is computationally intensive. Also, it is unclear whether the (bounded) suboptimality of connections upholds the theoretical asymptotic optimality guarantee of RRT*.

In this paper, we present Kinodynamic RRT*, an extension of RRT* for systems with *controllable linear dynamics*, that overcomes the above limitations. Our approach finds asymptotically optimal trajectories in state spaces of arbitrary dimension containing static obstacles and bounds on the control input, with respect to a cost function that is expressed as a tunable trade-off between the duration of the trajectory and the expended control effort. We use a fixed-final-state-free-final-time controller [15] in our algorithm to exactly and optimally connect any pair of states, and we show that for the rich subclass of systems with a *nilpotent* dynamics matrix, closed-form expressions for optimal trajectories between pairs of states can be derived in terms of a solution to a $2n^2$-degree polynomial equation. This means that our algorithm computes asymptotically optimal trajectories for such kinodynamic systems at little additional computational cost compared to RRT* for holonomic robots. In addition,

The authors are with the School of Computing at the University of Utah. E-mail: dustin.j.webb@utah.edu, berg@cs.utah.edu.

978-1-4673-5643-5/13/$31.00 ©2013 IEEE

it allows us to explicitly characterize the *reachable sets* of a given state, enabling efficient neighbor search.

We note that while we focus our presentation on extending RRT* to kinodynamic systems, also the application of PRM [11] and path *smoothing* by iterative shortcutting [6] have thus far been limited to holonomic systems, for these methods too require connecting pairs of states by feasible trajectories. Our approach is equally suited for making PRM and smoothing applicable to robots with dynamics, and may particularly align well with recent interest in constructing roadmaps containing near-optimal trajectories [16].

We demonstrate the potential of our approach by computing asymptotically optimal trajectories in three challenging motion planning scenarios: (i) a planar robot with a 4-D state space and double integrator dynamics, (ii) an aerial vehicle with a 10-D state space and linearized quadrotor dynamics (see Fig. 1), and (iii) a car-like robot with a 5-D state space and non-linear dynamics.

The remainder of this paper is organized as follows. We begin by discussing related work in Section II and formally defining the problem we discuss in this paper in Section III. Section IV describes how an optimal trajectory is computed between any pair of states, and Section V describes our adapted RRT* algorithm. We describe the extension to non-linear dynamics in Section VI, discuss experimental results in Section VII, and conclude in Section VIII.

II. Related Work

The term kinodynamic planning was first introduced in 1993 in [4], which presented a resolution-complete algorithm for optimal planning of robots with discretized double integrator dynamics in low-dimensional workspaces. Kinodynamic planning has since been an active area of research. Incremental sampling-based algorithms, in particular the rapidly-exploring random tree (RRT) approach [13], proved to be effective in state spaces of high dimensionality, and is applicable to general dynamics systems as it builds a random tree of trajectories, and complex dynamics can be forward integrated to expand the tree.

Unfortunately, RRT does not produce optimal trajectories. In fact, the probability that it finds an optimal path is zero [10]. Recently, RRT* was introduced to overcome this problem and guarantees asymptotic optimality [8]; it iteratively builds a tree of trajectories through the state space whose probability of containing an optimal solution approaches 1 as the number of iterations of the algorithm approaches infinity. However, to achieve asymptotic optimality, RRT* requires that pairs of states can be *exactly* and *optimally* connected, and was therefore introduced for holonomic systems only.

Several attempts have been made to extend RRT*'s applicability to kinodynamic systems for which a straight-line connection between a pair of states is typically not a valid trajectory due to the system's differential constraints. In [9], sufficient conditions were established to ensure asymptotic optimality of the RRT* algorithm for systems with differential constraints, and it was shown how to apply RRT* to two specific instances of kinodynamic systems: the Dubin's car and the double integrator. The approach of [5] generalizes this to arbitrary kinodynamic systems, but has several limitations: to connect pairs of states, it uses the shooting method [2] with a constant control input. Such connections only reach a neighborhood of the target state in general, which requires costly repropagation of the trajectories in the tree descending from such states. Also, these connections are (bounded) suboptimal with respect to the specified cost function, as optimal connections in general require time-varying control inputs. It is unclear what the theoretical impact of this is on the asymptotic optimality guarantee of RRT*. More recently, LQR-RRT* was proposed in [18], and uses an infinite-horizon LQR controller to connect pairs of states. This controller will in general not reach the target state exactly or within a guaranteed neighborhood, even in case of linear dynamics. Also, it effectively uses a different cost function for each pair of states it connects (the infinite-horizon LQR cost function is defined with respect to the target state, and each connection has a different target state), and it is unclear whether a trajectory formed by concatenating such connections can satisfy any well-defined notion of optimality.

Our approach improves upon prior work by connecting any pair of states exactly and optimally for systems with controllable linear dynamics, which guarantees that asymptotic optimality is in fact achieved. We accomplish this by extending the well-studied formulation for a fixed-final-state and *fixed*-final-time optimal control problem [15] to derive an optimal, open-loop, fixed-final-state *free*-final-time control policy. A similar approach has been adopted by [19] for extending RRTs in state space under a dynamic cost-to-go distance metric [7]. In comparison to the latter work, we present a numerical solution that is guaranteed to find a *global optimum* for the general case, and an efficient closed-form solution for the special case of systems with a nilpotent dynamics matrix.

III. Problem Definition

Let $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{U} = \mathbb{R}^m$ be the state space and control input space, respectively, of the robot, and let the dynamics of the robot be defined by the following linear system, which we require to be formally *controllable*:

$$\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}, \tag{1}$$

where $\mathbf{x}[t] \in \mathcal{X}$ is the state of the robot, $\mathbf{u}[t] \in \mathcal{U}$ is the control input of the robot, and $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, and $\mathbf{c} \in \mathbb{R}^n$ are constant and given.

A *trajectory* of the robot is defined by a tuple $\pi = (\mathbf{x}[\,], \mathbf{u}[\,], \tau)$, where τ is the arrival time or duration of the trajectory, $\mathbf{u} : [0, \tau] \to \mathcal{U}$ defines the control input along the trajectory, and $\mathbf{x} : [0, \tau] \to \mathcal{X}$ are the corresponding states along the trajectory given $\mathbf{x}[0]$ with $\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}$.

The *cost* $c[\pi]$ of a trajectory π is defined by the function:

$$c[\pi] = \int_0^\tau (1 + \mathbf{u}[t]^T R \mathbf{u}[t])\, \mathrm{d}t, \tag{2}$$

which penalizes both the duration of the trajectory and the expended control effort, where $R \in \mathbb{R}^{m \times m}$ is positive-definite, constant, and given, and weights the cost of the control inputs relative to each other and to the duration of the trajectory.

Let $\mathcal{X}_{\text{free}} \subset \mathcal{X}$ define the *free* state space of the robot, which consists of those states that are within user-defined bounds and are collision-free with respect to obstacles in the environment. Similarly, let $\mathcal{U}_{\text{free}} \subset \mathcal{U}$ define the free control input space of the robot, consisting of control inputs that are within bounds placed on them. This brings us to the formal definition of the problem we discuss in this paper: given a start state $\mathbf{x}_{\text{start}} \in \mathcal{X}_{\text{free}}$ and a goal state $\mathbf{x}_{\text{goal}} \in \mathcal{X}_{\text{free}}$, find a collision-free trajectory π^*_{free} between $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ with minimal cost:

$$\pi^*_{\text{free}} = \text{argmin}\{\pi \,|\, \mathbf{x}[0] = \mathbf{x}_{\text{start}} \wedge \mathbf{x}[\tau] = \mathbf{x}_{\text{goal}} \wedge \forall\{t \in [0,\tau]\}\,(\mathbf{x}[t] \in \mathcal{X}_{\text{free}} \wedge \mathbf{u}[t] \in \mathcal{U}_{\text{free}})\}\; c[\pi]. \quad (3)$$

We note that the cost function of Eq. (2) obeys the *optimal substructure property*; let $\pi^*[\mathbf{x}_0, \mathbf{x}_1] = (\mathbf{x}[\,], \mathbf{u}[\,], \tau)$ be the optimal trajectory between $\mathbf{x}_0 \in \mathcal{X}$ and $\mathbf{x}_1 \in \mathcal{X}$, irrespective of bounds and obstacles, and let $c^*[\mathbf{x}_0, \mathbf{x}_1]$ be its cost:

$$c^*[\mathbf{x}_0, \mathbf{x}_1] = \min\{\pi \,|\, \mathbf{x}[0] = \mathbf{x}_0 \wedge \mathbf{x}[\tau] = \mathbf{x}_1\}\; c[\pi]. \quad (4)$$

$$\pi^*[\mathbf{x}_0, \mathbf{x}_1] = \text{argmin}\{\pi \,|\, \mathbf{x}[0] = \mathbf{x}_0 \wedge \mathbf{x}[\tau] = \mathbf{x}_1\}\; c[\pi], \quad (5)$$

then for all $0 < t < \tau$ we have: $c^*[\mathbf{x}_0, \mathbf{x}_1] = c^*[\mathbf{x}_0, \mathbf{x}[t]] + c^*[\mathbf{x}[t], \mathbf{x}_1]$. Hence, an optimal collision-free trajectory π^*_{free} between start and goal consists of a concatenation of optimal trajectories between a series of successive states $(\mathbf{x}_{\text{start}}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{\text{goal}})$ in $\mathcal{X}_{\text{free}}$.

IV. Optimally Connecting a Pair of States

A critical component of our approach to solve the problem as defined in Eq. (3) is to be able to compute the optimal trajectory $\pi^*[\mathbf{x}_0, \mathbf{x}_1]$ (and its cost $c^*[\mathbf{x}_0, \mathbf{x}_1]$) between any two states $\mathbf{x}_0 \in \mathcal{X}$ and $\mathbf{x}_1 \in \mathcal{X}$, as defined in Eqs. (5) and (4). In this section we discuss how to compute these. It is known from [15] what the optimal control policy is in case a *fixed* arrival time τ is given, as we review in Section IV-A. We extend this analysis to find the optimal free arrival time in Section IV-B and show how to compute the corresponding optimal trajectory in IV-C. We discuss practical implementation in Section IV-D.

A. Optimal Control for Fixed Final State, Fixed Final Time

Given a *fixed* arrival time τ and two states $\mathbf{x}_0$ and $\mathbf{x}_1$, we want to find a trajectory $(\mathbf{x}[\,], \mathbf{u}[\,], \tau)$ such that $\mathbf{x}[0] = \mathbf{x}_0$, $\mathbf{x}[\tau] = \mathbf{x}_1$, and $\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}$ (for all $0 \le t \le \tau$), minimizing the cost function of Eq. (2). This is the so-called fixed final state, fixed final time optimal control problem.

Let $G[t]$ be the *weighted controllability Gramian* given by:

$$G[t] = \int_0^t \exp[A(t-t')]BR^{-1}B^T \exp[A^T(t-t')]\; \mathrm{d}t', \quad (6)$$

which is the solution to the Lyapunov equation:

$$\dot{G}[t] = AG[t] + G[t]A^T + BR^{-1}B^T, \quad G[0] = 0. \quad (7)$$

We note that $G[t]$ is a positive-definite matrix for $t > 0$ if the dynamics system of Eq. (1) is controllable.

Further, let $\bar{\mathbf{x}}[t]$ describe what the state $\mathbf{x}$ (starting in $\mathbf{x}_0$ at time 0) would be at time t if no control input were applied:

$$\bar{\mathbf{x}}[t] = \exp[At]\mathbf{x}_0 + \int_0^t \exp[A(t-t')]\mathbf{c}\; \mathrm{d}t', \quad (8)$$

which is the solution to the differential equation:

$$\dot{\bar{\mathbf{x}}}[t] = A\bar{\mathbf{x}}[t] + \mathbf{c}, \qquad \bar{\mathbf{x}}[0] = \mathbf{x}_0. \quad (9)$$

Then, the optimal control policy for the fixed final state, fixed final time optimal control problem is given by:

$$\mathbf{u}[t] = R^{-1}B^T \exp[A^T(\tau - t)]G[\tau]^{-1}(\mathbf{x}_1 - \bar{\mathbf{x}}[\tau]), \quad (10)$$

which is an *open-loop* control policy. We refer the reader to [15] for details on the derivation of this equation.

B. Finding the Optimal Arrival Time

To find an optimal trajectory $\pi^*[\mathbf{x}_0, \mathbf{x}_1]$ between $\mathbf{x}_0$ and $\mathbf{x}_1$ as defined by Eq. (5), we extend the above analysis to solve the fixed final state, *free* final time optimal control problem, in which we can choose the arrival time τ freely to minimize the cost function of Eq. (2).

To find the optimal arrival time τ^*, we proceed as follows. By filling in the control policy of Eq. (10) into the cost function of Eq. (2) and evaluating the integral, we find a closed-form expression for the cost of the optimal trajectory between $\mathbf{x}_0$ and $\mathbf{x}_1$ for a given (fixed) arrival time τ:

$$c[\tau] = \tau + (\mathbf{x}_1 - \bar{\mathbf{x}}[\tau])^T G[\tau]^{-1}(\mathbf{x}_1 - \bar{\mathbf{x}}[\tau]). \quad (11)$$

The optimal arrival time τ^* is the value of τ for which this function is minimal:

$$\tau^* = \text{argmin}\{\tau > 0\}\; c[\tau], \quad (12)$$

and the cost of the optimal trajectory between $\mathbf{x}_0$ and $\mathbf{x}_1$ as defined in Eq. (4) is given by $c^*[\mathbf{x}_0, \mathbf{x}_1] = c[\tau^*]$.

The optimal arrival time τ^* is found by taking the derivative of $c[\tau]$ with respect to τ (which we denote $\dot{c}[\tau]$), and solving $\dot{c}[\tau] = 0$ for τ. The derivative is given by:

$$\dot{c}[\tau] = 1 - 2(A\mathbf{x}_1 + \mathbf{c})^T \mathbf{d}[\tau] - \mathbf{d}[\tau]^T BR^{-1}B^T \mathbf{d}[\tau], \quad (13)$$

where we define:

$$\mathbf{d}[\tau] = G[\tau]^{-1}(\mathbf{x}_1 - \bar{\mathbf{x}}[\tau]). \quad (14)$$

It should be noted that the function $c[\tau]$ may have multiple local minima. Also, note that $c[\tau] > \tau$ for all $\tau > 0$, since $G[\tau]$ is positive-definite.

C. Computing the Optimal Trajectory

Given the optimal arrival time τ^* as defined above, we find the corresponding optimal trajectory $\pi^*[\mathbf{x}_0, \mathbf{x}_1] = (\mathbf{x}[\,], \mathbf{u}[\,], \tau^*)$, as defined in Eq. (5), as follows. Let us define:

$$\mathbf{y}[t] = \exp[A^T(\tau^* - t)]\mathbf{d}[\tau^*], \quad (15)$$

such that the optimal control policy (see Eq. (10)) is:

$$\mathbf{u}[t] = R^{-1}B^T\mathbf{y}[t]. \quad (16)$$

Filling in this optimal control policy into Eq. (1) gives us the differential equation for the state $\mathbf{x}[\,]$:

$$\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + BR^{-1}B^T\mathbf{y}[t] + \mathbf{c}, \quad \mathbf{x}[\tau^*] = \mathbf{x}_1. \tag{17}$$

Noting that Eq. (15) is the solution to the differential equation:

$$\dot{\mathbf{y}}[t] = -A^T\mathbf{y}[t], \qquad \mathbf{y}[\tau^*] = \mathbf{d}[\tau^*], \tag{18}$$

and combining this with Eq. (17) gives us the composite differential equation:

$$\begin{bmatrix}\dot{\mathbf{x}}[t]\\ \dot{\mathbf{y}}[t]\end{bmatrix} = \begin{bmatrix}A & BR^{-1}B^T\\ 0 & -A^T\end{bmatrix}\begin{bmatrix}\mathbf{x}[t]\\ \mathbf{y}[t]\end{bmatrix} + \begin{bmatrix}\mathbf{c}\\ \mathbf{0}\end{bmatrix}, \quad \begin{bmatrix}\mathbf{x}[\tau^*]\\ \mathbf{y}[\tau^*]\end{bmatrix} = \begin{bmatrix}\mathbf{x}_1\\ \mathbf{d}[\tau^*]\end{bmatrix}, \tag{19}$$

which has as solution:

$$\begin{bmatrix}\mathbf{x}[t]\\ \mathbf{y}[t]\end{bmatrix} = \exp\left[\begin{bmatrix}A & BR^{-1}B^T\\ 0 & -A^T\end{bmatrix}(t-\tau^*)\right]\begin{bmatrix}\mathbf{x}_1\\ \mathbf{d}[\tau^*]\end{bmatrix} + \int_{\tau^*}^{t}\exp\left[\begin{bmatrix}A & BR^{-1}B^T\\ 0 & -A^T\end{bmatrix}(t-t')\right]\begin{bmatrix}\mathbf{c}\\ \mathbf{0}\end{bmatrix}\mathrm{d}t'. \tag{20}$$

This gives us $\mathbf{x}[t]$ and, using Eq. (16), $\mathbf{u}[t]$ for all $0 < t < \tau^*$, which completely determines $\pi^*[\mathbf{x}_0, \mathbf{x}_1]$.

D. Practical Implementation

For the implementation of the above computations in practice, we distinguish the special case in which matrix A is *nilpotent*, in which case we can derive a closed-form solution for the optimal trajectory, from the general case, in which case we can find the optimal trajectory numerically.

If matrix $A \in \mathbb{R}^{n\times n}$ is *nilpotent*, i.e. $A^n = 0$, which is not uncommon as we will see in Section VII, $\exp[At]$ has a closed-form expression in the form of an $(n-1)$-degree matrix polynomial in t. As a result, the integrals of Eqs. (6) and (8) can be evaluated exactly to obtain closed-form expressions for $G[\tau]$ and $\bar{\mathbf{x}}[\tau]$. Solving $\dot{c}[\tau] = 0$ for τ to find the optimal arrival time τ^* then amounts to finding the roots of a $2n^2$-degree polynomial in τ. Using the Jenkins-Traub method to find all roots of a polynomial [1] gives us the global minimum of $c[\tau]$ and the corresponding optimal arrival time τ^*. Subsequently, the nilpotence of A implies that the matrix $\left[\begin{smallmatrix}A & BR^{-1}B^T\\ 0 & -A^T\end{smallmatrix}\right]$ is nilpotent as well, which means that Eq. (20) can be evaluated exactly to obtain a closed form expression for the optimal trajectory (states and control inputs) between any two states $\mathbf{x}_0$ and $\mathbf{x}_1$.

For a general (not nilpotent) matrix A, we integrate $\dot{G}[t]$ and $\dot{\bar{\mathbf{x}}}[t]$ forward in time according to Eqs. (7) and (9) using the 4th-order Runge-Kutta method [2], which gives us $G[\tau]$, $\bar{\mathbf{x}}[\tau]$, and $c[\tau]$ for increasing $\tau > 0$. We keep track of the minimal cost $c^* = c[\tau]$ we have seen so far and the corresponding arrival time, as we perform the forward integration for increasing $\tau > 0$. Since $c[\tau] > \tau$ for all $\tau > 0$, it suffices to terminate the forward integration at $\tau = c^*$ to guarantee that a global minimum c^* of $c[\tau]$, and the corresponding optimal arrival time τ^*, has been found. This procedure also gives us $\mathbf{d}[\tau^*]$, which we use to subsequently reconstruct the optimal trajectory between $\mathbf{x}_0$ and $\mathbf{x}_1$, by integrating the differential equation (19) backward in time for $\tau^* > t > 0$ using 4th-order Runge-Kutta.

KINODYNAMICRRT*$[\mathbf{x}_{\text{start}} \in \mathcal{X}_{\text{free}}, \mathbf{x}_{\text{goal}} \in \mathcal{X}_{\text{free}}]$

1: $\mathcal{T} \leftarrow \{\mathbf{x}_{\text{start}}\}$.
2: **for** $i \in [1, \infty)$ **do**
3: Randomly sample $\mathbf{x}_i \in \mathcal{X}_{\text{free}}$.
4: $\mathbf{x} \leftarrow \text{argmin}\{\mathbf{x} \in \mathcal{T} \,|\, c^*[\mathbf{x}, \mathbf{x}_i] < r \wedge$ COLLISIONFREE$[\pi^*[\mathbf{x}, \mathbf{x}_i]]\}\,(\text{cost}[\mathbf{x}] + c^*[\mathbf{x}, \mathbf{x}_i])$.
5: $\text{parent}[\mathbf{x}_i] \leftarrow \mathbf{x}$.
6: $\text{cost}[\mathbf{x}_i] \leftarrow \text{cost}[\mathbf{x}] + c^*[\mathbf{x}, \mathbf{x}_i]$.
7: **for all** $\{\mathbf{x} \in \mathcal{T} \cup \{\mathbf{x}_{\text{goal}}\} \,|\, c^*[\mathbf{x}_i, \mathbf{x}] < r \wedge \text{cost}[\mathbf{x}_i] + c^*[\mathbf{x}_i, \mathbf{x}] < \text{cost}[\mathbf{x}] \wedge$ COLLISIONFREE$[\pi^*[\mathbf{x}_i, \mathbf{x}]]\}$ **do**
8: $\text{cost}[\mathbf{x}] \leftarrow \text{cost}[\mathbf{x}_i] + c^*[\mathbf{x}_i, \mathbf{x}]$.
9: $\text{parent}[\mathbf{x}] \leftarrow \mathbf{x}_i$.
10: $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{x}_i\}$.

Fig. 2. The Kinodynamic-RRT* algorithm. The tree $\mathcal{T}$ is represented as a set of states. Each state $\mathbf{x}$ in the tree has two attributes: a pointer $\text{parent}[\mathbf{x}]$ to its parent state in the tree, and a number $\text{cost}[\mathbf{x}]$ which stores the cost of the trajectory in the tree between the start state and $\mathbf{x}$. Further, we define COLLISIONFREE$[\mathbf{x}[\,], \mathbf{u}[\,], \tau] = \forall\{t \in [0, \tau]\}\,(\mathbf{x}[t] \in \mathcal{X}_{\text{free}} \wedge \mathbf{u}[t] \in \mathcal{U}_{\text{free}})$.

V. KINODYNAMIC RRT*

To find the optimal collision-free trajectory π^*_{free} as defined in Eq. (3), given the ability to find an optimal trajectory between any pair of states as described above, we use an adapted version of RRT*, since RRT* is known to achieve *asymptotic optimality*; that is, as the number of iterations of the algorithm approaches infinity, the probability that an optimal path has been found approaches 1.

A. Adapted RRT* Algorithm

Our algorithm is given in Fig. 2. The algorithm builds a tree $\mathcal{T}$ of trajectories in the free state space rooted in the start state. In each iteration i of the algorithm, a state $\mathbf{x}_i$ is sampled from the free state space $\mathcal{X}_{\text{free}}$ (line 3) to become a new node of the tree (line 10). For each new node a parent is found among neighboring nodes already in the tree, i.e. the nodes $\mathbf{x}$ for which $c^*[\mathbf{x}, \mathbf{x}_i] < r$ for some neighbor radius r. The node $\mathbf{x}$ that is chosen as parent is the node for which the optimal trajectory $\pi[\mathbf{x}, \mathbf{x}_i]$ to the new node is collision-free (i.e. the states and control inputs along the trajectory are in the respective free spaces) and results in a minimal cost between the root node ($\mathbf{x}_{\text{start}}$) and the new node (lines 4-6). Subsequently, it is attempted to decrease the cost from the start to other nodes in the tree by connecting the new node to neighboring nodes in the tree, i.e. the nodes $\mathbf{x}$ for which $c^*[\mathbf{x}_i, \mathbf{x}] < r$. For each state $\mathbf{x}$ for which the connection is collision-free and results in a lower cost to reach $\mathbf{x}$ from the start, the new node $\mathbf{x}_i$ is made the parent of $\mathbf{x}$ (lines 7-9). Then, the algorithm continues with a new iteration. If this is repeated indefinitely, an optimal path between $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ will emerge in the tree.

The algorithm as given in Fig. 2 differs subtly from the standard RRT* algorithm. First of all, we have defined our problem as finding a trajectory that exactly arrives at a goal state, rather than a goal region as is common in RRT*. As a consequence, we explicitly add the goal state to the set of states that is considered for a forward connection from a newly sampled node in line 7, even if the goal is not (yet) part of tree. Also, typical RRT* implementations include a "steer"

module, which lets the tree grow *towards* a sampled state (but not necessarily all the way), and adds the endpoint of a partial trajectory as a node to the tree [10]. Since it is non-trivial given our formulation to compute a partial trajectory of a specified maximum cost, our algorithm attempts a full connection to the sampled state, and adds the sampled state itself as a node to the tree. These changes do not affect the asymptotic optimality guarantee of the algorithm.

B. Reachable Sets and Determining the Neighbor Radius

In the original RRT* algorithm, the neighbor radius r can be decreased over the course of the algorithm as a function of the number of nodes i currently in the tree, without affecting the asymptotic optimality guarantee, by setting r such that neighbors are selected from a ball with volume $\gamma \log[i]/i$ (where $\gamma > 2^n(1+1/n)\mu[\mathcal{X}_{\text{free}}]$ and $\mu[\mathcal{X}_{\text{free}}]$ is the volume of the state space) [8]. For non-Euclidean distance measures, such as in our case, let $\mathcal{R}^+[\mathbf{x}, r]$ be the forward-reachable set of state $\mathbf{x}$, i.e. set of states that can be reached from $\mathbf{x}$ with cost less than r, and let $\mathcal{R}^-[\mathbf{x}, r]$ be the backward-reachable set, i.e. set of states that can reach $\mathbf{x}$ with cost less than r:

$$\mathcal{R}^+[\mathbf{x}, r] = \{\mathbf{x}' \in \mathcal{X} \,|\, c^*[\mathbf{x}, \mathbf{x}'] < r\}, \tag{21}$$

$$\mathcal{R}^-[\mathbf{x}, r] = \{\mathbf{x}' \in \mathcal{X} \,|\, c^*[\mathbf{x}', \mathbf{x}] < r\}. \tag{22}$$

In this case the neighbor radius r must be set such that a ball of volume $\gamma \log[i]/i$ is contained within either $\mathcal{R}^+[\mathbf{x}_i, r]$ or $\mathcal{R}^-[\mathbf{x}_i, r]$ [9], where $\mathbf{x}_i$ is the i'th node sampled in the RRT*-algorithm (see line 3).

For general dynamics systems, it is hard to determine the reachable sets explicitly [9], but for our formulation we can. Let us focus on the forward-reachable set $\mathcal{R}^+[\mathbf{x}_0, r]$ of some state $\mathbf{x}_0$. As follows from Eq. (11), the set of states that can be reached from $\mathbf{x}_0$ with cost less than r for a *given* arrival time τ forms an *ellipsoid*:

$$\begin{aligned} &\{\mathbf{x}_1 \,|\, \tau + (\mathbf{x}_1 - \bar{\mathbf{x}}[\tau])^T G[\tau]^{-1} (\mathbf{x}_1 - \bar{\mathbf{x}}[\tau]) < r\} \\ =\ &\{\mathbf{x}_1 \,|\, (\mathbf{x}_1 - \bar{\mathbf{x}}[\tau])^T \frac{G[\tau]^{-1}}{r-\tau} (\mathbf{x}_1 - \bar{\mathbf{x}}[\tau]) < 1\} \\ =\ &\mathcal{E}[\bar{\mathbf{x}}[\tau], G[\tau](r-\tau)], \end{aligned} \tag{23}$$

where $\mathcal{E}[\mathbf{x}, M]$ is an ellipsoid with center $\mathbf{x}$ and (positive-definite) weight matrix M, formally defined as:

$$\mathcal{E}[\mathbf{x}, M] = \{\mathbf{x}' \,|\, (\mathbf{x}' - \mathbf{x})^T M^{-1} (\mathbf{x}' - \mathbf{x}) < 1\}. \tag{24}$$

Hence, the forward-reachable set $\mathcal{R}^+[\mathbf{x}_0, r]$ is defined as the union of the reachable ellipsoids for all possible arrival times τ (see Fig. 3):

$$\mathcal{R}^+[\mathbf{x}_0, r] = \bigcup_{0<\tau<r} \mathcal{E}[\bar{\mathbf{x}}[\tau], G[\tau](r-\tau)]. \tag{25}$$

Backward-reachable sets $\mathcal{R}^-[\mathbf{x}, r]$ are defined similarly.

Now, after a state $\mathbf{x}_i$ is sampled in line 3 of the RRT*-algorithm, we must choose r such that a ball of volume $\gamma \log[i]/i$ is contained within $\mathcal{R}^+[\mathbf{x}_i, r]$. Since we can always linearly transform the state-space such that an ellipsoid becomes a ball, it is equivalent to make sure that an ellipsoid with volume $\gamma \log[i]/i$ is contained within $\mathcal{R}^+[\mathbf{x}_i, r]$. The

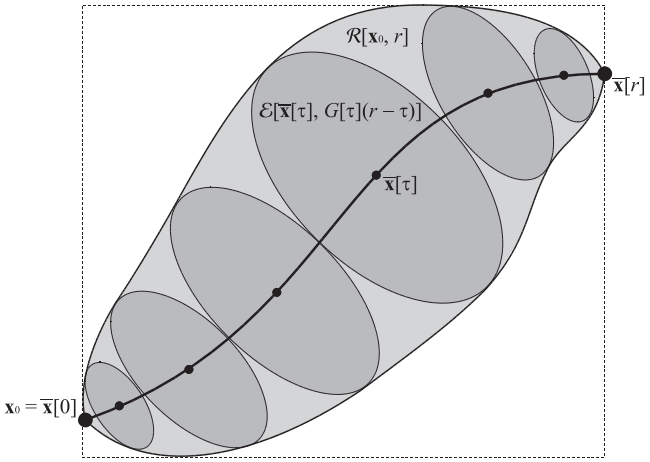

Fig. 3. The forward-reachable set $\mathcal{R}^+[\mathbf{x}_0, r]$ is a union of ellipsoids.

squared volume of an ellipsoid $\mathcal{E}[\mathbf{x}, M]$ equals $\zeta_n^2 \det[M]$, where ζ_n is the volume of an n-dimensional sphere. Now, let $v[r]$ be the squared volume of the *largest* ellipsoid that constitutes $\mathcal{R}^+[\mathbf{x}_i, r]$ for a given value of r:

$$v[r] = \max\{0 < \tau < r\}\ \zeta_n^2 \det[G[\tau](r-\tau)] \tag{26}$$

(note that the function v is independent of $\mathbf{x}_i$). The value of τ where the maximum is achieved is found by taking the derivative of $\det[G[\tau](r-\tau)]$ with respect to τ, equating to 0, and solving for τ. This equation has exactly one solution where τ is unequal to 0 or to r, and if dynamics matrix A is nilpotent, the equation is polynomial. Hence, by *deflating* the polynomial using the solutions $\tau = 0$ and $\tau = r$, we find a linear relation between r and the maximizing value of τ, and hence a closed-form polynomial expression for $v[r]$. The neighbor radius r that is selected in the i'th iteration of the algorithm is then computed by solving the polynomial equation $v[r] = (\gamma \log[i]/i)^2$ for r.

C. Efficient Neighbor Search

Having computed the neighbor radius r for the i'th iteration of the RRT* algorithm, line 4 requires to find the neighboring nodes in the tree contained in the backward-reachable set $\mathcal{R}^-[\mathbf{x}_i, r]$ of $\mathbf{x}_i$, and line 7 requires to find the neighboring nodes in the forward-reachable set $\mathcal{R}^+[\mathbf{x}_i, r]$. For this, we use a k-d tree datastructure [3] that allows for the efficient search of nodes within axis-aligned ranges.

Therefore, given a node $\mathbf{x}_i$, we need to find the axis-aligned bounding box of $\mathcal{R}^+[\mathbf{x}_i, r]$ (and $\mathcal{R}^-[\mathbf{x}_i, r]$) to query the datastructure for neighboring nodes. The axis-aligned bounding box of an ellipsoid $\mathcal{E}[\mathbf{x}, M]$ is given by the Euclidean product $\prod_{k=1}^n [\mathbf{x}_{(k)} - \sqrt{M_{(k,k)}}, \mathbf{x}_{(k)} + \sqrt{M_{(k,k)}}]$, where $\mathbf{x}_{(k)}$ is the k'th element of $\mathbf{x}$, and $M_{(k,k)}$ the k'th element on the diagonal of M. Hence, the bounding box of the entire reachable set $\mathcal{R}^+[\mathbf{x}_i, r]$ is given by:

$$\begin{aligned} &\prod_{k=1}^{n} \big[\, \min\{0 < \tau < r\}\ (\bar{\mathbf{x}}[\tau]_{(k)} - \sqrt{G[\tau]_{(k,k)}(r-\tau)}), \\ &\max\{0 < \tau < r\}\ (\bar{\mathbf{x}}[\tau]_{(k)} + \sqrt{G[\tau]_{(k,k)}(r-\tau)})\big]. \end{aligned} \tag{27}$$

Each of the minimizing and maximizing values for τ are found by taking the derivative of $(\bar{\mathbf{x}}[\tau]_{(k)} \pm \sqrt{G[\tau]_{(k,k)}(r-\tau)})$ with respect to τ, equating to 0, and solving for τ. If dynamics matrix A is nilpotent, these are $4n$-degree polynomial equations. The bounding box for $\mathcal{R}^-[\mathbf{x}_i, r]$ is computed similarly.

VI. Systems with Non-Linear Dynamics

We have presented our algorithm for linear dynamics systems of the type of Eq. (1), but we can apply our algorithm to non-linear dynamics as well through linearization. Let the non-linear dynamics of the robot be defined by a function $\mathbf{f}$:

$$\dot{\mathbf{x}}[t] = \mathbf{f}[\mathbf{x}[t], \mathbf{u}[t]]. \tag{28}$$

We can locally approximate the dynamics by linearizing the function $\mathbf{f}$ to obtain a system of the form of Eq. (1), with:

$$A = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}[\hat{\mathbf{x}}, \mathbf{0}], \quad B = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}[\hat{\mathbf{x}}, \mathbf{0}], \quad \mathbf{c} = \mathbf{f}[\hat{\mathbf{x}}, \mathbf{0}] - A\hat{\mathbf{x}}, \tag{29}$$

where the dynamics are linearized about state $\hat{\mathbf{x}}$ and control input $\hat{\mathbf{u}} = \mathbf{0}$. The resulting linear system is a first-order Taylor approximation of the non-linear dynamics, and is approximately valid only when $\mathbf{x} \approx \hat{\mathbf{x}}$ and $\mathbf{u} \approx \mathbf{0}$.

We adapt the algorithm of Fig. 2 to non-linear dynamics by (re)linearizing $\mathbf{f}$ in each iteration of the algorithm about $\hat{\mathbf{x}} = \mathbf{x}_i$ after a new state $\mathbf{x}_i$ is sampled in line 3. The resulting linear dynamics are then used in the subsequent computations of the functions c^* and π^* (note that this only works if the linearized dynamics are controllable). We choose $\hat{\mathbf{x}} = \mathbf{x}_i$ since it is either the start or the end point of any trajectory computed in that iteration of the algorithm, and we choose $\hat{\mathbf{u}} = \mathbf{0}$ since the cost function (see Eq. (2)) explicitly penalizes deviations of the control input from zero.

The linearization is only a valid approximation if the computed trajectories do not venture too much away from the linearization point. As over the course of the algorithm the distances between states get shorter (due to a decreasing neighbor radius r) and the trajectories in the tree get more optimal (hence having control inputs closer to zero), this approximation becomes increasingly more reasonable. It is not clear, however, whether any convergence guarantees can be given for non-linear systems.

VII. Experimental Results

We experimented with our implementation on three kino-dynamic systems; a double integrator disk robot operating in the plane, a quadrotor robot operating in three space, and a non-holonomic car-like robot operating in the plane, which are discussed in detail in Sections VII-A to VII-C. Simulation results are subsequently analyzed in Section VII-D.

A. Linear Double Integrator Model

The double integrator robot is a circular robot capable of moving in any direction by controlling its acceleration. Its state space is four-dimensional, and its linear dynamics are described by:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}, \quad A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad R = \rho I, \tag{30}$$

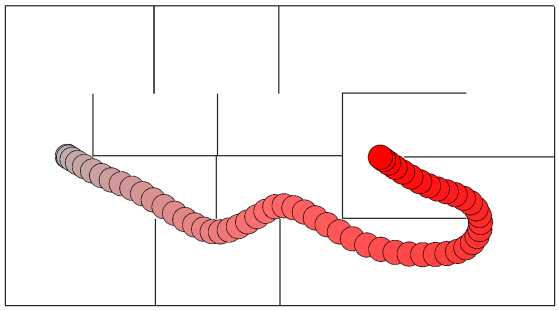

Fig. 4. Asymptotically optimal trajectory for double integrator robot after 100,000 nodes were added to the tree.

$\mathbf{u} = \mathbf{a}$, and $\mathbf{c} = \mathbf{0}$, where $\mathbf{p}$ describes its position in the plane, $\mathbf{v}$ its velocity, and $\mathbf{a}$ its acceleration. Further, we set bounds such that $\mathbf{p} \in [0, 200] \times [0, 100]$ (m), $\mathbf{v} \in [-10, 10]^2$ (m/s), and $\mathbf{u} = \mathbf{a} \in [-10, 10]^2$ (m/s^2). The control penalty ρ was set to 0.25 as this permitted the robot to reach velocities near its bounds but not frequently exceed them.

We experimented with this model in the environment of Fig. 4, which has two homotopy classes. A is nilpotent as $A^2 = 0$, so we can use both the closed-form and the numerical method for computing connections between states.

B. Linearized Quadrotor Model

The quadrotor helicopter was modeled after the Ascending Technologies' ResearchPilot. Its state $\mathbf{x} = (\mathbf{p}^T, \mathbf{v}^T, \mathbf{r}^T, \mathbf{w}^T)^T$ is 12-dimensional, consisting of three-dimensional position $\mathbf{p}$, velocity $\mathbf{v}$, orientation $\mathbf{r}$ (rotation about axis $\mathbf{r}$ by angle $||\mathbf{r}||$), and angular velocity $\mathbf{w}$. Its dynamics are non-linear [17], but are well-linearizable about the hover point of the quadrotor. The linearization is (very) sensitive though to deviations in the yaw. Fortunately, the yaw is a redundant degree of freedom, so in our linearization, we constrain the yaw (and its derivatives) to zero. This gives a reduced ten-dimensional state and three-dimensional control input, with the following linearized dynamics:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{r} \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_f \\ u_x \\ u_y \end{bmatrix}, \quad A = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & \left[\begin{smallmatrix} 0 & g \\ -g & 0 \\ 0 & 0 \end{smallmatrix}\right] & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} \mathbf{0} & 0 \\ \left[\begin{smallmatrix} 0 \\ 0 \\ 1/m \end{smallmatrix}\right] & 0 \\ \mathbf{0} & 0 \\ \mathbf{0} & \ell I/j \end{bmatrix}, \quad \mathbf{c} = \mathbf{0}, \quad R = \begin{bmatrix} \frac{\rho}{4} & 0 & 0 \\ 0 & \frac{\rho}{2} & 0 \\ 0 & 0 & \frac{\rho}{2} \end{bmatrix}, \tag{31}$$

where $\mathbf{r}$ and $\mathbf{w}$ are two-dimensional (with their third component implicitly zero), $g = 9.8$m/s^2 is the gravity, m is the mass of the quadrotor (kg), ℓ the distance between the center of the vehicle and each of the rotors (m), and j is the moment of inertia of the vehicle about the axes coplanar with the rotors (kg m^2). The control input $\mathbf{u}$ consists of three components: u_f is the total thrust of the rotors relative to the thrust needed for hovering, and u_x and u_y describe the relative thrust of the rotors producing roll and pitch, respectively. Further, the bounds are defined as $\mathbf{p} \in [0, 5]^3$ (m),

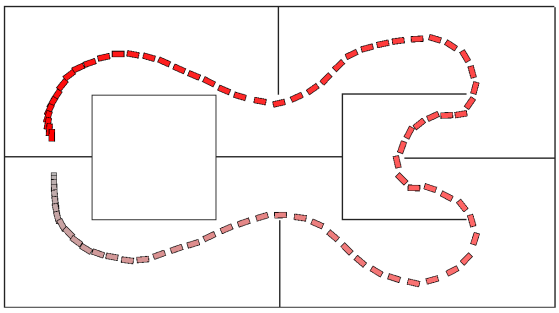

Fig. 5. Asymptotically optimal trajectory for the car-like robot after 60,000 nodes were added to the tree.

$\mathbf{v} \in [-5,5]^3$ (m/s), $\mathbf{r} \in [-1,1]^2$ (rad), $\mathbf{w} \in [-5,5]^2$ (rad/s), $u_f \in [-4.545, 9.935]$ (N), and $u_x, u_y \in [-3.62, 3.62]$ (N). The matrix R was chosen such that producing force is penalized equally for each rotor.

The quadrotor simulations were performed in the environment of Fig. 1 for the linearized dynamics. Clearly, A is nilpotent as it is strictly upper diagonal, so we can use both the closed-form and the numerical method for computing connections between states.

C. Non-Linear Car-Like Model

The car-like robot has a five-dimensional state $\mathbf{x} = (x, y, \theta, v, \kappa)^T$, consisting of its planar position (x, y) (m), its orientation θ (rad), speed v (m/s), and curvature κ (m^{-1}). The control input $\mathbf{u} = (u_v, u_\kappa)^T$ is two-dimensional and consists of the derivatives of speed and curvature, respectively. The dynamics are non-linear, and described by $\dot{\mathbf{x}} = \mathbf{f}[\mathbf{x}, \mathbf{u}]$, with $\mathbf{f}$ given by:

$$\dot{x} = v\cos\theta,\ \dot{y} = v\sin\theta,\ \dot{\theta} = v\kappa,\ \dot{v} = u_v,\ \dot{\kappa} = u_\kappa. \quad (32)$$

For this system, we repeatedly linearize the dynamics about the last sampled state, as described in Section VI. Let this state be $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{\theta}, \hat{v}, \hat{\kappa})^T$, then the linearized dynamics are given by:

$$A = \begin{bmatrix} 0 & 0 & -\hat{v}\sin\hat{\theta} & \cos\hat{\theta} & 0 \\ 0 & 0 & \hat{v}\cos\hat{\theta} & \sin\hat{\theta} & 0 \\ 0 & 0 & 0 & \hat{\kappa} & \hat{v} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (33)$$

and $\mathbf{c} = \mathbf{f}[\hat{\mathbf{x}}, \mathbf{0}] - A\hat{\mathbf{x}}$, where bounds were set such that $p_x \in [0, 200]$, $p_y \in [0, 100]$, $\theta \in [-\pi, \pi]$, $v \in (0, 10]$, $\kappa \in [-0.25, 0.25]$. We note that the velocity v must be non-zero, otherwise the resulting linear dynamics are not controllable.

The car-like robot experiments were performed in the environment of Fig. 5. Also in this case, the dynamics matrix A is nilpotent for all linearizations, so we can use both the closed-form and the numerical method for computing connections between states.

D. Analysis of Results

We used our algorithm to compute asymptotically optimal trajectories for the double integrator, the quadrotor, and the car-like robots. They are shown in Figs. 4, 1, and 5, respectively. While the paths found for the double integrator and quadrotor robots appear continuous and smooth, in the case of the car-like robot effects of linearization are clearly visible; the robot appears to skid sideways to some extent, as if drifting through the curves. The double integrator experiment was executed thirty times to show that our algorithm converges to the optimal solution even when solutions from multiple homotopy classes exist. While the upper path was initially found during some experiments, the lower path was always selected as the final solution with a mean cost of 61.87 and a variance of 0.58.

TABLE I

TIMING DATA FOR THE FIRST 5000 NODES OF EACH SIMULATION (S).

	Closed Form			Runge Kutta 4		
nodes	DblInt	QuadRtr	Car	DblInt	QuadRtr	Car
1000	19.75	116.2	5.42	969.85	4643.52	274.8
2000	43.26	274.9	12.43	3638.46	10819.7	331.2
3000	72.59	507.3	21.74	7872.15	20284.9	405.6
4000	108.57	841.6	31.99	13479.4	33400.9	497.4
5000	150.36	1168	42.94	20771.7	49477.8	606.3

TABLE II

TIMING DATA WITH RADIUS REDUCTION AND NEIGHBOR SEARCH (S).

	Linear		k-d Tree	
nodes	DblInt	QuadRtr	DblInt	QuadRtr
1,000	10.251	51.603	11.018	62.694
5,000	75.437	968.106	66.978	1159.94
10,000	210.785	3632.24	150.869	4344.18
15,000	402.068	8045.18	260.388	10745.2
20,000	645.075	14083	339.977	16606.8

Table I shows the time required to expand the first 5,000 nodes for all three systems using both the closed-form method and the numerical 4th-order Runge-Kutta (RK4) method for computing connections between states, where we used a constant neighbor radius r and linear (brute-force) neighor search. It is clear that the closed-form method executed much more quickly than the RK4 method in all cases. On average, we see a factor of 45 difference in running time. Both methods resulted in solutions with comparable costs after expanding the same number of nodes. Processing nodes for the quadrotor experiment are most computationally intensive, as a direct result of the high-dimension of its state space. Nodes were more quickly processed for the car-like robot than for the double integrator despite a higher dimensionality, because a smaller initial neighbor radius r was used to ensure short connections.

Table II shows timing data for the double integrator and quadrotor experiments using the closed-form method in which the radius r is gradually reduced as the tree grows (see Section V-B), and in which we search for neighbors using either the linear (brute-force) approach or using the k-d tree (see Section V-C). To analyze the effect of radius reduction, compare the "linear" columns of Table II with the "closed-form" columns of Table I at 5,000 nodes. For the double integrator we see 48% speed-up and for the quadrotor a 15% speedup. The speedup is less for the quadrotor, as for such a high-dimensional system it is relatively more expensive to compute the cost of a connection compared to collision-checking it. Table II also shows the effect of using a k-d

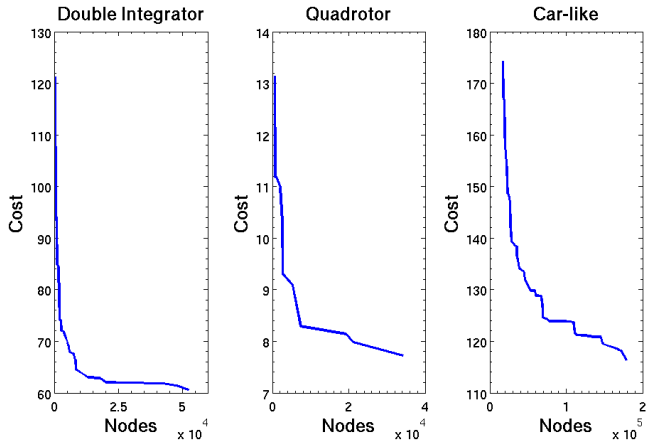

Fig. 6. Graphs showing the cost of the current-best solution as a function of the number of nodes in the tree for (from left to right) the double integrator, the quadrotor, and the car-like robot experiments.

tree for neighbor searching. For the double integrator, using the k-d tree significantly speeds up the neighbor search, particularly with a higher number of nodes in tree. For the quadrotor, linear search is faster. This is because the benefit of using a k-d tree diminishes with increasing dimension of the space [3], and because of the overhead of computing the bounding box of the reachable sets (see Section V-C).

Figure 6 shows the cost of the current-best solution for each experiment as more nodes are added to the tree. A total of 100,000 nodes were expanded in the double integrator robot simulation, 20,000 nodes in the quadrotor robot simulation, and 200,000 nodes in the car-like robot simulation. In all cases we see that a high-cost solution is found in relatively few nodes, and that these solutions are quickly refined as more nodes are added. Necessarily these refinements plain off as the solutions approach the asymptotic optimum.

VIII. Discussion, Conclusion, and Future Work

We have presented Kinodynamic RRT*, an incremental sampling-based approach that extends RRT* for asymptotically optimal motion planning for robots with linear differential constraints. Our approach achieves asymptotically optimality by using a fixed-final-state-free-final-time optimal control formulation that connects any pair of states exactly and optimally for systems with controllable linear dynamics. We have shown that for the rich subclass of systems with a nilpotent dynamics matrix, such trajectories can be computed efficiently, making asymptotically optimal planning computationally feasible for kinodynamic systems, even in high-dimensional state spaces. We plan to make the source code of our implementation publicly available for download.

For our experiments, we have not fully optimized our implementation, and we believe that running times can be further improved. In particular, extensions suggested in earlier work [10], such as using an admissible heuristic that can be quickly computed and provides a conservative estimate of the true cost of moving between two states may prune many (relatively costly) attempts to connect pairs of states. Such a heuristic can then also be used in a branch-and-bound technique [10] to prune parts of tree of which one knows it will never contribute to an optimal solution.

Other areas of potential improvement include studying non-uniform sampling to accelerate the convergence to optimal solutions. One could sample more heavily around the current optimal solution, or use stochastic techniques to infer distributions of samples that are likely to contribute to an optimal trajectory. For a quadrotor helicopter for instance, one can imagine that there is a strong correlation between its velocity and orientation, which should be reflected in the sampling. In addition, we note that, as mentioned in the introduction, the ability to connect any pair of states can be used to perform trajectory smoothing by iterative shortcutting as post-processing step. This may improve the quality of solutions further and provide better estimates of the convergence rate of the algorithm.

Lastly, we plan to apply our planner to real-world robots, in particular quadrotors. This would require constructing a stabilizing controller around the computed trajectory, either using traditional techniques such as LQR, or by repeatedly computing reconnections between the current state of the robot and a state on the trajectory.

References

[1] C. Bond. Jenkins-Traub real polynomial root finder. *http://www.crbond.com*. 2002.

[2] R. Burden, D. Faires. *Numerical Analysis*. Brooks/Cole, 2001.

[3] M. de Berg, M. van Kreveld, M. Overmars, O. Schwartzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[4] B. Donald, P. Xavier, J. Canny, J. Reif. Kinodynamic motion planning. *Journal of the ACM* 40(5):1048-1066, 1993.

[5] J. Jeon, S. Karaman, E. Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. *IEEE Conf. on Decision and Control*, 2011.

[6] R. Geraerts, M. Overmars. Creating high-quality paths for motion planning. *Int. J. of Robotics Research*, 26(8):845-863, 2007.

[7] E. Glassman, R. Tedrake. A quadratic regulator-based heuristic for rapidly exploring state space. *IEEE Int. Conf. on Robotics and Automation*, 2010.

[8] S. Karaman, E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics: Science and Systems*, 2010.

[9] S. Karaman, E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. *IEEE Conf. on Decision and Control*, 2010.

[10] S. Karaman, E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. of Robotics Research* 30(7):846-894, 2011.

[11] L. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation* 12(4):566580, 1996.

[12] J.-C. Latombe. *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[13] S. LaValle, J. Kuffner. Randomized kinodynamic planning. *Int. J. of Robotics Research* 20(5):378-400, 2001.

[14] S. LaValle. Planning Algorithms. Cambridge University Press, New York, 2006.

[15] F. Lewis, V. Syrmos. *Optimal Control*. John Wiley & Sons, 1995.

[16] J. Marble, K. Bekris. Towards small asymptotically near-optimal roadmaps. *IEEE Int. Conf. on Robotics and Automation*, 2012.

[17] N. Michael, D. Mellinger, Q. Lindsey, V. Kumar. The GRASP multiple mirco-UAV test bed: experimental evaluation of multirobot aerial control algorithms. *IEEE Robotics and Automation Magazine* 17(3):56-65, 2010.

[18] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. *IEEE Conf. on Robotics and Automation*, 2012.

[19] R. Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. *Robotics: Science and Systems*, 2009.