Accelerating Kinodynamic RRT* through Dimensionality Reduction

Donglinag Zheng[1] and Panagiotis Tsiotras[2]

***Abstract*— Sampling-based motion planning algorithms such as RRT* are well-known for their ability to quickly find an initial solution and then converge to the optimal solution asymptotically. However, the convergence rate can be slow for high-dimensional planning problems, particularly for dynamical systems where the sampling space is not just the configuration space but the full state space. In this paper, we introduce the idea of using a partial-final-state-free (PFF) optimal controller in kinodynamic RRT* [1] to reduce the dimension of the sampling space. Instead of randomly sampling the full state space, the proposed accelerated kinodynamic RRT*, called Kino-RRT*, only samples part of the state space, while the rest of the states are selected by the PFF optimal controller. We also propose a delayed and intermittent update of the optimal arrival time of all the edges in the RRT* tree to decrease the computation complexity of the algorithm. We tested the proposed algorithm using 4-D and 10-D state-space linear systems and showed that Kino-RRT* converges much faster than the kinodynamic RRT* algorithm.**

I. Introduction

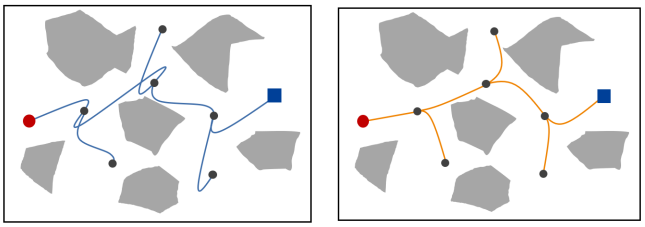
Robotic motion planning with the goal of finding a dynamically feasible and optimal trajectory for the robot through an environment with obstacles, has gained much progress over the past decades. As a fundamental problem in robotics applications, it is still a very challenging problem to solve when the environment is complex with irregular obstacles and the dynamics of the robot are to be considered [2].

Sampling-based motion planning algorithms, such as rapidly exploring randomized trees (RRTs) [3], have been developed to solve planning problems in high-dimensional continuous state spaces by incrementally building a tree through the search space. The asymptotic optimal variant of RRT, namely RRT* [4], almost surely converges asymptotically to the optimal solution. RRT* is well-suited for planning in high-dimensional spaces and obstacle-rich environments. Many applications of RRT* have been studied in recent years [5], [6], [7].

One limitation of RRT* is that it requires any two points sampled in the planning space to be connected with an optimal trajectory. Thus, many works on RRT* consider robots with simple dynamics [5], [8] or assume a holonomic model and connect sampled points with straight lines [9]. For robots with differential constraints, the optimal trajectory between two states is obtained by solving a two-point boundary value problem (TPBVP), which is an non-trivial undertaking for complex nonlinear systems. The solution to this local TPBVP is also referred to as the steering function. A version of the RRT* algorithm that explicitly considers differential dynamics is the kinodynamic RRT* [1], [8].

Solving TPBVPs is the computationally dominant component of kinodynamic RRT*, and thus researchers have looked into more efficient ways to solve these TPBVPs. A steering function based on LQR is used in [10]. A fixed-final-state free-final-time controller that optimally connects any pair of states is used in [1]. Learning-based RRT* algorithms are introduced in [11], [12], [13], where the TPBVP is solved using supervised learning [11], [12] and reinforcement learning [13].

Another challenge of RRT* is the slow convergence rate of the solution to the optimal one, which is especially evident for the kinodynamic case where the sampling space is not just the configuration space but the full state space. Heuristic and informed sampling methods have been developed to improve the convergence rate [9], [14], [15]. However, these methods only consider the geometric planning problem. The dynamics of the robot is not considered. Good heuristics for improving the convergence of kinodynamic RRT* is an open research problem [16], [17].



(a) (b)

Fig. 1: Motivation of the partial-final-state-free (PFF) optimal controller. (a) Existing kinodynamic RRT* algorithms sample the full state space, which results in inefficient trajectories. (b) Kino-RRT* with a PFF controller samples the reduced state-space to improve convergence performance.

In this paper, we build on previous work on the kinodynamic RRT* [1] and propose a new algorithm, called Kino-RRT*, which shows faster convergence. We propose the idea of using a partial-final-state-free (PFF) optimal controller to reduce the sampling dimension of the state space. The motivation is illustrated in Figure 1. Instead of randomly sampling the full state space, the proposed Kino-RRT* only samples part of the state space. The rest of the states are selected by the PFF optimal controller. Because part of the final states are computed by the PFF controller to optimize

This work has been supported by NSF awards IIS-1617630 and IIS-2008695

[1]Dongliang Zheng is with School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA dzheng@gatech.edu

[2]Panagiotis Tsiotras is with School of Aerospace Engineering and Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA tsiotras@gatech.edu

the cost function, Kino-RRT* samples in the state space with reduced dimension. The method can also be interpreted as a heuristic for state-space sampling. Choosing the partial-free final states by the PFF optimal controller is more efficient than random sampling, and thus the resulting algorithm achieves faster convergence. We derive an analytical solution of the PFF optimal controller for the case of linear systems. Note, however, that the idea of using PFF controller in kinodynamic RRT* is not limited to linear systems. It can be adopted similarly to [18], [11], [12] to deal with nonlinear dynamics as well.

Finding the optimal arrival time for the TPBVP in the kinodynamic RRT* requires solving a root-finding problem of a high-order polynomial. Because the TPBVP is required to be solved repeatedly, the root-finding procedure can be computationally expensive. We therefore also propose a delayed and intermittent update of the optimal arrival time of all the edges in the tree to decrease the computation complexity of the kinodynamic RRT* algorithm.

The remainder of the paper is organized as follows. Some related works are given in Section II. The statement of the problem studied in this paper is given In Section III. In Section IV, the PFF optimal controller is derived. The PFF controller is a key ingredient of the proposed Kino-RRT* algorithm, which is outlined in Section V. The implementation of Kino-RRT* on different robot systems is given in Section VI. Section VII concludes the paper.

II. Related Works

Incremental sampling-based motion planning algorithms find a solution in high dimensional planning spaces by incrementally improving the solution . For motion planning of robot systems, considering the differential constraints is necessary for generating feasible trajectories. The extension of RRT* to dynamic systems is studied in [8], where sufficient conditions ensuring asymptotic optimality of the RRT* for dynamic systems were established. Every *local steering* and *distance function* in kinodynamic RRT* requires the solution of a TPBVP [8]. Assuming a solver of the TPBVP is available, references [19], [20], [21] study the radius of the neighbor nodes in kinodynamic RRT* to guarantee asymptotic optimality.

Solving the TPBVPs is the computationally expensive component of the kinodynamic RRT* algorithm. Infinite-horizon and finite-horizon LQR controllers were used as the steering function in kinodynamic RRT* for linear or linearized systems in [10] and [22], respectively. However, these methods cannot achieve the exact connection of two states, which is required in the kinodynamic RRT* algorithm. A fixed-final-state free-final-time controller is used in [1] to achieve the exact connection of any pair of states for linear or linearized systems. The optimal arrival time is computed by solving a root-finding problem. To deal with nonlinear dynamics, [18] directly uses a numerical solver to solve the TPBVP online, and [23] uses discrete motion primitives. Learning-based methods also have been studied to solve the TPBVP in kinodynamic RRT*. References [11] and [12] use offline generated optimal trajectories and supervised learning to train neural networks to solve the TPBVP. In [13], the steering function is realized by a local policy trained using Deep Reinforcement Learning.

Other works solve the sampling-based kinodynamic motion planning problem without relying on TPBVP solvers [24], [25]. These methods extended RRT-style shooting methods to kinodynamic planning by randomly sampling piece-wise constant control inputs of the system. However, the convergence to high-quality trajectories in practice can be slow by the use of random controls [26], [18], [27].

III. Problem formulation

The optimal kinodynamic motion planning problem is defined as finding a dynamically feasible trajectory for the robot to reach the goal state starting from an initial state, while satisfying the state and control constraints and minimizing a cost function [8], [1]. Specifically, given the planning domain X, free space X_{free}, goal region X_{goal}, initial state x_0, consider the dynamics of the robot

$$\dot{x} = Ax + Bu + c, \tag{1}$$

and the cost function,

$$J(u) = \int_0^T (1 + u^\top R u)\, \mathrm{d}\tau, \tag{2}$$

the goal of the motion planning problem is to find a control $u(t)$, $t \in [0,T]$, such that the solution $x(t)$ to (1) is obstacle-free, i.e. $x(t) \in X_{\text{free}}$ for all $t \in [0,T]$, reaches the goal region, i.e. $x(T) \in X_{\text{goal}}$, and minimizes the cost functional (2).

RRT*-type algorithms try to solve this problem by growing a tree, which involves randomly sampling intermediate states (nodes), and making optimal connections between states (edges). This results in converging to the optimal solution asymptotically. In kinodynamic RRT*, every edge between two states is the solution of a TPBVP given by

$$\begin{aligned} u^* = &\arg\min_u \ J(u), \\ \text{s.t.} \ \ &\dot{x}(t) = Ax + Bu + c, \\ &x(0) = x_a, \ x(t_{\text{f}}) = x_b, \end{aligned} \tag{3}$$

where J is the same as in (2) but over the time interval $[0, t_{\text{f}}]$, and x_a and x_b are the sampled initial state and final state of this edge, respectively. The solution of (3) with free-final-time t_{f} is given in [1]. Besides this fixed-final-state free-final-time controller, next we will present a partial-final-state-free controller, which is the key ingredient of the proposed Kino-RRT* algorithm.

IV. Partial-Final-State-Free Optimal Controller

Rewrite the state $x \in \mathbb{R}^n$ as the concatenation of two vectors $x = [x_1^\top \ x_2^\top]^\top$, where $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$ with $n_1 + n_2 = n$. The partial-final-state-free (PFF) optimal control

problem is given by

$$\begin{aligned} u^* &= \arg\min_u \ J(u), \\ \text{s.t.} \ \ &\dot{x}(t) = Ax + Bu + c, \\ &x(0) = x_a, \ x_1(t_\text{f}) = x_c. \end{aligned} \tag{4}$$

First, we consider the case where the arrival time t_f is given. Instead of fixing the states $x(0)$ and $x(t_\text{f})$ as in (3), $x(0)$ and $x_1(t_\text{f})$ are fixed, while $x_2(t_\text{f})$ is free in (4).

A. The PFF Optimal Controller

We solve this PFF optimal control problem using Pontryagin's Maximum Principle [28]. The Hamiltonian of the system is given by

$$H(x,u,t,\lambda) = 1 + u^\top Ru + \lambda^\top (Ax + Bu + c). \tag{5}$$

The necessary conditions for optimality are

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

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = -A^\top \lambda, \tag{7}$$

$$0 = \frac{\partial H}{\partial u} = 2Ru + B^\top \lambda, \tag{8}$$

$$0 = \lambda_2(t_\text{f}), \tag{9}$$

where $\lambda = [\lambda_1^\top \ \ \lambda_2^\top]^\top$, $\lambda_1 \in \mathbb{R}^{n_1}$ is the costate of x_1, and $\lambda_2 \in \mathbb{R}^{n_2}$ is the costate of x_2. Solving for u using (8), we get

$$u = -\frac{1}{2} R^{-1} B^\top \lambda. \tag{10}$$

Substituting (10) into (6), yields

$$\dot{x} = Ax - \frac{1}{2} R^{-1} B R^{-1} B^\top \lambda + c. \tag{11}$$

The analytical solutions for the differential equations (7) and (11) are available and are given by

$$\lambda(t) = e^{A^\top (t_\text{f} - t)} \lambda(t_\text{f}), \tag{12}$$

$$x(t) = e^{At} x(0) - \frac{1}{2} G(t) \lambda(t_\text{f}) + \int_0^t e^{A(t-\tau)} c \, \mathrm{d}\tau, \tag{13}$$

where $G(t) = \int_0^t e^{A(t-\tau)} B R^{-1} B^\top e^{A^\top (t_\text{f} - \tau)} \, \mathrm{d}\tau$.

Note that if $\lambda(t_\text{f})$ is known, then the problem can be solved with the control given by (10) and (12), and the state trajectory given by (13). Thus, the problem remains to determine $\lambda_1(t_\text{f})$. To this end, evaluate (13) at t_f to obtain

$$x(t_\text{f}) = \bar{x}(t_\text{f}) - \frac{1}{2} G(t_\text{f}) \lambda(t_\text{f}), \tag{14}$$

where

$$\bar{x}(t_\text{f}) \triangleq e^{At_\text{f}} x(0) + \int_0^{t_\text{f}} e^{A(t_\text{f} - \tau)} c \, \mathrm{d}\tau. \tag{15}$$

We may obtain $x_2(t_\text{f})$ and $\lambda_1(t_\text{f})$ by solving the linear equations (14). Using (9), rewrite (14) as

$$\begin{bmatrix} \bar{x}_1(t_\text{f}) - x_1(t_\text{f}) \\ \bar{x}_2(t_\text{f}) - x_2(t_\text{f}) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} G_{11}(t_\text{f}) & G_{12}(t_\text{f}) \\ G_{21}(t_\text{f}) & G_{22}(t_\text{f}) \end{bmatrix} \begin{bmatrix} \lambda_1(t_\text{f}) \\ 0 \end{bmatrix}, \tag{16}$$

where $\bar{x}(t_\text{f}) = [\bar{x}_1^\top(t_\text{f}) \ \ x_1^\top(t_\text{f})]^\top$. Note that $\bar{x}_1(t_\text{f}) - x_1(t_\text{f})$ is known and $\bar{x}_2(t_\text{f}) - x_2(t_\text{f})$ is unknown. Then, (16) becomes

$$2(\bar{x}_1(t_\text{f}) - x_1(t_\text{f})) = G_{11}(t_\text{f}) \lambda_1(t_\text{f}), \tag{17}$$

$$2(\bar{x}_2(t_\text{f}) - x_2(t_\text{f})) = G_{21}(t_\text{f}) \lambda_1(t_\text{f}). \tag{18}$$

Assuming (A,B) is controllable, it follows that $G(t_\text{f})$ is invertible and hence $G_{11}(t_\text{f})$ is invertible. From (17), we can solve for $\lambda_1(t_\text{f})$ as follows

$$\lambda_1(t_\text{f}) = 2G_{11}^{-1}(t_\text{f})(\bar{x}_1(t_\text{f}) - x_1(t_\text{f})). \tag{19}$$

Finally, from (10) and (12), the open-loop optimal control is given by

$$u(t) = -\frac{1}{2} R^{-1} B^\top e^{A^\top (t_\text{f} - t)} \lambda(t_\text{f}). \tag{20}$$

The corresponding optimal cost is

$$J(u^*) = t_\text{f} + \frac{1}{4} \lambda(t_\text{f})^\top G(t_\text{f}) \lambda(t_\text{f}). \tag{21}$$

B. The Optimal Arrival Time

Next, consider the case when t_f is free. In this case, we have the transversality condition [28]

$$H(t_\text{f}) = 0. \tag{22}$$

Substituting (10) into (5) and evaluating (5) at t_f, then (22) becomes

$$H(t_\text{f}) = 1 + \lambda(t_\text{f})^\top (Ax(t_\text{f}) + c) - \frac{1}{4} \lambda(t_\text{f})^\top B R^{-1} B^\top \lambda(t_\text{f}) = 0. \tag{23}$$

We find the optimal arrival time t_f by solving (23), which requires finding the roots of a polynomial [1].

C. PFF with Quadratic Terminal Penalty

In some cases, it may be desired to add implicit constraints on the free-final-state. Here, we extend the PFF controller by adding a quadratic penalty on the free-final-state to the cost function. Consider the PFF optimal control problem with the cost function,

$$J(u) = \frac{1}{2} x_2(t_\text{f})^\top S x_2(t_\text{f}) + \int_0^{t_\text{f}} (1 + u^\top Ru) \, \mathrm{d}\tau. \tag{24}$$

The necessary conditions for optimality for the PFF control problem (4) with this new cost function are still given by (6)-(8), except that (9) is now replaced by

$$\lambda_2(t_\text{f}) = \phi_x^\top (x(t_\text{f})) = S x_2(t_\text{f}), \tag{25}$$

where $\phi(x(t_\text{f})) = \frac{1}{2} x_2(t_\text{f})^\top S x_2(t_\text{f})$.

Following the same derivation as before, we get the same expression given by (14). The problem remains to solve for $\lambda(t_\text{f})$. Substituting (25) into (14), we get

$$\bar{x}(t_\text{f}) - x(t_\text{f}) = \frac{1}{2} G(t_\text{f}) \begin{bmatrix} \lambda_1(t_\text{f}) \\ S x_2(t_\text{f}) \end{bmatrix}, \tag{26}$$

which is equvalent to

$$\begin{bmatrix} \bar{x}_1(t_\text{f}) - x_1(t_\text{f}) \\ \bar{x}_2(t_\text{f}) \end{bmatrix} = M \begin{bmatrix} \lambda_1(t_\text{f}) \\ x_2(t_\text{f}) \end{bmatrix}, \tag{27}$$

where

$$M = \left(\frac{1}{2} G(t_{\mathrm{f}}) \begin{bmatrix} I & 0 \\ 0 & S \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix} \right). \tag{28}$$

Note that M is invertible. Thus, we can calculate $\lambda_1(t_{\mathrm{f}})$ and $x_2(t_{\mathrm{f}})$ from (27). Along with (25), we obtain $\lambda(t_{\mathrm{f}})$.

V. The Kino-RRT* Algorithm

In this section, we present the details of the Kino-RRT* algorithm, which is built on both the PFF controller and the fixed-final-state free-final-time controller. First, we summarize some primitive procedures used in the Kino-RRT* algorithm. Some of these primitive procedures follow the work in [4].

Sampling: The sampling procedure `SamplePFF` returns a partial state that is randomly sampled in a reduced state space and is collision-free in the corresponding reduced state space. For example, for a robot whose state space includes the position space and the velocity space, `SamplePFF` may sample a position of the robot that is collision-free.

Parent: `parent`(x) returns the parent node of x.

Nearest Neighbor: Given a tree $G = (V,E)$, where V is the node set and E is the edge set, the procedure `Nearest`(V,x) returns the node in V that is closest to the state x.

Near Nodes: The function `Near`(V,x,r) returns all the nodes in V that are contained in a ball of radius r centered at x.

Collision Checking: The function `CollisionFree`(τ) takes a trajectory τ (an edge segment) as an input and returns true if and only if τ lies entirely in the collision-free space. The function `CollisionPoint`(x) returns true if and only if the point x is collision-free.

Cost: The procedure `Cost`(x) returns the cost-to-come from the root node to x.

Segment Cost: The procedure `SegCost`(x_i,x_j) returns the cost to go from x_i to x_j. Depending on x_j, this cost is obtained by either solving the PFF control problem or the fixed-final-state free-final-time control problem.

Shrink: The procedure `Shrink`(x_i,x_j) returns x_j if the distance between x_i and x_j is less than or equal to ℓ. Otherwise, it returns a new state x_{new} that lies on the line formed by x_i and x_j and is at a distance ℓ away from x_i towards x_j. The `Shrink` procedure is consistent with the RRT* algorithm dictates that segments should have a maximum length ℓ. If one tries to connect two points that are far away, this connecting segment will collide with obstacles with a high probability.

Steering: The procedure `SteerPFF`(x_i,x_j) solves the TPBVP using the PFF optimal controller, and it returns a trajectory τ that starts from x_i and ends at x_j. The procedure `Steer`(x_i,x_j) solves the TPBVP using the fixed-final-state free-final-time controller, and it returns a trajectory τ that starts from x_i and ends at x_j. Note that x_j in `Steer`(x_i,x_j) is a point in the full state space, while x_j in `SteerPFF`(x_i,x_j) is a point in the reduced sampling space.

FreeState: The function `FreeSate` takes the trajectory τ returned by `SteerPFF`(x_i,x_j) as input and returns the rest of the state x_{free} at the endpoint of the trajectory that is not specified by x_j.

Algorithm 1: Kino-RRT*

$V \leftarrow \{x_{\mathrm{init}}\}$; $E \leftarrow \emptyset$; $\mathscr{G} \leftarrow (V,E)$;
for $i = 1 : N$ **do**
  $z_{\mathrm{rand}} \leftarrow$ `SamplePFF`;
  $x_{\mathrm{nearest}} \leftarrow$ `Nearest`(V, z_{rand});
  $z_{\mathrm{new}} \leftarrow$ `Shrink`$(x_{\mathrm{nearest}}, z_{\mathrm{rand}})$;
  if *CollisionPoint*(z_{new}) **then**
    $\tau \leftarrow$ `SteerPFF`$(x_{\mathrm{nearest}}, z_{\mathrm{new}})$;
    if *CollisionFree*(τ) **then**
      $x_{\mathrm{free}} \leftarrow$ `FreeState`(τ);
      $X_{\mathrm{near}} \leftarrow$ `Near`(V, z_{new}, r);
      $(x_{\mathrm{min}}, x_{\mathrm{free}}) \leftarrow$ `ChooseParent`$(X_{\mathrm{near}}, x_{\mathrm{nearest}}, z_{\mathrm{new}})$;
      $x_{\mathrm{new}} \leftarrow (z_{\mathrm{new}}, x_{\mathrm{free}})$;
      $V \leftarrow V \cup \{x_{\mathrm{new}}\}$;
      $E \leftarrow E \cup \{(x_{\mathrm{min}}, x_{\mathrm{new}})\}$;
      $E \leftarrow$ `Rewire`$(X_{\mathrm{near}}, E, x_{\mathrm{new}}, x_{\mathrm{min}})$;
      $\mathscr{G} \leftarrow (V,E)$;
return $\mathscr{G}$;

Algorithm 2: ChooseParent

ChooseParent $(X_{\mathrm{near}}, x_{\mathrm{nearest}}, z_{\mathrm{new}})$ **:**
  $x_{\mathrm{min}} \leftarrow x_{\mathrm{nearest}}$;
  $c_{\mathrm{min}} \leftarrow$ `Cost`(x_{nearest}) + `SegCost`$(x_{\mathrm{nearest}}, z_{\mathrm{new}})$;
  foreach $x_{\mathrm{near}} \in X_{\mathrm{near}} \setminus x_{\mathrm{nearest}}$ **do**
    if *Cost*(x_{near}) + *SegCost*$(x_{\mathrm{near}}, z_{\mathrm{new}}) < c_{\mathrm{min}}$ **then**
      $\tau \leftarrow$ `SteerPFF`$(x_{\mathrm{near}}, z_{\mathrm{new}})$;
      if *CollisionFree*(τ) **then**
        $x_{\mathrm{free}} \leftarrow$ `FreeState`(τ);
        $x_{\mathrm{min}} \leftarrow x_{\mathrm{near}}$;
        $c_{\mathrm{min}} \leftarrow$ `Cost`(x_{near}) + `SegCost`$(x_{\mathrm{near}}, z_{\mathrm{new}})$;
  return $(x_{\mathrm{min}}, x_{\mathrm{free}})$;

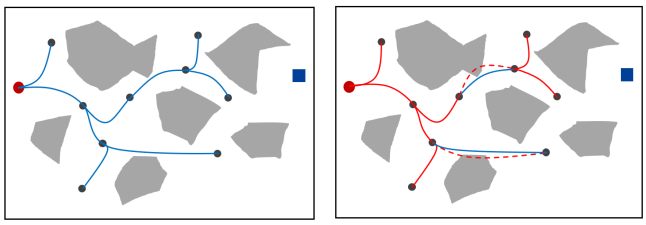
Algorithm 3: Rewire

Rewire $(X_{\mathrm{near}}, E, x_{\mathrm{new}}, x_{\mathrm{min}})$ **:**
  foreach $x_{\mathrm{near}} \in X_{\mathrm{near}} \setminus x_{\mathrm{min}}$ **do**
    if *Cost*(x_{new}) + *SegCost*$(x_{\mathrm{new}}, x_{\mathrm{near}}) <$ *Cost*(x_{near}) **then**
      $\tau \leftarrow$ `Steer`$(x_{\mathrm{new}}, x_{\mathrm{near}})$;
      if *CollisionFree*(τ) **then**
        $x_{\mathrm{parent}} \leftarrow$ `Parent`(x_{near});
        $E \leftarrow E \setminus \{(x_{\mathrm{parent}}, x_{\mathrm{near}})\}$;
        $E \leftarrow E \cup \{(x_{\mathrm{new}}, x_{\mathrm{near}})\}$;
  return E;

The complete algorithm is given by Algorithm 1, Algorithm 2, and Algorithm 3. We use z to denote a point in the reduced sampling space. The rest of the state (free-state) x_{free}, which comes from the endpoint of the state trajectory, is decided by the PFF optimal controller. After the `ChooseParent` step (line 11, Algorithm 1), the free-state is found and is combined with the sampled state to form a point in the full state space (line 12, Algorithm 1). Then, this point is added to the tree as a node (line 13, Algorithm 1).

A. Delayed and Intermittent Update of the Arrival Time

For both the PFF controller and the fixed-final-state controller, finding the optimal arrival time of the TPBVP requires solving a root-finding problem of a high-order polynomial (see (23)). This root-finding procedure will slow down the kinodynamic RRT* algorithm, as the TPBVP is required to be solved repeatedly. Here we propose a delayed and intermittent update of the optimal arrival time, which is shown in Figure 2. The planning algorithm first grows a tree using a heuristic of the arrival time (for example, by setting a desired average speed) without solving the root-finding problem (Figure 2(a)). Then, we intermittently update all the edges in the tree using the optimal arrival time (Figure 2(b)). If the updated edge is in-collision, we will use the original edge. We call this method KinoD-RRT*.



(a) (b)

Fig. 2: Delayed and intermittent update of the arrival time. (a) Grow a tree using a heuristic of the arrival time (blue lines). (b) Delayed update of the optimal arrival time (red lines). If the updated edge is in-collision (red dash lines), the original edge is used (blue lines).

VI. Experimental Results

We tested the Kino-RRT* algorithm on three kinodynamic systems: a 2D double integrator robot operating in a plane environment and a linearized quadrotor robot with a 10-dimensional state-space. We compared the Kino-RRT* algorithm with a variant of the kinodynamic RRT* algorithm. The only difference between the Kino-RRT* and the compared algorithm (a variant of kinodynamic RRT*) is the utilization of the PFF controller in Kino-RRT*. The compared kinodynamic RRT* algorithm samples the full state space and uses the fixed-final-state free-final-time controller to solve the TPBVPs. The gain of performance is solely due to the PFF controller. Thus, this comparison is informative.

A. Implementation Details

In kinodynamic RRT*, the near nodes are found by using the forward-reachable set or the backward-reachable set [1], [21]. Specifically, in line 12, Algorithm 1, `Near`(V,x,r) returns all nodes in V such that the cost J to go from these nodes to x is less than r (backward-reachable set). Check membership in the forward/backward reachable set for a set of nodes can be computationally expensive.

We use Euclidean distance to find the near nodes and the nearest node. In this case, the forward-reachable set and the backward-reachable set are the same. This essentially means that we do not use the true distance. For kinodynamic motion planning, the true distance between two states is the minimum cost J from the solution of the TPBVP [8]. Using the true distance, the forward (or backward) reachable set defines an ε-radius sub-Riemannian ball centered at x. It is showed in [25] that there always exists a certain size Euclidean hyper-ball inside such sub-Riemannian ball under mild conditions, which justifies the use of Euclidean norms. Euclidean distance is also used in [25]. After the nearest node and the near nodes are selected, the true distance is used in the `ChooseParent` and `Rewire` algorithms. The Euclidean distance is used only to pre-select relevant nodes and to help with the computations.

We also used a constant radius for the Euclidean hyper-ball for the near nodes, which implies a constant radius of the sub-Riemannian ball with respect to the true distance. Note that the kinodynamic RRT* is asymptotically optimal with a constant neighbor radius. The implementation is the same for the Kino-RRT* and the compared algorithm for an informative comparison. All the experiment examples are done on a laptop computer with an Intel Core i5-8250U 1.6 GHz CPU and 8 GB of RAM.

B. 2D Double Integrator

The state of the 2D double integrator is given by $x = [p^\top \; v^\top]^\top$, where $p = [x_1 \; x_2]^\top$ is the position and $v = [x_3 \; x_4]^\top$ is the velocity. The control input is the acceleration. The system matrices are given by

$$A = \begin{bmatrix} 0 & I_2 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I_2 \end{bmatrix}, \quad c = 0. \tag{29}$$

The weighting matrix in the cost function is set to $R = I_2$.

For both Kino-RRT* and kinodynamic RRT* the position is uniformly sampled within the boundary of the environment, that is, $p \in [0,20]^2$ m. The free-final-state of the PFF controller is the velocity. Thus, the Kino-RRT* algorithm does not sample the velocity space. For the kinodynamic RRT* algorithm, the velocity is uniformly sampled in $v \in [-2,2]^2$ m/s^2. Note that a larger interval for the velocity essentially requires searching in a larger state space, which will result in slower convergence. However, if the sampling velocity interval is too small, the search is confined to a small state space that may not contain the optimal solution. Here, the velocity interval is chosen to be small while containing the optimal solution.

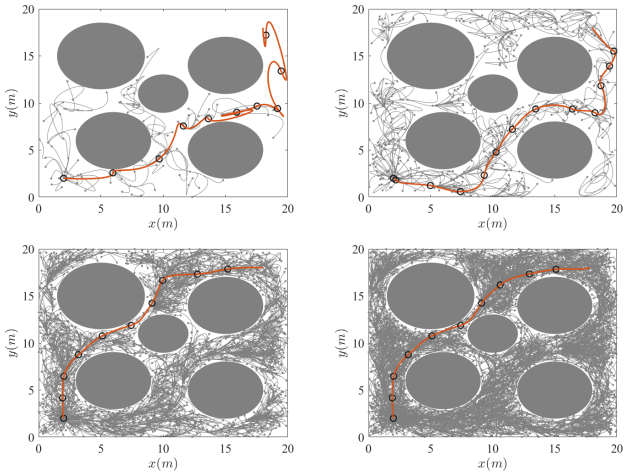


Fig. 3: Kinodynamic RRT* results of the 2D double integrator. The first figure corresponds to the first solution found. From the upper left to bottom right, the nodes expanded are 94, 400, 2000, 4000. The corresponding time to generate these trees are 0.053, 0.22, 2.38, 8.17 sec. The cost of best trajectory in the trees are 86.76, 47.56, 27.99, 25.72.

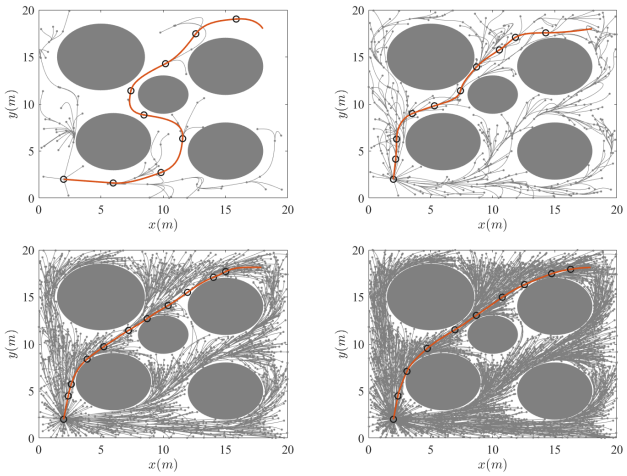


Fig. 4: Kino-RRT* results of the 2D double integrator. The first figure corresponds to the first solution found. From the upper left to bottom right, the nodes in the tree are 85, 400, 2000, 4000. The corresponding time to generate these trees are 0.015, 0.14, 2.12, 7.64 sec. The cost of best trajectory in the trees are 37.46, 25.97, 18.72, 16.42.

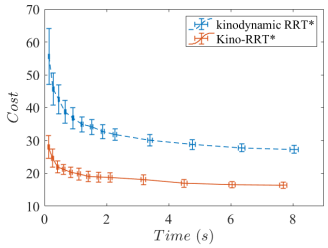
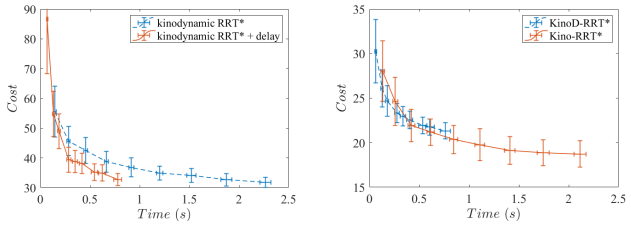


Fig. 5: Comparison of Kino-RRT* and kinodynamic RRT* for the 2D double integrator case.



(a) (b)

Fig. 6: Delayed and intermittent update of the optimal arrival time of the 2D double integrator

The results of the kinodynamic RRT* algorithm and the Kino-RRT* algorithm are given in Figure 3 and Figure 4, respectively. The comparison of the Kino-RRT* and the kinodynamic RRT* is shown in Figure 5. In Figure 5, we can see that our algorithm finds a better trajectory from the beginning (the first solution). In fact, the solution found by Kino-RRT* within 0.14 sec is comparable to the solution found by kinodynamic RRT* that took 8 sec after expanding 4000 nodes. After the Kino-RRT* finds the first solution, the cost enters a sharp decrease phase. For the kinodynamic RRT* algorithm, the cost curve is close to flat after 8 sec, and the probability of sampling good states to decrease the cost is low. Kino-RRT* is more than 50 times faster than the kinodynamic RRT* to find a trajectory with the same cost. By sampling in a reduced state-space, the solution returned by Kino-RRT* is close to the optimal solution after a few seconds of computation. However, for the kinodynamic RRT* algorithm, it is difficult to sample good velocities that are comparable to the ones chosen by the PFF controller, which leads to slow convergence.

Figure 6 shows the results of the delayed and intermittent update of the optimal arrival time. The Kinodynamic RRT* combined with the delayed and intermittent update of the optimal arrival time is called Kinodynamic RRT* with delay. Four methods, Kinodynamic RRT*, Kinodynamic RRT* with delay, Kino-RRT*, and KinoD-RRT*, are compared. Kinodynamic RRT* with delay is 3 times faster than Kinodynamic RRT* when expanding the same number of nodes. The planned trajectories have a similar cost for expanding the same number of nodes.

Kino-RRT* with delay is also 3 times faster than Kino-RRT* when expanding the same number of nodes. We can see that in Figure 6(b), KinoD-RRT* (blue dash line) finds better trajectory in the beginning because it can expand more nodes in a given time. However, Kino-RRT* outperform KinoD-RRT* after some point. This is because the velocities (free-final-state) chosen by KinoD-RRT* are not as optimized as the velocities chosen by Kino-RRT*. The velocity chosen by the PFF controller is affected by the arrival time. Non-optimal arrival times (which is the case with KinoD-RRT*) will result in a sub-optimal final velocity.

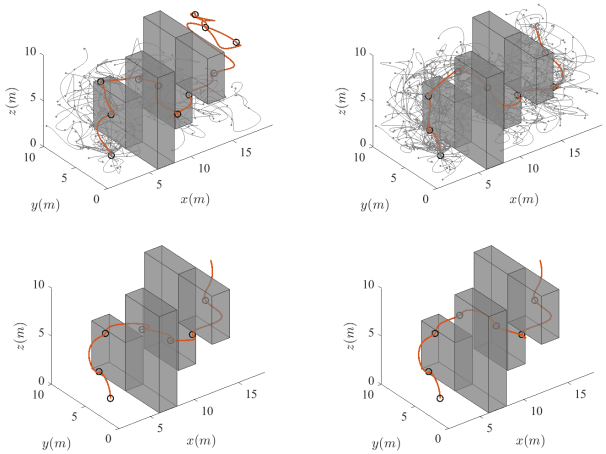


Fig. 7: Kinodynamic RRT* results of the quadrotor. The first figure corresponds to the first solution found.

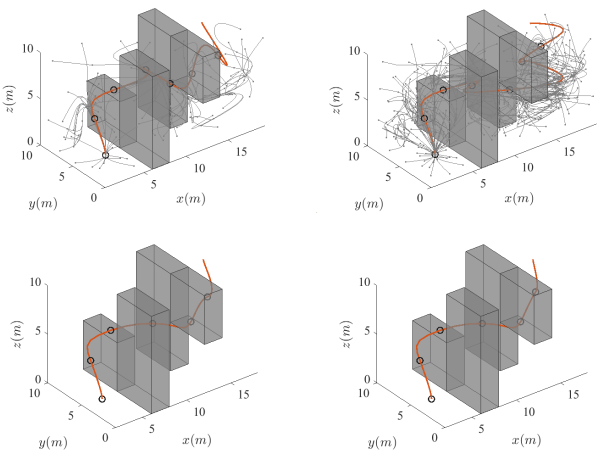


Fig. 8: Kino-RRT* results of the quadrotor. The first figure corresponds to the first solution found.

C. Linearized Quadrotor

A linearized quadrotor model adopted from [1] is used. The 10-dimensional state is given by $x = [p^\top \ v^\top \ r^\top \ w^\top]^\top$, which consists of the three-dimensional position p and velocity v, and the two-dimensional orientation r and angular velocity w. The yaw rotation, which is a redundant degree of freedom, is not considered in the model. The system matrices are given by

$$A = \begin{bmatrix} 0 & I_3 & 0 & 0 \\ 0 & 0 & \begin{bmatrix} 0 & g \\ -g & 0 \\ 0 & 0 \end{bmatrix} & 0 \\ 0 & 0 & 0 & I_2 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m} \end{bmatrix} & 0 \\ 0 & 0 \\ 0 & \frac{\ell I_2}{J} \end{bmatrix},$$

$$c = 0,$$

where g is the gravitational acceleration, m is the mass of the quadrotor, ℓ is the distance between the center of the vehicle and the rotors, and J is the moment of inertia about the axes coplanar with the rotors. The control input of the system is $u = [u_f \ u_x \ u_y]^\top$, where u_f is the total thrust of the rotors relative to the thrust needed for hovering, and u_x and u_y are the relative torques of roll and pitch, respectively.

The free-final-state of the PFF controller is v, r, and w. Thus the Kino-RRT* algorithm only samples the position space. Since the quadrotor is linearized at the hovering state and the dynamics is sensitive to the roll and pitch angles, we will use the PPF controller with quadratic terminal penalty introduced in Section IV-C. The terminal penalty matrix is $S = \text{diag}(0,0,0,20,20,0,0)$. The weighting matrix of the control is $R = \text{diag}(15,30,30)$.

For both Kino-RRT* and kinodynamic RRT* the position is uniformly sampled within the boundary of the 3D environment. The sampling intervals of v, r, and w for the kinodynamic RRT* are $v \in [-2,2]^3$ m/s, $r \in [-1,1]^2$ rad, and $w \in [-4,4]^2$ rad/s, respectively.

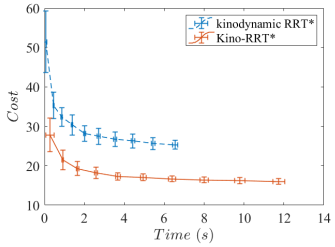


Fig. 9: Comparison of Kino-RRT* and kinodynamic RRT* for the linearized quadrotor.

The results of the kinodynamic RRT* algorithm and the Kino-RRT* algorithm are given in Figure 7 and Figure 8, respectively. In Figure 7, from upper left to bottom right, the number of nodes in the tree are 159, 400, 1000, 2000. The corresponding time to generate these trees are 0.147, 0.48, 1.92, 6.18 sec. The cost of the best trajectory in these trees are 58.10, 30.92, 24.84, 24.61, respectively. In Figure 7, from upper left to bottom right, the number of nodes in the tree are 133, 400, 1000, 2000. The corresponding time to generate these trees are 0.19, 0.84, 3.72, 11.35 sec. The cost of the best trajectory in these trees are 20.31, 19.13, 15.56, 15.52, respectively. The comparison of Kino-RRT* and kinodynamic RRT* is shown in Figure 9. The solution of the PPF controller with quadratic terminal penalty is more complex than the fixed-final-state free-final-time controller. Thus, the Kino-RRT* algorithm takes more time to expand the same number of nodes compared to the kinodynamic RRT*. Because each node in Kino-RRT* is more optimized, it still converges faster than the kinodynamic RRT*.

Figure 10 shows the results of the delayed and intermittent update of the optimal arrival time. For the linearized quadrotor example, the kinodynamic RRT* with delay is 2 times faster than the kinodynamic RRT* when expanding the same number of nodes, and is also 2 times faster for finding a trajectory with a similar cost. Similar performance improvement is observed for the KinoD-RRT* compared to

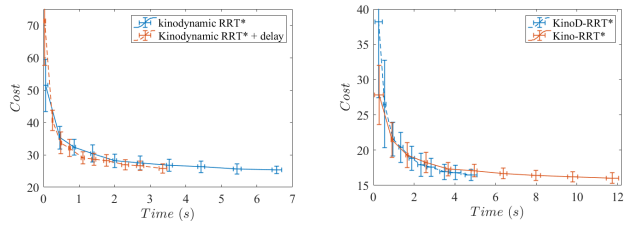


Fig. 10: Delayed and intermittent update of the optimal arrival time for the linearized quadrotor example.

Kino-RRT*. This performance improvement depends on the heuristic of the arrival time for KinoD-RRT*.

VII. Conclusion

In this paper, we developed the Kino-RRT* algorithm, which utilizes a partial-final-state-free (PFF) optimal controller to improve the convergence performance of sampling-based motion planning of kinodynamic systems. Instead of sampling the full state of the robot, Kino-RRT* only samples part of the state-space and the rest of the states are optimized by the PFF optimal controller. Although the algorithm is demonstrated on linear systems, the idea of PFF can be used as in [18], [11], [12] for nonlinear kinodynamic systems as well. We tested the algorithm on robot systems with 4-D and 10-D state-spaces. In all cases, Kino-RRT* showed better convergence compared to the standard kinodynamic RRT*, achieving trajectories with better cost using much less time to compute. The proposed Kino-RRT* algorithm shows potential in real-time kinodynamic motion planning for high-dimensional dynamical systems.

References

[1] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *IEEE International Conference on Robotics and Automation*, (Karlsrühe, Germany), pp. 5054–5061, May 2013.

[2] S. M. LaValle, "Motion planning: Wild frontiers," *IEEE Robotics Automation Magazine*, vol. 18, no. 2, pp. 108–118, 2011.

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

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[5] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 1478–1483, May 2011.

[6] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[7] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 1–25, 2021.

[8] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control*, (Atlanta, GA), pp. 7681–7687, December 2010.

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Chicago, IL), pp. 2997–3004, September 2014.

[10] 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 *IEEE International Conference on Robotics and Automation*, (Saint Paul, MN), pp. 2537–2542, May 2012.

[11] W. J. Wolfslag, M. Bharatheesha, T. M. Moerland, and M. Wisse, "RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1655–1662, 2018.

[12] D. Zheng and P. Tsiotras, "Sampling-based kinodynamic motion planning using a neural network controller," in *AIAA Scitech 2021 Forum*, p. 1754, 2021.

[13] H. T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[14] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch Informed Trees (BIT*): informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[15] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[16] B. Paden, V. Varricchio, and E. Frazzoli, "Verification and synthesis of admissible heuristics for kinodynamic motion planning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 648–655, 2017.

[17] D. Yi, R. Thakker, C. Gulino, O. Salzman, and S. Srinivasa, "Generalizing informed sampling for asymptotically-optimal sampling-based kinodynamic planning via markov chain monte carlo," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Brisbane, Australia), pp. 7063–7070, 2018.

[18] C. Xie, J. van den Berg, S. Patil, and P. Abbeel, "Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver," in *IEEE International Conference on Robotics and Automation*, (Seattle, WA), pp. 4187–4194, May 2015.

[19] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *2013 IEEE International Conference on Robotics and Automation*, (Karlsrühe, Germany), pp. 5041–5047, 2013.

[20] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the driftless case," in *IEEE International Conference on Robotics and Automation*, (Seattle, WA), pp. 2368–2375, 2015.

[21] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics," in *54th IEEE Conference on Decision and Control*, (Osaka, Japan), pp. 2574–2581, 2015.

[22] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *2013 IEEE International Conference on Robotics and Automation*, (Karlsrühe, Germany), pp. 2429–2436, 2013.

[23] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[24] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.

[25] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[26] A. Sivaramakrishnan, Z. Littlefield, and K. E. Bekris, "Towards learning efficient maneuver sets for kinodynamic motion planning," *arXiv preprint, arXiv:1907.07876*, 2019.

[27] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "MPC-MPNet: model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *arXiv preprint arXiv:2101.06798*, 2021.

[28] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. John Wiley & Sons, 2012.